Generalized Low Density Parity Check Codes

Eran Sharon, Ran Zamir, Dudy Avraham - Western Digital®

A. Abstract

Error Correction Coding (ECC) technology is a key enabler for high-density storage, allowing reliable storage over increasingly unreliable media due to memory process scaling. The race for improved memory cost efficiency and performance fuels the search for efficient ECC solutions, maximizing the correction capability for a given overprovisioning, while having low complexity and low power encoding and decoding methods. Today's state of the art ECC solutions used in storage applications are based on Low Density Parity Check (LDPC) codes. In this comprehensive study, we explore Generalized LDPC (GLDPC) codes, including development of code design and generation tools and efficient low complexity encoding and decoding solutions. We show that the designed GLDPC codes outperform existing state of the art LDPC codes.

B. Generalized LDPC codes

LDPC codes are commonly described using a sparse bipartite graph, where the left nodes represent the codeword bits and the right nodes represent parity check constraints that the bits should satisfy in order to constitute a valid codeword (as shown in Figure 1a). LDPC codes are decoded iteratively by exchanging information between the left bit nodes and right check nodes of the graph. An equivalent description is to view each left node as representing a *repetition code* constraint and each right node as representing a *parity check code* constraint. According to this view, checking for a valid codeword can be performed by encoding the codeword bits onto the edges using the left repetition codes and verifying that the encoded edge values satisfy all the constraints imposed by the right parity check codes. Additionally, the iterative decoding can be viewed as exchanging information between Maximum A-Posteriori (MAP) decoders of the repetition codes and MAP decoders of the parity check codes.

With this view in mind, one may consider a generalized definition. Why should we limit ourselves to repetition codes and parity check codes? We may use other constituent codes as the graph nodes, such as Hamming codes, Simplex codes or Reed Muller codes, as shown in Figure 1b. We refer to such a code, described by a sparse bipartite graph, with general constituent codes imposing its nodes' constraints, as a GLDPC code.

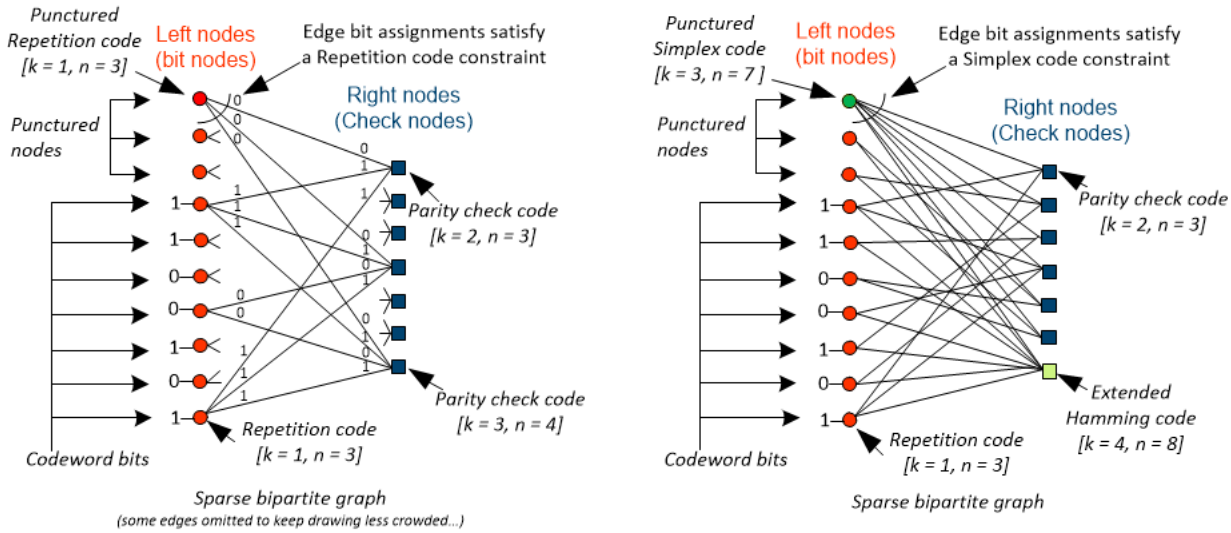


Figure1: a. LDPC code **b. GLDPC code**

C. Low complexity Encoding and Decoding of GLDPC Codes

One important consideration when selecting the constituent codes for a GLDPC is whether a low complexity MAP decoder is available for them. For a general length n block code, the complexity of a MAP decoder is exponential ($O(2^n)$). In that sense, repetition and parity check codes are very good choices as they have linear complexity MAP decoders ($O(n)$). With that in mind, for our GLDPC constructions, we limited ourselves to several additional constituent code options (on top of the repetition and parity-check codes): Hamming, extended Hamming, Simplex and 1st order Reed Muller (of different lengths). For these additional codes we developed low complexity MAP decoders of order $O(n \cdot \log(n))$ based on the concepts of [1]. These MAP decoders make use of a Fast Walsh-Hadamard transform (enabled by the structure of these code families).

The GLDPC decoding can then be viewed as an iterative exchange of information between the low complexity MAP decoders of the small constituent codes connected through a random interleaver (represented by the sparse bipartite graph), as shown in Figure 2. By keeping the fraction of "generalized" nodes in the graph small and keeping the overall number of edges in the graph similar to that of a conventional LDPC code, we ensured that the overall GLDPC decoding complexity is kept similar to that of conventional LDPC codes. We also maintained the Quasi-Cyclic code structure, which is customary in today's commercial LDPC designs in order to enable a high throughput GLDPC decoder implementation.

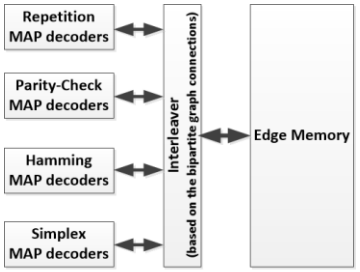


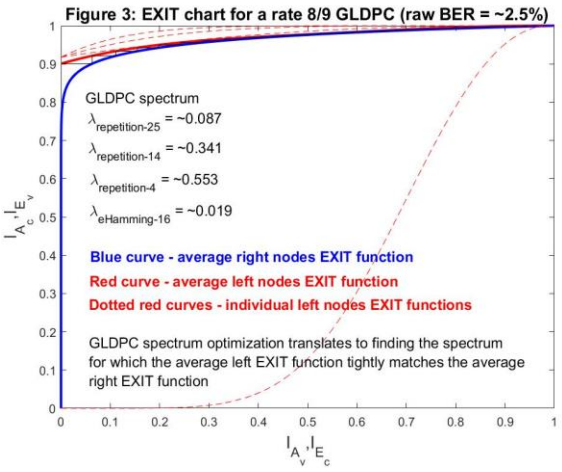
Figure 2: GLDPC decoder

For encoding, we developed a linear complexity algorithm based on the sparse parity check matrix representation of the GLDPC code. This was done by forcing a lower triangular structure on the parity check matrix and by implementing a linear complexity ($O(n)$) encoding algorithm for the "generalized" nodes (again by using their recursive Hadamard structure).

D. Designing and Generating Capacity Approaching GLDPC Codes

One of the main challenges we had to overcome as part of this research is how to design and generate good GLDPC codes. That is, how to select the optimal mixture of constituent codes of the GLDPC code and how to connect them. For that purpose, we developed the following tools:

1. **GLDPC spectrum optimizer** – finds the optimal fraction of nodes from each constituent code type (referred to as "spectrum") for maximizing the correction capability of a GLDPC code with a given rate. The tool is based on asymptotic analysis of the performance of a GLDPC ensemble, by tracking the Extrinsic Information Transfer (EXIT) between its constituent codes. For that purpose we had to derive expressions for the EXIT functions of different constituent codes (such as the Hamming, Simplex and Reed Muller codes), as described in [2,3]. We then expressed the relations between the EXIT functions of the constituent codes that need to be satisfied in order for the GLDPC decoder to converge, as a set of linear constraints. This enabled us to use a linear programming optimizer for finding the spectrum that would maximize the GLDPC ensemble's correction capability. The convergence process of the GLDPC decoder, as predicted by the EXIT analysis, including the optimal spectrum (for a given code rate of 8/9) is shown in Figure 3.



2. **GLDPC graph generator** – constructs a GLDPC graph satisfying a defined spectrum. This tool is based on a modified version of the Error Minimization Progressive Edge Growth (EMPEG) algorithm that we developed previously for LDPC codes [4]. The EMPEG algorithm generates the graph edge by edge, while avoiding detrimental configurations that may lead to an error floor.

E. Simulation Results

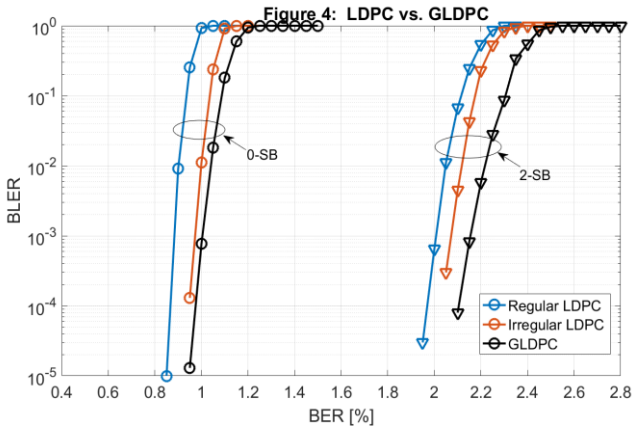
In order to evaluate the potential of GLDPC codes, we compared three [N = 4608B,K = 4096B, R = 8/9] codes:

a. (4,36)-regular LDPC code

b. Optimized Irregular LDPC code

c. Optimized Irregular GLDPC code

All codes are Quasi-Cyclic with lifting factor Z = 256. As shown by the simulation results on Figure 4, the extra degrees of freedom that exist in GLDPC codes enable them to outperform state of the art LDPC codes. Note that these are preliminary results. The GLDPC framework opens a huge search space and we expect further improvements across various ECC metrics.



References

[1] A.Ashikhmin and S. Litsyn "Simple MAP decoding of First Order Reed Muller and Hamming Codes," IEEE Trans. Info. Theory, vol. 50, pp 1812-1818, Aug. 2004.

[2] E. Sharon, A. Ashikhmin, S. Litsyn, ``EXIT functions for binary input memoryless symmetric channels," IEEE Trans. on Communications, vol. 54, pp. 1207--1214, July, 2006.

[3] E. Sharon, A. Ashikhmin, S. Litsyn, ``Analysis of low-density parity-check codes based on EXIT functions," IEEE Trans. on Communications, vol. 54, pp. 1407-1414, Aug. 2006.

[4] E. Sharon and S. Litsyn, ``Constructing LDPC codes by error minimization progressive edge growth," IEEE Trans. on Communications, vol. 56, pp. 359-368, March 2008.