Rapid Prototyping of a Fixed-Throughput 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*— A field-programmable gate array (FPGA) implementation of a new detection algorithm for uncoded multiple input-multiple output (MIMO) systems based on the complex version of the sphere decoder (SD) is presented in this paper. The algorithm overcomes the main drawback of the SD: its variable throughput, depending on the noise level and the channel conditions. Implementation results show that the algorithm is highly parallelizable and can be fully pipelined. This reduces the use of FPGA resources and results in a constant throughput, which is significantly higher than previous SD implementations at a cost of a very small bit error ratio (BER) degradation.**

I. Introduction

The use of multiple input-multiple output (MIMO) technology has become the new frontier of wireless communications. It enables high-rate data transfers and improved link quality through the use of multiple antennas at both transmitter and receiver [1]. Nowadays, the prototyping of those multiple-antenna systems has become increasingly important to verify the enhancements advanced by analytical results [2], [3]. For that purpose, field-programmable gate arrays (FPGAs), with their high level of parallelism, high densities and embedded multipliers, are a suitable prototyping platform.

For spatially multiplexed uncoded MIMO systems, the sphere decoder (SD) is widely considered the most promising approach to obtain optimal maximum likelihood (ML) performance with reduced complexity [4], [5]. However, previous application-specific integrated circuit (ASIC) implementations of the SD have shown that it provides a variable throughput and makes a suboptimum use of the hardware resources due to its sequential nature [6], [7]. Those factors are of special importance if the SD needs to be integrated into a complete wireless communication system where the data needs to be detected in a fixed number of operations and the resource use needs to be optimized.

This paper presents a real-time FPGA prototype of a recently proposed fixed-throughput sphere decoder (FSD) that overcomes the two problems mentioned above while providing quasi-ML performance [8]. In addition, its throughput is higher than previously proposed alternatives to obtain a fixed-throughput MIMO detector based on the SD [9], [10].

II. Fixed-Throughput Sphere Decoder (FSD)

The FSD proposed in [8] combines a novel channel matrix preprocessing with a search through a small subset of the complete receive constellation. It achieves quasi-ML performance in a fixed number of operations making it suitable for hardware implementation.

A. MIMO System Model

We consider a wireless system with M transmit and N receive antennas, denoted as $M \times N$, with $N \geq M$. The transmitted symbols are independent and belong to a quadrature amplitude modulation (QAM) constellation of P points forming an M-dimensional complex constellation $\mathcal{C}$ of P^M vectors. The received N-vector is given by

$$\mathbf{r} = \mathbf{H}\mathbf{s} + \mathbf{v} \tag{1}$$

where $\mathbf{s} = (s_1, s_2, \ldots, s_M)^T$ denotes the vector of transmitted symbols with $\mathrm{E}[|s_i|^2] = 1/M$, $\mathbf{v} = (v_1, v_2, \ldots, v_N)^T$ is the vector of independent and identically distributed (i.i.d.) complex Gaussian noise samples with variance $\sigma^2 = N_0$ and $\mathbf{r} = (r_1, r_2, \ldots, r_N)^T$ is the vector of received symbols. $\mathbf{H}$ denotes the $N \times M$ 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.

B. FSD Algorithm

The FSD performs a search over only a fixed number of lattice vectors $\mathbf{Hs}$, generated by a small subset $\mathcal{S} \subset \mathcal{C}$, around the received vector $\mathbf{r}$. The transmitted vector $\mathbf{s} \in \mathcal{S}$ with the smallest Euclidean distance is then selected as the solution. The process can be written as

$$\hat{\mathbf{s}}_{\mathbf{fsd}} = \arg\{\min_{\mathbf{s}\,\in\,\mathcal{S}} \|\mathbf{U}(\mathbf{s} - \hat{\mathbf{s}})\|^2\} \tag{2}$$

where $\mathbf{U}$ is an $M \times M$ 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 (squared) Euclidean distance in (2) is obtained recursively starting from $i = M$ and working backwards until $i = 1$ using

$$D_i = u_{ii}^2|s_i - z_i|^2 + \sum_{j=i+1}^{M} u_{jj}^2|s_j - z_j|^2 = d_i + D_{i+1} \tag{3}$$

where $D_{M+1} = 0$, $D_1 = \|\mathbf{U}(\mathbf{s} - \hat{\mathbf{s}})\|^2$ and

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

In (3), the term D_{i+1} can be considered as an accumulated (squared) Euclidean distance (AED) down to level $j = i + 1$ and the term d_i as the partial (squared) Euclidean distance (PED) contribution from level i.

The subset of transmitted vectors $\mathcal{S}$ is determined defining the number of points s_i, denoted as n_i, that are considered per level. In [8], it was shown that, in the SD, the number of candidates considered per level during the tree search follow

$$\mathrm{E}[n_M] \geq \mathrm{E}[n_{M-1}] \geq \ldots \geq \mathrm{E}[n_1] \qquad (5)$$

with $1 \leq n_i \leq P$. The FSD, therefore, assigns a fixed number of points, n_i, to be searched per level following (5). This can be explained as follows: whereas in the first level, $i = M$, more points need to be considered due to interference from the other levels, the decision-feedback equalization performed on z_i reduces the number of points that need to be considered in the last levels to approximate the ML solution.

The total number of vectors whose Euclidean distance is calculated is, therefore, $N_{\mathcal{S}} = \prod_{i=1}^{M} n_i$, where simulations show that quasi-ML performance is achieved with $N_{\mathcal{S}} \ll P^M$, i.e. $\mathcal{S}$ is a very small subset of $\mathcal{C}$ [8]. The n_i points on each level i are selected according to increasing distance to z_i, following the Schnorr-Euchner (SE) enumeration [11].

A trade-off exists between the complexity and the performance of the FSD. If $\mathcal{S}$ is large, the performance will be closer to that of the original SD but the number of operations and, therefore, the required computational resources or the processing time will increase. That makes the FSD suitable for reconfigurable architectures where the size of $\mathcal{S}$ can be made adaptive depending on the MIMO channel conditions.

Fig. 1 shows a hypothetical subset $\mathcal{S}$ in 4×4 system with 4-QAM modulation where the number of points per level $\mathbf{n}_{\mathcal{S}} = (n_1, n_2, n_3, n_4)^T = (1, 1, 2, 3)^T$. In each level i, the n_i closest points to z_i are considered as components of the subset $\mathcal{S}$. In this case, the Euclidean distance of only $N_{\mathcal{S}} = 6$ transmitted vectors would be calculated, whereas the total number of transmitted vectors $4^4 = 256$ is much larger.

C. FSD Preprocessing of the Channel Matrix

The preprocessing of the channel matrix in the FSD determines the detection ordering of the signals $\hat{s}_i$ according to the distribution of points $\mathbf{n}_{\mathcal{S}}$ used.

It orders iteratively the M columns of the channel matrix. On the i-th iteration, considering only the signals still to be detected, the signal $\hat{s}_k$ with the smallest post-detection noise amplification, as calculated in [12], is selected if $n_i < P$. If $n_i = P$, the signal with the largest noise amplification is selected instead.

The following heuristic supports this ordering approach: if the maximum possible number of candidates, P, is searched on one level, the *robustness* of the signal is not relevant to the final performance, therefore, the signals that suffer the largest noise amplification can be be detected on the levels where $n_i = P$.

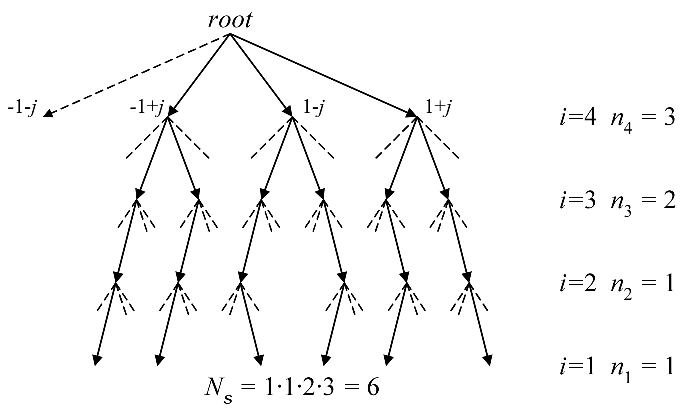

Fig. 1. Example of vectors $\mathbf{s} \in \mathcal{S}$ in a 4×4 system with 4-QAM modulation

III. 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 simulation 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.

A. Hardware Platform

The FPGA platform has been provided by Alpha Data Ltd. [13]. 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.

B. Rapid Prototyping Methodology

The rapid prototyping methodology selected is based on The Mathwork's MATLAB and Simulink [14] and Xilinx's DSP System Generator [15] tailored to Alpha Data's FPGA boards. Fig. 2 shows the methodology used for the rapid prototyping of the FSD. Initially, MATLAB is used to implement a complete MIMO system including transmitter, channel simulator and receiver. The FSD 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 FSD in the development stage, with the possibility of monitoring every signal in the FPGA model.

The FSD design is then synthesized for the FPGA using Xilinx's synthesis tools. This hardware design and a Simulink-based memory interface are integrated into the MATLAB system as shown in Fig. 3. This rapid prototyping methodology

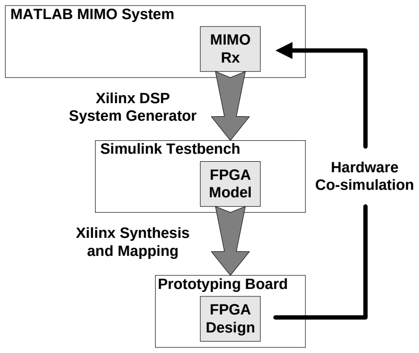

Fig. 2. Rapid prototyping methodology

allows us to implement quickly the FSD on an FPGA and perform real-time hardware-in-the-loop testing of the algorithm.

IV. FPGA Implementation

The FSD has been implemented for a 4×4 system using 16-QAM modulation where the subset $\mathcal{S}$ contains $N_{\mathcal{S}} = 16$ vectors following the point distribution $\mathbf{n}_{\mathcal{S}} = (1,1,1,16)^T$ providing quasi-ML performance [8]. Thus, all the possible points are searched in the first level ($i = M$) and only the closest point to z_i is considered for the remaining levels. With this distribution, the SE enumeration is not necessary, further simplifying the implementation. Therefore, only $N_{\mathcal{S}} = 16$ Euclidean distances are calculated, whereas the constellation size, $16^4 = 65536$, is much larger. In order to achieve a similar quasi-ML performance, a complex version of the K-best lattice decoder proposed in [10], would result in a complexity increase in the search stage by a factor of 13 compared to that of the FSD [8]. This would yield an implementation with a higher resource use and/or a lower throughput than the FSD.

A. FSD Architecture

Fig. 4 shows the block diagram of the FPGA implementation of the FSD 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 pseudoinverse of the channel matrix, $\mathbf{H}^\dagger$, and the entries of the Cholesky decomposition of the Gram matrix, $\mathbf{U}$.

Zero Forcing Unit (ZFU): This block performs the zero forcing (ZF) equalization to obtain $\hat{\mathbf{s}} = \mathbf{H}^\dagger\mathbf{r}$.

Partial Distance Unit (PDU) i: The 4 PDU blocks calculate the AED in (3) for each one of the levels. In the first level, $i = 4$, all the points in the constellation are considered ($n_4 = 16$). Therefore, the 16 PEDs, d_4, are calculated for all the possible points s_4 where $z_4 = \hat{s}_4$. These values directly form the set of D_4 values that are transferred as input to the next level.

For the rest of the levels, only the point $s_i \in$ 16-QAM closest to z_i is considered ($n_i = 1$ with $i \neq 4$). In this case, three tasks need to be performed:

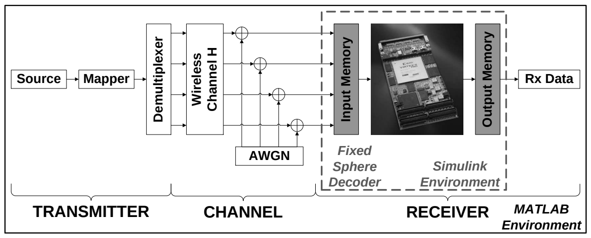

Fig. 3. Hardware-in-the-loop MIMO system diagram

1) The value z_i needs to be obtained using (4) taking into account the points used in the previous levels (s_j with $j = i+1,\ldots,4$) and the unconstrained ML solution $\hat{\mathbf{s}}$.
2) The closest point s_i to z_i is selected and the PED d_i is calculated for that level.
3) The current AED is calculated using $D_i = d_i + D_{i+1}$ and transferred as input to the next PDU block.

In order to make use of the parallelism of the FPGA, the distance calculations in the PDU blocks are performed in parallel for blocks of 4 vectors out of the $N_{\mathcal{S}} = 16$ vectors. This results in an optimized FPGA design providing an increase in the overall throughput of the FSD without making extensive use of the hardware resources.

Minimum Search Unit (MSU): This block searches for the minimum (squared) Euclidean distance D_1 among the 16 values calculated by the previous PDU blocks. The transmitted vector associated with the minimum D_1 is selected as the FSD solution $\hat{\mathbf{s}}_{\mathbf{fsd}}$.

Demapper Unit (DU): This block performs the 16-QAM demapping of the solution $\hat{\mathbf{s}}_{\mathbf{fsd}}$.

B. FSD Pipelining

From a hardware point of view, the FSD makes use of the inherent parallelism of the FPGA platform. In addition, its deterministic structure (i.e. a fixed number of operations are required to detect each MIMO symbol) compared to the SD makes possible a full pipelining of the algorithm, resulting in a highly optimized hardware implementation.

Applied to the FSD, pipelining implies that the detection process for one MIMO symbol starts before the previous MIMO symbols have been completely detected. The main advantage of a fully pipelined algorithm is the increase in the overall throughput due to two factors:

- If the hardware platform contains enough computational resources, a MIMO symbol can be detected in every clock cycle, dramatically increasing the throughput compared to a SD implementation. A trade-off exists between the use of hardware resources and the number of cycles per MIMO detection. Therefore, the use of hardware resources could also be reduced by detecting a MIMO symbol in more than one cycle.
- If the latency of the system (i.e. the number of cycles required to detect the first MIMO symbol) is not a critical issue, pipeline registers can be introduced between every operation of the algorithm, increasing the clock frequency of the design and, therefore, the throughput.

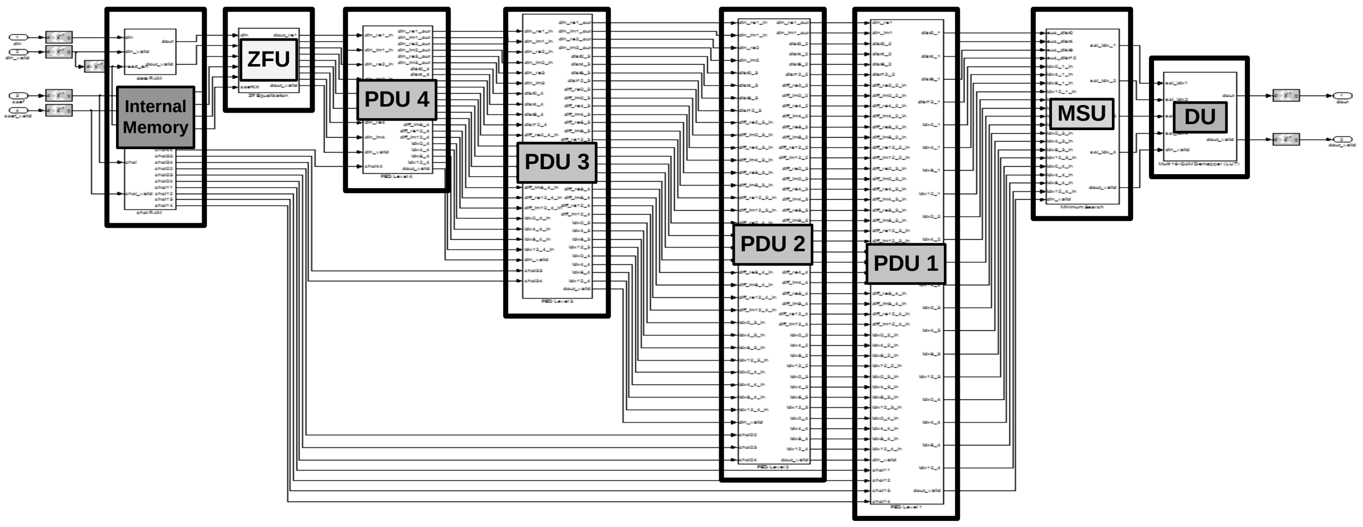

Fig. 4. FPGA block diagram of the FSD

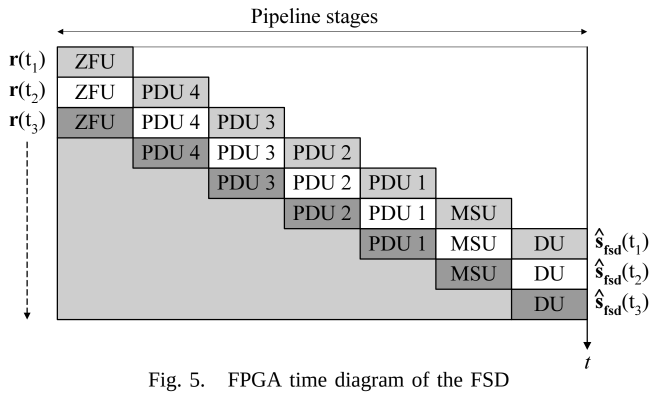
Pipeline stages
$\mathbf{r}(t_1)$
$\mathbf{r}(t_2)$
$\mathbf{r}(t_3)$
ZFU
PDU 4
PDU 3
PDU 2
PDU 1
MSU
DU
$\hat{\mathbf{s}}_{\mathbf{fsd}}(t_1)$
$\hat{\mathbf{s}}_{\mathbf{fsd}}(t_2)$
$\hat{\mathbf{s}}_{\mathbf{fsd}}(t_3)$
t

Fig. 5. FPGA time diagram of the FSD

In the case of the SD, given the sequential nature of the algorithm, full pipelining is not possible. The throughput can only be increased by integrating more detectors in parallel into the same hardware platform [6].

Fig. 5 shows a time diagram of the FSD algorithm on the FPGA where the information about the latency of the algorithm is not present for simplicity. It shows how the different parts of the algorithm (i.e. pipeline stages) start processing valid data sequentially as the received vectors $\mathbf{r}$ are available. In particular, the detection process is detailed for three time instants showing how the detected symbols $\hat{\mathbf{s}}_{\mathbf{fsd}}$ are being outputted at a constant rate.

The white area in the top right corner indicates the parts of the architecture that are waiting for valid data to fill the different pipeline stages. On the other hand, the grey area in the bottom left corner indicates that all the pipeline stages have been filled and that symbols are being processed in parallel for different time instants. Therefore, once the pipeline stages have been filled, all the blocks in the design are active in every clock cycle, resulting in an optimized use of the hardware resources of the design.

V. Results

The FSD has been implemented for a 4×4 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 of the FSD with a SD design previously presented in [6].

A. FPGA Resource Use

The resource use of the implementation of the FSD on the Xilinx Virtex-II-Pro FPGA board is summarized and compared with the resource use of 4 parallel SDs [6] in Table I.

It can be seen that the FSD uses significantly less resources than the 4-SDs with the exception of the flip-flops and the multipliers. The flip-flops are used in the design as delay nets to synchronize the different pipeline stages at the end of the detection process.

The number of multipliers used is slightly larger indicating that the computational complexity of the algorithm in terms of hardware resources is similar to that of the SD. Although the computational complexity of the FSD in terms of product operations is higher [8], this does not directly translate into a more complex hardware implementation. Other factors like the *regular structure* of the algorithm and the possibility of pipelining also determine the suitability of the algorithm for a hardware implementation. It should be noted that the number of multipliers in the PDUs could be reduced using an approximation of the Euclidean metric, like the Manhattan distance, at the cost of a non-negligible performance degradation [7].

On the other hand, the number of look-up tables (LUTs) has been considerably reduced. The use of LUTs can be seen as an indicator of the control logic required for the algorithm. In the case of FSD, the fixed number of operations and the possibility of pipelining greatly reduces the need for control blocks leaving the LUTs mainly to arithmetic operations. Taking into account that each slice contains two flip-flops and

TABLE I

FPGA RESOURCE USE OF THE FSD COMPARED WITH 4-SDS

Xilinx XC2VP70 FPGA	4-SDs [6]	FSD
Number of slices (33,088)	64% (21,467)	38% (12,721)
Number of flip-flops (66,176)	26% (17,691)	23% (15,332)
Number of 4-input LUTs (66,176)	54% (36,249)	24% (16,119)
Number of multipliers (328)	47% (156)	48% (160)
Number of block RAM (328)	55% (183)	25% (82)

two LUTs, we find that a considerable percentage of the slices are only partially used.

Finally, the number of memory blocks has been more than halved, where most of them are due to the input and output buffers defined on the FPGA to synchronize the FPGA board and the Simulink interface. From an algorithmic point of view, the FSD requires much less memory space for intermediate data storage during the detection process than the SD.

B. Performance Results

The bit error ratio (BER) performance of the FSD has been evaluated in real-time using 10,000 channel realizations with 200 symbols transmitted in every channel realization and is shown in Fig. 6. The pseudoinverse, the Cholesky decomposition and the FSD ordering of the channel matrix are performed offline in MATLAB. The input values to the FSD are quantized using 16 bits per real component.

It can be seen that the MATLAB performance of the FSD practically matches that of the SD (the degradation is only of 0.06 dB at a BER = 10^{-3}). In the case of the FPGA performance, a difference only appears for high signal to noise ratio (SNR) due to the quantization process. Furthermore, comparing the performance of both FPGA implementations, the FSD results in a more robust algorithm against the quantization process if the same fixed-point precision is used. From a BER performance point of view, the FSD provides practically ML performance without requiring information about the noise level on the channel and calculating only 16 Euclidean distances. The complexity of the maximum likelihood detector (MLD) is much higher, and would require 65,536 distance calculations per MIMO symbol, making it irrealizable in practice.

Fig. 7 shows the real-time average throughput of the FSD compared with the FPGA implementation of the SD [6] using different channel matrix ordering methods [5]. The throughput in megabits per second (Mbps) is calculated according to

$$Q = M \cdot \log_2 P \cdot f_{clock} \,/\, C \quad (\text{Mbps}) \tag{6}$$

where f_{clock} is the clock frequency of the design in MHz and C is the number of clock cycles required to detect a MIMO symbol. For this design, $f_{clock} = 100\,\text{MHz}$ and the number of cycles is $C = 4$ resulting in a throughput of $Q = 400\,\text{Mbps}$. It can be seen how the FSD outperforms the different SD alternatives and, more importantly, provides a constant throughput independent of the noise level. Therefore, the FSD is suitable for integration into a practical communication

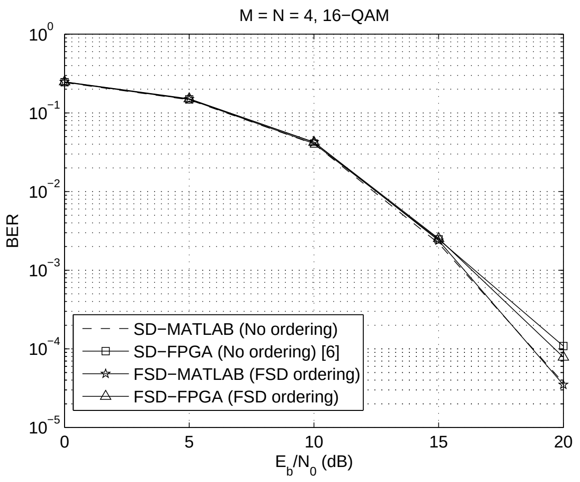

Fig. 6. BER performance of the FSD and the SD as a function of the SNR per bit.

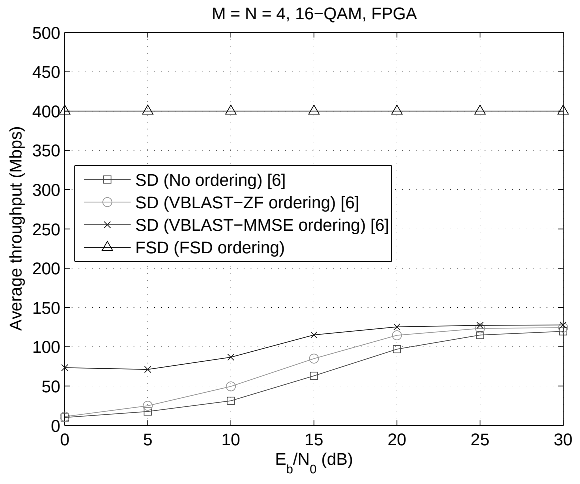

Fig. 7. Average throughput of the FSD and the SD with different orderings of the channel matrix as a function of the SNR per bit.

system where a deterministic throughput is required. Although the FSD requires a specific ordering of the channel matrix, its complexity is equivalent to the vertical Bell Labs layered space time-zero forcing (VBLAST-ZF) ordering of the SD and smaller than the VBLAST-minimum mean-square error (MMSE) ordering. However, the complexity of the ordering stage could be considered to be negligible for packet-based communications where the ordering is only performed once per frame. At an E_b/N_0 = 20 dB, the throughput of the FSD is 3.5 times larger than that of the SD which has the same complexity in the ordering stage (VBLAST-ZF).

It should be noted that the SD in [6] runs at a lower clock frequency $f_{clock} = 50\,\text{MHz}$. However, increasing f_{clock} to match that of the FSD would also increase the length of the critical path of the SD, without increasing the overall throughput of the system (only a marginal increase could be possible by finding the optimal trade-off point between clock frequency and number of cycles). On the other hand, given that

TABLE II

COMPARISON OF REAL-TIME SDS AND THE FSD PRESENTED IN THIS WORK

	K-best 1 [10]	K-best 2 [9]	SD 1 [7]	SD 2 [7]	SD 3 [6]	FSD 1	FSD 2
Hardware platform	ASIC	ASIC	ASIC	ASIC	FPGA	FPGA	FPGA
MIMO system	4×4	4×4	4×4	4×4	4×4	4×4	4×4
Modulation	16-QAM	16-QAM	16-QAM	16-QAM	16-QAM	16-QAM	16-QAM
Floating-point BER perfomance	quasi-ML	quasi-ML	ML	close to ML	ML	quasi-ML	quasi-ML
Clock frequency	100 MHz	100 MHz	51 MHz	71 MHz	50 MHz	100 MHz	150 MHz
Throughput at E_b/N_0 = 20dB	10 Mbps (constant)	53.3 Mbps (constant)	126 Mbps	253 Mbps	114.5 Mbps	400 Mbps (constant)	600 Mbps (constant)

the FSD has been fully pipelined (i.e. C is fixed), hardware optimizations to further increase f_{clock} would directly increase the throughput of the implementation.

The FPGA implementation of the FSD has been compared with previous 4×4 16-QAM SD implementations in Table II. Although a rapid prototyping methodology has been used, the FSD outperforms previous SD and K-best detectors while using less than half of the resources on the FPGA board. In addition, the BER performance only suffers a very small deviation from ML [8]. In particular, the FSD outperforms previous alternatives presented to achieve a constant throughput in the SD that require more computational power and memory resources [9], [10]. In addition, FSD 2 shows how internally pipelining the multipliers to increase f_{clock} directly increases the throughput, incurring only in a 10% increase in the number of flip-flops used. Finally, we believe that the implementation of the FSD on an ASIC using hardware tools could lead to further improvements in its performance.

VI. CONCLUSION AND FUTURE WORK

An FPGA implementation of the FSD using a rapid prototyping methodology has been presented in this paper. The FSD has been proposed as an alternative to the SD to achieve quasi-ML performance, while providing a constant throughput.

It has been shown that the structure of the FSD is especially suited for parallel hardware implementation, as opposed to the sequential tree search performed in the SD. In particular, given that the number of operations of the algorithm is fixed, the FSD can be fully-pipelined, providing a significantly higher throughput than previously presented SD implementations.

The implementation of the FSD shows that quasi-ML performance can be achieved with high throughput for systems where the number of antennas or the constellation size make the MLD irrealizable. In addition, the constant throughput of the FSD makes its integration into complete communication systems reasonably straightforward. This overcomes the problem with the SD, where special techniques (for example, early termination strategies [7]) are required to guarantee a minimum throughput.

Finally, the structure of the FSD can be adapted to provide soft information (*a posteriori probabilities*) about the detected bits, similar to the list-SD used for iterative turbo-decoding [16]. This last aspect is the main subject of ongoing work.

ACKNOWLEDGMENT

The authors thank Alpha Data Ltd. for partially sponsoring this research and Dr. Andrew McCormick from Alpha Data for his help in the integration of the prototyping platform.

REFERENCES

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

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

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

[4] E. Viterbo and J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Trans. Inform. Theory*, vol. 45, no. 5, pp. 1639–1642, July 1999.

[5] 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*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[6] L. G. Barbero and J. S. Thompson, "Rapid prototyping of the sphere decoder for MIMO systems," in *Proc. IEE/EURASIP Conference on DSP Enabled Radio (DSPeR '05)*, vol. 1, Southampton, UK, Sept. 2005, pp. 41–47.

[7] 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*, vol. 40, no. 7, pp. 1566–1577, July 2005.

[8] L. G. Barbero and J. S. Thompson, "A fixed-complexity MIMO detector based on the complex sphere decoder," *submitted to IEEE Workshop on Signal Processing Advances for Wireless Communications (SPAWC '06)*.

[9] 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*, vol. 1, Shanghai, China, June 2004, pp. 65–68.

[10] 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)*, vol. 3, Scottsdale, AZ, May 2002, pp. 273–276.

[11] C. P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Mathematical Programming*, vol. 66, pp. 181–199, 1994.

[12] 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)*, Atlanta, GA, Sept. 1998, pp. 295–300.

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

[14] The MathWorks, Inc., http://www.mathworks.com.

[15] Xilinx, Inc., http://www.xilinx.com.

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