FPGA DESIGN CONSIDERATIONS IN THE IMPLEMENTATION OF A FIXED-THROUGHPUT SPHERE DECODER FOR MIMO SYSTEMS

Luis G. Barbero and John S. Thompson

Institute for Digital Communications
University of Edinburgh, EH9 3JL Edinburgh, UK
e-mail: {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. It achieves quasi-maximum likelihood (ML) performance in systems where a hardware implementation of the maximum likelihood detector (MLD) is unfeasible due to its high complexity. It achieves this with a highly parallel and fully pipelined architecture. In addition, different design modifications are proposed and implemented to reduce the resource use and/or increase the throughput of the algorithm.

1. INTRODUCTION

The prototyping of multiple input-multiple output (MIMO) systems has become increasingly important [1] to validate theoretical results, anticipating higher-data rate and improved link quality when those multiple-antenna systems are applied to wireless communications [2]. For that purpose, field-programmable gate arrays (FPGAs), with their high level of parallelism, high densities and embedded multipliers, are a suitable prototyping platform.

The optimum detector for spatially multiplexed uncoded MIMO systems is the maximum likelihood detector (MLD) but it suffers from an extremely high complexity for large number of antennas and higher-order constellations. The MLD has been implemented in practice for small constellation sizes [3]. For medium constellation sizes, ML performance has been achieved through the use of the sphere decoder (SD) with a variable throughput [4] - [7]. However, the problem of large constellation sizes has not been addressed from an implementation point of view.

This paper presents a real-time FPGA implementation of a recently proposed fixed-throughput sphere decoder (FSD) that can be applied to large constellation sizes achieving quasi-ML performance [8]. Given the dimensionality of the problem, different design modifications are proposed to reduce the resource use of the algorithm and/or increase its throughput (i.e. number of bits detected per second).

2. FIXED-THROUGHPUT SPHERE DECODER

2.1. MIMO System Model

The system model considered has M transmit and N receive antennas, with $N \geq M$, denoted as $M \times N$. The transmitted symbols are taken independently from 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, using matrix notation, 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.

2.2. FSD Algorithm

The main idea behind the SD is to perform a search over only those noiseless received points (defined as $\mathbf{Hs}$) that lie within a hypersphere of radius R around the received signal $\mathbf{r}$ [4].

The FSD implemented in this paper, on the other hand, 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 $\mathbf{H}$.

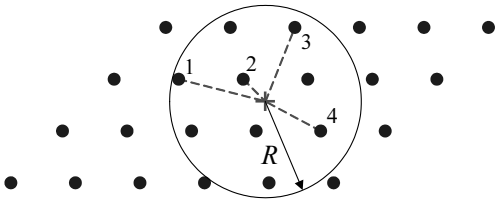

Fig. 1. Schematic of the FSD principle for the 2-dimensional case - only the numbered dots inside the circle are searched

The (squared) Euclidean distance in (2) can be obtained recursively starting from $i = M$ and working backwards until $i = 1$ using

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

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

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

In (3), the term D_{i+1} can be seen 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

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

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).

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 [9].

Conceptually, the FSD is equivalent to a SD where, for every MIMO symbol, the initial radius is set to the maximum D_1 distance among the $N_{\mathcal{S}}$ values obtained, and only those $N_{\mathcal{S}}$ vectors are searched. Fig. 1 shows the basic principle of the FSD where the dots represent the noiseless received constellation, the cross represents the actual received point contaminated with noise and only the numbered dots inside the hypersphere are considered as ML candidates ($N_{\mathcal{S}} = 4$).

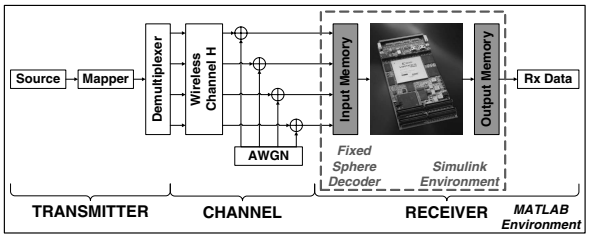

Fig. 2. Hardware-in-the-loop MIMO system diagram

2.3. 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 [10], is selected if $n_i < P$. If $n_i = P$, the signal with the largest noise amplification is selected instead.

3. RAPID PROTOTYPING SYSTEM

The algorithm has been implemented using a rapid prototyping system that 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. It allows us to perform real-time hardware-in-the-loop testing of the algorithm embedded in a computer-simulated system as shown in Fig. 2. The prototyping platform and methodology have been described in detail in [11].

- Hardware Platform: the FPGA platform has been provided by Alpha Data Ltd. [12] and includes a Xilinx Virtex-II (XC2V4000) and a Xilinx Virtex-II-Pro (XC2VP70) devices.

- Rapid Prototyping Methodology: It is based on The MathWork's MATLAB and Simulink [13] and Xilinx's DSP System Generator [14] tailored to Alpha Data's FPGA boards.

4. FPGA ARCHITECTURE

The FSD has been implemented for a 4×4 system using 64-QAM modulation where the subset $\mathcal{S}$ contains $N_{\mathcal{S}} = 64$ vectors following the point distribution $\mathbf{n}_{\mathcal{S}} = (1, 1, 1, 64)^T$ providing quasi-ML performance [8]. Therefore, only $N_{\mathcal{S}}$= 64 Euclidean distances are calculated, whereas the constellation size, $64^4 = 16,777,216$, is much larger, making the implementation of a MLD unfeasible. Previous approaches

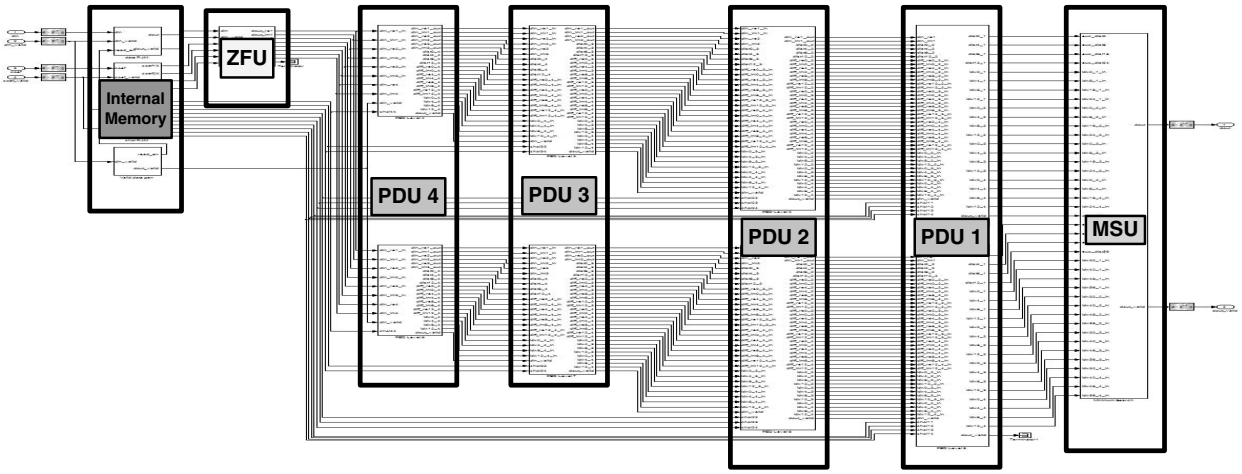

Fig. 3. FPGA block diagram of the FSD

to achieve ML detection dealt with a very small constellation sizes, $4^4 = 256$ [3] and $16^4 = 65,536$ [6].

Fig. 3 shows the block diagram of the FPGA implementation of the FSD where the only blocks left out are the input and output memories. 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 ($n_4 = 64$). Therefore, the 64 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 $s_i \in$ 64-QAM closest to z_i is considered ($n_i = 1$ with $i \neq 4$). In this case, three tasks need to be performed:

1. The value z_i needs to be obtained using (4) taking into account the points used in the previous levels (s_j when $j = i+1, \cdots, 4$) 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.

Minimum Search Unit (MSU): This block searches for the minimum (squared) Euclidean distance D_1 among the 64 values calculated by the previous PDU blocks. The transmitted vector associated with the minimum D_1 is selected as the FSD solution $\hat{\mathbf{s}}_{\mathbf{fsd}}$.

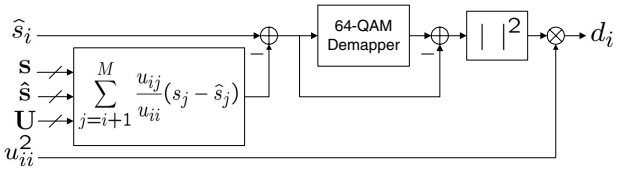

Fig. 4. PDU i branch design structure (with $i \neq 4$)

Three different versions of this architecture have been implemented that subsequently reduce the resource use of the FPGA without greatly affecting the performance.

4.1. FSD-A

The initial version of the architecture, denoted as FSD-A, consists of a direct implementation of the algorithm. 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 8 vectors out of the $N_{\mathcal{S}} = 64$ vectors. Therefore, 8 iterations are required to perform all the distance calculations. 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.

Therefore, every PDU block contains 8 branches to calculate the different PEDs. The structure of those branches is shown in Fig. 4 for levels $i = 1 \ldots 3$. In this case, only the closest constellation point to z_i (obtained by the 64-QAM demapper block) is required. For the first level, $i = 4$, the calculation of z_i is not required and only a block that enumerates the points of the 64-QAM constellation is needed to calculate the 64 $|s_4 - \hat{s}_4|^2$ values.

The fixed structure of the FSD makes possible a fully pipelined version of the algorithm. Thus, the detection process for one MIMO symbol starts before the previous MIMO

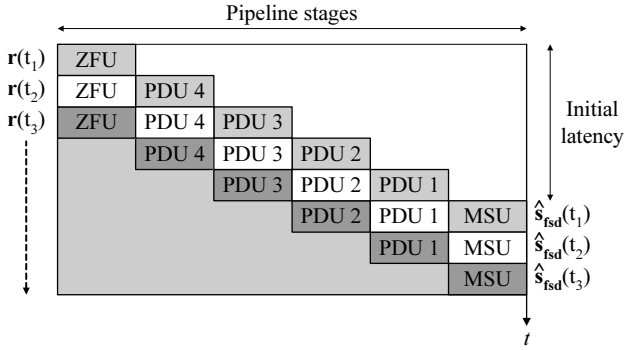

Fig. 5. FPGA time diagram of the FSD

symbols have been completely detected resulting in an overall throughput increase.

Fig. 5 shows a time diagram of the FSD algorithm on the FPGA. It shows how the different blocks of the architecture (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. That area is related to the initial latency of the implementation (i.e. the number of cycles required to detect the first MIMO symbol). 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.

4.2. FSD-B

In general, the implementation of MIMO detection algorithms is limited by the computational power of the target platform. In the particular case of FPGAs, the limiting factor is normally the number of embedded multipliers available. Therefore, ways of reducing the number of multipliers in that type of algorithms are of special interest.

The second implementation of the FSD, noted as FSD-B, modifies the structure of the complex multipliers in order to reduce the number of embedded multipliers. A direct implementation of a complex multiplication can be written as

$$(a+jb)(c+jd) = (ac-bd)+j(bc+ad) \qquad (6)$$

where 4 multipliers and 2 adders/subtractors are required. In this case, 2 clock cycles are required to perform the operation: the multiplications in the first cycle and the addition/subtraction in the second one.

However, we can reduce the number of multipliers if we rewrite (6) as

$$\begin{aligned}(a+jb)(c+jd) &= [a(c-d)+d(a-b)]+ \\ &\quad j[b(c+d)+d(a-b)]\,, \end{aligned} \qquad (7)$$

requiring only 3 multipliers, due to the repeated factor $d(a-b)$, and 5, comparatively inexpensive, adders/subtractors. Although in this case 3 clock cycles are needed to perform the complete operation, it might not pose a problem if the initial latency of the algorithm is not a critical issue. At the same time, the increase in the number of adders/subtractors does not generally represent an implementation problem.

It should be noted that this FSD-B implementation has the same bit error ratio (BER) performance as the FSD-A one, since no mathematical simplification has been applied, only a modified structure of the complex multiplication.

4.3. FSD-C

This last version, noted as FSD-C, further reduces the number of multipliers of the FSD-B by replacing the ℓ^2-norm calculation performed to obtain the PEDs (represented by the $|\ |^2$ block in Fig. 4) by a simpler method [5]. In our implementation, a ℓ^1-norm approximation is used so that the PED is written as

$$d_i \approx u_{ii}^2(\,|\Re\{s_i - z_i\}| + |\Im\{s_i - z_i\}|\,)\,. \qquad (8)$$

In this case, the AED value, D_1, does not represent a squared Euclidean distance anymore.

This version of the algorithm does result in a BER performance degradation given that the exact Euclidean distance metric is replaced by a Manhattan distance metric. However, in most scenarios, the reduction in the number of multipliers is more relevant than the BER degradation.

5. RESULTS

The different FPGA designs have been implemented and integrated into a MATLAB system model in order to perform hardware co-simulation of the algorithm and measure their BER and throughput performance. The resource use and the performance of the different versions of the FSD on the Xilinx Virtex-II-Pro FPGA board are summarized in Table 1. The throughput in megabits per second (Mbps) is calculated according to

$$Q = M \cdot \log_2 P \cdot f_{clock} \,/\, C \quad \text{(Mbps)} \qquad (9)$$

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 ($C = 8$ for all the versions).

It can be seen how the percentage of multipliers is subsequently reduced from a 92% down to a 57% in the FSD-C

Table 1. FPGA resource use and performance of the different FSD versions

Xilinx XC2VP70 FPGA	FSD-A	FSD-B	FSD-C	optimized FSD-B
Number of slices (33,088)	62% (20,580)	65% (21,794)	65% (21,694)	74% (24,815)
Number of flip-flops (66,176)	40% (26,732)	47% (31,372)	48% (31,900)	60% (39,800)
Number of 4-input LUTs (66,176)	41% (27,643)	45% (30,415)	48% (32,127)	47% (31,759)
Number of multipliers (328)	92% (304)	76% (252)	57% (188)	76% (252)
Number of block RAM (328)	26% (88)	26% (88)	26% (88)	26% (88)
Clock frequency (f_{clock})	100 MHz	100 MHz	100 MHz	150 MHz
Throughput (Q)	300 Mbps	300 Mbps	300 Mbps	450 Mbps
Initial latency	62 cycles	66 cycles	66 cycles	78 cycles

version. As expected, the number of look-up tables (LUTs) increases from FSD-A to FSD-B. This is due to the increase in the number of adders/subtractors required that are implemented on the FPGA using LUTs. In FSD-C, the number of LUTs increases slightly due to the additional logic required to calculate the ℓ^1-norm approximation.

The same trend can be observed in the number of flip-flops used. The increase in FSD-B is due to the delay nets required to synchronize the parts of the architecture that surround the new complex multipliers (their latency has been increased from 2 to 3 cycles). The slight increase in FSD-C is due to some additional delay nets required in the ℓ^1-norm approximation. Finally, for the number of slices, we should take into account that each one contains 2 LUTs and 2 flip-flops. Therefore, their percentage of use can only be seen as an indicator of the occupied slices where a high percentage of them are only partially used. There is a reduction in the number of slices from FSD-B to FSD-C because the reduction in the number of multipliers cause the routing tools to find a design that reduces the number of slices partially used (both the number of flip-flops and LUTs increase).

In terms of performance, only a negligible latency increase occurs when the original complex multiplication is replaced by the alternative proposed in section 4.2. It should be noted that the reduction in multipliers suggests that the FPGA tools should be able to find a more optimized design for FSD-B and FSD-C, marginally increasing the clock frequency and the throughput. However that increase in f_{clock} would be provided by the mapping and routing tools, given that the limiting factor is still the internal latency of the multipliers. Therefore, that possibility has not been studied given that the results depend mainly on the commercial tools used and the options selected (all the designs have been targeted to the same $f_{clock} = 100$ MHz).

Finally, an optimized version of FSD-B is presented in the last column. It has been obtained by increasing only the internal pipeline stages of the embedded multipliers. With this modification, the mapping and routing tools obtain a design that has a higher clock frequency and throughput. The number of flip flops has been considerably increased in order to synchronize the different parts of the design with the new multipliers. Although, the initial latency has been increased, given the increase in the clock frequency, the real latency has, in fact, been reduced from $t_l = 66/100\,\text{MHz} = 0.66\mu$s to $t_l = 78/150\,\text{MHz} = 0.52\mu$s.

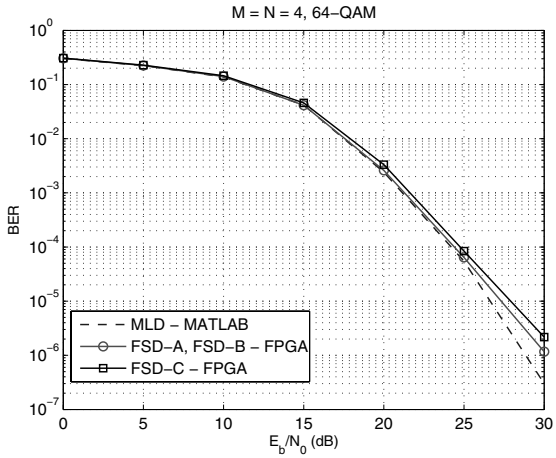

Fig. 6. BER performance of the fixed-point FSD and the floating-point MLD as a function of the SNR per bit.

The fixed-point BER performance of the different implementations 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 to compare it to the floating-point performance of the MLD. The pseudoinverse, the Cholesky decomposition and the FSD ordering of the channel matrix are performed offline in MATLAB and the input values to the FSD are quantized using 16 bits per real component. It can be seen how the FSD algorithm gives quasi-ML performance. A difference only appears for high signal to noise ratio (SNR) due to the quantization process that is not considered in the MATLAB simulation. As mentioned in section 4.3, the FSD-C version has a small performance degradation of only 0.35 dB at a

Table 2. Comparison of real-time FPGA implementations

Xilinx XC2VP70	SD [6]	FSD-B
MIMO configuration	4×4	4×4
Modulation	16-QAM	64-QAM
Number of hypotheses (P^M)	65,536	16,777,216
BER perfomance	ML	quasi-ML
Percentage of slices	64%	65%
Percentage of flip-flops	26%	47%
Percentage of 4-input LUTs	54%	45%
Percentage of multipliers	47%	76%
Percentage of block RAM	55%	26%
f_{clock} (MHz)	50	100
Q at E_b/N_0 = 20dB (Mbps)	114.5	300 (constant)

BER = 10^{-3}.

Due to the fact that ML MIMO detection has not been approached in the literature for a system of the same dimensionality, Table 2 shows a comparison between the FSD-B implementation and a previously presented FPGA implementation of the original SD, for a smaller MIMO system, on the same FPGA board [6]. The comparison shows the suitability of the FSD algorithm for approaching ML performance in high-dimensional MIMO systems. The SD would need more hardware resources (if we proportionally compare the number of hypotheses and the hardware resources) and provide a lower throughput that is not constant, affecting its integration into a complete communication system.

6. CONCLUSION

An FPGA implementation of the FSD algorithm has been presented in this paper. It has been proposed as an alternative to the SD to achieve quasi-ML performance with a constant throughput in MIMO systems where the MLD is irrealizable. Different versions of the algorithm have been proposed to reduce the number of multipliers of the hardware implementation and/or increase the throughput.

Although other real-time implementations exist that achieve ML performance, the work presented here represents, to the best of our knowledge, the first approach to approximate ML MIMO detection in high-dimensional systems ($P^M > 10^6$) using FPGAs.

7. 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.

8. REFERENCES

[1] 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.

[2] 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.

[3] T. Koike, Y. Seki, H. Murata, S. Yoshida, and K. Araki, "FPGA implementation of 1Gbps real-time 4×4 MIMO-MLD," in *Proc. 61st IEEE Vehicular Technology Conference (VTC '05-Spring)*, Stockholm, Sweden, May 2005, pp. –.

[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] 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.

[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] 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.

[8] L. G. Barbero and J. S. Thompson, "A fixed-complexity MIMO detector based on the complex sphere decoder," to appear in *Proc. IEEE Workshop on Signal Processing Advances for Wireless Communications (SPAWC '06)*, Cannes, France, July 2006.

[9] 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.

[10] 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.

[11] L. G. Barbero and J. S. Thompson, "Rapid prototyping system for the evaluation of MIMO receive algorithms," in *Proc. IEEE International Conference on Computer as a Tool (EUROCON '05)*, vol. 2, Belgrade, Serbia and Montenegro, Nov. 2005, pp. 1779–1782.

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

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

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