Low-Density Parity-Check Code Constructions for Hardware Implementation

Edward Liao[1], Engling Yeo[2], Borivoje Nikolić
Department of Electrical Engineering & Computer Sciences
University of California, Berkeley
Berkeley, CA, 94720, USA

***Abstract*—We present several hardware architectures to implement low-density parity-check (LDPC) decoders for codes constructed with hierarchical structure. The proposed hierarchical formulation of the LDPC code allows a structured hardware realization of the decoder. For a fully-parallel implementation, there is reduced routing congestion, allowing implementations for blocks sizes up to 1024 bits in 0.13µm technology. Partially and fully serial implementations benefit greatly from the structure of the code as well, leading to several flexible, efficient architectures. In a general purpose 0.13µm technology, the approximate area required by a 1024-bit fully-parallel LDPC decoder is found to be 12.5mm^2 while a serial decoder can be implemented in an area of 0.15mm^2.**

I. Introduction

Low-density parity-check (LDPC) codes [1] have been recently shown to allow communications systems to perform close to the channel capacity limit. High data rate systems using these codes must use dedicated hardware for LDPC decoders. However, this hardware can be quite complex, requiring large silicon area, and are normally power-hungry and throughput limited. In addition, an extra challenge is to build an efficient hardware implementation that is also flexible for variable code rates and block sizes.

To meet these challenges, there has recently been much work done on the study of constructions of LDPC codes with advantages in hardware implementation. Kou, Fossorier and Lin [2] recognized that LDPC encoding can be simplified for codes constructed on finite geometries. Yeo et. al. [3] used the same construction with a modified decoding schedule to significantly reduce the decoder complexity. Zhang and Parhi [4] and Mansour and Shanbhag [5], [6] demonstrated that the decoder can be simplified for regular codes based on algebraically constructed Ramanujan graphs. Hocevar [7] showed a flexible hardware implementation of a code based on permutation matrices.

This paper studies LDPC codes and proposes a code construction that produces codes with properties favoring hardware implementation along with good BER performance with low error floors. The structure of this proposed construction can be exploited in several ways for different architectures supporting various degrees of flexibility and throughput. For fully-parallel, very high throughput decoders, these codes can reduce the amount of routing, making this a more feasible option for blocks sizes up to 1024 bits. In addition, these codes allow partially-serial, flexible architectures with increased efficiency and relatilvey high throughputs.

II. LDPC Decoding

The design of LDPC codes can typically be defined by an $N \times M$ parity-check matrix **H**. The symbol N, represents the length of the block (i.e. the number of bit in the code), while the symbol M, represents the number of parity checks in the code. The rate of such a code is thus $(N-M)/N$. The LDPC code can be represented by a bipartite graph of bit nodes and check nodes as shown in Figure 1. An edge between a bit node n, and check node m, exists if the entry in the n^{th} column and m^{th} row of **H** is non-zero. In addition, the LDPC code is defined as regular if each bit node is adjacent to an equal number of check nodes, d_v, and likewise that each check node is adjacent to an equal number of bit nodes, d_c. This code is then described as a regular (d_v, d_c) LDPC code.

The sum-product algorithm that is used for LDPC decoding has two phases. In the first phase, the bit nodes compute updated information which is sent to adjacent check nodes. In the second phase, the check nodes compute updated information based on the new messages from the bit nodes. This update information is then sent back to adjacent bit nodes and the process is repeated.

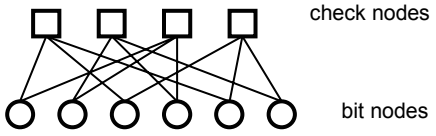

Figure 1. Bipartite graph for a regular (2, 3) LDPC code.

An LDPC decoder can be realized in hardware to reflect the bipartite graph representation. Such a decoder would consist of N bit-node processing elements and M check-node processing elements interconnected through a network of wires. The number of interconnect wires required in such a decoder is $2N \cdot d_v \cdot w$ where w represents the bit width of each message. The problem with this implementation is that in a randomly generated LDPC code, there is little structure in the interconnect network. The average wire length becomes very large causing such parallel decoders to be very area inefficient and thus very costly [8]. While parallel decoders can achieve very high throughputs with low power consumption, their cost is frequently prohibitive for practical communications applications. One solution to this problem is to construct codes

[1] Presently with Qualcomm Inc., San Diego, CA
[2] Presently with ST Microelectronics, Berkeley, CA

that have an ordered structure to reduce the interconnect complexity of the decoder.

III. Algebraic Constructions

There have recently been several proposed LDPC codes with ordered structure based on algebraic constructions [9], [10]. These algebraic constructions make use of known graphs with properties that achieve good bit error rate (BER) performance. Several properties are desirable for the code to achieve good performance via the sum-product algorithm. First, the bipartite graph representing the code must have a large girth or minimum cycle length. Second, the graph must be a good expander. Finally, there should be a large minimum Hamming distance between codewords.

One method to control the structure of the LDPC code is to describe the parity check matrix algebraically. Various mathematical techniques can be used to produce a bipartite graph with the desired features. One of these techniques is to use Ramanujan graphs, which are *k*-regular graphs (i.e., graphs with all nodes having edge degree *k*) that have certain optimality in their expansion behavior. The formal definition of a Ramanujan graph is that the second largest eigenvalue of the adjacency matrix is not larger than $2\sqrt{k-1}$. An infinite family of Ramanujan graphs is well-known and can be algebraically constructed [11]. Each graph in this family is defined by two parameters, p and q, which must be equivalent to 1 modulo 4 and each Ramanujan graph is denoted by $\mathbf{X_{p,q}}$. The size of $\mathbf{X_{p,q}}$ is known to be $(p^3-p)/2$. Another interesting property exhibited by these graphs is that they have an equal number of nodes in both bipartite sets. By performing various transformations on these bipartite graphs, a parity check matrix can be constructed with feasible rates. For example, one could simply take each node in one of the bipartite sets and splits it into two nodes, with each node taking half the edges [6]. This method of construction would result in a rate-½ code with girth as large as the original Ramanujan graph.

Although these graphs display large girth and good expander properties, the resultant codes are constructed without consideration of the minimum Hamming distance. In fact, these codes turn out to have poor error floors caused by low-weight codewords and near codewords [12]. These near codewords are sets where the decoder gets trapped in a state where incorrect decisions remain within the decoder and the parity check matrix is not satisfied, but it cannot correct the existing errors.

An LDPC construction based on the Ramanujan graph $\mathbf{X_{13,5}}$ leads to a rate-½ 2184-bit code and has a very poor error floor around a block error rate of 0.001 due to low-weight code words. Similarly, codes constructed based on the Ramanujan graph $\mathbf{X_{17,5}}$ leading to a rate-½ 4896-bit code has an error floor around a block error rate of 10^{-7} due to a family of near codewords.

IV. Proposed Construction

We demonstrate a hierarchical design approach that combines the desirable features of several of the introduced code construction techniques, while overcoming the low-weight codeword problem. The most advantageous features of the previously described codes are the large minimum cycle length, expansion properties, and structured interconnect. The structure of interconnect for the decoder can be seen by looking at the parity check matrix of these codes. The resultant LDPC code constructed from Ramanujan graphs is a regular (3, 6) LDPC code with structured placement of the parity checks. Specifically, the matrix can be partitioned into 28×14 sub-matrices, where each entry in the sub-matrix is either a 78×78 zero-matrix of all 0s or a 78×78 permutation matrix (a square matrix with a single 1 in every column and a single 1 in every row with dimensions 78×78). In the top-level matrix, denoted here by **H***, each row has 6 permutation sub-matrices and 22 zero sub-matrices. Conversely, each column of **H*** has 3 permutation sub-matrices and 11 all-zero sub-matrices. These features allow the decoder to be designed in a hierarchical manner.

In addition to the structure, the codes based on the family of algebraically constructed Ramanujan graphs have good expansion properties. Furthermore, it is known that for small values of k, the probability of a randomly constructed k-graph being Ramanujan is rather high, with the probability of a graph with N nodes being Ramanujan approaching 1 as N grows. With $k = 8$ and $N > 1024$, the probability of the graph being Ramanujan is over 0.8. Using this fact, a random, bipartite regular k-graph of any size can be constructed with good probability of satisfying the Ramanujan criteria and simply splitting one set of the nodes into two will lead to a rate-½ LDPC code.

This leads to a proposed set of rate-½ LDPC codes with structured interconnect and good expansion and girth properties. These LDPC codes can be described as follows. Each code has N bits and $N/2$ check nodes. The parity check matrix is partitioned into $P \times P/2$ sub-matrices. Each sub-matrix is either a square permutation matrix or zero matrix with dimensions $N/M \times N/M$. The top-level matrix can be constructed randomly, with the constraint that there are d_v permutation matrix entries in each column and $2d_v$ permutation matrix entries in each row where d_v is the edge degree of the bit node desired in the resulting LDPC. Similarly, each permutation matrix entry can be randomly constructed. If necessary, the permutation matrix can be partitioned much like the top-level parity check matrix to add another level of hierarchy which might further ease the routing complexity. The resulting parity check matrix is equivalent to a random bipartite $2d_v$-regular graph with N total nodes and taking one bipartite set and splitting each of the nodes into two nodes. Since the original bipartite $2d_v$-regular graph has high probability of being Ramanujan for small d_v, the graph will most likely have a large minimum cycle length. An example of a code constructed in the described manner is shown in Figure 3. Here, a 24x12 parity-check matrix is constructed from a 6x3 top-level matrix, **H***. The top-level parity-check matrix is randomly constructed as are each permutation sub-matrix.

The performance of a code generated in the proposed fashion was simulated along with a completely randomly generated code and the results shown in Figure 2. The codes are both regular (4, 8) rate-½ LDPCs using 1024 bit blocks. The proposed code uses a parity check matrix partitioned with

a top-level 32x16 parity check matrix with 32x32 matrix entries. Down to a block error rate of 10^{-6}, this code shows no change in BER slope down to a block error rate of 10^{-6}. The error floor still exists below this level [12], but is not raised compared to previous constructions.

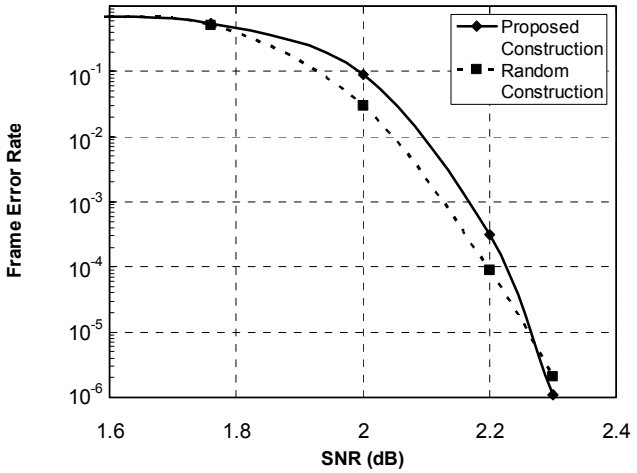

Figure 2. BER curves for proposed LDPC codes, maximum 64 iterations.

V. Decoder Architectures

In addition to having good performance, the hierarchical structure of the codes also allows easier hardware implementation of the decoder. In this section, several decoder architectures are analyzed, illustrating how each one takes advantage of the hierarchical structure of the code. A fully-parallel architecture is shown to be feasible due to the structure of the interconnect introduced by the code construction, two different architectures partially serializing the decoder can be created naturally from the structure of the code, and finally, a fully-serial architecture can be shown to take advantage of the code properties as well.

A. Parallel Decoder

As previously mentioned, the primary issue with the physical design of a fully parallel decoder is minimizing the level of routing congestion. This is typically achieved through careful partitioning of the large groups of processing elements such that the number of long global interconnects is minimized. In general, this is a difficult problem because LDPC codes have the tendency to be random and unstructured [13]. However, in our proposed construction, the processing elements can be partitioned naturally along the lines of the hierarchy according to the partitioned parity check matrix **H***. Each sub-matrix represents a group of check nodes and bit nodes. The messages are likewise clustered and routed together between groups of processing elements. This scheme provides increased structure for the decoder implementation.

The routing complexity within each group of N/P check nodes can be interpreted by partitioning the top-level parity check matrix into horizontal bands with N/P rows. Each partition comprises of a concatenation of $2d_v$ different permutation matrices, separated by zero-matrices, as shown in Figure 3. These zero-matrices do not affect the routing of the decoder and are therefore removed, forming a reduced matrix with dimensions $(N/P)\times(2\cdot d_v\cdot N/P)$. Thus, the routing problem is reduced to the equivalent of routing an LDPC decoder with $2\cdot d_v\cdot N/P$ bit nodes and N/P check nodes.

The proposed code design provides an N-bit regular $(d_v, 2d_v)$ code, which can be broken up hierarchically. At the top level is a rate-½ LDPC with M "bit" nodes. The next level of hierarchy is very similar to a regular $(1, 2\cdot d_v)$ $2\cdot d_v\cdot N/P$ bit LDPC decoder. This check node group can be designed hierarchically as well if needed by simply designing a code with partitioned permutation matrices. The floorplan for a 1024-bit rate-½ LDPC decoder is described in [13] and a similar placement strategy can be followed for this decoder and for each check node group in the decoder. In this floorplan, bit nodes are placed along the outer edges of the chip and the check nodes and routing are placed in the middle of the ring formed by the bit nodes.

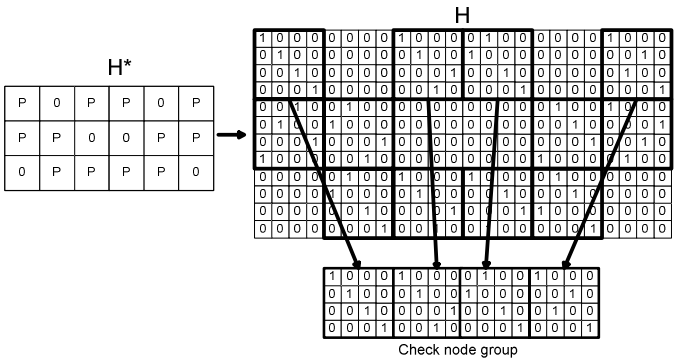

Figure 3. Formation of parity check matrix, **H**, from top level matrix **H*** and matrix for one check node group.

In order to obtain area estimates, the processing elements were synthesized in 0.13µm general-purpose standard-cell CMOS technology. Each check node group consisting of 32 check nodes is approximately 500µm x 500µm and each bit node group is approximately 275µm x 275µm. With no routing congestion, the total area for the parallel decoder would be approximately 8mm^2. To analyze the utilization, we must estimate wiring lengths for the proposed placement. First, we count the total number of wires at the top level. For a (4, 8) regular LDPC code with 1024 bits per block, there are $4\cdot1024\cdot2 = 8192$ messages passed between bit and check nodes and vice versa. Using 4 bit messages, this requires 32,768 wires. A pessimistic wire length can be approximated assuming a uniform distribution over the check node groups (i.e.: no optimizations on placement) and assuming no spacing between check node groups for routing. The average wire length is found to be approximately 1mm, which would require a total wire length of 32,768mm or an area of 6.55mm^2. Although multiple metal layers will reduce the actual area on a chip, this area is still significant, affecting the decoder size estimate. By iterating through these computations, a good approximation for the area of the decoder can be found. For this chip, the total routing area is approximately 4.5mm^2 and the total chip area is 12.5mm^2 for a utilization of 64%. Likewise, the same calculations can be done for a (3, 6) regular LDPC code, which

has smaller bit and check nodes (total area of 7mm^2). The routing area is found to be approximately 3.2mm^2, for a logic density utilization of 69%.

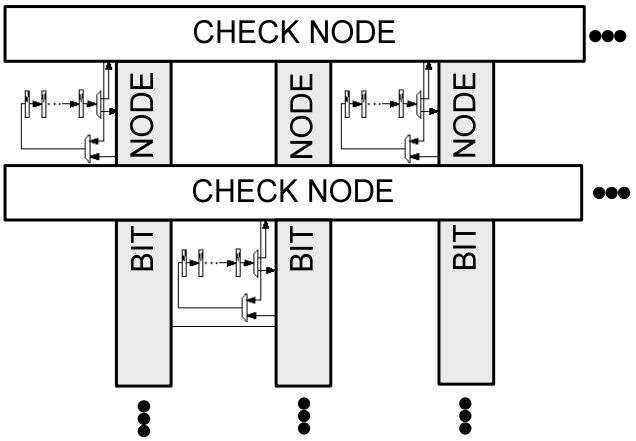

Figure 4. Shift-register structure for LDPC decoders.

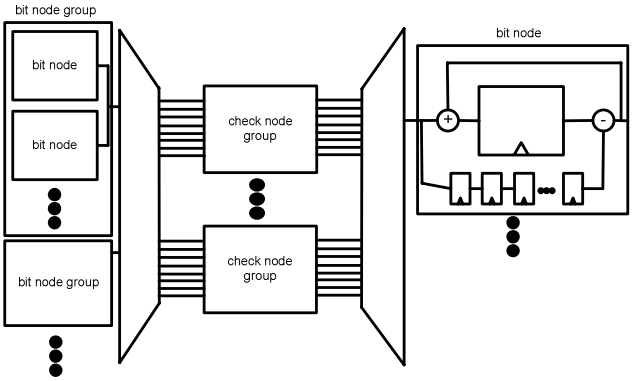

Figure 5. Second implementation of partially parallel decoder.

B. Parallel/Serial Implementations

The logic density can be increased through partially serial implementations. There are two natural ways of serializing the decoder. The first implementation would keep the multiple copies of the check node groups, but serialize the group so that it processes only one check node per cycle. The second implementation would be to multiplex several parallel check node groups among all the bit node groups.

The first possible implementation is shown in I.AThis figure shows a structure that uses a single check node processing element to serially evaluate all check-to-variable messages from a group of check nodes. Similarly, all message computations corresponding to a group of bit nodes are multiplexed onto a single bit node processing element. Thus, the decoder for the 1024-bit code example uses 16 check node processing elements and 32 bit node processing elements. Besides taking advantage of the hierarchical code design, this structure also exploits the property that each row of the second-level sub-matrices has a maximum edge degree of one. The messages corresponding to each permutation sub-matrix are stored in a shift-register chain. Unlike previous uses of shift registers in LDPC decoders [3], [14], [15], this does not require consecutive rows in the parity check matrix to be cyclic. The processing elements are arranged in a tight grid surrounding the shift registers. The zero-matrices do not require any storage of messages, and correspond to the empty spaces between the grid in I.AThe area of the decoder is approximated by performing synthesis in 0.13µm general-purpose CMOS technology. The 16 check-node and 32 bit-node processing elements occupy 1.1mm^2 and 0.6mm^2 respectively. The shift registers will occupy another 0.5mm^2. The structure of the decoder interconnect enhances the logic density, which is estimated to be 0.8. This results in a 1024-bit decoder that is 2.8mm^2 and capable of decoding at 1Gb/s.

The shift register architecture decoder can be made programmable with some overhead. To support this, a large grid of the check and bit nodes can be laid out with extra shift registers throughout the grid. This grid can be programmed to the parity check matrix by turning the shift registers on or setting them to shift out a no information message to the bit and check nodes.

The second partially parallel implementation is shown in Figure 5. Instead of using all check node groups operating in parallel, only a few check node groups are used along with multiplexers and demultiplexers to appropriately route the messages from the bit nodes to the check node groups and from the check node groups back to the bit nodes. A similar architecture has been recently reported for structured code constructions [15]. This implementation still takes advantage of the fact that the check nodes and bit nodes can easily be grouped together to simplify multiplexing involved at the top level. The check node groups would be the same as a check node groups in the parallel decoder, but the bit node could be simplified to be a simple accumulator register with a shift register. The check node message would be added to the message in the bit node and subtracted out when the message was passed back to that check node. Even further simplification could be used if the approximation in [3] is used where the check node message is not subtracted out. This would save the cost of the storing all the check-to-bit messages within the decoder, a large saving in storage. This results in a 32-bit node group of approximately 0.1mm^2 and a check node group of 0.25mm^2. With additional multiplexing and routing, the area of this decoder is approximately 3.5mm^2 running at approximately 300MHz. This implementation has the advantage it can easily be used to support various LDPC codes constructed in the proposed manner with various block sizes and code rates. The only overhead would be to add enough bit node groups to support as large a block as needed and configure the code through the control signals to the multiplexers.

C. Serial Implementation

In area-constrained applications, the size of the decoder can be further reduced by using a fully serial decoder with SRAM to store the messages, and a single processing element for all message computations. In general, serial architectures have the throughput limited by the implementation of multi-ported SRAM [8]. However, the hierarchical LDPC code allows the decoder implementation to make use of eight independent smaller SRAMs, which are inherently faster. Each bit node group would have a memory consisting of all the check to bit

node messages. Each cycle, the messages are read out, summed together (subtracting out the one check-to-bit message) and processed by a check node. The new check-to-bit message would then be stored in the memory. The requirement for this decoder is dominated by the 32 1kbit (32x8x4) SRAMs. In 0.13µm CMOS, a single 32x32 SRAM will occupy an approximate area of 15,000µm^2 and operate at up to 500MHz. Thus the 0.51mm^2 serial decoder will be capable of a decoding throughput above 100Mb/s, depending on desired number of decoding iterations. It is also possible to use the approximation of [3], [15] where the check to bit message is not subtracted. In this case, only a single message needs to be stored for each bit node and each of the 32 SRAMs need only be 128 bits (32x4), consuming an area of 3400µm^2 for a total of 0.15mm^2.

D. Comparison of Architectures

Table I summarizes the different architectures described here for a 1024-bit rate-½ LDPC decoder. It is clear from this table that with the proposed code construction, a wide range of applications can be supported with variable throughput and hardware cost requirements. In addition, many of these architectures can support multiple block sizes and code rates with appropriate code construction.

TABLE I. COMPARISON OF ARCHITECTURES

Architecture type	Approximate Area (mm^2)	Estimated Clock speed (MHz)	Estimated Throughput, 10 iterations (Gb/s)
Fully Parallel	12.5	300	30.0
Shift-register based	2.8	1000	3.2
Muxed check node group	3.5	300	1.1
Fully Serial	0.15	500	0.1

For applications requiring very high throughput decoding, a fully parallel decoder is a feasible solution, requiring 12.5mm^2 with an estimated throughput of 30 Gb/s. However, a drawback for this decoder is that it is not as easily configurable for different block sizes or code rates. The shift register architecture has a smaller area while still supporting up to 3.2 Gb/s. While the area is smaller than the multiplexed check node group architecture with a larger throughput, it is clear that the multiplexed check node is much more easily configured to support multiple LDPC codes. The shift register architecture can support multiple LDPC codes as described, but there is quite a bit of overhead to do so whereas the multiplexed check node group architecture just needs to change some control logic. Finally, for applications where throughput is not as important, a fully-serial architecture can be used.

VI. CONCLUSIONS

A family of randomly constructed hierarchical LDPC codes has been proposed. These codes have intrinsic properties which ensure good BER performance via the sum-product algorithm. For an example 1024-bit block, the code has error floor below the block error rate of 10^{-6}. It also exhibits good performance compared to randomly generated LDPC codes.

Due to the hierarchical nature of the code, the feasibility of hardware implementation of the decoder has been significantly improved. Four different architectures have been demonstrated. These architectures contrast in terms of area of implementation and achievable throughputs. A 1024-bit fully-parallel decoder architecture requires 12mm^2 and achieves 30 Gb/s decoding throughput. Partially serializing the decoding leads to two additional architectures which were shown to have smaller area than the fully-parallel decoder, while maintaining a relatively high throughput of 1Gb/s. The area can be further reduced if memory is eliminated from the design by using an approximation to the sum-product algorithm. Finally, a serial architecture was shown to have high memory efficiency and an area as small as 0.15mm^2, though throughput is reduced accordingly to 100Mb/s. The architectures demonstrate that these codes are suitable for a range of applications with different throughput and area constraints.

REFERENCES

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[2] Y. Kou, S. Lin and M. Fossorier, "Low density parity check codes based on finite geometries: A rediscovery and more," *IEEE Trans. on Information Theory*, pp. 2711-2736, Oct. 1999.

[3] E. Yeo, P. Pakzad, B. Nikolić, and V. Anantharam, "High throughput low-density parity-check decoder architectures," *Proc. IEEE GLOBECOM 2001*, Nov. 2001, pp. 3019-3024.

[4] T. Zhang and Parhi, "A 54 Mbps (3, 6)-regular FPGA LDPC decoder", *IEEE Workshop on Signal Proc. Systems, 2002. (SIPS '02)*, Oct. 2002, pp. 127 -132.

[5] M. Mansour, N. Shambhag "Memory-efficent turbo decoder architectures for LDPC codes" *IEEE Workshop on Signal Proc. Systems, 2002. (SiPS 2002),* Oct. 2002, pp. 159-64.

[6] M. Mansour and N. Shanbhag, "Architecture-aware low-density parity-check codes," *Proc. 2003 Int. Symp. on Circuits and Systems*, *ISCAS '03*, vol. 2, May 25-28, 2003. pp. 57-60.

[7] D.E. Hocevar, "LDPC code construction with flexible hardware implementation," *Proc. IEEE Int. Conf. on Comm.*, ICC'03, vol 4 , 2003, pp. 2708 -2712.

[8] E. Yeo, B. Nikolić, and V. Anantharam, "Iterative decoder architectures," *IEEE Communications Magazine*, vol. 41, no.8, pp. 132-140. Aug 2003.

[9] J. Rosenthal and P.O Vontobel, "Constructions of LDPC codes using Ramanujan graphs and ideas from Margulis," *Proc. 38th Annual Allerton Conference on Communication, Control, and Computing*, 2003. pp. 248-257.

[10] J. Lafferty, D. Rockmore, "Code and Iterative Decoding on Algebraic Expander Graphs". *International Symposium on Information Theory and its Applications.* Nov. 2000.

[11] A. Lubotzky, R. Phillips, and P. Sarnak, "Ramanujan graphs," *Combinatoria,* vol. 8, no. 3, pp. 261-277, 1988.

[12] D. MacKay and M. Postol, "Weaknesses of margulis and ramanujan-margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, 2003.

[13] A. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate 1/2 low-density parity-check code decoder," *IEEE J. Sold-State Circuits*, vol. 37, no. 3, pp. 404-412, March 2002.

[14] R. Lynch, R. Lynch, E. Kurtas, A. Kuznetsov, E. Yeo, and B. Nikolic, "The search for a practical iterative detector for magnetic recording," *Digests of The Magnetic Recording Conference, TMRC 2003*, p. E6, Santa Clara, CA, Aug 18-20, 2003.

[15] S. Olcer, "Decoding architecutre for array-code-based LDPC codes," *Proc. IEEE GLOBECOM 2003,* Dec. 2003, pp. 2046-2050.