

Design Tradeoffs and Hardware Architecture for Real-Time Iterative MIMO Detection using Sphere Decoding and LDPC Coding

Hyungjin Kim, *Student Member, IEEE*, Dong-U Lee, *Member, IEEE*,
and John D. Villasenor, *Senior Member, IEEE*

Abstract— We explore the performance and hardware complexity tradeoffs associated with performing iterative multiple-input multiple-output (MIMO) detection using a sphere decoder and a low-density parity-check (LDPC) decoder. Iterations are performed both within the LDPC decoder as well as via an outer iteration loop through which refined soft information is fed back from the LDPC decoder to a MIMO detector. A hardware architecture and associated implementation results on Xilinx Virtex-5 field programmable gate array for a 4x4 QPSK MIMO system are presented. The system offers a performance improvement of approximately 1 dB over systems without the outer iteration loop, and provides an information bit throughput that ranges from 60 to 300 megabits per second when a length 1944 rate 1/2 LDPC code is used.

Index Terms— Field programmable gate arrays, Maximum likelihood decoding, Tree searching, Channel coding.

I. INTRODUCTION

THE BENEFITS of multiple-input multiple-output (MIMO) communications systems are well established. Recent wireless communication standards such as IEEE 802.11n include MIMO techniques in combination with low-density parity-check (LDPC) coding [1], and it is certain that the MIMO/LDPC combination will be used in an increasing number of communications environments in coming years. While there is a very large and rapidly growing body of literature on many issues related to MIMO generally and its use in conjunction with LDPC codes specifically, there has been proportionally much less work addressing the real-time hardware challenges of such systems.

The most common method for MIMO detection is to use minimum mean square error (MMSE) detection [2], [3], which has the advantage of being quite efficient computationally, and is thus the basis for most of the early-generation commercial systems for decoding of MIMO/LDPC in IEEE 802.11n. It is well established, however, that at the cost of increasing

computational complexity, methods based on sphere decoding can offer significant performance advantages over MMSE. For example, it has been shown in [4] that sphere decoding outperforms MMSE by approximately 5 dB at a frame error rate (FER) of 0.1 and a greater amount at lower FERs. The overwhelming majority of the work on sphere decoding has been in the context of non-iterative detection, in which the received signal information is processed in a MIMO detector and then by a channel decoder, but is not fed back again to the MIMO detector.

Many variations on soft-output sphere decoding have been proposed which aim, among other things, to reduce the algorithmic complexity while minimizing performance degradation. These include list sphere decoding (LSD) [4], single tree search [5], K -best sphere decoding [6], relaxed K -best sphere decoding [7], and soft sphere detection with bounded search [8].

Garret *et al.* [4] implemented LSD where the tree search is bounded to provide constant throughput. The design in [4] for 4×4 16-QAM MIMO achieves 38.8 megabits per second (Mbps) using 10 mm^2 in a $0.18 \text{ }\mu\text{m}$ CMOS process. Studer *et al.* [5] examined various optimizations for non-iterative MIMO detection. The single tree search in [5] allows concurrent searching for the ML solution and all counter-hypotheses, which results in significant speed-ups over conventional non-iterative MIMO detection methods. Guo and Nilsson [6] were the first to implement a soft-output K -best detector. Their design achieves over 100 Mbps for 4×4 16-QAM MIMO using 0.56 mm^2 in a $0.13 \text{ }\mu\text{m}$ CMOS process. Recently, Chen *et al.* [7] described a soft-output K -best detector for a 4×4 64-QAM MIMO system. Sorting, which is often the bottleneck in K -best detectors, is approximated by performing coarse granularity sorting. That design has an average throughput of 8 Mbps while occupying 2.38 mm^2 in a $0.13 \text{ }\mu\text{m}$ process. It is estimated that in order to achieve 100 Mbps, approximately 31 mm^2 will be required.

One of the most interesting open questions, both from a hardware and performance standpoint, concerns the extent to which soft information generated by a channel decoder can be iteratively fed back to a MIMO detector in order to increase the overall performance of the system. Such iterative MIMO detection was first proposed by Hochwald and Brink [9], who described information exchange between a soft-output sphere decoder and a channel decoder (a turbo decoder in the case of

Manuscript received July 1, 2007; revised November 30, 2007. This work was supported in part by the Office of Naval Research (Contract number N00014-06-1-0253) and in part by the National Science Foundation (Grant number CCR-0120778 and CCF-0541453).

Hyungjin Kim and John D. Villasenor are with the Electrical Engineering Department, University of California, Los Angeles, CA, 90095-1594, USA (e-mail: hjkimnov@ee.ucla.edu; villa@icsl.ucla.edu).

Dong-U Lee was with the Electrical Engineering Department, University of California, Los Angeles, CA, USA when the work was conducted. He is now with Mojix, Inc., 11075 Santa Monica Blvd., Suite 350, Los Angeles, CA 90025 (e-mail: dongu@mojix.com).

Digital Object Identifier 10.1109/JSAC.2008.080816.

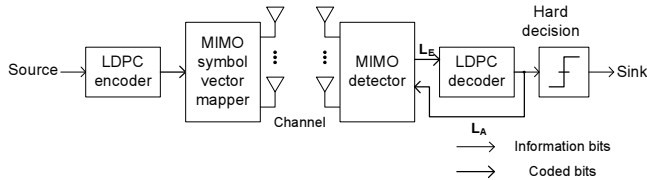


Fig. 1. Block diagram of an LDPC-coded MIMO system employing outer iterations.

[9]). The simulation results in [9] indicate that performance gains of several dBs can be obtained by using such information exchange. Iterative MIMO detection using LDPC codes as opposed to turbo codes was described by ten Brink, Kramer and Ashikhmin in [10] and Lu, Yue, and Wang in [11].

While the implementations above considered non-iterative systems, Radosavljevic and Cavallaro [8] examined the iterative case and the associated hardware implications. The approach in [8] is based on a K -best tree search where K is variable across different levels. Though the hardware architecture and computational complexity issues of the iterative case are discussed, actual hardware implementation results are not given.

Thus, one of the critical questions in iterative MIMO detection concerns the increase in performance that can be obtained by introducing the feedback and the specific associated cost in hardware. The present paper represents the first detailed exploration of the hardware architecture and implementation issues that arise when outer iterations are introduced, and thus provides a set of quantitative and experimental results that can be useful in the design of emerging and future systems. The main contributions of this paper include:

- Modification on the extrinsic log-likelihood ratio (LLR) equation, which halves computational complexity.
- Parameter exploration and optimization of iterative MIMO detection based on the K -best tree search.
- Hardware architecture for iterative MIMO detection.
- Implementation on a Xilinx Virtex-5 field-programmable gate array (FPGA), capable of a peak throughput of 300 Mbps information bit rate. To the best of our knowledge, this is the first hardware realization of an iterative MIMO detector.

The rest of this paper is organized as follows. Section II discusses algorithms of MIMO iterative detection and their performance. Section III explores design parameters and an algorithm of MIMO detector which allow good decoding performance and reduce complexity. Section IV describes the hardware architecture. Section V provides implementation results and concluding remarks are given in Section VI.

II. ITERATIVE MIMO DETECTION

A. System Model

Fig. 1 shows a block diagram of the MIMO system considered in this work. At the transmitter side, the information bits are encoded using an LDPC encoder. The MIMO symbol vector mapper maps the LDPC-coded bits to multiple transmit antennas and performs modulation. At the receiver side, each receive antenna receives the sum of signals from all transmit

antennas scaled by the complex channel gain and augmented with noise. The MIMO detector generates soft outputs from the received signal and feeds them to the LDPC decoder, which in turn provides refined soft data back to the MIMO detector. This process is known as an “outer iteration”, in contrast with the “LDPC decoding iterations” performed within the LDPC decoder. The outer iteration can be repeated until all check nodes in the LDPC decoder have been satisfied, until a given number of maximum outer iterations have been executed, or until some other stopping criterion is met. Finally, hard decisions are performed on the soft outputs to generate the decoded bits. A more detailed diagram of such an iterative MIMO detection system can be found in Figure 5 of [10].

In a MIMO system with n_T transmit and n_R receive antennas, the received signal vector can be written in the customary manner as

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n} \quad (1)$$

where \mathbf{y} is an $n_R \times 1$ complex vector whose elements are signals from receive antennas. \mathbf{H} is an $n_R \times n_T$ complex channel matrix, in which the real and imaginary components of each element follow a Gaussian distribution with zero-mean and variance $1/2$. \mathbf{s} is an $n_T \times 1$ complex vector whose elements are signals from transmit antennas. \mathbf{s} is obtained by modulating the LDPC encoded bits \mathbf{x} . The length of \mathbf{x} is given by $n_B = n_T \times m_C$ where m_C is the number of bits per symbol. \mathbf{n} is a complex vector of independent zero-mean complex Gaussian noise entries with variance σ^2 per real component. In this work, we assume that the channel matrix \mathbf{H} and the noise variance σ^2 are known at the receiver, but unknown at the transmitter.

B. MAP Bit Detection

To perform maximum *a posteriori* probability (MAP) bit detection, extrinsic information is used, which is generally expressed as an LLR. The extrinsic LLR of the i th bit in \mathbf{x} , $L_E(x_i|y)$, can be approximated using the max-log approximation as

$$L_E(x_i|y) \approx \frac{1}{2} \max_{\mathbf{x} \in \mathcal{X}_{i,+1}} \left\{ -\frac{1}{\sigma^2} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 + \mathbf{x}_{[i]}^T \cdot \mathbf{L}_{A,[i]} \right\} - \frac{1}{2} \max_{\mathbf{x} \in \mathcal{X}_{i,-1}} \left\{ -\frac{1}{\sigma^2} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 + \mathbf{x}_{[i]}^T \cdot \mathbf{L}_{A,[i]} \right\} \quad (2)$$

where $\mathcal{X}_{i,+1}$ and $\mathcal{X}_{i,-1}$ are the sets of 2^{n_B-1} bit vectors whose i th bit is $+1$ and -1 , respectively. $\mathbf{x}_{[i]}$ is the subvector of \mathbf{x} obtained by omitting its i th element and $\mathbf{L}_{A,[i]}$ is the vector of *a priori* LLRs without its i th element. Derivations of Eqn. (2) can be found in [9]. In Eqn. (2), $\mathbf{x}_{[i]}^T \cdot \mathbf{L}_{A,[i]}$ can be written as

$$\begin{aligned} \mathbf{x}_{[i]}^T \cdot \mathbf{L}_{A,[i]} &= \mathbf{x}^T \cdot \mathbf{L}_A - x_i L_A(x_i) \\ &= \left(\sum_{j=0}^{n_B-1} x_j L_A(x_j) \right) - x_i L_A(x_i). \end{aligned} \quad (3)$$

where x_i is the i th bit of \mathbf{x} and $L_A(x_i)$ is the *a priori* LLR of x_i .

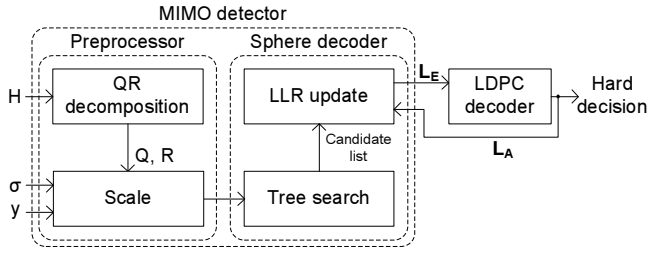


Fig. 2. Block diagram of the MIMO detector.

Substituting Eqn. (3) into Eqn. (2), we obtain

$$L_E(x_i|\mathbf{y}) \approx \frac{1}{2} \max_{\mathbf{x} \in \mathcal{X}_{i,+1}} \left\{ -\frac{1}{\sigma^2} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 + \mathbf{x}^T \cdot \mathbf{L}_A \right\} - \frac{1}{2} \max_{\mathbf{x} \in \mathcal{X}_{i,-1}} \left\{ -\frac{1}{\sigma^2} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 + \mathbf{x}^T \cdot \mathbf{L}_A \right\} - L_A(x_i) \quad (4)$$

which is more amendable for implementation purposes than Eqn. (2) since the terms for the max operation need to be computed once per bit vector instead of n_B times. This essentially halves the amount of computation.

Fig. 2 shows the block diagram of the MIMO detector used here, which consists of a preprocessor and sphere decoder that perform the MAP bit detection explained above. Preprocessor passes scaled QR decomposition results to the sphere decoder. The sphere decoder computes the extrinsic LLR as given in Eqn. (4), which requires the distance

$$d = \frac{1}{\sigma^2} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 \quad (5)$$

and $\mathbf{x}^T \cdot \mathbf{L}_A$ for all possible bit vectors. Eqn. (5) is generally obtained through a tree search and the remaining calculations in Eqn. (4) are performed in the LLR update unit.

C. Tree search

In order to provide an LDPC coder with the extrinsic LLRs (Fig. 2), a sphere decoder is required to generate a list of bit vectors and the corresponding distances d to the received signal \mathbf{y} in order to approximate Eqn. (4). The original (hard output detection) sphere decoder computes \mathbf{s} which minimizes Eqn. (5) to provide the maximum-likelihood (ML) symbol vector estimate. It avoids an exhaustive search of 2^{n_B} different hypotheses by utilizing QR decomposition of the channel matrix. Modification of the original sphere decoder enables soft output detection by providing a list of symbol vectors and corresponding bit vectors and distances [9]. Since the generation of the complete list of bit vectors and the distances is generally too complex for real-time hardware implementations, a subset of the complete list, \mathcal{L} , can be generated instead as noted earlier with respect to references [6], [9]. The list size directly impacts the hardware complexity because the tree search and the LLR update computations increase approximately linearly with the list size. If the size of \mathcal{L} is too small, large degradations in detection performance can result. Thus, one of the design goals is to identify a size for \mathcal{L} that is large enough to deliver minimal performance loss relative to an ideal system, while being small enough to facilitate real-time computation. Since bit vectors that lead to small distances

d in Eqn. (5) will more likely maximize each “max” term in Eqn. (4), for a given size of \mathcal{L} , denoted here by K , one should consider the smallest K distances and their corresponding bit vectors.

An efficient approach to obtaining \mathcal{L} is to perform QR decomposition on \mathbf{H} according to $\mathbf{H} = \mathbf{Q}\mathbf{R}$ where \mathbf{Q} is an unitary matrix and \mathbf{R} is an upper triangular matrix. By substituting $\mathbf{Q}\mathbf{R}$ with \mathbf{H} in Eqn. (5), we obtain

$$\frac{1}{\sigma^2} \|\mathbf{y} - \mathbf{Q}\mathbf{R}\mathbf{s}\|^2 = \|\mathbf{y}' - \mathbf{R}'\mathbf{s}\|^2 \quad (6)$$

where $\mathbf{y}' = \mathbf{Q}^H \mathbf{y} / \sigma$, $\mathbf{R}' = \mathbf{R} / \sigma$, and $(\cdot)^H$ denotes conjugate-transpose. Assume that y'_i is the i th element of the vector \mathbf{y}' and r'_{ij} is the element at the intersection of the i th row and the j th column of the matrix \mathbf{R}' . Eqn. (6) can thus be written as

$$\|\mathbf{y}' - \mathbf{R}'\mathbf{s}\|^2 = \sum_{i=0}^{n_R-1} \left\| y'_i - \sum_{j=i}^{n_R-1} r'_{ij} s_j \right\|^2. \quad (7)$$

Note that the index of summation inside $\|\cdot\|^2$ starts with i because matrix \mathbf{R} is an upper triangular matrix. By defining real coefficients a_i and b_{ij} as

$$a_{2i} = \text{Re}\{y'_i\} \quad (8)$$

$$a_{2i+1} = \text{Im}\{y'_i\} \quad (9)$$

$$b_{2i,2j} = b_{2i+1,2j+1} = \text{Re}\{r'_{ij}\} \quad (10)$$

$$b_{2i,2j+1} = -b_{2i+1,2j} = -\text{Im}\{r'_{ij}\} \quad (11)$$

$$s'_{2j} = \text{Re}\{s_j\} \quad (12)$$

$$s'_{2j+1} = \text{Im}\{s_j\}. \quad (13)$$

Eqn. (7) can be written as

$$\sum_{i=0}^{n_R-1} \left\| y'_i - \sum_{j=i}^{n_R-1} r'_{ij} s_j \right\|^2 = \sum_{i=0}^{2n_R-1} \left(a_i - \sum_{j=i}^{2n_R-1} b_{ij} s'_j \right)^2, \quad (14)$$

where all the coefficients of the term at the right side are real. Note that $b_{2i,2i+1} = 0$ because r'_{ii} is real. Assume that the accumulated distances of Eqn. (14) from $i = m$ to $(2n_R - 1)$ is denoted as d_m , in other words,

$$d_m = \sum_{i=m}^{2n_R-1} \left(a_i - \sum_{j=i}^{2n_R-1} b_{ij} s'_j \right)^2. \quad (15)$$

Because d_m is involved with only a subvector of \mathbf{s}' , $[s'_m \cdots s'_{2n_R-1}]^T$ from $m = (2n_R - 1), \dots, 0$, it can be interpreted as a tree structure as illustrated for the case of an $n_T = n_R = 4$ system using QPSK modulation in Fig. 3. In Fig. 3, due to use of real coefficients, there are eight levels for four receive antennas. $d_m^{(k)}$ denotes the accumulated distances corresponding to the k th subvector of \mathbf{s}' at the m th level. A tree structure such as the one shown in Fig. 3 makes it possible to efficiently perform tree search and pruning in order to find the smallest K distances [12], [13].

LSD [9] and K -best decoding [6] have been widely used to find a good subset \mathcal{L} based on tree search and pruning. LSD gives the best \mathcal{L} in the sense that it provides the smallest K using depth-first tree search whose search time (or throughput)

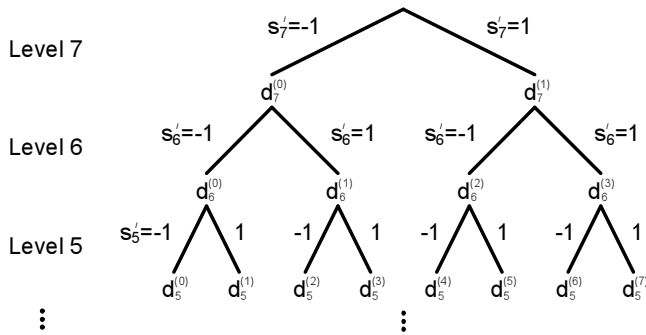


Fig. 3. Tree structure for calculation of Eqn. (14) for the case of 4 by 4 antennas with QPSK constellation.

can greatly vary with channel conditions. By limiting the number of tree search steps, the variation of tree search time can be greatly reduced at the expense of detection performance [4]. On the other hand, a K -best decoder uses breadth-first tree search. In this approach, at each level, the branches associated with the smallest K accumulated distances are retained and the other branches are pruned. Since the amount of computation is fixed at each level, a K -best detector can potentially provide constant throughput in a hardware implementation. In the work presented here, we use the K -best detector due to its constant throughput characteristics.

The tree search can be performed more efficiently by using a sorted QR-decomposition where the conventional QR-decomposition is preceded by a column permutation of \mathbf{H} [5], [14]. Discussions of implementation issues of QR-decomposition for MIMO detector can be found in [15], [16].

III. OPTIMIZING K -BEST SPHERE DECODING

In the K -best sphere decoder, it is obviously of high interest to identify the optimal design parameters. We consider this design goal in the context of a 4×4 MIMO channel modulated via QPSK and 16-QAM. Throughout this paper, the (1944,972) quasi-cyclic irregular LDPC code (a rate $R = 1/2$) proposed for the IEEE 802.11n standard [1] is used. In the plots following, E_b/N_0 is defined as

$$\frac{E_b}{N_0} \Big|_{dB} = \frac{E_s}{N_0} \Big|_{dB} + C \quad (16)$$

where $C = 0$ for QPSK, $C = -10\log_{10}2$ for 16-QAM, $E_s = E \|\mathbf{s}\|^2$, and $N_0 = 2\sigma^2$ [9].

A. Number of Outer and LDPC Decoding Iterations

Fig. 4 shows the FER performance of iterative MIMO detection (Eqn. (4)) using different numbers of outer and LDPC decoding iterations. QPSK with $K = 256$ (complete list) over a 4×4 MIMO channel is used. We consider the complete list of candidates exclude the effects of \mathcal{L} size. For each frame, zero, two, four and six outer iterations and 10, 15, and 20 LDPC decoding iterations are performed. While the general trend that increasing numbers of outer and/or LDPC decoding iterations improves performance is expected, the more interesting aspect of these results is the ability to identify specific tradeoffs in complexity versus performance, particularly for higher numbers of iterations.

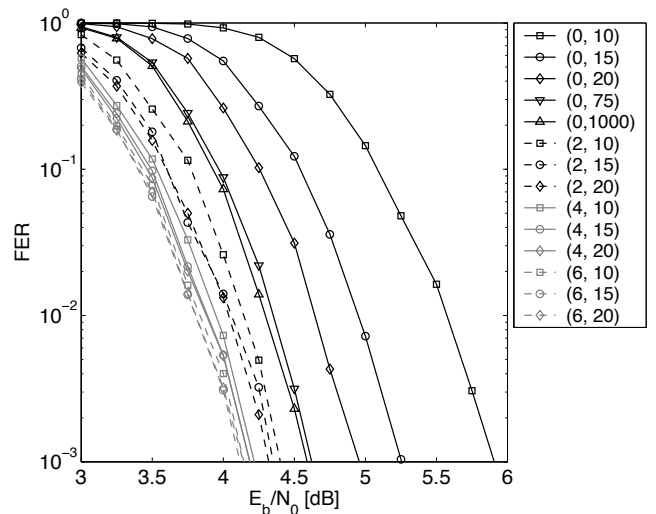


Fig. 4. FER performance of different combinations of outer and LDPC decoding iterations. QPSK with $K = 256$ (complete list) over a 4×4 MIMO channel is used. (m, n) refers to m outer iterations and n LDPC decoding iterations per outer iteration. Decoding begins and ends with n LDPC decoding iterations, so the total number of LDPC iterations is $(m + 1)n$.

For the system considered here, four outer iterations were chosen for the detector, since the performance gain obtained in going from four to six outer iterations is insignificant (~ 0.1 dB). The dB gain of performing four outer iterations opposed to performing zero outer iterations varies between 0.7 dB to 1.6 dB depending on the number of LDPC iterations. The number of LDPC decoding iterations is set at 15, since the figure indicates that the performance gain from 15 to 20 LDPC iterations is negligible. Thus, the full system, which uses four outer iterations and 15 LDPC iterations is able to achieve an SNR gain of about 1 dB compared to a system using 15 LDPC iterations and no outer iterations. In [9], it was observed that the gain due to iteration was several dB, while in the results presented here the gain is 1 dB. This difference is due to the fact that the block size used in the present work is approximately ten times smaller than that used in [9]. As shown in Fig. 4, the performance improvement of (4,15) over (0,75), both of which utilize a total of 75 LDPC decoding iterations, is approximately 0.4 dB. It can be seen that even (0,1000) (in other words, 1000 LDPC iterations) leads to a marginal further dB gain (the 0.4 dB gap still exists), suggesting outer iterations are necessary to obtain substantial performance improvements. The complexity differences associated with this tradeoff will be discussed in Section V.

Given that there will be 75 LDPC iterations performed, it is of interest to confirm that evenly distributing these iterations over the outer iteration loops gives the best performance. Fig. 5 shows the FER performance of different allocation of 75 LDPC iterations under one potential set of variation in which the zeroth to third outer iterations are associated with p LDPC iterations each and the fourth outer iteration is associated with q LDPC iterations. The figure shows that when LDPC decoding iterations are evenly allocated across the outer iterations ($p = 15, q = 15$), the best performance is achieved. By contrast, the ($p = 5, q = 55$) case illustrates that

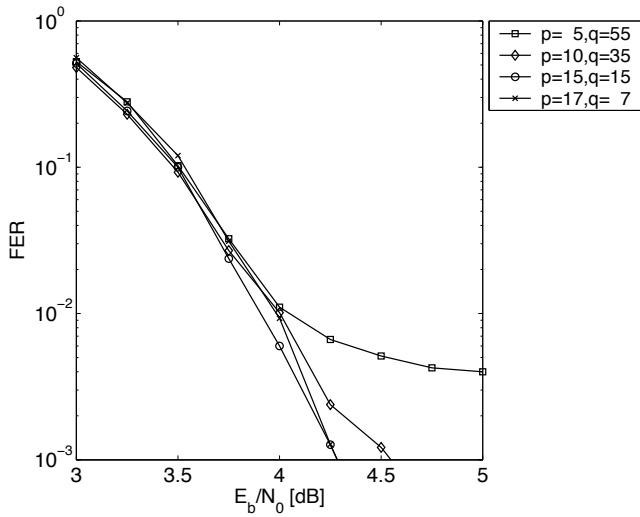


Fig. 5. FER performance of different allocations of 75 LDPC iterations. QPSK with $K = 256$ (complete list) over a 4×4 MIMO channel is used. The zeroth to third outer iterations use p LDPC iterations each, and the fourth outer iteration uses q LDPC iterations.

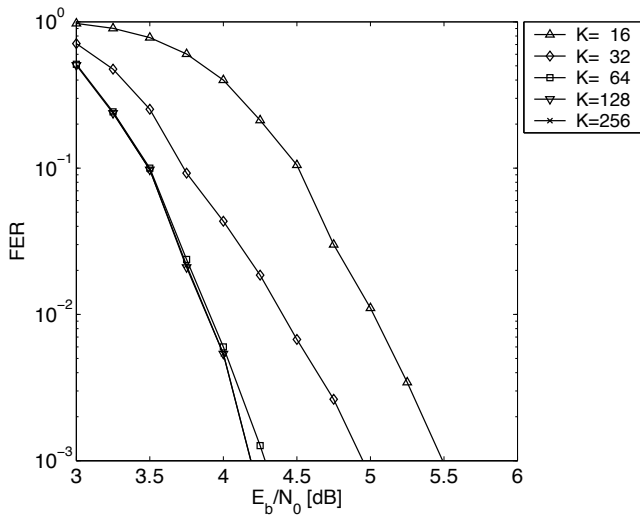


Fig. 6. FER performance of different list sizes K using QPSK. Four outer and fifteen LDPC decoding iterations over a 4×4 MIMO channel are used.

when the number of LDPC decoding iterations is too small in the zeroth to third outer iterations, an error floor is observed.

B. List Size K

Because the computational complexity increases approximately linearly with the list size K , it is very important to find a K that is small while providing good detection performance. Fig. 6 compares the FER performance of different list sizes K using QPSK. Four outer iterations and 15 LDPC decoding iterations (as found in Section III-A) are performed. As the figure shows that $K = 64$ achieves essentially the same performance as $K = 256$ (complete list), $K = 64$ is used for the remainder of this paper for QPSK. Although $K = 256$ would yield a simpler tree search implementation over $K = 64$ due to its regularity, the complexity of its LLR update increases considerably, making a $K = 256$ system computationally less efficient than a $K = 64$ system. This

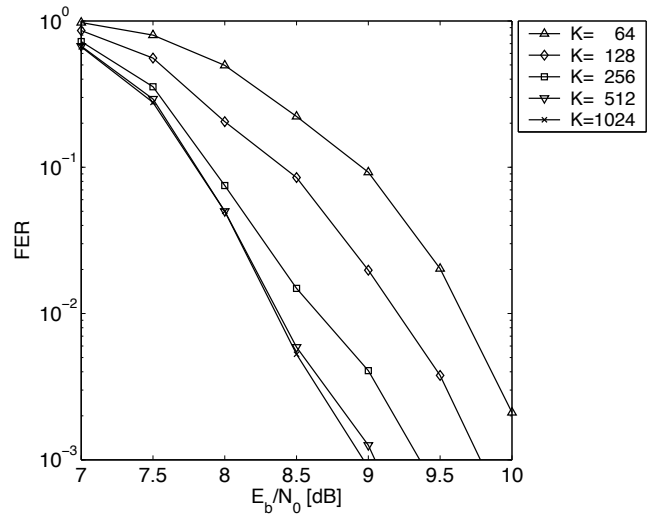


Fig. 7. FER performance of different list sizes K using 16-QAM. Four outer and fifteen LDPC decoding iterations over a 4×4 MIMO channel are used.

is discussed further in Section V. An analogous plot for 16-QAM is shown in Fig. 7. As in the QPSK case, four outer iterations and fifteen LDPC decoding iterations are used, since simulations showed that they led to good performance for 16-QAM as well. The figure indicates that $K = 512$ is a good choice for 16-QAM.

C. Best K Selection

The two main computational challenges in the K -best detector are to compute accumulated distances d_m at each level (Eqn. (14)) and to select the best K candidates with the smallest distances (often referred as “sorting” in the literature). This selection process is required when the number of candidates becomes greater than K . For example in 4×4 MIMO, the best K selection process starts at level 1 for QPSK with $K = 64$, and at level 3 for 16-QAM with $K = 512$.

For small K , the best K selection is a relatively simple process. However as was noted above, effective MIMO detection requires a large K ($K = 64$ for QPSK and $K = 512$ for 16-QAM). The complexity of the best K selection increases quadratically with K [17]. Therefore, although using real coefficients over complex coefficients for the tree search (Eqn. (14)) doubles the depth of the tree, real coefficients help to reduce the complexity of the best K selection approximately by half.

An approximate approach for performing best K selection is to partition the elements ($2K$ elements for QPSK and $4K$ elements for 16-QAM) into M groups (N elements per group) and to select the best K/M in each group. This approximate best K selection reduces complexity considerably as will be discussed further in Section IV.

Neighboring candidates are likely to have similar distances, since they originated from the same parent node as shown in Fig. 3. This is undesirable since a group of small distances will prune small “good” candidates, while a group of large distances will keep large “bad” candidates. In order to avoid this problem, we permute the candidates before allocating them to groups. The same permutation is applied to all frames.

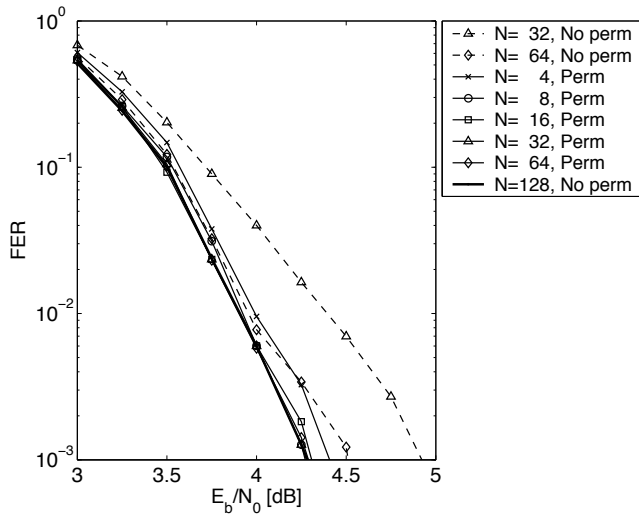


Fig. 8. FER performance of different group size N for selecting the best 64 elements from a pool of 128 elements ($K = 64$, QPSK). Four outer and fifteen LDPC decoding iterations over a 4×4 MIMO channel are used. “Perm” means that random permutations are performed before grouping.

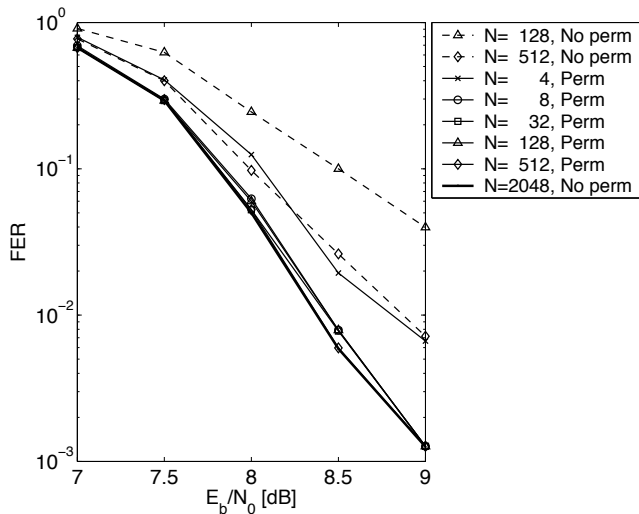


Fig. 9. FER performance of different group sizes N for selecting the best 512 elements from a pool of 2,048 elements ($K = 512$, 16-QAM). Four outer and fifteen LDPC decoding iterations over a 4×4 MIMO channel are used. “Perm” means that random permutations are performed before grouping.

Fig. 8 and Fig. 9 show the results of performing the approximate best K selection using different group sizes N with and without permutation, for selecting 64 from 128 elements and 512 from 2,048 elements respectively. Though the approximate best K leads to a different selection than to the ideal best K (thick lines in the figures), in both plots, the detection performance loss is very small when $N \geq 8$ is used with permutation. The figures indicate that permutation has a very large impact on the detection performance. Thus in hardware implementation for the QPSK case, 16 computational modules that pick the best four from eight candidates are required. For the 16-QAM case, 256 modules that pick the best two from eight candidates are required. As the selection module in 16-QAM is only marginally smaller than the one in QPSK [17], the best K selection complexity for 16-QAM

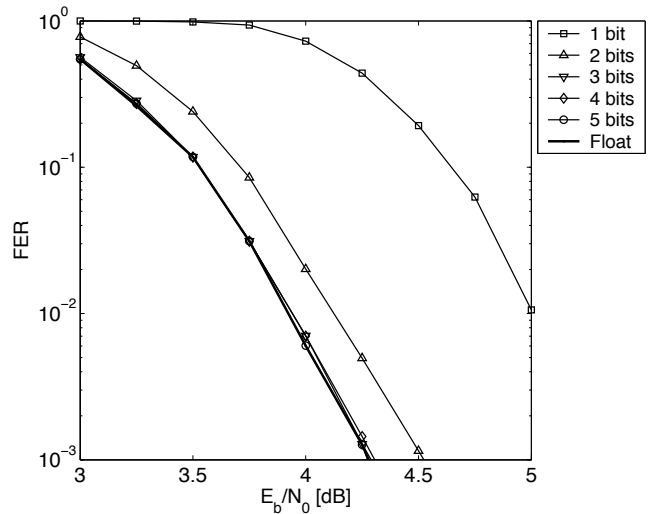


Fig. 10. FER performance of different numbers of fractional bits for the signals when QPSK is used. The integer parts for the distances is set to unsigned eight bits and the rest are set to signed five bits. Four outer iterations, fifteen LDPC decoding iterations, and best K group size $N = 8$ over a 4×4 MIMO channel are used.

is almost 16 times that of QPSK per search tree level. As will be discussed in Section V, the best K selection for QPSK occupies a significant portion of the device area of a recent-generation FPGA, while the best K selection for 16-QAM exceed the capacity of currently available FPGAs. Therefore for the subsequent sections, we shall focus on the 4×4 QPSK case, though the methods can be generalized to larger FPGA devices when they become available (or of course could be applied to ASIC design).

D. Fixed-Point Arithmetic

Since the iterative MIMO detector will be implemented in hardware using fixed-point arithmetic, the required bit-widths for the fixed-point signals must be examined. In order to determine the required integer bits, the dynamic ranges of the inputs a_i and b_{ij} , and the intermediate signals have been identified through simulation. The simulation results indicated that 8 unsigned integer bits are required for the distance $d_m^{(k)}$ and 5 signed bits are required for the inputs and all other signals.

Regarding the fractional bits, we assume that all signals share the same fractional bit-width and that truncation is performed at all times for quantization. Fig. 10 examines the FER performance variation when using different numbers of fractional bits. The integer bit-widths are set as determined above (except the floating point results). The figure shows that using 3 fractional bits is sufficient to provide comparable FER performance to floating-point arithmetic.

IV. HARDWARE ARCHITECTURE

A. Speed Requirements of Units

In order to minimize idle time of the tree search, LLR update, and LDPC decoder units (Fig. 2), we utilize the following processing strategy.

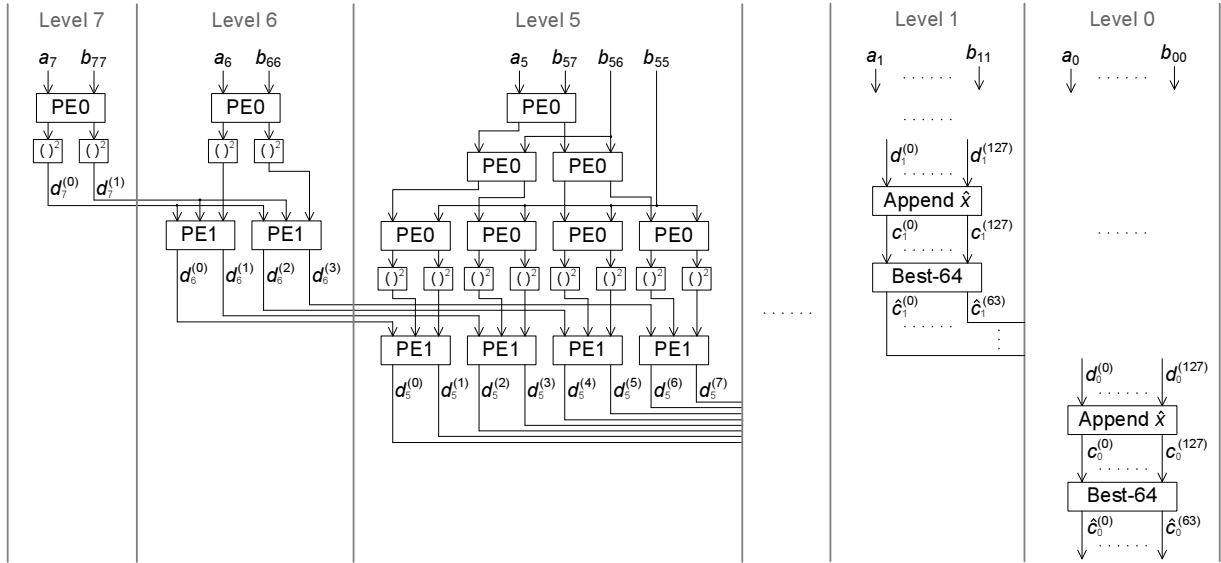


Fig. 11. Tree search unit architecture.

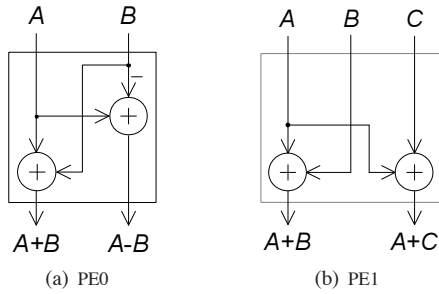


Fig. 12. Structures of PE0 and PE1.

- The LLR update generates L_E of frame i while the LDPC decoder generates L_A of frame j . This implies that LLR update should have a similar throughput to LDPC decoding.
- The tree search should supply candidate lists at a sufficiently high speed such that it does not become the bottleneck of the LLR update. In the worst channel conditions, the MIMO detector will require the full four outer iterations. However, in the best channel conditions, the MIMO detector will require zero outer iterations where one LLR update is followed by zero LDPC decoding iterations (LDPC decoder terminates with a syndrome check). Thus, the tree search should achieve similar throughput to one LLR update.

Recent hardware-based LDPC decoders are capable of information bit throughputs of hundreds of megabits per second (Mbps). For instance, the FPGA-based LDPC decoder implementation in [18] is able to perform 15 decoding iterations with the (1944,972) code at around 300 Mbps. Noting the two processing strategies above, this implies that both the LLR update and tree search need to achieve 300 Mbps information bit rate.

B. Tree Search

Fig. 11 depicts the hardware architecture of the tree search unit. Using Eqn. (15) and noting that $b_{2i,2i+1} = 0$, it computes the distances of each level in the following manner

$$\begin{aligned}
 d_7 &= (a_7 - b_{77}s'_7)^2 \\
 d_6 &= d_7 + (a_6 - b_{66}s'_6)^2 \\
 d_5 &= d_6 + (a_5 - b_{57}s'_7 - b_{56}s'_6 - b_{55}s'_5)^2 \\
 d_4 &= d_5 + (a_4 - b_{47}s'_7 - b_{46}s'_6 - b_{44}s'_4)^2 \\
 &\vdots \\
 d_0 &= d_1 + (a_0 - b_{07}s'_7 - b_{06}s'_6 - \dots - b_{00}s'_0)^2.
 \end{aligned}$$

The structures of the two processing elements used in the tree search unit are shown in Fig. 11. PE0 subtracts the term A which corresponds to a_i or the partial sum $a_i - \sum_{j=i}^n b_{ij}s'_j$ with the term B which corresponds to $b_{i(j-1)}s'_{j-1}$. Since $s'_{j-1} \in \{-1, 1\}$, PE0 produces $A + B$ and $A - B$. PE1 adds the accumulated parent distance A to the child distances B and C .

The distances of all nodes are computed up to level 1, where it is necessary to identify the best 64 candidates from the resulting 128 candidates. The binary bit-vector $\hat{\mathbf{x}} = (\mathbf{x} + 1)/2$ corresponding to the i th node is appended to the distance $d_1^{(i)}$ to create the candidate $c_1^{(i)}$. The candidates are fed to the Best-64 unit which picks the best 64 candidates in groups of 8 as discussed in Section III-C. The Best-64 unit comprises of 16 Best-4 units, which chooses the best 4 candidates from a set of 8 candidates. The Best-4 unit can be based on a variety of sorting methods. Multi-cycle sorting algorithms such as the bubble sort employed in [6] and the approximate sort in [7] are less suitable for this system due to its high throughput requirements. Instead, the Best-4 unit is based on the bitonic sorter [17] which allows single cycle sorting. Furthermore, since the goal is to choose the smallest 4 elements from a set of 8 elements, some logic required for a full 8 bitonic sorter can be removed. Note that the permutations discussed in

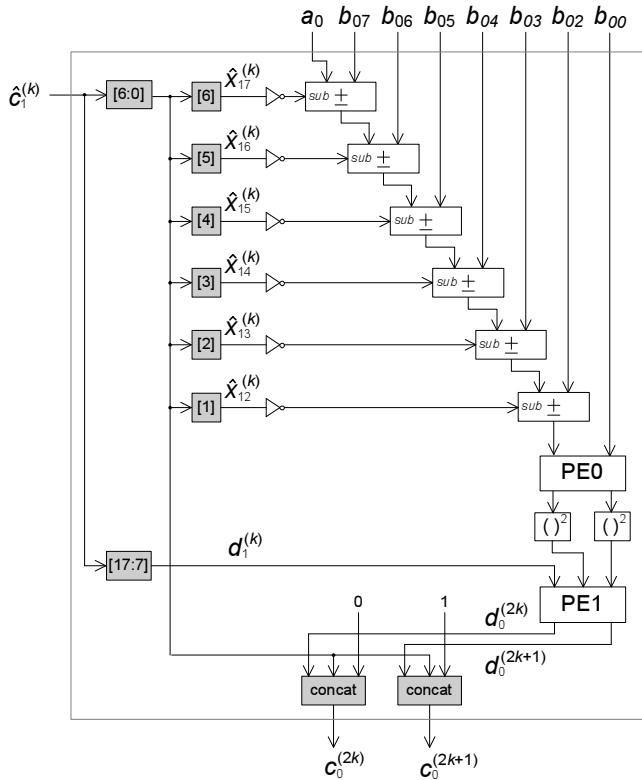


Fig. 13. Structure of the processing element used in level 0. “[$m:n$]” selects the m th to n th bit of the incoming signal. “concat” concatenates the incoming signals into a signal.

Section III-C are simply random interconnections in hardware, which requires no hardware logic resources.

Levels 7 to 1 enumerate all possible bit vectors and find their distances. However, this is not the case with level 0, since part of the tree has been pruned in the Best-64 unit of level 1. Thus, a different structure for computing the candidates and their distances is used in level 0. Fig. 13 shows the processing element used in level 0 for computing the final candidates. A candidate from level 1 $\hat{c}_1^{(k)}$ holds the distance $d_1^{(k)}$ in the leading 11 bits (eight integer bits and three fractional bits (Section III-D)) the corresponding bit vector $\hat{x}_1^{(k)}$ in the remaining 7 bits. The processing element uses each bit in the bit vector as a control signal to the Add/Sub module and finds the distances of the two child nodes. The level 0 bit $\hat{x}_{00}^{(k)}$ is appended at the end of each candidate.

The computation of the tree search architecture in Fig. 11 flows from top to bottom where a significant amount of parallelism can be exploited. For instance, the computation of $d_6^{(i)}$ and the computation of the child distances of level 5 can be performed in parallel because there are no data dependencies. Moreover, since there are no feedbacks present in the system, it can be fully pipelined while providing a constant throughput of one list \mathcal{L} per clock cycle.

C. LLR Update

Fig. 14 illustrates the hardware architecture of the LLR update module which computes Eqn. (4) except the distance computation which is performed by the tree search. First, the LLR update module reads one candidate every cycle and eight

LLRs from the LDPC decoder every K cycles. Initially the module computes the inner product $\mathbf{x}^T \cdot \mathbf{L}_A$. For the zeroth outer iteration, \mathbf{L}_A is set to zero. For subsequent iterations, it is obtained from the LDPC decoder. Since x_i , the i th bit of \mathbf{x} , is either -1 or 1 , the inner product becomes additions of $\pm L_A(x_i)$ where the sign of $L_A(x_i)$ is determined by x_i . As shown in Fig. 14, this can be realized by a tree of adders, where the sign of $L_A(x_i)$ is determined by $\hat{x}_{0i}^{(k)}$ which is the i th bit of the k th bit vector in the list \mathcal{L} . The result of the inner product is subtracted by the distance $d_0^{(k)}$ (the distance corresponding to the bit vector $\hat{\mathbf{x}}_0^{(k)}$). The result of the subtraction $p^{(k)}$ together with the bit $\hat{x}_{0i}^{(k)}$ is fed to PE2, which is depicted in Fig. 15. It finds the maximum $p^{(k)}$ for the cases when $\hat{x}_{0i}^{(k)} = 0$ and $\hat{x}_{0i}^{(k)} = 1$. The two maximum values are subtracted and divided by two via a right shift. $L_A(x_i)$ is subtracted from $q^{(k)}$ to produce the extrinsic LLR $L_E(x_i)$. The LLR update module produces extrinsic LLRs \mathbf{L}_E every K cycles. Multiple instances of the module can be used to improve throughput characteristics.

D. Scheduling and Interfacing

Table I illustrates the task scheduling between the tree search, LLR update, and LDPC decoder units. The interfacing between these units is shown in Fig. 16. In Table I, it is assumed that the incoming frames are indexed from the letter A , and that frames A , B , C , and D require 4, 2, 0, and 3 outer iterations respectively. Each column in the table represents the time taken for an LLR update or 15 LDPC decoding iterations. X_i refers to the LLR update or LDPC decoding in the i outer iteration of frame X . The figure shows that the LLR update and LDPC decoder units are always utilized, while the utilization of the tree search is dependent on the number of outer iterations.

Fig. 16 shows that triple buffering is used between the tree search and LLR update (interface 0). Each RAM holds the candidate lists of a single frame. As mentioned in Section IV-A, the LLR update and LDPC decoder operate in an interleaved manner, where each unit operates on two different frames concurrently. Thus, the two frames required by the LLR update must be available in interface 0. The third RAM can be written with a new frame by the tree search. The LLR update and LDPC decoder are linked via interface 1, which uses two dual buffers, one for write and one for read. These dual buffers ensure smooth operation of the scheduling in Table I. The data flow shown in the dashed and dotted lines switches every outer iteration.

V. IMPLEMENTATION RESULTS

The iterative MIMO detector has been implemented on a 65 nm Xilinx Virtex-5 XC5VLX330T-2 FPGA. Virtex-5 FPGAs contain the following three main types of resources: (1) user configurable elements known as “slices”, (2) storage elements known as “block RAMs”, and (3) multiply-and-add units known as “DSP slices”. A slice comprises of two 6-input lookup tables (LUTs), four flip-flops, and some additional logic. Each block RAM can store 36 kbits of data and a DSP slice can perform a 25-bit by 18-bit multiplication followed by a 48-bit addition. The Xilinx Virtex-5 XC5VLX330T-2 device

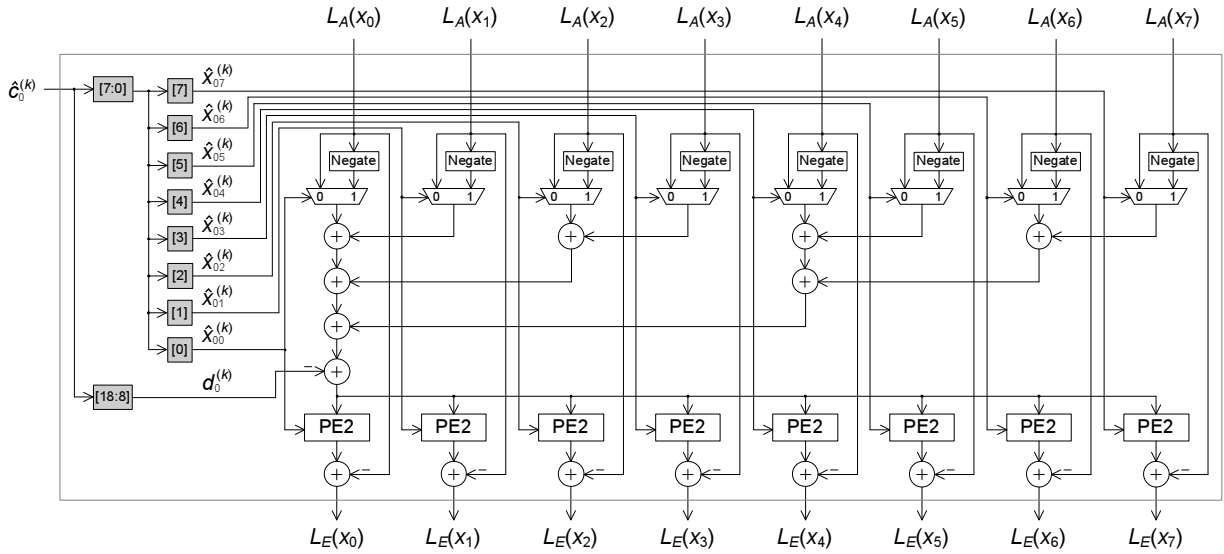


Fig. 14. LLR update module architecture.

TABLE I

TASK SCHEDULING OF THE MIMO DETECTOR UNITS ASSUMING THAT FRAMES *A*, *B*, *C*, AND *D* REQUIRE 4, 2, 0, AND 3 OUTER ITERATIONS RESPECTIVELY.

Tree Search	<i>A</i>	<i>B</i>	<i>C</i>	-	-	-	-	<i>D</i>	-	<i>E</i>	<i>F</i>	-	...
LLR Update	-	<i>A</i> ₀	<i>B</i> ₀	<i>A</i> ₁	<i>B</i> ₁	<i>A</i> ₂	<i>B</i> ₂	<i>C</i> ₀	<i>A</i> ₄	<i>D</i> ₀	<i>E</i> ₀	<i>D</i> ₁	...
LDPC Decoder	-	-	<i>A</i> ₀	<i>B</i> ₀	<i>A</i> ₁	<i>B</i> ₁	<i>A</i> ₂	<i>B</i> ₂	<i>C</i> ₀	<i>A</i> ₄	<i>D</i> ₀	<i>E</i> ₀	...

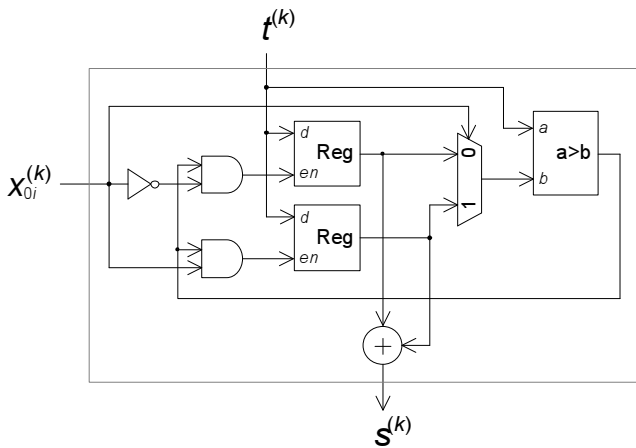


Fig. 15. PE2 used in the LLR update module.

contains 51,840 slices, 324 block RAMs, and 192 DSP slices. Synplicity Synplify 8.8.0.4 has been used for synthesis, and Xilinx ISE 9.1.03i has been used for placement and routing.

Table II compares the computational complexities and delays of various units for choosing the best 64 samples from a pool of 128 samples in M groups of N samples. $M = 16$ which is used in this work, requires about six times fewer comparators and multiplexors and half the delay compared to the $M = 1$ case, while providing no discernible detection performance loss as observed in Fig. 8.

Table III examines the computational complexity and hardware resource requirements for each level of the tree search unit. Note that the comparators and multiplexors in level 1 and

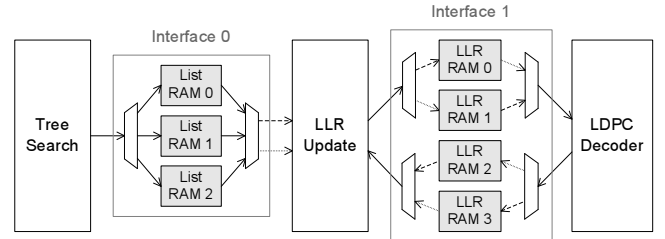


Fig. 16. Interfacing MIMO detector units.

TABLE II

COMPUTATIONAL COMPLEXITIES AND COMBINATORIAL DELAYS OF VARIOUS UNITS FOR CHOOSING THE BEST 64 SAMPLES FROM A POOL OF 128 SAMPLES IN M GROUPS OF N SAMPLES. WHEN $M > 1$, THE APPROXIMATE BEST 64 SAMPLES ARE CHOSEN.

M	N	Comparators	Multiplexors	Delay
16	8	256	448	4 comp, 4 mux
8	16	448	832	7 comp, 7 mux
4	32	704	1,344	11 comp, 11 mux
2	64	1,024	1,984	16 comp, 16 mux
1	128	1,408	2,752	22 comp, 22 mux

0 are occupied by the Best-64 modules. In both level 1 and 0, around 2300 slices are occupied by the Best-64 modules. 192 of the squaring operations are executed via DSP slices in level 3, 2, and 0. As 192 is the total number of DSP slices available in the FPGA considered here, squaring for the other five levels are realized via table lookups. The operands of each

TABLE III
COMPUTATIONAL COMPLEXITY AND HARDWARE RESOURCE REQUIREMENTS FOR EACH LEVEL OF THE TREE SEARCH UNIT.

Level	Adders	Squarers	Comparators	Multiplexors	Slices	Block RAMs	DSP Slices
7	2	2	-	-	26	1	-
6	6	2	-	-	57	1	-
5	22	8	-	-	131	4	-
4	30	8	-	-	185	4	-
3	94	32	-	-	274	-	32
2	126	32	-	-	395	-	64
1	382	128	256	448	4,480	64	-
0	640	128	256	448	5,524	-	128
Total	1,302	340	512	896	11,074	74	192

squarer are 7-bit unsigned numbers (four integer bits and three fractional bits) and the output is 11 bits wide, so a table of $2^7 \times 11 = 1408$ bits is required. We store these tables in block RAMs. Since a block RAM can be dual-ported, two squaring operations can be performed per cycle using a single block RAM. There are a total of 340 squaring operations involved in the detector of which 192 are implemented via 192 DSP slices, and the remaining 148 squaring operations are implemented via 74 block RAMs.

Table IV shows the hardware area requirements of the units inside the MIMO detector. The maximum clock speeds and throughputs that can be supported by each unit are shown. The tree search is fully pipelined into 38 pipeline stages, generating one symbol vector candidate list \mathcal{L} per clock cycle. Its maximum clock speed is 350 MHz which leads to a constant throughput of 1.4 Gbps information bit rate. Each LLR update module (Fig. 14) is pipelined into 4 stages, the feedback shown in Fig. 15 being the critical path. Since each LLR update module can generate 4 information bits every 64 cycles, a single module can achieve 12.5 Mbps at 200 MHz. Since, we need to match the 300 Mbps throughput of the LDPC decoder, we use 24 instances of the LLR update module to perform LLR update. For the LDPC decoder, the pipelined implementation in [18] is adopted, which is capable of 300 Mbps information bit rate at 15 LDPC decoding iterations for the (1944,972) code. The LDPC decoder uses 8 bits for the internal nodes. Note that the slice count of the LDPC decoder is lower than the one reported in [18], because the Virtex-5 FPGA employs 6-input LUTs versus the 4-input LUTs employed by the Virtex-4 FPGA used in [18].

The slices in interface 0 and 1 are used to multiplex the dual and triple buffers respectively. Since there are $1944/n_B = 243$ symbol vectors in a frame, $K = 64$, and each candidate is 19 bits wide (11 bits for distance and 8 bits for bit vector), we need a total of 295,488 bits for each list RAM in interface 0. We partition the data into 12 dual-ported block RAMs such that the 24 LLR update modules can read a candidate every cycle. In interface 1, since each LLR is 11 bits wide and there are 1944 LLRs in the frame, each LLR RAM must hold $11 \times 1944 = 21,384$ bits which comfortably fit into a single block RAM.

When the units in Table IV are integrated to perform MIMO detection, the tree search, interface 0, and interface 1 are down-clocked to match the speeds of the LLR update and

TABLE IV
HARDWARE AREA REQUIREMENTS AND SPEEDS OF THE UNITS INSIDE THE ITERATIVE MIMO DETECTOR. THE LLR UPDATE COMPRISES OF 24 INSTANCES OF THE LLR UPDATE MODULE (FIG. 14). THROUGHPUTS ARE GIVEN IN TERMS OF THE INFORMATION BIT RATE.

Unit	Slices	Block RAMs	DSP Slices	Clock Speed	Throughput
Tree Search	11,074	74	192	350 MHz	1.4 Gbps
Interface 0	608	36	-	350 MHz	-
LLR Update	10,656	1	-	200 MHz	300 Mbps
Interface 1	1,408	4	-	350 MHz	-
LDPC Decoder	10,055	123	-	160 MHz	300 Mbps
Total	33,801	238	192	-	-
Utilization	65%	73%	100%	-	-

LDPC decoder. The tree search for instance, is clocked at 75 MHz to give a throughput of 300 Mbps. The different clocks for the three units are generated via the discrete clock managers available in the FPGA. The latency of the detector ranges from $30 \mu\text{s}$ (zero outer iterations) to $82 \mu\text{s}$ (four outer iterations), while the throughput ranges from 60 Mbps (four outer iterations) to 300 Mbps (zero outer iterations). The throughput is bottlenecked by the LDPC decoder. If a faster LDPC decoder is employed, the tree search and update LLR units can be scaled by increasing the clock speed and using more instances, respectively. For instance, if the (1944,1620) LDPC decoder described in [18] is used, which has a throughput of 900 Mbps, the tree search unit would need to be clocked at 225 MHz and 72 instances of the update LLR module would be required.

It is also of interest to compare the complexity of our K -best ($K = 64$) decoder against a direct implementation which generates the complete list ($K = 256$). We assume that a 1-bit adder, a 1-bit negate, a 1-bit comparator, and a 1-bit multiplexer consist of 5, 5, 5, and 3 gates, respectively, and that an n -bit squarer consists of $n^2/2$ 1-bit adders. Using this estimation methodology, the complexity of our $K = 64$ tree search at 1.4 Gbps is approximately 252,000 gates. The same tree search unit at 300 Mbps can be considered to exhibit an equivalent complexity of 54,000 gates ($252,000 \times 300/1,400$). Similarly, our 300 Mbps LLR update unit can be estimated at 66,000 gates, which is slightly larger than the tree search at 300 Mbps.

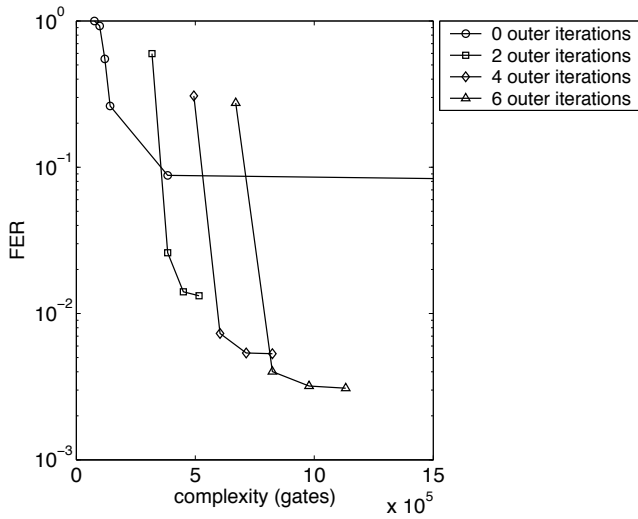


Fig. 17. Variation in FER performance with complexity at $E_b/N_0 = 4$ dB over a 4×4 MIMO channel with QPSK and $K = 64$. The curves are plotted using different numbers of outer iterations and the LDPC iterations used in Fig. 4 (5, 10, 15, 20, 75, and 1000 LDPC iterations. 75 and 1000 are only for 0 outer iterations.)

By contrast, a direct implementation requires the LLR update unit to process $K = 256$, which leads to a factor of four increase in complexity over the $K = 64$ case, i.e. 264,000 gates. Considering that the complexity of the LLR update unit in the direct implementation is much larger than the sum of our tree search and LLR update (264,000 vs 120,000), and the fact that the FER performance gap between our tree search and the direct implementation is very small (as seen in Fig. 6), we believe that it is more computationally efficient to use the K -best tree search as opposed to direct implementation for iterative MIMO detection. When mapped on to a $0.13 \mu\text{m}$ custom ASIC, where a two-input NAND gate is assumed to be $4.5 \mu\text{m}^2$ and the cell area of one-port memory is assumed to be $3.6 \mu\text{m}^2$ [8], our iterative decoder excluding the LDPC decoder is projected to occupy 1.08 mm^2 (including consideration of a 30% routing overhead).

Using the complexity estimation approach above, the complexity of a 4×4 16-QAM iterative MIMO detector is approximately nine times greater than that of QPSK (1,230,000 gates versus 131,000 gates). This large complexity increase is primarily due to the $K = 512$ requirement of 16-QAM (Fig. 7). Therefore a 16-QAM iterative detector would not fit even on the largest FPGA available today and will likely occupy a prohibitive amount of area on custom ASICs. That said, the exponential growth of transistor density will make such implementations economically feasible in the near future.

The complexity of the receiver C_{recv} can be expressed by $C_{recv}(m, n) = C_{ts} + (C_{LLR} + C_{LDPC}(n))(m + 1)$ for $m \neq 0$ and $C_{recv}(m, n) = C_{ts} + C_{LDPC}(n)$ for $m = 0$ where C_{ts} , C_{LLR} , and C_{LDPC} are the complexity of tree search, LLR update and n LDPC iterations, respectively, and m is the number of outer iterations. For the case when no outer iterations are performed (i.e., $m = 0$), the LLR update module can be simplified since there is no feedback from the LDPC decoder, and for simplicity, the complexity of LLR computation for the no outer iteration case can be ignored.

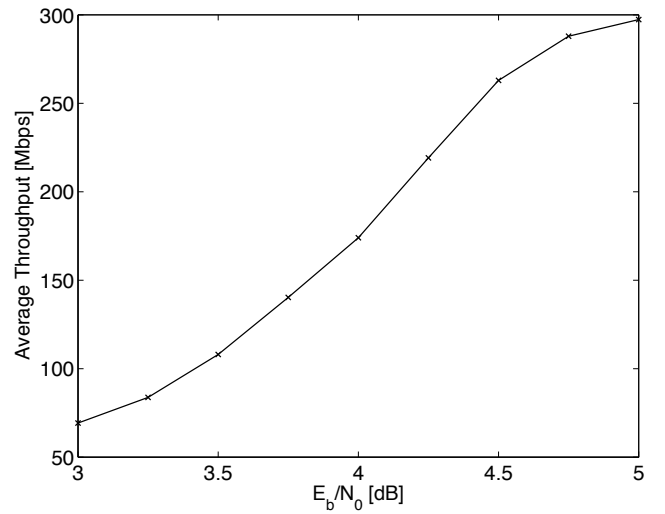


Fig. 18. Average information bit throughput variation with SNR of the 4×4 iterative QPSK MIMO detector. The (1944,972) LDPC code is used. The detector uses a maximum of four outer iterations with fifteen LDPC iterations for each outer iteration. K is set to 64 and the Best-64 group size is set to eight.

Fig. 17 examines the variation in FER with complexity at $E_b/N_0 = 4$ dB with $K = 64$. The curves are plotted using different numbers of outer iterations and the LDPC iterations used in Fig. 4 (5, 10, 15, 20, 75, and 1000). For a given number of outer iterations, the FER performance gain diminishes rapidly with complexity. Furthermore, the performance improvement decreases with increasing numbers of outer iterations.

Fig. 18 characterizes the average throughput variation with SNR of our iterative MIMO detector. At low SNRs the detector approaches the lower bound of 60 Mbps, while the upper bound of 300 Mbps can be achieved at high SNRs. A non-iterative detector (zero outer iterations) would provide a constant throughput of 300 Mbps, which is one to five times faster than that of the iterative detector. Alternatively, from a complexity view point, since Table IV indicates that the complexities of the tree search, LLR update, and LDPC decoder are similar in terms of logic, the complexity of iterative detector ranges from one to four times that of the non-iterative detector, depending on the SNR. In return for that complexity increase, the iterative detector achieves a 1 dB gain in SNR as discussed Section III-A. There will be many applications in which this tradeoff will be well worth making, particularly given that the cost of computations will continue to decrease as hardware technologies continue to evolve, while bandwidth will always remain a scarce and fundamentally limited resource.

VI. CONCLUSIONS

We have presented a hardware architecture and associated implementation for MIMO iterative detection utilizing feedback between a sphere decoder and LDPC decoder. The parameters associated with the K -best sphere decoder have been optimized for hardware implementation, while exhibiting negligible performance loss. The system delivers approximately 1 dB of performance improvement over systems without such

feedback for a 4×4 QPSK MIMO. The iterative detector uses a (1944,972) LDPC code and has been realized on a Xilinx Virtex-5 FPGA. The design is capable of a peak throughput of 300 Mbps information bit rate.

REFERENCES

- [1] "IEEE P802.11n/D1.10," *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, Jan. 2007.
- [2] A. Zanella, M. Chiani, and M. Z. Win, "MMSE reception and successive interference cancellation for MIMO systems with high spectral efficiency," *IEEE Trans. Wireless Commun.*, vol. 4, no. 3, pp. 1244–1253, May 2005.
- [3] A. Matache, C. Jones, and R. Wesel, "Reduced complexity MIMO detectors for LDPC coded systems," in *IEEE Military Communications Conf.*, 2004, pp. 1073–1079.
- [4] D. Garrett, L. Davis, S. ten Brink, B. Hochwald, and G. Knagge, "Silicon complexity for maximum likelihood MIMO detection using sphere decoding," *IEEE J. Solid-State Circuits*, vol. 39, no. 9, pp. 1544–1552, Sept. 2004.
- [5] C. Studer, M. Wenk, A. Burg, and H. Bölcskei, "Soft-output sphere decoding: performance and implementation aspects," in *Proc. IEEE Asilomar Conf. on Signals, Systems and Computers*, 2006, pp. 2071–2076.
- [6] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-best sphere decoding for MIMO detection," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 3, pp. 491–503, Mar. 2006.
- [7] S. Chen, T. Zhang, and Y. Xin, "Relaxed K-best MIMO signal detector design and VLSI implementation," *IEEE Trans. VLSI Syst.*, vol. 15, no. 3, pp. 328–337, Mar. 2007.
- [8] P. Radosavljevic and J. R. Cavallaro, "Soft sphere detection with bounded search for high-throughput MIMO receivers," in *Proc. IEEE Asilomar Conf. Signals, Systems and Computers*, 2006, pp. 1175–1179.
- [9] 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.
- [10] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 670–678, Apr. 2004.
- [11] B. Lu, G. Yue, and X. Wang, "Performance analysis and design optimization of LDPC-coded MIMO OFDM systems," *IEEE Trans. Signal Processing*, vol. 52, no. 2, pp. 348–361, Feb. 2004.
- [12] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," vol. 44, pp. 463–471, 1985.
- [13] C. P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Math. Programming*, vol. 66, pp. 181–191, 1994.
- [14] D. Wübben, R. Böhnke, J. Rinas, V. Kühn, and K. Kammeyer, "Efficient algorithm for decoding layered space-time codes," *IEE Electronics Letters*, vol. 37, no. 22, pp. 1348–1350, Oct. 2001.
- [15] B. Cerato, G. Maser, and P. Nilsson, "Hardware architecture for matrix factorization in MIMO receivers," in *Proc. Great Lakes Symposium on VLSI*, 2007, pp. 196–199.
- [16] L. M. Davis, "Scaled and decoupled Cholesky and QR decompositions with application to spherical MIMO detection," in *Proc. IEEE Wireless Communications and Networking Conf.*, 2003, pp. 326–331.
- [17] K. E. Batchler, "Sorting networks and their applications," in *Spring Joint Computer Conf.*, AFIPS Proc. vol. 32, 1968, pp. 307–314.
- [18] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, "Configurable, high throughput, irregular LDPC decoder architecture: tradeoff analysis and implementation," in *Proc. IEEE Int'l Conf. Application-specific Systems, Architectures and Processors*, 2006, pp. 360–367.



Hyungjin Kim received the BS degree and the MS both in Electrical Engineering Department from Seoul National University in 1997 and 1999, respectively. He is currently a PhD student at the Electrical Engineering Department, University of California, Los Angeles (UCLA). His research interests include MIMO communications, channel coding, sensor networks, compression and encryption algorithms, video/image processing and signal processing. He is a student member of the IEEE.



Dong-U Lee received the BEng degree in Information Systems Engineering and the PhD degree in Computing, both from Imperial College London in 2001 and 2004, respectively. From 2005 to 2007, he was a postdoctoral researcher at the Electrical Engineering Department, University of California, Los Angeles (UCLA), where developed high-performance hardware designs for wireless communications and mathematical function evaluations. He is now a research scientist at Mojix, Inc., Los Angeles, CA, where he is specializing in hardware

implementation aspects of RFID receivers. His research interests include computer arithmetic, communications, design automation, reconfigurable computing, and video image processing. He is a member of the IEEE.



John D. Villasenor received the BS degree in 1985 from the University of Virginia, the MS in 1986 from Stanford University, and the PhD degree in 1989 from Stanford, all in Electrical Engineering. From 1990 to 1992, he was with the Radar Science and Engineering section of the Jet Propulsion Laboratory in Pasadena, California, where he developed methods for imaging the earth from space. He joined the Electrical Engineering Department at the University of California, Los Angeles (UCLA) in 1992, and is currently Professor. He served as

Vice Chair of the Department from 1996 to 2002. At UCLA, his research efforts lie in communications, computing, imaging and video compression, and networking. He is a senior member of the IEEE.