RAPID PROTOTYPING OF THE SPHERE DECODER FOR MIMO SYSTEMS

Luis G. Barbero and John S. Thompson

Institute for Digital Communications
University of Edinburgh
Email: {l.barbero, john.thompson}@ed.ac.uk

Abstract

The use of multiple input-multiple output (MIMO) technology is rapidly becoming the new frontier of wireless communication systems increasing their capacity and spectral efficiency. In order to validate this technology from an implementation point of view, field-programmable gate arrays (FPGAs), with their high level of parallelism, high densities and embedded multipliers, are a suitable platform for the study and prototyping of MIMO algorithms. This paper presents an FPGA implementation of the sphere decoder (SD) for MIMO detection. This algorithm provides optimal maximum likelihood (ML) performance with reduced complexity, compared to the maximum likelihood detector (MLD).

1 Introduction

In the last seven years, the use of multiple input-multiple output (MIMO) technology in wireless links has been extensively studied, mostly from a theoretical point of view, showing that significant capacity increases could be achieved under certain conditions by using multiple antennas at both transmitter and receiver [5]. For the uncoded MIMO case, the sphere decoder (SD) is widely considered the most promising approach to obtain optimal maximum likelihood (ML) performance with reduced complexity [12, 3].

Nowadays, the prototyping of those multiple-antenna systems has become increasingly important to verify the enhancements advanced by analytical results [9, 8]. However, in most cases, the target platform is rarely used as feedback to investigate ways of optimizing the algorithm. The main aim of our rapid prototyping approach is to speed up the initial implementation of the SD to be able to study possible optimizations from an algorithmic point of view using the real-time prototype.

Although application-specific integrated circuit (ASIC) implementations of the SD exist [2], this paper presents what, to the best of our knowledge, is the first FPGA implementation of the SD using a rapid prototyping methodology.

2 Sphere Decoder (SD)

2.1 MIMO System Model

The system model considered has M transmit and N receive antennas, with $N \geq M$, denoted as MxN. The transmitted symbols are taken independently from a quadrature amplitude modulation (QAM) constellation of P points. The received N-vector, using matrix notation, is given by

$$\mathbf{r} = \mathbf{Hs} + \mathbf{n} \qquad (1)$$

where $\mathbf{s} = (s_1, s_2, ..., s_M)^T$ denotes the vector of transmitted symbols with $\mathrm{E}[|s_i|^2] = 1/M$, $\mathbf{n} = (n_1, n_2, ..., n_N)^T$ is the vector of independent and identically distributed (i.i.d.) complex Gaussian noise samples with variance $\sigma^2 = N_0$ and $\mathbf{r} = (r_1, r_2, ..., r_N)^T$ is the vector of received symbols. $\mathbf{H}$ denotes the NxM channel matrix where h_{ij} is the complex transfer function from transmitter j to receiver i. The entries of $\mathbf{H}$ are modelled as i.i.d. Rayleigh fading with $\mathrm{E}[|h_{ij}|^2] = 1$ and are perfectly estimated at the receiver.

2.2 SD Algorithm

The complex version of the SD [7] is used, given that, compared to the real version, it has a speed advantage and results in a more efficient hardware implementation [2]. The main idea behind the SD is to reduce the computational complexity of the MLD by searching over only those noiseless received points (defined as $\mathbf{Hs}$) that lie within a hypersphere of radius R around the received signal $\mathbf{r}$. This process can be written as

$$\hat{\mathbf{s}}_{\mathbf{ml}} = \arg\{\min_{\mathbf{s}} \|\mathbf{U}(\mathbf{s} - \hat{\mathbf{s}})\|^2 \leq R^2\} \qquad (2)$$

where $\mathbf{U}$ is an MxM upper triangular matrix, with entries denoted u_{ij}, obtained through Cholesky decomposition of the Gram matrix $\mathbf{G} = \mathbf{H}^H\mathbf{H}$ and $\hat{\mathbf{s}} = \mathbf{H}^\dagger\mathbf{r}$ is the unconstrained ML estimate of $\mathbf{s}$ where $\mathbf{H}^\dagger = (\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ is the pseudoinverse of $\mathbf{H}$.

The solution of the sphere constraint (SC) in (2) can be obtained recursively using a tree search algorithm, starting from $i = M$ and working backwards until $i = 1$. For each level, the constellation points s_i that satisfy

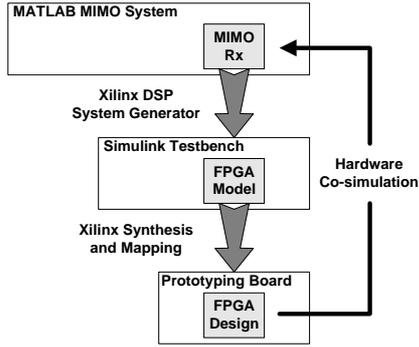$$|s_i - z_i|^2 \leq \frac{T_i}{u_{ii}^2} \qquad (3)$$

Fig. 1. Rapid prototyping methodology

are selected as partial ML candidates, where

$$z_i = \hat{s}_i - \sum_{j=i+1}^{M} \frac{u_{ij}}{u_{ii}}(s_j - \hat{s}_j) \qquad (4)$$

and

$$T_i = R^2 - \sum_{j=i+1}^{M} u_{jj}^2 |s_j - z_j|^2 . \qquad (5)$$

When a new point is found inside the hypersphere (at $i = 1$) the radius is updated with the new minimum Euclidean distance and the algorithm continues the search with the new SC. This process can be seen as a tree search through M levels where each level contains P nodes and each node has P branches. The leaves at the bottom ($i = 1$) correspond to all possible vector symbols $\mathbf{s}$ with their associated Euclidean distance. When, in any level i, $T_i \le 0$, the accumulated (squared) Euclidean distance (AED) from the root to that node has exceeded the SC and the entire branch plus all its descendants can be discarded, yielding a speed increase compared to an exhaustive search. The search finishes when the radius has been reduced so that no more points are found that satisfy the SC: the last point found satisfying the SC is the ML solution $\hat{\mathbf{s}}_{\mathbf{ml}}$.

Two factors are important in order to achieve the speed increase of the SD:

- The initial radius, R, is chosen according to the noise variance per antenna, σ^2, so that the probability of not finding a point inside the hypersphere is negligible.

- The points that satisfy (3) are searched according to increasing distance to z_i, following the Schnorr-Euchner (SE) enumeration [10], reducing the number of operations required to find the ML solution.

Recently, different alternatives have been proposed to further reduce the complexity of the tree search in the SD by preprocessing the channel matrix [3]. Among them, the methods that perform an ordering of the columns of the channel matrix using the vertical Bell Labs layered space time (VBLAST) architecture combined with the zero forcing (ZF) [14] or the minimum mean-square error (MMSE) [4] criterion are of special interest from an implementation point of view.
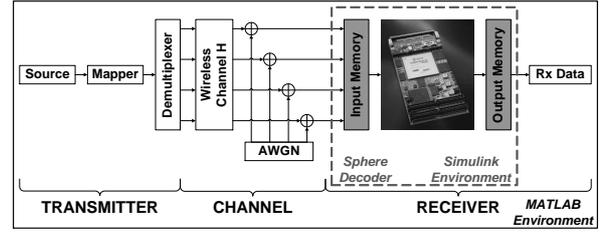

Fig. 2. Hardware-in-the-loop MIMO system diagram

3 Rapid Prototyping System

The rapid prototyping system used has the simplicity and, at the same time, the flexibility required to move quickly from a computer-based implementation of an algorithm to its real-time implementation. As opposed to previous prototyping approaches, the focus of our approach is on the analysis of the MIMO algorithm.

3.1 Hardware Platform

The FPGA platform has been provided by Alpha Data Ltd. [1], the company that partially sponsors this work. It consists of an ADC-PMC peripheral component interconnect (PCI) adapter board that hosts two FPGA boards: an ADM-XRC-II with a Xilinx Virtex-II (XC2V4000) and an ADM-XP with a Xilinx Virtex-II-Pro (XC2VP70), both with external SRAM memory for data storage.

3.2 Rapid Prototyping Methodology

The rapid prototyping methodology selected is based on The Mathwork's MATLAB and Simulink [11] and Xilinx's DSP System Generator [15] tailored to Alpha Data's FPGA boards. Fig. 1 shows the methodology used for the rapid prototyping of the SD.

Initially, MATLAB is used to implement a complete MIMO system including transmitter, channel simulator and receiver. The SD is then implemented on the FPGA using the DSP System Generator. The tool is embedded in Simulink and provides different blocks to perform basic mathematical and bit operations that can be directly mapped on the FPGA for real-time execution.

The development of the FPGA model is embedded in a Simulink testbench that facilitates the debugging of the SD in the development stage, with the possibility of monitoring every signal in the FPGA model.

The SD design is then synthesized for the FPGA using Xilinx synthesis tools. This hardware design and a Simulink-based memory interface are integrated into the MATLAB system as shown in Fig. 2. This rapid prototyping methodology allows us to quickly implement the SD on an FPGA and perform real-time hardware-in-the-loop testing of the algorithm.
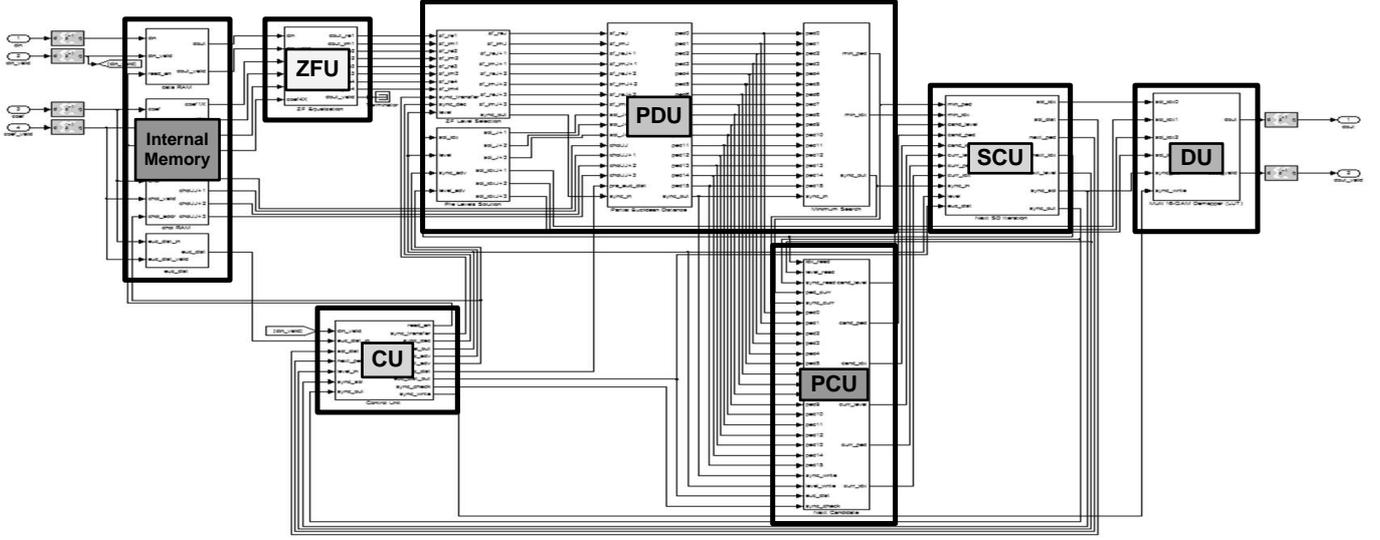
Fig. 3. FPGA block diagram of the SD

4 FPGA Implementation

The FPGA implementation of the SD is based on the fact that (3) can be rewritten as

$$D_i = d_i + D_{i+1} \leq R^2 \qquad (6)$$

where

$$D_{i+1} = \sum_{j=i+1}^{M} u_{jj}^2 |s_j - z_j|^2 \qquad (7)$$

can be seen as an AED down to level $j = i+1$ with $D_{M+1} = 0$ and

$$d_i = u_{ii}^2 |s_i - z_i|^2 \qquad (8)$$

can be seen as the partial (squared) Euclidean distance (PED) contribution from level i. Therefore, on each level i, the value D_i is calculated to obtain which points s_i are selected to continue the tree search. One alternative to implement (6) is to calculate the P different d_i in parallel (for the s_i points belonging to the P-QAM constellation), add them to D_{i+1} and check which ones satisfy the SC. For higher order constellations, this computationally expensive approach could be simplified using a method presented in [7] to directly enumerate the points that satisfy (3) and reduce the number of d_i calculations.

The complete tree search performed by the FPGA implementation of the SD is described below (starting from $i = M$):

1. A set of P values D_i is calculated. The minimum of these values is obtained, representing the first point in the SE enumeration for that level i.

2. The rest of the s_i and their associated D_i are saved in increasing order into a partial candidates memory for level i, in case they need to be visited later in the detection process.

3. The minimum D_i obtained in step 1) is checked against the SC:

a) If $D_i \leq R^2$ and $i \neq 1$, goto step 1) with $i \leftarrow i - 1$.

b) If $D_i \leq R^2$ and $i = 1$, a new solution has been found. $R^2 \leftarrow D_1$ and goto step 4).

c) If $D_i > R^2$ and $i \neq M$, goto step 4).

d) If $D_i > R^2$ and $i = M$, goto step 5).

4. The candidates memory from previous levels ($i_{cand} = i + 1, \ldots, M$) is searched to obtain the partial candidate with $D_{i_{cand}} \leq R^2$ closer to completion (i.e. lower i_{cand}). If a partial candidate is found, the detection process continues, goto step 1) with $i \leftarrow i_{cand} - 1$. If no candidate is found, goto step 5).

5. The detection process has finished and the last solution found is the ML solution.

From an algorithmic point of view, this implementation of the SD guarantees that no node in the tree is evaluated twice and that, in every loop of the algorithm from step 1) to step 5), a new node in the tree is evaluated. This minimizes the number of steps required in the tree search.

4.1 SD Architecture

Fig. 3 shows the block diagram of the FPGA implementation of the SD where the only blocks left out are the input and output memories used for synchronization with the Simulink environment. The function of the different blocks of the design is described below.

Internal Memory: This block contains intermediate memory to store the received symbols $\mathbf{r}$, the entries of the pseudo-inverse of the channel matrix, $\mathbf{H}^{\dagger}$, the entries of the Cholesky decomposition of the Gram matrix, $\mathbf{U}$, and the initial squared radius R^2.

Zero Forcing Unit (ZFU): This block performs the ZF equalization to obtain $\hat{\mathbf{s}} = \mathbf{H}^{\dagger}\mathbf{r}$ every time a new MIMO symbol

needs to be detected. This is performed in parallel with the detection of the previous MIMO symbol in order to reduce the latency and increase the overall throughput of the system.

Partial Distance Unit (PDU): This block performs the two tasks of step 1). It calculates the values D_i for each level i. Given that D_{i+1} is an input to the block, the process is reduced to obtaining the P different d_i that can be written as

$$d_i = u_{ii}^2 \left| s_i - \hat{s}_i + \sum_{j=i+1}^{M} \frac{u_{ij}}{u_{ii}}(s_j - \hat{s}_j) \right|^2. \qquad (9)$$

As noted in [2], if we define

$$a = s_i \ , \ b = -\hat{s}_i + \sum_{j=i+1}^{M} \frac{u_{ij}}{u_{ii}}(s_j - \hat{s}_j), \qquad (10)$$

the expression in (9) can be rewritten as

$$d_i = u_{ii}^2 \left(|a|^2 + 2\Re(b^*a) + |b|^2 \right) \qquad (11)$$

where the number of operations required to calculate (11) can be reduced taking into account that $a = s_i$ corresponds to the points of the P-QAM constellation. In particular, for the case of 16-QAM, the term $|a|^2$ can only have three different values that can be precalculated and stored as constants. In addition, the 16 different combinations of $\Re(b^*a)$ can be obtained through

$$\Re(b^*a) = \pm\Re(b) \cdot \{1,3\} \pm \Im(b) \cdot \{1,3\} \qquad (12)$$

where only two real multiplications are required. Therefore, the most computationally intensive parts are the calculation of b and $|b|^2$. In addition, this block searches for the minimum value of D_i representing the first point in the SE enumeration.

Partial Candidates Unit (PCU): This block stores the distances $D_{i_{cand}}$ obtained in the PDU for levels $i_{cand} = 2,\ldots,M$. In total, $(M-1) \times P$ values are stored. In an intermediate step, the block performs the SE of the candidates for each level i_{cand}. This is done by searching always for the minium distance $D_{i_{cand}}$ of the points that have not been previously visited by the tree search. The resulting $M-1$ values are stored in an intermediate cache memory.

This block also obtains the next candidate s_{next} that needs to be searched among the values stored in the cache memory. The selected value must satisfy the SC and be the one closer to completion (i.e. lower $i_{cand} > i$). This process corresponds to step 4).

Sphere Constraint Unit (SCU): This block checks if the AED D_i of the point s_i obtained in the PDU satisfies the SC. Depending on the result of this check and the current level i, this unit selects between the point s_i and the candidate s_{next} from the PCU as the next input for the PDU. Additionally, it indicates the control unit (CU) which level needs to be detected next.

Control Unit (CU): This block is responsible for the transition between the levels. It reads the channel coefficients ($\mathbf{H}^\dagger$ and $\mathbf{U}$) that are required in every iteration. In addition, it controls in which point of the detection process the SD is, synchronizing the other blocks appropriately.
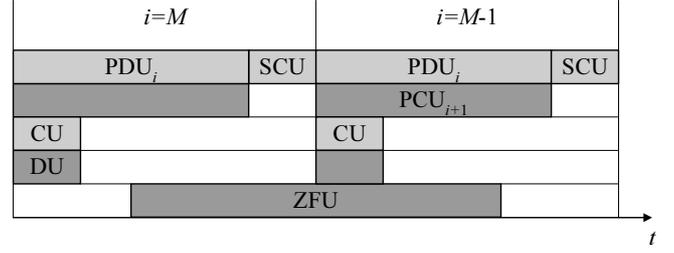

Fig. 4. FPGA time diagram of the SD

Demapper Unit (DU): This block performs the P-QAM demapping of the ML solution $\hat{\mathbf{s}}_{\mathbf{ml}}$.

4.2 SD Scheduling

From a hardware point of view, the SD makes use of the inherent parallelism of the FPGA platform. The independent parts of the algorithm have been scheduled to run in parallel, therefore reducing the number of blocks that form part of the critical path. This reduction in the critical path results in an increase in the overall throughput of the system.

Fig. 4 shows the time diagram of the SD algorithm on the FPGA. The diagram represents two iterations of the SD, for $i = M$ and $i = M - 1$, showing when the different blocks are active. The light grey rectangles represent the blocks that are executed in every iteration of the SD. On the other hand, the dark grey rectangles represent the blocks that are not executed in every iteration of the SD, resulting in partially used hardware resources. Finally, the white spaces represent unused hardware resources [1].

It can be seen that the two most computationally intensive blocks, PDU and PCU, can be pipelined with one iteration-delay. While the PDU is calculating the AEDs for level i, the PCU obtains the SE enumeration of the candidates from level $i + 1$. When $i = M$, the PCU is not executed, indicated by a dark grey rectangle with no label on it.

The ZFU is executed only when $i = M$ and extends into the following iteration. It precalculates $\hat{s}$ for the next MIMO symbol to be detected. The DU is only executed when a solution has been found and the detection process for the MIMO symbol has finised (i.e. $i = M$ and the next MIMO symbol starts to be detected).

The critical path of the algorithm is formed by the PDU and the SCU, directly determining the throughput of the system. The white spaces and the dark grey blocks in the diagram indicate a suboptimum use of the FPGA resources available. This is due to the interdependency between the different blocks that makes difficult the process of mapping the SD into a high throughput, highly-pipelined implementation. In addition, the light grey blocks contain sequential subblocks that can not be fully pipelined, also representing a suboptimum use of the resources.

---

[1] The term "unused or partially used hardware resources" means that a part of the design is running but processing data not relevant for the detection process, therefore representing a suboptimum use of the resources.

5 Results

The SD has been implemented for a 4x4 system using 16-QAM modulation. The FPGA design has been integrated into the MATLAB system model in order to perform hardware co-simulation of the algorithm and compare the real-time fixed-point performance with the floating-point MATLAB one.

5.1 FPGA Resource Use

The resource use of the parallel implementation of 4 SDs on the Xilinx Virtex-II-Pro FPGA board is summarized in Table 1. The integration of the 4 SDs uses approximately half of the FPGA resources making intensive use of the RAM memory blocks. The number of memory blocks used is due to the input and output buffers defined on the FPGA to synchronize the FPGA board with the Simulink interface and the internal memory requirements of the SD.

The number of multipliers can be used as an indicator of the computational complexity of the algorithm. Each single SD uses 39 embedded multipliers: 16 multipliers in the ZFU and 23 multipliers in the PDU. It should be noted that the number of multipliers could be reduced by reusing the multipliers when they are idle. In addition, an approximation of the Euclidean metric like the Manhattan distance could be used in order to reduce the number of multipliers in the PDU at the cost of a performance degradation [2].

The percentage of slices used can be seen as an indicator of the amount of control logic and intermediate buffers required in the SD. It should be noted that each slice contains two flip-flops and two look-up tables (LUTs) and that, looking at their percentage of use, we can see that a high percentage of the slices are only partially used. However, the high percentage usage of LUTs gives an idea of the *irregularities* of the SD, factor that affects its mapping on hardware and the resulting throughput.

5.2 Performance Results

The bit error ratio (BER) performance of the SD has been evaluated in real-time using 10,000 channel realizations with 200 symbols transmitted in every channel realization. and is shown in Fig. 5. The pseudoinverse and Cholesky decomposition of the channel are calculated offline in MATLAB. The input values to the SD are quantized using 16 bits per real component. The initial radius is set to the end of the scale to always find a point inside the hypersphere.

It can be seen that the FPGA performance approximately matches that of MATLAB, a difference only appears for high signal to noise ratio (SNR) due to the quantization process. The SD on the FPGA has also been simulated using VBLAST-ZF and VBLAST-MMSE channel matrix ordering. In floating-point, both offer a reduction in complexity, although the latter incurs in a slight performance degradation [3]. The channel ordering is performed offline in MATLAB. The aforementioned performane degradation is only noticeable at low to medium

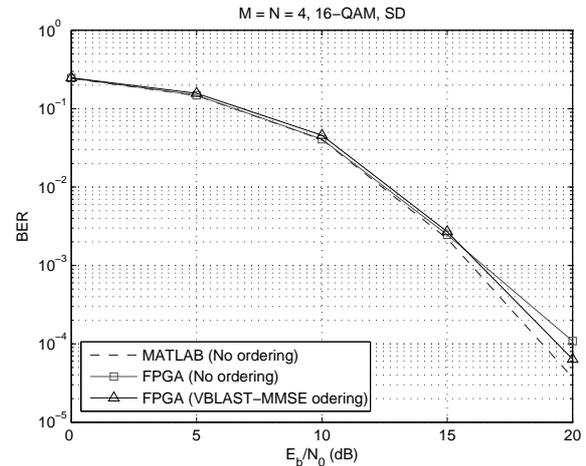| XC2VP70	Use	Percentage
Number of slices	21,467 / 33,088	64%
Number of flip-flops	17,691 / 66,176	26%
Number of 4-input LUTs	36,249 / 66,176	54%
Number of multipliers	156 / 328	47%
Number of block RAM	183 / 328	55%

Table 1. FPGA resource use of 4-SDs

Fig. 5. BER performance of the SD in MATLAB and on the FPGA as a function of the SNR per bit.

SNR. At high SNR however, the performance is actually improved, showing that the VBLAST-MMSE ordering results in a more robust SD implementation for the same fixed-point precision. Simulation results have shown that the SD fixed-point performance with VBLAST-ZF ordering is similar to that of the SD with no ordering.

Fig. 6 shows the average throughput of the SD for different channel matrix orderings. The throughput in megabits per second (Mbps) is calculated according to

$$Q_{avg} = 4 \cdot M \cdot \log_2 P \cdot f_{clock} / C_{avg} \quad \text{(Mbps)} \qquad (13)$$

where f_{clock} is the clock frequency of the design in MHz and C_{avg} is the average number of clock cycles required to detect a MIMO symbol. For this design, $f_{clock} = 50$ MHz and the minimum number of cycles is $C_{min} = 25$ resulting in a maximum throughput $Q_{max} = 128$ Mbps. Increasing the clock frequency would not result in a direct increment in the throughput because the average number of cycles required for detection would also increase. Therefore, the quotient f_{clock} / C_{avg} could be seen as an indicator of the level of optimization of the hardware design.

The results in Fig. 6 show that the throughput of the SD is not constant and depends on the noise level and also the channel conditions. The two ordering alternatives considered increase the throughput especially for low SNR. These ordering methods would cause an increase in complexity in the receiver although it could be considered negligible for packet-
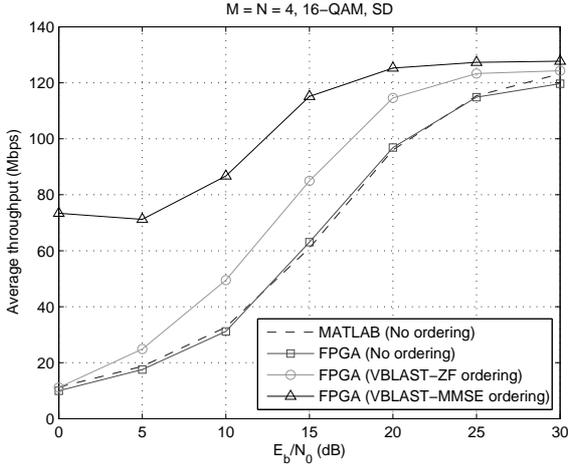
Fig. 6. Average throughput of the SD with different orderings of the channel matrix as a function of the SNR per bit.

SD	FPGA	ASIC 1 [2]	ASIC 2 [2]
MIMO system	4x4	4x4	4x4
Modulation	16-QAM	16-QAM	16-QAM
Granularity	4-SD	Single-SD	Single-SD
BER perfomance	ML	ML	close to ML
f_{clock} (MHz)	50	51	71
Q_{avg} (Mbps) at $E_b/N_0 = 20$dB	114.5	126	253

Table 2. Comparison of real-time SD implementations

based communications where the ordering is only performed once per frame. In particular, VBLAST-MMSE ordering provides the largest throughput increase though it should be noted that this method requires an estimate of the noise level in the receiver, difficulting its integration into a practical system. Generally, the non-deterministic throughput of the SD is the main problem when integrating it into a complete communication system where data needs to be detected in a fixed number of operations.

The theoretical throughput of a floating-point implementation of the SD with no ordering is plotted for comparison purposes. It can be seen how the quantization process also has an effect on the achievable throughput at high SNR. This is due to the effect the quantization has on the SC, allowing for additional points to be considered as candidates once a solution has been found.

The FPGA implementation with VBLAST-ZF ordering has been compared with previous ASIC implementations of the SD in Table 2. Although other implementations exist trying to obtain a constant throughput in the SD [6, 13], their throughput is lower while incurring in significantly higher computational complexity and memory requirements.

The FPGA implementation achieves a similar performance to that of ASIC 1. The system could be improved in terms of

throughput adding more SDs in parallel on the same platform or simply using the VBLAST-MMSE ordering method. The main difference between the two implementations is that, apart from the preprocessing used, for each level i, the equivalent PDU of ASIC 1 does not perform a minimum search. It directly preselects the point closer to the ZF solution to continue the tree search resulting in a throughput increase, even though more points need to be searched for moderate SNRs to find the ML solution [2].

On the other hand, ASIC 2 uses an l^∞-norm approximation for the Euclidean distance calculation resulting in a throughput and clock frequency increase while having only a small performance degradation. In addition, it uses a scheme for direct SE enumeration of the points based on the method in [7], contributing to the throughput and clock frequency increase. These optimizations could also be integrated into our FPGA design improving the throughput of the SD.

6 Conclusion and Future Work

An FPGA implementation of the SD using a rapid prototyping methodology has been presented in this paper. The advantage of the rapid prototyping methodology is the flexibility that provides to analyze in detail the hardware implementation of the SD while running it in real-time.

It has been shown that the performance of the FPGA implementation matches that of a previously presented ASIC implementation. Although improvements exist that could be added to the FPGA, they are based on mathematical approximations and hardware optimizations. In order to further improve hardware implementations of the SD and make its integration into a practical system easier, we need to identify the bottlenecks of the system from an algorithmic point of view. The two major drawbacks of a hardware implementation of the SD are:

- The tree search of the algorithm makes its throughput dependent on the noise level and the channel conditions. This can greatly affect the performance of a complete communication system where data needs to be detected in a fixed number of operations.

- The resource use of the FPGA is suboptimal due to the sequential nature of the algorithm. It has been shown that the algorithm, with only some parts of the design processing valid data at the same time, can not be fully pipelined.

This deep understanding of the SD thanks to the prototyping experience can be used as a means of identifying more optimized algorithms that could overcome the main drawbacks of the SD without greatly affecting its performance. We believe that a compromise can be established between the performance and the complexity to dramatically increase the throughput of the SD. This last aspect is the main subject of ongoing work.

Acknowledgment

References

[1] Alpha Data Ltd. http://www.alpha-data.com.

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

[3] M. O. Damen, H. E. Gamal, and G. Caire. On maximum-likelihood detection and the search for the closest lattice point. *IEEE Trans. Inform. Theory*, 49(10):2389–2402, Oct. 2003.

[4] M. Debbah, B. Muquet, M. de Courville, M. Muck, S. Simoens, and P. Loubaton. A MMSE successive interference cancellation scheme for a new adjustable hybrid spread OFDM system. In *Proc. 51st IEEE Vehicular Technology Conference (VTC '00-Spring)*, volume 2, pages 745–749, Tokyo, Japan, May 2000.

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

[6] Z. Guo and P. Nilsson. A VLSI architecture of the Schnorr-Euchner decoder for MIMO systems. In *Proc. IEEE 6th Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication*, volume 1, pages 65–68, Shanghai, China, June 2004.

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

[8] T. Kaiser, A. Wilzeck, M. Berentsen, and M. Rupp. Prototyping for MIMO systems: An overview. In *Proc. 12th European Signal Processing Conference (EUSIPCO '04)*, Viena, Austria, Sept. 2004.

[9] M. Rupp, A. Burg, and E. Beck. Rapid prototyping for wireless designs: the five-ones approach. *Signal Processing*, 83:1427–1444, 2003.

[10] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–191, 1994.

[11] The Mathworks, Inc. http://www.mathworks.com.

[12] E. Viterbo and J. Boutros. A universal lattice code decoder for fading channels. *IEEE Trans. Inform. Theory*, 45(5):1639–1642, July 1999.

[13] K. wai Wong, C. ying Tsiu, R. S. kwan Cheng, and W. ho Mow. A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS '02)*, volume 3, pages 273–276, Scottsdale, AZ, May 2002.

[14] P. W. Wolniansky, G. J. Foschini, G. D. Golden, and R. A. Valenzuela. V-BLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel. In *Proc. URSI International Symposium on Signals, Systems and Electronics (ISSSE '98)*, pages 295–300, Atlanta, GA, Sept. 1998.

[15] Xilinx, Inc. www.xilinx.com.