Joint $(3, k)$-Regular LDPC Code and Decoder/Encoder Design

Tong Zhang* and Keshab K. Parhi

ECE Dept. University of Minnesota

4-174 EE/CSci Building

200 Union Street S. E.

Minneapolis, MN 55455, USA.

E-mail: {tzhang, parhi}@ece.umn.edu

EDICS: 5-VLSI VLSI FOR SIGNAL PROCESSING

Abstract

In the past few years, Gallager's Low-Density Parity-Check (LDPC) codes have received a lot of attention and tremendous efforts have been devoted to analyze and improve their error-correcting performance. However, little consideration has been given to the practical LDPC codec hardware implementations. The straightforward fully parallel LDPC decoder architecture usually incurs too high complexity for many practical purposes, thus effective partly parallel decoder design approaches are highly desirable. Due to the randomness of LDPC code, it is extremely difficult, if not impossible, to develop a direct transformation from fully parallel architecture to partly parallel ones for a given LDPC code. Meanwhile, also because of its randomness, the direct LDPC encoding scheme has quadratic complexity in the block length, which makes the efficient encoder design not trivial. We believe that jointly conceiving code construction and decoder/encoder design is a promising direction to develop high performance LDPC coding systems. In this paper, we propose a joint $(3, k)$-regular LDPC code and decoder/encoder design approach to construct a class of $(3, k)$-regular LDPC codes that not only have very good performance but also exactly fit to a high-speed partly parallel decoder and low-complexity encoder. Moreover, we propose a modified joint design approach in order to further reduce the decoder hardware complexity for those high-rate $(3, k)$-regular LDPC codes applied to silicon area critical applications.

Submitted to the IEEE Transactions on Signal Processing

1 Introduction

Low-Density Parity-Check (LDPC) codes have recently received a lot of attention because of their excellent performance and have been widely considered as a promising candidate error-correcting coding scheme for many applications in telecommunications and magnetic storage. Nearly 40 years ago, Gallager invented LDPC codes [1][2], in which two innovative ideas were exploited: iterative decoding and constrained random code construction. However, except for the papers of Zyablov and Pinsker [3], Tanner [4] and Margulis [5], Gallager's work was neglected by the majority of the scientific community for the next 30 years. Few years after the birth of Turbo codes [6] in which both the above ideas are employed, LDPC codes were independently rediscovered by Wiberg [7] and MacKay and Neal [8].

An LDPC code is defined as the null space of a very sparse $M \times N$ parity check matrix, and typically is represented by a bipartite graph[1], usually called *Tanner graph*, between N *variable* (or *message*) nodes in one set and M *check* (or *constraint*) nodes in another set. The LDPC code is called (j, k)-regular if each variable node has a degree of j and each check node has a degree of k. The construction of LDPC code (or its Tanner graph) is typically random. LDPC codes can be effectively decoded by the iterative belief-propagation (BP) (also known as sum-product) algorithm [9]. The structure of BP decoding algorithm directly matches the Tanner graph: decoding message is computed on each variable node and check node and iteratively exchanged through the edges between the neighboring nodes. It is well known that BP decoding algorithm works well if the underlying Tanner graph does not contain too many short cycles. Thus, it is typically required that the random Tanner graph should be 4-cycle free which is easy to achieve, but the construction of random Tanner graph with higher order cycle free is not trivial.

In the past few years, LDPC codes have become a hot topic and tremendous efforts have been devoted to effectively analyze and improve the code performance, see [10]-[18], *etc.* Besides their excellent performance, another important reason why LDPC codes attract so many attentions is that the BP decoding algorithm is *inherently* fully parallelizable and the computation associated with each node (variable node or check node) is very simple, thus a great potential decoding speed can be expected. In this work, we are interested in the hardware implementations of LDPC decoder/encoder. The authors of [19] design the decoder by *directly* mapping BP algorithm to the hardware: each variable node and check node are assigned their own processors and all the processors are connected through an interconnection network which exactly reflects the Tanner graph. Applying such fully parallel decoder architecture, the authors of [19] physically implemented a 1024-

[1] A bipartite graph is one in which the nodes can be partitioned into two disjoint sets.

bit, rate-1/2 LDPC decoder with the maximum symbol throughput of 1 Gbit/s. By completely exploiting the parallelism of the decoding algorithm, such fully parallel design could achieve very high decoding speed. However, with the increase of LDPC code length, both *computation* complexity due to the implementation of all the processors and *communication* complexity due to the implementation of the entire interconnection network will incur very high hardware complexity. Especially, as mentioned in [19], the routing overhead for implementing the entire interconnection network will become quite formidable due to the randomness of the Tanner graph. Therefore, such fully parallel design scheme may not be suitable for many practical purposes, even short code length (less than 10,000 bits) is used, thus high-speed partly parallel decoder design approaches that can efficiently trade speed for hardware complexity is highly desirable.

In the partly parallel decoding, the computations associated with a certain number of variable nodes or check nodes are time-multiplexed to a single processor. Meanwhile, since the computation associated with each node is not complicated, the fully parallel interconnection network reflecting the Tanner graph should be transformed to partly parallel ones for both communication complexity reduction and high-speed partly parallel decoding. Unfortunately, the randomness of Tanner graph make it nearly impossible to develop such transformation. In another word, an arbitrary random LDPC code has a little chance to be suited for a high-speed partly parallel hardware decoder implementation. To circumvent this problem, the authors of [21] propose to reverse the conventional design sequence: Instead of trying to develop a partly parallel decoder for a given LDPC code, we can use an available partly parallel decoder to define a constrained random LDPC code. This methodology is referred as *decoder-first code design* and a partly parallel decoder architecture is presented in [21]. However, that decoder contains many random number generators which will incur much complexity for real implementations and make the entire design and implementation process very complicated. Moreover, like the conventional random LDPC codes, it is difficult to develop a systematic efficient-encoding scheme for the obtained LDPC codes.

Recently, several explicit LDPC code construction methods have been proposed [22]-[25]. These constructions are based on either finite projective planes or expander graphs and some developed explicit LDPC codes almost achieve the same performance as their randomly constructed counterparts. Exploiting the explicit structure of these LDPC codes, it is possible to develop high performance partly parallel decoders and find systematic efficient-encoding schemes. However, we note that all the construction approaches put very strict constraints on the selection of code length and code rate, which is obviously not desirable for many applications. Therefore, in this work we do not consider the decoder/encoder design issues for those explicit LDPC codes.

In practical applications, there always exist certain constraints on the decoding latency and hardware com-

plexity, both of them will increase significantly with the code length. Thus, in this work we only consider LDPC codes with short block length (less than 10,000 bits) and we believe these LDPC codes are of great interest from practical point of view. At short code lengths, the randomly constructed Tanner graph more likely contains quite a few short cycles, *e.g.*, 6 and 8, and the performance usually varies significantly over one code ensemble, especially at high signal-to-noise ratio (SNR). The authors of [26] propose an effective heuristic method to find good short LDPC codes under the intuitive assumption that the LDPC code with less short cycles very likely has better performance over one code ensemble. Inspired by the criteria for less short cycles and the decoder-first code design methodology [21], we propose a *joint* $(3, k)$-regular LDPC code and decoder/encoder design approach. The basic design methodology is: First we propose a method to explicitly construct a high-girth[2] $(2, k)$-regular LDPC code that exactly fits to a high-speed partly parallel decoder; then we extend this decoder to a partly parallel $(3, k)$-regular LDPC decoder that is configured by a set of constrained random parameters and defines a $(3, k)$-regular LDPC code ensemble, and a good code can be selected from this ensemble based on the criteria for less short cycles and computer simulations; finally, exploiting the specific structures of such LDPC codes, we develop a systematic efficient-encoding scheme to reduce the encoding complexity. Since each code in such code ensemble is actually constructed by randomly inserting certain check nodes into the deterministic high-girth $(2, k)$-regular LDPC code under the constraint specified by the decoder, it is reasonable to expect that the codes in this ensemble more likely don't contain too many short cycles and we can easily select a good code from it. As demonstrated in two design examples presented in this paper, the jointly developed $(3, k)$-regular LDPC codes can achieve nearly the same performance as their conventional random counterparts. Compared with decoder-first code design, our proposed joint design approach leads to a much more efficient decoder which does not contain any random number generators and the jointly developed $(3, k)$-regular LDPC code can be efficiently encoded. Moreover, we propose a modified joint design approach in order to further reduce the decoder hardware complexity for those high-rate $(3, k)$-regular LDPC codes applied to silicon area critical applications.

This paper is an extended version of [27]. This paper is organized as follows. In Section 2, a BP decoding algorithm suitable for hardware implementation is described and some practical decoder design issues are briefly discussed. We present the joint $(3, k)$-regular LDPC code and decoder/encoder design approach in Section 3. In Section 4, we modify the original joint design approach for high-rate $(3, k)$-regular LDPC codes. The conclusions and future work directions are drawn in Section 5.

[2]Girth is the length of a shortest cycle in a graph.

2 Decoding Algorithm

LDPC codes can be effectively decoded using BP algorithm. Since the direct implementation of BP algorithm will result in high hardware complexity due to a large number of multiplications, we may introduce some logarithmic quantities to convert these multiplications into additions, which leads to the Log-BP algorithm [1][28]: In fact, both BP and Log-BP algorithm realize the same decoding rule.

Before presenting the Log-BP decoding algorithm, we need to introduce some definitions as follows: Let $\mathbf{H}$ denote the $M \times N$ parity check matrix and $H_{i,j}$ denote the entry of $\mathbf{H}$ at the position (i, j). We define the set of bits n that participate in parity check m as $\mathcal{N}(m) = \{n : H_{m,n} = 1\}$, and the set of parity checks m in which bit n participates as $\mathcal{M}(n) = \{m : H_{m,n} = 1\}$. We denote the set $\mathcal{N}(m)$ with bit n excluded by $\mathcal{N}(m) \setminus n$, and the set $\mathcal{M}(n)$ with parity check m excluded by $\mathcal{M}(n) \setminus m$.

Algorithm 2.1 *Iterative Log-BP Decoding Algorithm*

Input: The prior probabilities $p_n^0 = P(x_n = 0)$ *and* $p_n^1 = P(x_n = 1) = 1 - p_n^0$, $n = 1, \cdots, N$;

Output: Hard decision $\hat{\mathbf{x}} = \{\hat{x}_1, \cdots, \hat{x}_N\}$;

Procedure:

1. *Initialization: For each* n, *compute the intrinsic (or channel) information* $\gamma_n = \log \frac{p_n^0}{p_n^1}$ *and for each* $(m, n) \in \{(i, j) | H_{i,j} = 1\}$, *compute*

$$\alpha_{m,n} = sign(\gamma_n) \log \Big(\frac{1 + e^{-|\gamma_n|}}{1 - e^{-|\gamma_n|}} \Big), \ where \ \ sign(\gamma_n) = \begin{cases} +1 & \gamma_n \geq 0 \\ -1 & \gamma_n < 0 \end{cases}.$$

2. *Iterative Decoding*

   - *Horizontal (or check node processing) step: For each* m, n, *compute*

$$\beta_{m,n} = \log \Big(\frac{1 + e^{-\alpha}}{1 - e^{-\alpha}} \Big) \prod_{n' \in \mathcal{N}(m) \setminus n} sign(\alpha_{m,n'}), \tag{1}$$

     where $\alpha = \sum_{n' \in \mathcal{N}(m) \setminus n} |\alpha_{m,n'}|$.

   - *Vertical (or variable node processing) step: For each* m, n, *compute*

$$\alpha_{m,n} = sign(\gamma_{m,n}) \log \Big(\frac{1 + e^{-|\gamma_{m,n}|}}{1 - e^{-|\gamma_{m,n}|}} \Big), \tag{2}$$

     where $\gamma_{m,n} = \gamma_n + \sum_{m' \in \mathcal{M}(n) \setminus m} \beta_{m',n}$. *For each* n, *update the "pseudo-posterior log-likelihood ratio (LLR)"* λ_n *as:*

$$\lambda_n = \gamma_n + \sum_{m \in \mathcal{M}(n)} \beta_{m,n}. \tag{3}$$

- *Decision step.*

  (a) Perform hard decision on $\{\lambda_1, \cdots, \lambda_N\}$ *to obtain* $\hat{\mathbf{x}} = \{\hat{x}_1, \cdots, \hat{x}_N\}$ *such that* $\hat{x}_n = 0$ *if* $\lambda_n > 0$ *and* $\hat{x}_n = 1$ *if* $\lambda \leq 0$*;*

  (b) If $\mathbf{H} \cdot \hat{\mathbf{x}} = 0$*, then algorithm terminates, else go to Horizontal step. A failure will be declared if pre-set maximum number of iterations occurs without successfully decoding.* ■

The variables $\alpha_{m,n}$ and $\beta_{m,n}$ in the above algorithm are called variable-to-check extrinsic information and check-to-variable extrinsic information, respectively. Notice that the function $f(x) = \log(\frac{1+e^{-|x|}}{1-e^{-|x|}})$ in the above algorithm can be implemented as a Look-Up Table (LUT) in hardware.

It is clear that each decoding iteration can be performed in fully parallel by mapping each check or variable node to one decoding processor as illustrated in Fig. 1: each check node is realized by *Check Node processing Unit* (CNU) to compute the check-to-variable extrinsic information $\beta_{m,n}$ according to (1), each variable node is realized by *Variable Node processing Unit* (VNU) to compute the variable-to-check extrinsic information $\alpha_{m,n}$ and pseudo-posterior LLR λ_n according to (2) and (3), respectively, and generate $\hat{x}_n$ by performing hard decision on λ_n. By also delivering the hard decision from each variable node to its neighboring check nodes, the parity check operation $\mathbf{H} \cdot \hat{\mathbf{x}}$ after each iteration can be incorporated into CNU's.

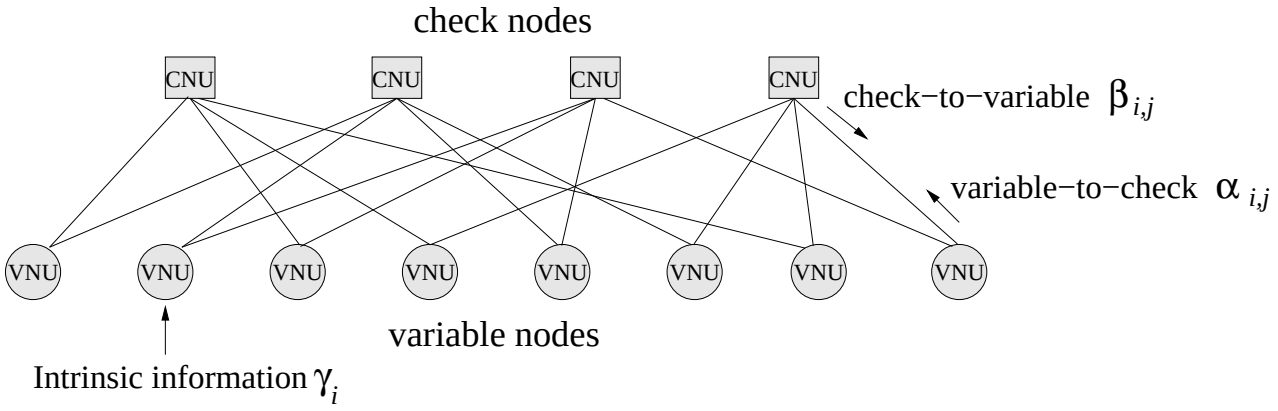

Figure 1: Tanner-graph based fully parallel decoder structure.

Since each CNU and VNU only involve several additions and accesses to LUT, a great decoding speed can be easily achieved if LDPC decoder is directly implemented based on such fully parallel architecture. However, even for the short LDPC codes (less than 10,000 bits) which are of interest, it may incur too high hardware complexity because: 1. The hardware complexity of CNU's and VNU's will be high due to the total number, *i.e.*, $O(N)$, of CNU's and VNU's, although each unit itself is not complicated; 2. Due to the large

number of soft extrinsic information and the randomness of the Tanner graph, implementation of the entire interconnection network connecting CNU's and VNU's will cause a formidable routing overhead.

Thus partly parallel decoder implementations with appropriate trade-off between complexity and speed are highly desirable for many applications. Due to the randomness of LDPC code construction, effective high-speed partly parallel decoder design approaches turn out to be not trivial. We believe that jointly conceiving code construction and decoder design should be a promising direction for high-speed partly parallel LDPC decoder implementation. In this work, we propose such a joint design approach for $(3, k)$-regular LDPC codes, which also leads to low-complexity encoding scheme. In practice, regular LDPC code with column weight of 3 is usually preferred because of its very good performance, highest code rate and lowest complexity. However, the more general joint design approach for (j, k)-regular LDPC codes or even irregular LDPC codes still remains as an open challenging problem.

3 Joint $(3, k)$-Regular LDPC Code and Decoder/Encoder Design

The basic joint design methodology can be described as follows: We first explicitly construct a high-girth $(2, k)$-regular LDPC code that exactly fits to a high-speed partly parallel $(2, k)$-regular LDPC decoder, then *extend* this decoder to a $(3, k)$-regular LDPC decoder that is configured by a set of constrained random parameters and defines a $(3, k)$-regular LDPC code ensemble. Each code in such code ensemble is essentially constructed by randomly inserting certain check nodes into the deterministic high-girth $(2, k)$-regular LDPC code under the constraint specified by the decoder. Thus it is reasonable to expect that the codes in this ensemble more likely don't contain too many short cycles and the selected code is prone to assume good performance, which will be further illuminated by two design examples. Moreover, the parity check matrix of each code in this ensemble can be systematically transformed to an approximate upper triangular matrix, based on which we may obtain an efficient-encoding scheme.

Before describing this joint design approach in more detail, we need to introduce the definition of *girth average* of a graph G [26]: Let g_u denote the shortest cycle that passes through node u in graph G, then $\sum_{u \in G} g_u / N$ is denoted as girth average of G, where $N = |G|$ is the total node number of G. Referring the partly parallel decoder architecture proposed in [21] as the *initial* decoder architecture, the joint design process is given as follows and the corresponding schematic diagram is shown in Fig. 2.

1. Explicitly construct two deterministic submatrices, $\mathbf{H}_0$ and $\mathbf{H}_1$, so that $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ defines a $(2, k)$-regular LDPC code, denoted as C_2, with the girth of 12;

2. Exploit the specific structure of $\tilde{\mathbf{H}}$ to develop a partly parallel decoder architecture for C_2;

3. Extend the $(2, k)$-regular LDPC decoder to a $(3, k)$-regular LDPC decoder that is configured by a set of constrained random parameters and defines a $(3, k)$-regular LDPC code ensemble. The parity check matrix of each code in this ensemble can be written as $\mathbf{H} = [\mathbf{H}_0^T, \mathbf{H}_1^T, \mathbf{H}_2^T]^T$.

4. The decoder randomly generates a certain number of $(3, k)$-regular LDPC codes from which a good code is selected based on girth average comparison and computer simulations;

5. Introduce a deterministic column permutation π_c to transform the parity check matrix of selected code to approximate upper triangular form, based on which we develop an efficient-encoding scheme.

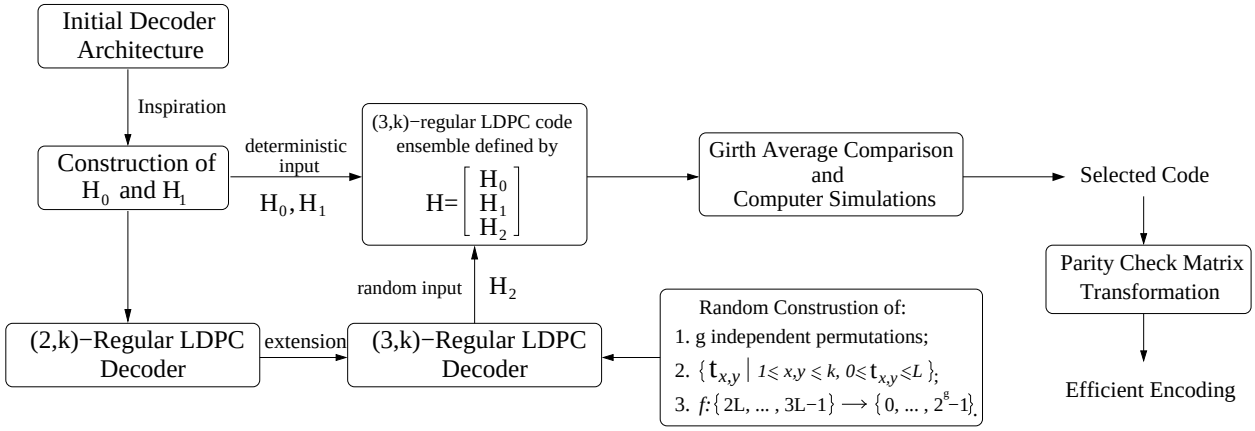

Figure 2: Code and decoder/encoder joint design flow diagram.

3.1 Construction of $\mathbf{H}_0$ and $\mathbf{H}_1$

Since $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ defines a $(2, k)$-regular LDPC code with the girth of 12, let's first review the existing methods for constructing high-girth $(2, k)$-regular LDPC codes. Tanner [4] proposed to construct a $(2, k)$-regular LDPC code with an exact value of girth (less than or equal to 32) by inserting variable nodes in the edges of a Moore graph [32] or a graph based on generalized polygon [33]. More generally, we may obtain a high-girth $(2, k)$-regular LDPC code by inserting variable nodes in the edges of any high-girth k-regular graphs, *e.g.*, Ramanujan graph [34] where the girth is larger than a certain bound. However, it is not clear yet that whether such $(2, k)$-regular LDPC codes could fit to a high-speed partly parallel decoder. Moreover, all these construction methods come with very strict constraints on the code length selection, which is not desirable in many applications. Finally, we also note that nearly all such constructions involve some complicated arithmetic operations which will likely incur high complexity in the decoder realizations.

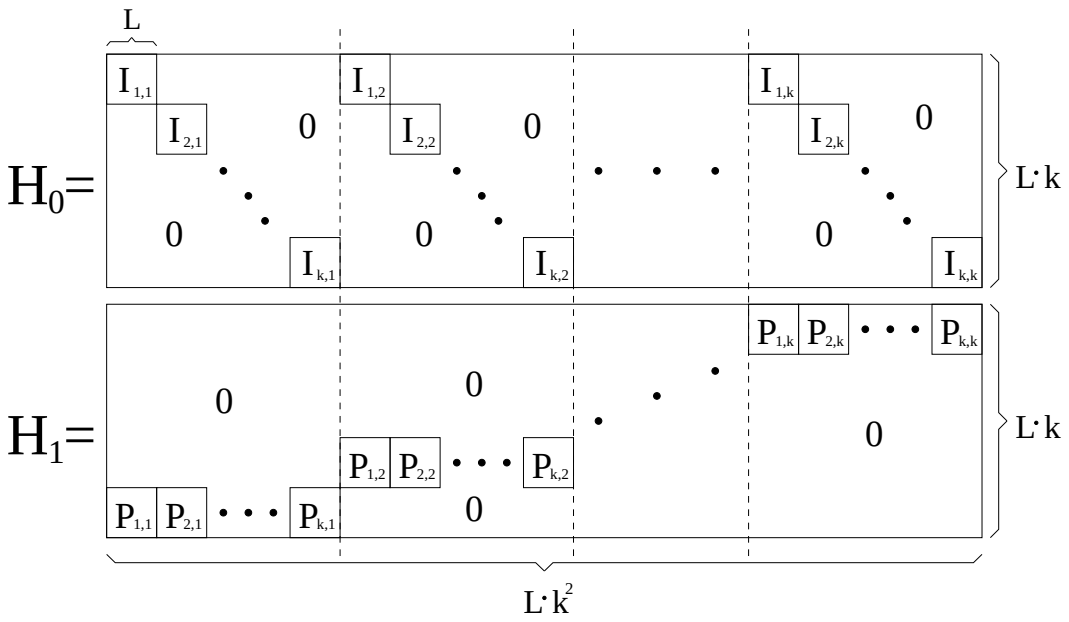

Figure 3: Structure of submatrices $\mathbf{H}_0$ and $\mathbf{H}_1$.

We propose a method to construct matrix $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ that defines a $(2, k)$-regular LDPC code with the girth of 12. Although 12 is not a very large girth value, our simulations show that it should be sufficient for generating good $(3, k)$-regular LDPC codes with our interested short code length. More important, this construction approach will lead to a decoder with extremely simple control structures and provide much more freedom on the selection of code length: Given k, any code length that could be factored as $L \cdot k^2$ is permitted, where L can not be factored as $L = a \cdot b, \forall a, b \in \{0, \cdots, k-1\}$.

Fig. 3 shows the structures of $\mathbf{H}_0$ and $\mathbf{H}_1$: Each block matrix $\mathbf{I}_{x,y}$ in $\mathbf{H}_0$ is an $L \times L$ identity matrix and each block matrix $\mathbf{P}_{x,y}$ in $\mathbf{H}_1$ is obtained by a cyclic shift of an $L \times L$ identity matrix. Let T denote the right cyclic shift operator where $T^u(\mathbf{Q})$ represents right cyclic shifting matrix $\mathbf{Q}$ by u columns, then $\mathbf{P}_{x,y} = T^u(\mathbf{I})$ where $u = ((x-1) \cdot y) \bmod L$ and $\mathbf{I}$ represents the $L \times L$ identity matrix, *e.g.*, let $L = 5$, $x = 3$ and $y = 4$, we have $u = (x-1) \cdot y \bmod L = 8 \bmod 5 = 3$, then

$$\mathbf{P}_{3,4} = T^3(\mathbf{I}) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Clearly, in both $\mathbf{H}_0$ and $\mathbf{H}_1$, each row contains k $1's$ and each column contains a single 1. Thus matrix $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ defines a $(2, k)$-regular LDPC code, denoted as C_2, with $L \cdot k^2$ variable nodes and $2L \cdot k$

check nodes. Let G denote the Tanner graph of C_2, we have the following theorem pertaining to the girth of G:

Theorem 3.1 *If L can not be factored as $L = a \cdot b$, where $a, b \in \{0, \cdots, k-1\}$, then the girth of G is 12 and there is at least one 12-cycle passing each check node.*

Proof. see Appendix A.

It is clear that the girth average of C_2 is always 12. For the comparison between C_2 and random code in terms of girth average, we randomly construct two 4-cycle free $(2, 6)$-regular LDPC code ensembles with code length of 2304 and 9216, respectively. Each ensemble contains 500 codes and has the histograms of girth average as shown in Fig. 4.

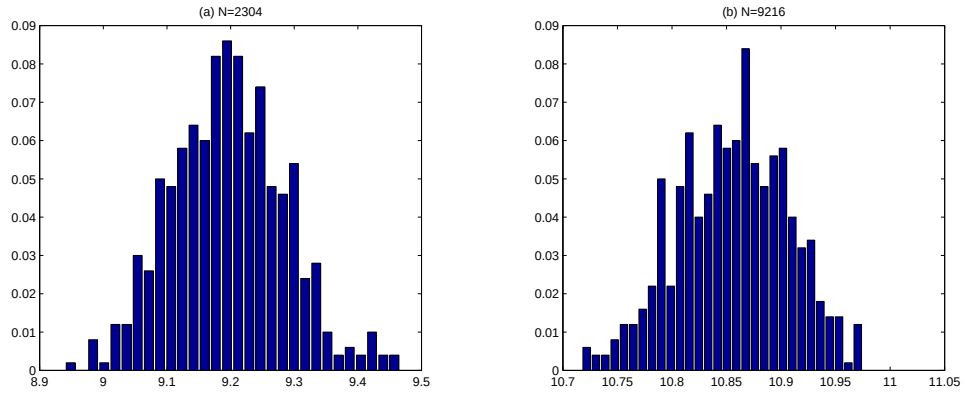

Figure 4: Histograms of girth average for $(2, 6)$-regular LDPC codes with code length (a) 2304 and (b) 9216.

3.2 $(2, k)$-Regular LDPC Decoder Architecture

In this section we present a partly parallel decoder architecture for C_2. To facilitate the following description, we arrange the $L \cdot k^2$ variable nodes of C_2 into k^2 groups, each group $\text{VG}_{x,y}$ contains the L variable nodes corresponding to the consecutive L columns in $\tilde{\mathbf{H}}$ going through the block matrix $\mathbf{I}_{x,y}$ and $\mathbf{P}_{x,y}$. Notice that any two variable nodes in the same group never connect to the same check node and all the k variable nodes connected to the same check node as specified by $\mathbf{H}_0$ ($\mathbf{H}_1$) always come from k groups with the same x-index (y-index). Based on such observations, we obtain a partly parallel decoder for C_2 as shown in Fig. 5.

This decoder contains k^2 memory banks, each one, denoted as MEM BANK-(x, y) for $1 \leq x, y \leq k$, stores all the p-bit intrinsic information (in RAM I), q-bit extrinsic information (in RAMs E_1 and E_2) and hard decisions (in RAM C) associated with the L variable nodes in $\text{VG}_{x,y}$. In this decoder, the check-to-variable extrinsic information $\beta_{m,n}$ and variable-to-check extrinsic information $\alpha_{m,n}$ associated with one pair of neighboring nodes are alternatively stored in the same memory location. The two check-to-variable or

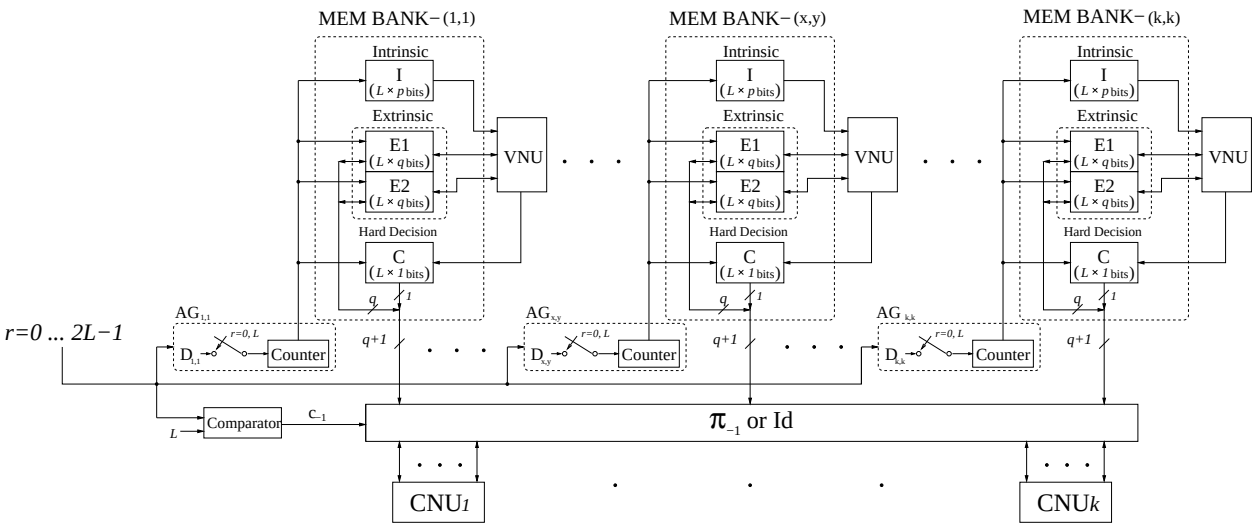

Figure 5: $(2, k)$-regular LDPC decoder architecture.

variable-to-check extrinsic information associated with each variable node are stored in the same address in E_1 and E_2. This decoder completes each decoding iteration in $2L$ clock cycles, and in each clock cycle it performs:

1. In each memory bank, if all the check-to-variable extrinsic information $\beta_{m,n}$ associated with one variable node become available after previous clock cycle, then

   (a) Retrieve 1 intrinsic information γ_n from I and 2 check-to-variable extrinsic information $\beta_{m,n}$ associated with this variable node from $E1$ and $E2$;

   (b) VNU computes 2 variable-to-check extrinsic information $\alpha_{m,n}$ and LLR λ_n, and obtains $\hat{x}_n$ by performing hard decision on λ_n;

   (c) Store the 2 variable-to-check extrinsic information $\alpha_{m,n}$ back to RAM $E1$ and $E2$ and $\hat{x}_n$ to RAM C.

2. Retrieve k^2 variable-to-check extrinsic information $\alpha_{m,n}$ and hard decision $\hat{x}_n$ (obtained in the previous iteration) from the k^2 memory banks at the addresses provided by AG$_i$'s;

3. Shuffle the k^2 variable-to-check extrinsic information and hard decision by the shuffle network configured by c_{-1} leading to the permutation π_{-1} if $c_{-1} = 1$ (π^1_{-1}), or to the identity permutation (Id=π^0_{-1}) otherwise;

4. Each CNU$_i$ computes k check-to-variable extrinsic information $\beta_{m,n}$ and performs the parity check on the k hard decision $\hat{x}_n$'s;

5. Unshuffle the k^2 check-to-variable extrinsic information $\beta_{m,n}$ and store them back into the k^2 memory banks at the original location.

It can be shown that, with the following architecture features, this decoder realize the variable-check nodes connections specified by $\mathbf{H}_0$ and $\mathbf{H}_1$ in the 1^{st} and 2^{nd} L clock cycles, respectively:

- Each Address Generator ($\mathrm{AG}_{x,y}$) provides memory address to $E1$ and $E2$ during the 1^{st} and 2^{nd} L clock cycles, respectively. In order to access all the k extrinsic information associated with the same check node in the same clock cycle, exploiting the special structures of $\mathbf{H}_0$ and $\mathbf{H}_1$, we only need to implement each $\mathrm{AG}_{x,y}$ as a modulo-L binary counter which is set to initial value $D_{x,y}$ every L clock cycles, *i.e.*, at $r = 0, L$, and

$$D_{x,y} = \begin{cases} 0, & r = 0, \\ ((x-1) \cdot y) \bmod L, & r = L. \end{cases} \tag{4}$$

- To realize the variable-check nodes connections specified by $\mathbf{H}_0$ during the 1^{st} L clock cycles ($r < L$), each k extrinsic information from k memory banks with the same x-index should be arranged into a successive sequence so that they can be sent to the same CNU. Thus we set the configuration bit $c_{-1} = 1$ and make π_{-1} permute an input data sequence $\{x_0, \cdots, x_{k^2-1}\}$ to output sequence $\{x_{\pi_{-1}(0)}, \cdots, x_{\pi_{-1}(k^2-1)}\}$, where

$$\pi_{-1}(i) = (i \bmod k) \cdot k + \lfloor \frac{i}{k} \rfloor. \tag{5}$$

- To realize the variable-check nodes connections specified by $\mathbf{H}_1$ during the 2^{nd} L clock cycles ($r \geq L$), each k extrinsic information from k memory banks with the same y-index should be arranged to be in a successive sequence. Notice that the original arrangement of the k^2 extrinsic information already has such property. Thus we just bypass the shuffle network by setting $c_{-1} = 0$.

3.3 $(3, k)$-Regular LDPC Decoder Architecture

Extending the above $(2, k)$-regular LDPC decoder in a straightforward manner, we may obtain a $(3, k)$-regular LDPC decoder as shown in Fig. 6, where the parts in shade are the added blocks. Configured by a set of constrained random parameters, this decoder defines a $(3, k)$-regular LDPC code ensemble in which each code has $L \cdot k^2$ variable nodes and $3L \cdot k$ check nodes and is essentially constructed by randomly introducing extra $L \cdot k$ check nodes into C_2.

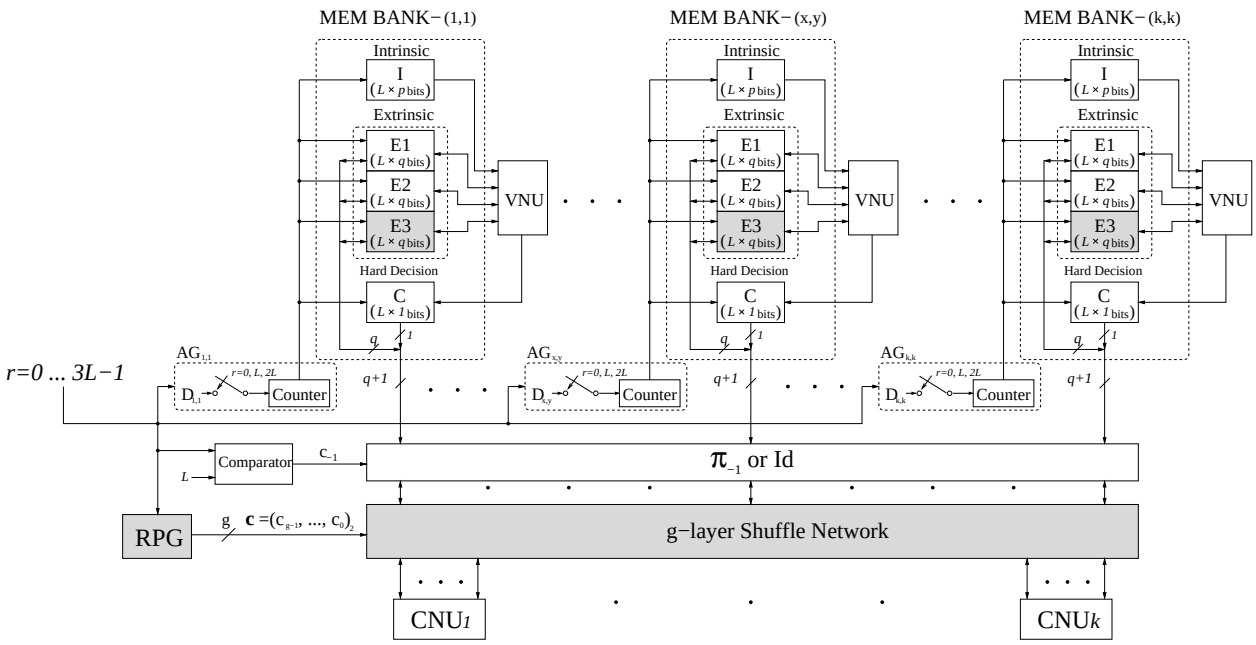

Figure 6: $(3, k)$-regular LDPC decoder architecture.

In this decoder, a Random Permutation Generator (RPG) and g-layer shuffle network are inserted between the 1-layer shuffle network (π_{-1} or Id) and all the CNU's. The realization of the g-layer shuffle network is similar with that in [21]: each layer is a shuffle network and configured by a single control bit c_i leading to a given permutation π_i if $c_i = 1$ (π_i^1), or to the identity permutation (Id=π_i^0) otherwise. Thus, configured by the g-bit word $\mathbf{c} = (c_{g-1}, \cdots, c_0)_2$ generated by RPG, the overall permutation pattern $\tilde{\pi}$ in each clock cycle is the product of g permutations: $\tilde{\pi} = \pi_{g-1}^{c_{g-1}} \circ \cdots \circ \pi_0^{c_0}$.

This decoder completes each decoding iteration in $3L$ clock cycles, and CNU's access the extrinsic information stored in $E1$, $E2$ and $E3$ in the 1^{st}, 2^{nd} and 3^{rd} L clock cycles, respectively. We denote the $2L \cdot k$ check nodes associated with the extrinsic information stored in $E1$ and $E2$ as deterministic check nodes, and the other $L \cdot k$ check nodes associated with the extrinsic information stored in $E3$ as random check nodes.

During the first $2L$ clock cycles, we bypass the g-layer shuffle network by setting the output of RPG as a zero vector, and set all other control signals in the same way as the above $(2, k)$-regular LDPC decoder. Thus this decoder realizes the interconnection pattern specified by $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ between all variable nodes and the $2L \cdot k$ deterministic check nodes during the first $2L$ clock cycles.

During the last L clock cycles, this decoder realizes the interconnection between all variable nodes and the $L \cdot k$ random check nodes, which is represented by an $L \cdot k$ by $L \cdot k^2$ matrix $\mathbf{H}_2$. This decoder has the following properties during the L clock cycles:

- RPG performs as a hash function f: $\{2L, \cdots, 3L-1\} \longrightarrow \{0, \cdots, 2^g - 1\}$ and its g-bit output vector $\mathbf{c}$ configures the g-layer shuffle network;
- The configuration bit $c_{-1} = 0$ so that the 1-layer shuffle network performs the identity permutation;
- AG$_{x,y}$ provides address to $E3$ with the counter set to $D_{x,y} = t_{x,y}$ at $r = 2L$, where $t_{x,y} \in \{0, \cdots, L-1\}$.

It is clear that in $\mathbf{H}_2$ defined by this decoder each row contains k 1's and each column contains a single 1. Thus this decoder defines a $(3, k)$-regular LDPC code ensemble, in which each code has the parity check matrix $\mathbf{H} = [\mathbf{H}_0^T, \mathbf{H}_1^T, \mathbf{H}_2^T]^T$, where $\mathbf{H}_0$ and $\mathbf{H}_1$ have the deterministic structures as shown in Fig. 3, and matrix $\mathbf{H}_2$ is jointly specified by each $t_{x,y}$, the hash function f and the g-layer shuffle network. In order to guarantee that this code ensemble only contains 4-cycle free codes and facilitate the design process, we propose to construct the hash function f and the g-layer shuffle network in fully random and generate the value of each $t_{x,y}$ in random with the following constraints:

1. Given x, we have $t_{x,y_1} \neq t_{x,y_2}, \forall y_1, y_2 \in \{1, \cdots, k\}$;
2. Given y, we have $t_{x_1,y} - t_{x_2,y} \not\equiv ((x_1 - x_2) \cdot y) \bmod L, \forall x_1, x_2 \in \{1, \cdots, k\}$.

In the following, we briefly prove that the above two constraints are sufficient for generating 4-cycle free LDPC codes. Notice that the existence of 4-cycle is equivalent to existence of two variable nodes being connected to (or sharing) the same two check nodes. For each code defined by the above decoder, two variable nodes share one check node iff their extrinsic information are delivered to the same CNU in the same clock cycle. During the 1^{st} L clock cycles, if the extrinsic information associated with two variable nodes are delivered to the same CNU in the same clock cycle, they must be stored in the same address of $E1$ in two memory banks with the same x-index. According to the first constraint, we know that two extrinsic information associated with such two variable nodes in $E3$ will never be accessed in the same clock cycle. So such two variable nodes have no chance to share one random check node no matter how we generate the hash function f and the g-layer shuffle network. Since the interconnection pattern realized in the 1^{st} L clock cycles is specified by $\mathbf{H}_0$, we may conclude that the first constraint guarantees that any two variable nodes sharing one deterministic check node specified by submatrix $\mathbf{H}_0$ never share one random check node. Similarly, the second constraint guarantees that any two variable nodes sharing one deterministic check node specified by submatrix $\mathbf{H}_1$ never share one random check node. Therefore, we know that each code defined by the above decoder does not contain two variable nodes sharing the same two check nodes, or each code is 4-cycle free.

We call the code ensemble defined by the above decoder as *implementation-oriented* $(3, k)$-regular LDPC code ensemble. For real applications, we must select a good code from this code ensemble. In this work, we propose to combine the girth average comparison and computer simulations together to effectively find a good code: first randomly generate a certain number of implementation-oriented $(3, k)$-regular LDPC codes, then pick few codes with high girth averages and finally select the one leading to the best simulation result in the time-consuming computer simulations.

3.4 Design Examples

Before presenting the efficient-encoding scheme for the implementation-oriented $(3, k)$-regular LDPC codes, we develop two such codes with different code length to illustrate the above design methodology. We select two values of L: $L_1 = 64$ and $L_2 = 128$, where the requirement that L can not be factored as $a \cdot b$, where $a, b \in \{0, \cdots, k-1 = 5\}$, is satisfied. Using the above decoder, we may obtain two code ensembles with different code length: $N_1 = L_1 \cdot k^2 = 2304$ and $N_2 = L_2 \cdot k^2 = 4608$. In both cases we set $g = 3$ so that the decoder contains a 1-layer deterministic shuffle network (π_{-1} or Id) and a 3-layer random shuffle network.

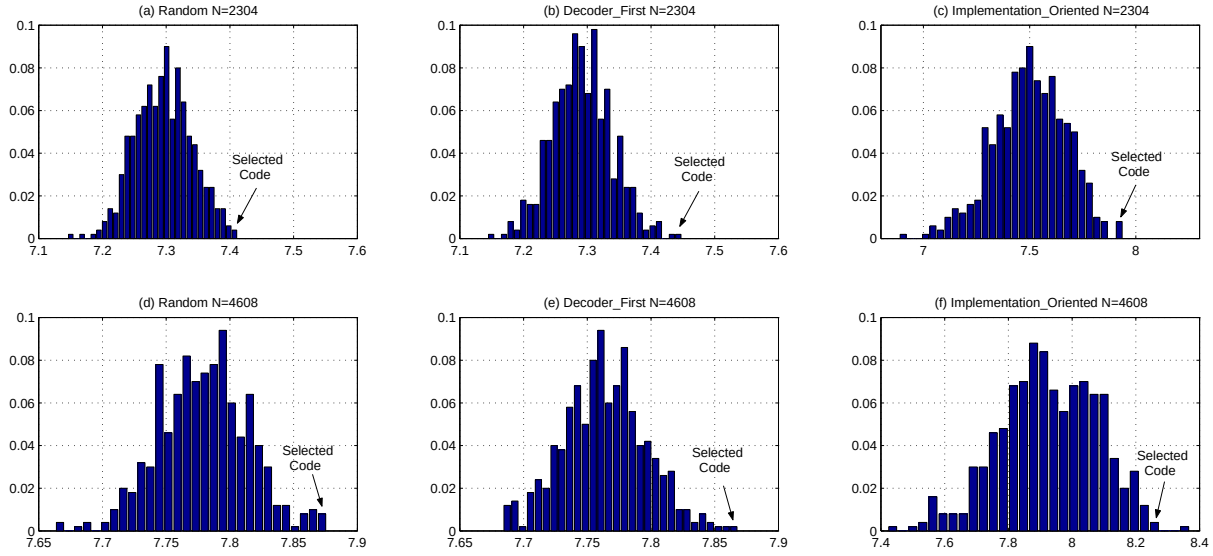

Figure 7: Histograms of girth average for (a)(d) fully random codes, (b)(e) codes developed by decoder-first code design, and (c)(f) implementation-oriented codes.

For each case, we independently generate 500 groups of hash function f, 3-layer shuffle network and all $t_{x,y}$ in random under the above two constraints on $t_{x,y}$. Then we apply these parameters to the $(3, 6)$-regular LDPC decoder and subsequently obtain two code ensembles, each one contains 500 codes. The histograms

of the girth averages of the two code ensembles are shown in Fig. 7 (c) and (f). For each case, we choose 5 codes with relatively high girth averages and select the one leading to the best performance based on the computer simulations. In the computer simulation, we assume that the LDPC codes are modulated by BPSK and transmitted over AWGN channel.

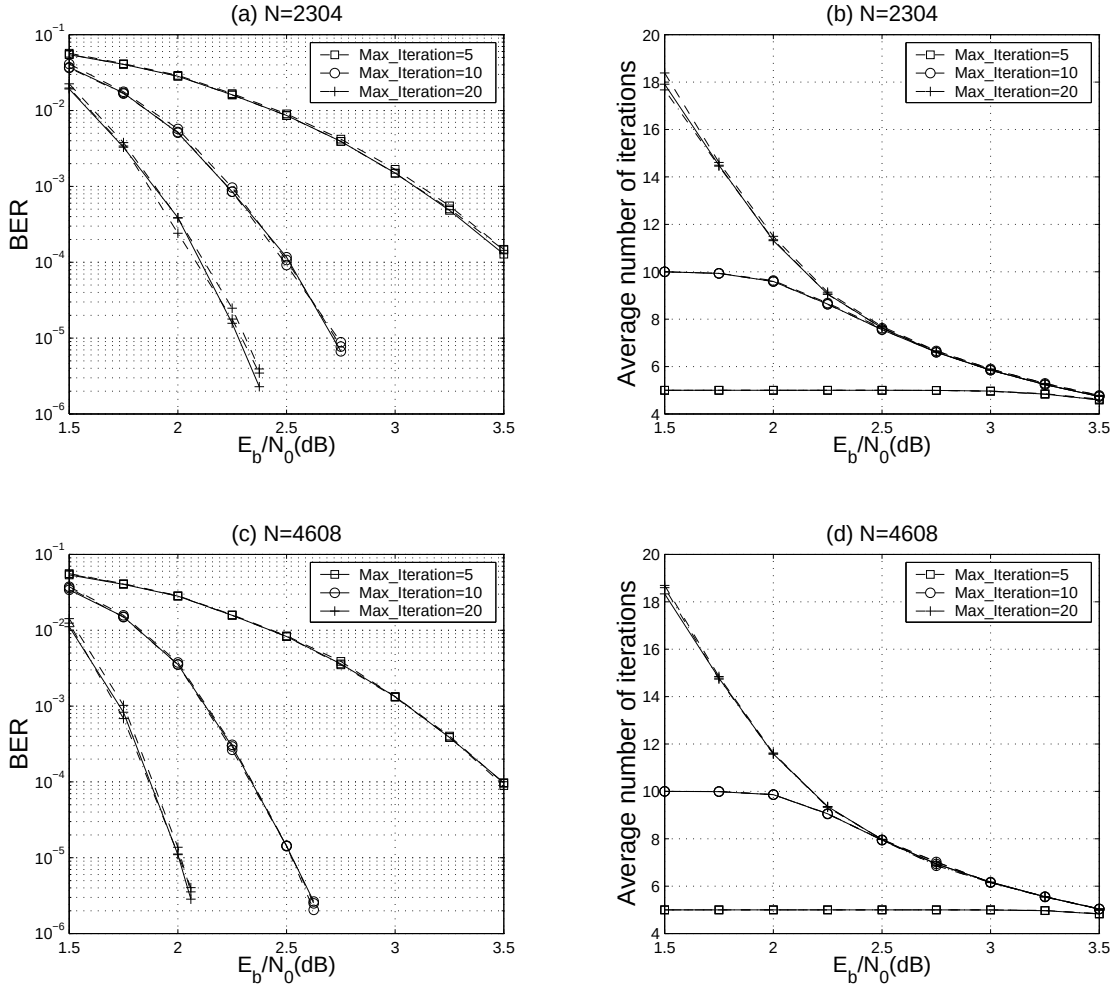

Figure 8: Finite precision simulation results where solid lines correspond to C_R^i, dash lines and dash dot lines for C_I^i and C_{DF}^i, respectively.

We denote the selected implementation-oriented $(3, 6)$-regular LDPC codes with $N_1 = 2304$ and $N_2 = 4608$ as C_I^1 and C_I^2, respectively. Since the parity check matrices of both C_I^1 and C_I^2 contain 2 redundant checks, C_I^1 and C_I^2 are $(2304, 1154)$ and $(4608, 2306)$ codes. For each code configuration, we also randomly generate two 4-cycle free $(3, 6)$-regular LDPC code ensembles, one is generated in fully random and another is generated using the decoder-first code design approach. Each code ensemble contains 500 codes and the histogram of the girth averages is shown in Fig. 7. Denote the selected fully random codes with N_1 and N_2 as

C_R^1 and C_R^2 and the selected codes developed from decoder-first code design approach with N_1 and N_2 as C_{DF}^1 and C_{DF}^2. Both C_R^1 and C_{DF}^1 are $(2304, 1152)$ codes and both C_R^2 and C_{DF}^2 are $(4608, 2304)$ codes.

The finite precision simulation results of each C_I^i, C_R^i and C_{DF}^i are shown in Fig. 8, including the BER vs. SNR and *average number of iterations* vs. SNR. In the finite precision simulations, we adopt the quantization scheme developed in [35]: received data x_n is quantized with 4 bits, extrinsic information $\alpha_{m,n}$ and $\beta_{m,n}$ are represented with 6 bits with a variable precision representation scheme proposed in [35]. Moreover, the intrinsic information γ_n is also represented with 6 bits. The function $f(x) = \log(\frac{1+e^{-|x|}}{1-e^{-|x|}})$ is realized by a 32×32 Look-Up Table (LUT). In order to guarantee the simulation accuracy, especially at high SNR, each point in the simulation results of BER vs. SNR is obtained under the condition that the frame error number at least exceeds 100.

3.5 Systematic Efficient-Encoding Scheme

For the random LDPC codes, the generator matrix is typically used to perform encoding, which has quadratic complexity in the block length. One efficient-encoding scheme, suggested in [29] and [30], is to construct random LDPC code using an approximate upper triangular parity check matrix so that the encoding complexity can be reduced significantly. In this section, we show that the above idea can be used to obtain an efficient-encoding scheme for the implementation-oriented $(3, k)$-regular LDPC codes. The developed encoding scheme is more efficient in the sense that it does not contain any less-efficient *back-substitution* operations which are extensively used in [29] and [30] for general random LDPC codes but will incur long encoding latency.

In [30] the greedy algorithms are used to construct approximate upper triangular LDPC parity check matrix. Different from that approach, based on the specific structure of implementation-oriented $(3, k)$-regular LDPC codes, we propose a systematic approach for their efficient encoding: First we obtain an approximate upper triangular matrix $\mathbf{H}^{en} = \pi_c(\mathbf{H})$ by introducing a deterministic column permutation π_c, then obtain $\hat{\mathbf{x}}$ by performing efficient-encoding based on $\mathbf{H}^{en}$, and finally get the codeword $\mathbf{x} = \pi_c^{-1}(\hat{\mathbf{x}})$.

Construction of $\mathbf{H}^{en}$: Recall that the parity check matrix of implementation-oriented $(3, k)$-regular LDPC code can be written as $\mathbf{H} = [\mathbf{H}_0^T, \mathbf{H}_1^T, \mathbf{H}_2^T]^T$, where $\mathbf{H}_0$ and $\mathbf{H}_1$ are shown in Fig. 3 and $\mathbf{H}_2$ is specified by the decoder. Denote the submatrix consisting all the columns of $\mathbf{H}$ which go through the block matrix $\mathbf{I}_{x,y}$ in $\mathbf{H}_0$ as $\mathbf{H}^{(x,y)}$, *e.g.,* $\mathbf{H}^{(1,2)}$ as shown in Fig. 9. We introduce a column permutation π_c to sequentially move each $\mathbf{H}^{(1,x)}$ forward to the position exactly next to $\mathbf{H}^{(k,1)}$ where x increases from 3 to k successively. From the construction of $\mathbf{P}_{x,y}$ in Section 3.1, we know that each $\mathbf{P}_{1,y}$ is actually an identity matrix. Therefore, the

matrix $\mathbf{H}^{en} = \pi_c(\mathbf{H})$ has the structure as shown in Fig. 9, and we can write matrix $\mathbf{H}^{en}$ in block matrix form:

$$\mathbf{H}^{en} = \begin{bmatrix} \mathbf{T} & \mathbf{B} & \mathbf{D} \\ \mathbf{A} & \mathbf{C} & \mathbf{E} \end{bmatrix}. \tag{6}$$

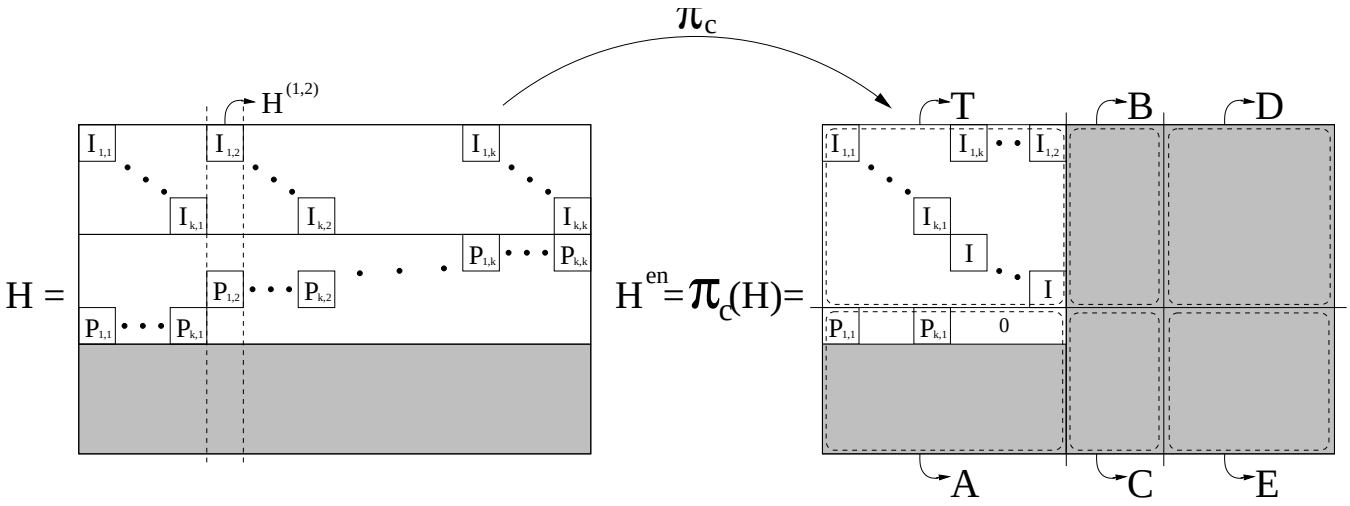

Figure 9: Structure of matrix $\mathbf{H}^{en}$.

Let $M = 3k \cdot L$ and suppose matrix $\mathbf{H}$ has r redundant checks, the left M by $(M - r)$ submatrix of $\mathbf{H}^{en}$ is

$$\begin{bmatrix} \mathbf{T} & \mathbf{B} \\ \mathbf{A} & \mathbf{C} \end{bmatrix}, \tag{7}$$

in which $\mathbf{T}$ is a $(2k - 1) \cdot L$ by $(2k - 1) \cdot L$ upper triangular submatrix.

Encoding Process: Next, we describe how the efficient-encoding is carried out based on the matrix $\mathbf{H}^{en}$. Let $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_a, \hat{\mathbf{x}}_b, \hat{\mathbf{x}}_c)$ be a *tentative* codeword decomposed according to (6), where $\hat{\mathbf{x}}_c$ is the information bits with the length of $N - M + r$, redundant bits $\hat{\mathbf{x}}_a$ and $\hat{\mathbf{x}}_b$ have the length of $(2k-1) \cdot L$ and $(k+1) \cdot L - r$, respectively.

Procedure 3.1

1. *Compute* $\mathbf{y}_c = \mathbf{D} \cdot \hat{\mathbf{x}}_c$ *and* $\mathbf{z}_c = \mathbf{E} \cdot \hat{\mathbf{x}}_c$*, which is efficient because both* $\mathbf{D}$ *and* $\mathbf{E}$ *are sparse;*

2. *Solve* $\mathbf{T} \cdot \hat{\mathbf{x}}'_a = \mathbf{y}_c$*. Since* $\mathbf{T}$ *has the form as shown in Fig. 9, we can prove that* $\mathbf{T}^{-1} = \mathbf{T}$*. Thus we have* $\hat{\mathbf{x}}'_a = \mathbf{T} \cdot y_c$*, which can be easily computed since* T *is sparse;*

3. *Evaluate* $\hat{\mathbf{s}} = \mathbf{A} \cdot \hat{\mathbf{x}}'_a + \mathbf{z}_c$*, which is also efficient since* A *is sparse;*

4. *Compute* $\hat{\mathbf{x}}_b = \mathbf{G} \cdot \hat{\mathbf{s}}$*, where* $\mathbf{G} = (\mathbf{A} \cdot \mathbf{T} \cdot \mathbf{B} + \mathbf{C})^{-1}$*. In this step, the complexity is scaled by* $((k+1) \cdot L - r)^2$.

5. *Finally we can obtain* $\hat{\mathbf{x}}_a$ *by solving* $\mathbf{T} \cdot \hat{\mathbf{x}}_a = \mathbf{B} \cdot \hat{\mathbf{x}}_b + \mathbf{y}_c$. *Since* $\mathbf{T}^{-1} = \mathbf{T}$, $\hat{\mathbf{x}}_a = \mathbf{T} \cdot (\mathbf{B} \cdot \hat{\mathbf{x}}_b + \mathbf{y}_c)$. *This is efficient since both* $\mathbf{T}$ *and* $\mathbf{B}$ *are sparse.*

We note that the above encoding process is similar to that in [30]. However, in [30], $\hat{\mathbf{x}}'_a$ and $\hat{\mathbf{x}}_a$ have to be solved using the less-efficient *back-substitution* method since $\mathbf{T}^{-1} \neq \mathbf{T}$ in general cases. Moreover, since the non-zero entries in submatrices $\mathbf{T}$, $\mathbf{B}$ and $\mathbf{D}$ are only contained in certain block matrices, the routing overhead in hardware implementation of matrix-vector multiplication for these matrices will be much lower compared with general sparse matrix-vector multiplications.

Finally, we obtain the real codeword $\mathbf{x} = \pi_c^{-1}(\hat{\mathbf{x}})$ so that $\mathbf{x}$ is a valid implementation-oriented $(3, k)$-regular LDPC codeword and the information bits on the decoder side can be easily obtained by performing the column permutation π_c on the decoder output.

4 Modified Joint Design for High-Rate Applications

In the above joint design, the partly parallel $(3, k)$-regular LDPC decoder contains k CNU's, k^2 VNU's and several shuffle networks with the routing complexity scaled by k^2. For large k, *i.e.,* the high-rate $(3, k)$-regular LDPC codes, the implementation complexity due to all the processor units and shuffle networks may still be too high for some silicon area critical applications. In this section, we propose a modified joint design scheme to further reduce the decoder hardware complexity for high-rate cases.

For high-rate $(3, k)$-regular LDPC code, provided that k is divisible by s, applying the proposed modified joint design we can obtain a decoder consisting of k/s CNU's, k^2/s VNU's and several shuffle networks with the routing complexity scaled by k^2/s. The schematic diagram of the modified joint design is shown in Fig. 10: We apply a deterministic column permutation π to $\tilde{\mathbf{H}}$ so that $\hat{\mathbf{H}} = \pi(\tilde{\mathbf{H}})$, which also defines a $(2, k)$-regular LDPC code with girth of 12, can be exploited to develop a modified partly parallel $(2, k)$-regular LDPC decoder with reduced complexity. Then following the same joint design methodology as the above, we can obtain a $(3, k)$-regular LDPC decoder and an efficient-encoding scheme. It is clear that the crucial step is how to find such a deterministic column permutation π. In the following we first describe an approach to construct π, and then briefly present the jointly developed decoder and efficient encoding scheme.

Construction of $\hat{\mathbf{H}} = \pi(\tilde{\mathbf{H}})$: Let $\tilde{\mathbf{H}}^{(x,y)}$ denote the submatrix of $\tilde{\mathbf{H}}$ consisting of all the columns going through the block matrix $\mathbf{I}_{x,y}$ in $\mathbf{H}_0$. Each $\tilde{\mathbf{H}}^{(x,y)}$ is referred as *unit submatrix* since it will be treated as an indivisible unit in the column permutation π. Let each k consecutive unit submatrices constitute an *u-block* submatrix $\tilde{\mathbf{H}}^{(n)}$: $\tilde{\mathbf{H}}^{(n)} = [\tilde{\mathbf{H}}^{(1,n)}, \cdots, \tilde{\mathbf{H}}^{(k,n)}]$.

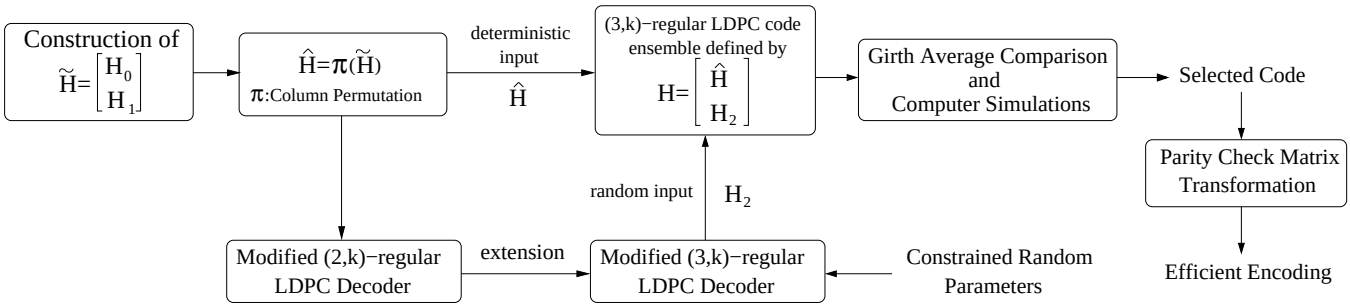

Figure 10: Modified joint design flow diagram.

Provided that k is divisible by s, we denote $t = k/s$ and write matrix $\tilde{\mathbf{H}}$ in block matrix form: $\tilde{\mathbf{H}} = [\tilde{\mathbf{H}}_1, \tilde{\mathbf{H}}_2, \cdots, \tilde{\mathbf{H}}_t]$, where each block matrix $\tilde{\mathbf{H}}_i$ consists of s consecutive u-block submatrices:

$$\tilde{\mathbf{H}}_i = [\tilde{\mathbf{H}}^{((i-1)\cdot s+1)}, \tilde{\mathbf{H}}^{((i-1)\cdot s+2)}, \cdots, \tilde{\mathbf{H}}^{(i\cdot s)}].$$

Then we apply an unit submatrix permutation θ to each submatrix $\tilde{\mathbf{H}}_i$ and obtain

$$\hat{\mathbf{H}}_i = \theta(\tilde{\mathbf{H}}_i) = [\hat{\mathbf{H}}^{((i-1)\cdot k+1)}, \hat{\mathbf{H}}^{((i-1)\cdot k+2)}, \cdots, \hat{\mathbf{H}}^{(i\cdot k)}],$$

where each u-block matrix $\hat{\mathbf{H}}^{(n)}$ consists of s unit submatrices:

$$\hat{\mathbf{H}}^{(n)} = [\tilde{\mathbf{H}}^{(x_n^1, y_n^1)}, \tilde{\mathbf{H}}^{(x_n^2, y_n^2)}, \cdots, \tilde{\mathbf{H}}^{(x_n^s, y_n^s)}]. \tag{8}$$

Let $n = (i-1)\cdot k + j$, where $1 \le i \le t$ and $1 \le j \le k$, the unit submatrix permutation θ is constructed in such a way that the index of each unit submatrix, (x_n^l, y_n^l), in (8) can be computed as

$$\begin{cases} x_n^l = ((j+l-2) \bmod k) + 1, \\ y_n^l = (((i-1)\cdot s + l - 1) \bmod k) + 1. \end{cases}$$

Applied with the same column permutation θ, each $\tilde{\mathbf{H}}_i$ is permuted to $\hat{\mathbf{H}}_i$ and the overall permutation is denoted as π:

$$\begin{aligned} \hat{\mathbf{H}} &= [\hat{\mathbf{H}}_1, \hat{\mathbf{H}}_2, \cdots, \hat{\mathbf{H}}_t] \\ &= [\theta(\tilde{\mathbf{H}}_1), \theta(\tilde{\mathbf{H}}_2), \cdots, \theta(\tilde{\mathbf{H}}_t)] = \pi(\tilde{\mathbf{H}}). \end{aligned} \tag{9}$$

Modified $(3, k)$-regular LDPC Decoder: In (9), we can write each matrix $\hat{\mathbf{H}}_i$ in the block matrix form: $\hat{\mathbf{H}}_i = [\hat{\mathbf{H}}_i^{(1)}, \cdots, \hat{\mathbf{H}}_i^{(k)}]$, each $\hat{\mathbf{H}}_i^{(j)}$ consists of consecutive $L \cdot s$ columns. We can prove that each $\hat{\mathbf{H}}_i^{(j)}$ has the structure as shown in Fig. 11.

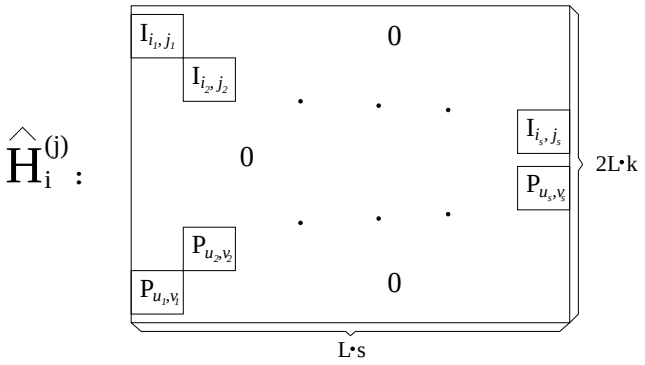

Figure 11: Structure of each $\hat{\mathbf{H}}_i^{(j)}$.

Denote the $(2, k)$-regular LDPC code defined by $\hat{\mathbf{H}}$ as $\hat{C}_2$. We arrange the $L \cdot k^2$ variable nodes of $\hat{C}_2$ into $k \cdot t$ groups, each group $\mathrm{VG}_{i,j}$ contains the $L \cdot s$ variable nodes corresponding to the $L \cdot s$ columns $\hat{\mathbf{H}}_i^{(j)}$. Notice that in $\hat{\mathbf{H}}_i^{(j)}$ each column contains a single 1 and $L \cdot s$ rows out of the $2L \cdot k$ rows contain a single 1 and all other rows are zero row vectors. Based on the above observation and referring to the original joint design, we can obtain a partly parallel decoder for $\hat{C}_2$ and extend it to a $(3, k)$-regular LDPC decoder as shown in Fig. 12, in which both the number of node processor unit and shuffle network complexity are scaled by $1/s$ compared with the decoder developed in the above original design. Similarly, configured by a set of constrained random parameters, this decoder defines a $(3, k)$-regular LDPC code ensemble, the parity check matrix of each code in this ensemble is $3L \cdot k$ by $L \cdot k^2$ and has the form $\mathbf{H} = [\hat{\mathbf{H}}^T, \mathbf{H}_2^T]^T$.

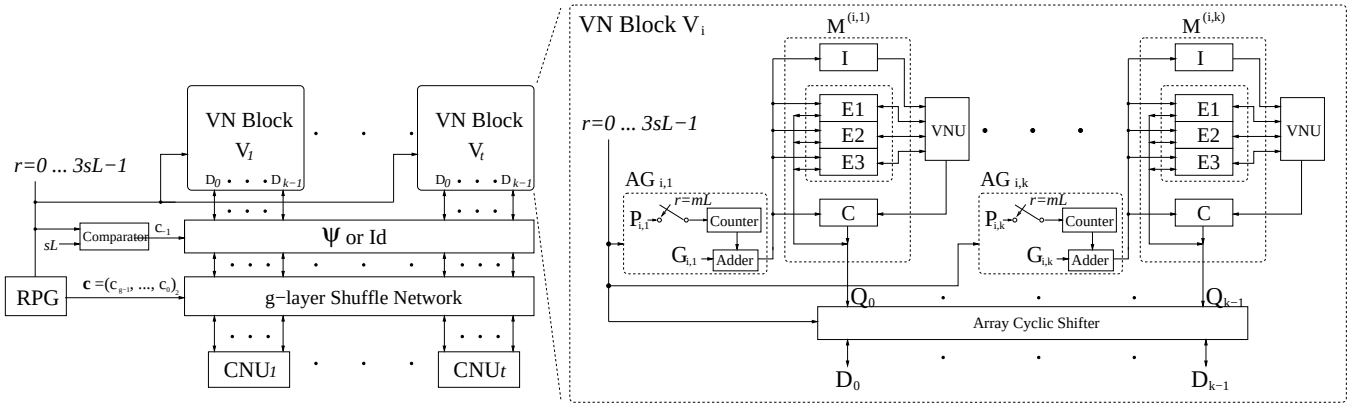

Figure 12: $(3, k)$-regular LDPC decoder architecture.

This decoder principally contains $t(= k/s)$ identical VN Blocks, a 1-layer shuffle network (ψ or Id), a g-layer shuffle network configured by a Random Permutation Generator (RPG) and t CNU's. Each VN Block V_i contains k memory banks, each one $M^{(i,j)}$ for $1 \leq j \leq k$ stores all the decoding information associated with $L \cdot s$ variable nodes in group $\mathrm{VG}_{i,j}$. This decoder completes each decoding iteration in $3s \cdot L$ clock

cycles: during the first $2s \cdot L$ clock cycles, it realizes the interconnections between all the variable nodes and $2L \cdot k$ check nodes specified by $\hat{\mathbf{H}}$; during the last $s \cdot L$ clock cycles, it defines the interconnections between all variable nodes and the other $L \cdot k$ check nodes. To achieve the above requirements and ensure that all the codes in the code ensemble are 4-cycle free, this decoder possesses the features as summarized in the following.

- As shown in Fig. 12, each $\mathrm{AG}_{i,j}$ is realized by a modulo-L binary counter followed by an adder. Each counter is preset with an initial value $P_{i,j}(m)$ every L clock cycles, *i.e.*, at $r = m \cdot L$ where $m \in \{0, \cdots, 3s-1\}$. The address is generated by adding the counter's output with a parameter $G_{i,j}(m)$. Each $G_{i,j}(m)$ is defined as

$$G_{i,j}(m) = \begin{cases} ((m+1-j) \bmod s) \cdot L, & m \leq s-1, \\ (m \bmod s) \cdot L, & s \leq m \leq 3s-1. \end{cases}$$

  Each $P_{i,j}(m)$ is generated as follows: For each m, let $d = m \bmod s$, we first introduce an *one to one mapping* $\varphi^d : (i,j) \to (x,y)$ defined as

$$\begin{cases} x = ((j+d-1) \bmod k) + 1, \\ y = (((i-1) \cdot s + d) \bmod k) + 1. \end{cases}$$

  For each $P_{i,j}(m)$, let $(x,y) = \varphi^d(i,j)$ and construct $P_{i,j}(m) = \hat{P}_{x,y}$ by

$$P_{i,j}(m) = \hat{P}_{x,y} = \begin{cases} 0, & m \leq s-1, \\ ((x-1) \cdot y) \bmod L, & s \leq m \leq 2s-1, \\ t_{x,y}, & 2s \leq m \leq 3s-1, \end{cases} \tag{10}$$

  where each $t_{x,y}$ is chosen in random with the following constraints:

  1. Given x, we have $t_{x,y_1} \neq t_{x,y_2}, \forall y_1, y_2 \in \{1, \cdots, k\}$;
  2. Given y, we have $t_{x_1,y} - t_{x_2,y} \not\equiv ((x_1 - x_2) \cdot y) \bmod L, \forall x_1, x_2 \in \{1, \cdots, k\}$.

- As shown in Fig. 12, the Array Cyclic Shifter in each VN Block, configured by $r = m \cdot L$, performs left cyclic shift on $\{Q_0, \cdots, Q_{k-1}\}$ to obtain $\{D_0, \cdots, D_{k-1}\}$ by

$$\begin{cases} D_i = Q_{(i+m+1) \bmod k}, & m \leq s-1, \\ D_i = Q_i, & s \leq m \leq 3s-1. \end{cases}$$

- During the first $2s \cdot L$ clock cycles, the output of RPG is a zero vector, and during the last $s \cdot L$ clock cycles, RPG performs as a hash function f: $\{2sL, \cdots, 3sL-1\} \longrightarrow \{0, \cdots, 2^g - 1\}$.

- The 1-layer shuffle network performs the permutation $\psi^{c_{-1}}$. ψ permutes $\{x_0, \cdots, x_{t \cdot k-1}\}$ to $\{x_{\psi(0)}, \cdots, x_{\psi(t \cdot k-1)}\}$:

$$\psi(i) = (\lfloor i/s \rfloor \bmod t) \cdot k + \lfloor i/k \rfloor \cdot s. \tag{11}$$

- The 1-bit output of comparator $c_{-1} = 1$ if $r < s \cdot L$, $c_{-1} = 0$ otherwise;

Similarly with the original joint design, we can use girth average comparison and computer simulations to select a good code from the code ensemble defined by the above decoder. Since $\hat{\mathbf{H}}$ is only a column permutation of $\tilde{\mathbf{H}}$, we can obtain an efficient-encoding scheme that is nearly same with that described in Section 3.5 except that the approximate upper triangular matrix is obtained as $\mathbf{H}^{en} = \pi_c(\pi^{-1}(\mathbf{H}))$.

Design Examples: Consider two different $\{k, L, s\}$: $k_1 = 10$, $L_1 = 23$ and $s_1 = 2$; $k_2 = 15$, $L_2 = 23$ and $s_2 = 5$. We obtain two decoders that define two code ensembles with different code length and code rate: $N_1 = L_1 \cdot k_1^2 = 2300$ and $R_1 = 1 - 3/k_1 = 0.7$; $N_2 = L_2 \cdot k_2^2 = 5175$ and $R_2 = 1 - 3/k_2 = 0.8$. In both cases, we set $g = 3$ and generate a code ensemble containing 500 codes. We denote the selected codes with $R_1 = 0.7$ and $R_2 = 0.8$ as C_I^1 and C_I^2, respectively. Moreover, we randomly generate two fully random 4-cycle free $(3, k)$-regular LDPC code ensembles with the same code length and code rate as C_I^1 and C_I^2, from which two good codes are selected, denoted as C_R^1 and C_R^2, respectively. The finite precision simulation results of each C_I^i and C_R^i are shown in Fig. 13. Again, in order to guarantee the accuracy, especially at high SNR, each point in the BER simulation results is obtained under the condition that the frame error number at least exceeds 100.

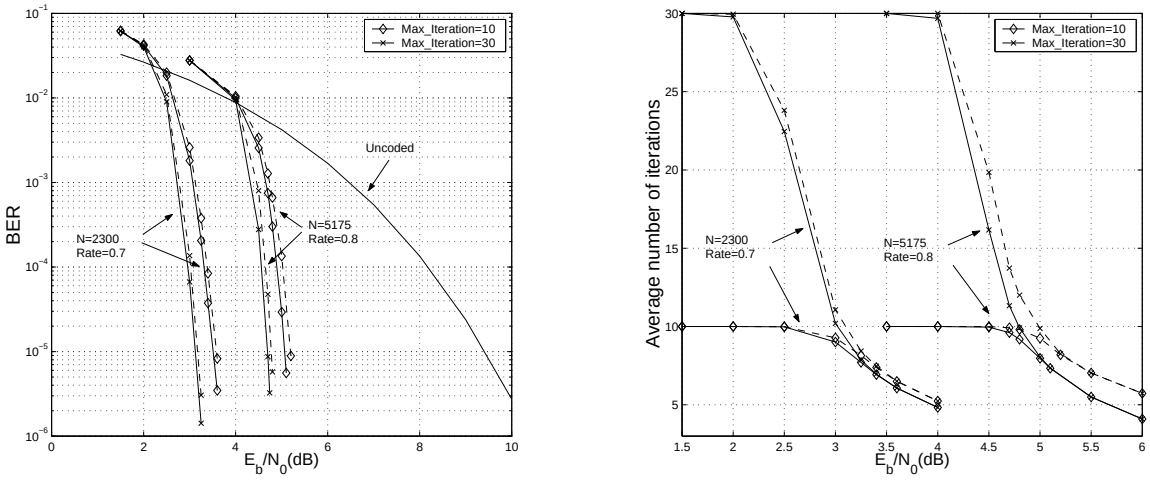

Figure 13: Finite precision simulation results where solid lines and dash lines correspond to C_R^i and C_I^i, respectively.

5 Conclusions and Future Work

A joint $(3, k)$-regular LDPC code and decoder/encoder design approach has been proposed in this paper. We first propose a novel method to explicitly construct high-girth $(2, k)$-regular LDPC code that exactly fits to a high-speed partly parallel decoder. Then by extending this decoder, we obtain a partly parallel $(3, k)$-regular LDPC decoder that is configured by a set of constrained random parameters and defines a $(3, k)$-regular LDPC code ensemble. Each code in this ensemble is essentially constructed by inserting certain check nodes into the high-girth $(2, k)$-regular LDPC code under the constraint specified by the decoder. Thus it is reasonable to expect that codes in this code ensemble more likely don't contain too many short cycles and we can easily select a good code from it, which has been illustrated by two design examples. Compared with the decoder-first code design, the proposed joint design approach eliminates the implementations of those random number generators so that the hardware complexity is much lower and the entire design process is much more simple. Moreover, we present how to effectively exploit the structure of its parity check matrix to obtain a systematic efficient-encoding scheme for such LDPC code. We also propose a modified joint design scheme to further reduce the decoder hardware complexity for high-rate codes applied to silicon area critical applications.

We believe such joint design approach should be a key for practical LDPC coding system implementations. Future research work will be directed towards investigating the joint design approaches for the more general(j, k)-regular LDPC codes and irregular LDPC codes, where the later one seems more promising from practical point of view.

APPENDIX A: Proof of Theorem 3.1

First, we arrange all the check nodes in Tanner graph G into two disjoint sets $\mathcal{P}$ and $\mathcal{Q}$, where $\mathcal{P}$ and $\mathcal{Q}$ contain the $L \cdot k$ check nodes correspond to the $L \cdot k$ rows in $\mathbf{H}_0$ and $\mathbf{H}_1$, respectively. Moreover, we label the check node in set $\mathcal{P}$ ($\mathcal{Q}$) corresponding to the r^{th} row in $\mathbf{H}_0$ ($\mathbf{H}_1$) as $p_{x,y}$ ($q_{u,v}$), where $x = \lfloor \frac{r-1}{L} \rfloor$ ($u = \lfloor \frac{r-1}{L} \rfloor$) and $y = (r-1) \bmod L$ ($v = (r-1) \bmod L$).

Because each column in $\mathbf{H}_0$ and $\mathbf{H}_1$ contains a single 1, each variable node in G has degree of 2 and always connects to one check node $p_{x,y} \in \mathcal{P}$ and one check node $q_{u,v} \in \mathcal{Q}$. Thus if we introduce a new graph $\tilde{G}$ that only consists of all the check nodes in G and two nodes are connected *iff* they connect to the same variable node in G, graph $\tilde{G}$ will be also a bipartite graph and each edge always connects $p_{x,y} \in \mathcal{P}$ with $q_{u,v} \in \mathcal{Q}$. The original graph G can be directly reconstructed from $\tilde{G}$ by inserting one variable node on each edge in $\tilde{G}$. Thus

the girth of G is two times of the girth of $\tilde{G}$ and Theorem 3.1 is equivalent to following theorem:

Theorem 5.1 *If L can not be factored as $L = a \cdot b$, where $a, b \in \{0, \cdots, k-1\}$, then the girth of $\tilde{G}$ is always 6 and there is at least one 6-cycle passing each node in $\tilde{G}$.*

Proof: Based on the fact that the length of each cycle in a bipartite graph is a multiple of 2, we prove the above theorem in the following steps:

1. First prove that whether two nodes $p_{x,y} \in \mathcal{P}$ and $q_{u,v} \in \mathcal{Q}$ are connected is solely dependent on the relationship between their indices: (x, y) and (u, v);
2. Then prove no 2- and 4-cycle exist in graph $\tilde{G}$;
3. Finally show that there is at least one 6-cycle passing each node.

From the constructions of $\mathbf{H}_0$ and $\mathbf{H}_1$ in Section 3.1 and the above computation rules for the index (x, y) of $p_{x,y}$ and index (u, v) of $q_{u,v}$, we know that the k non-zero entries of the row in $\mathbf{H}_0$ corresponding to node $p_{x,y}$ have the column indices $c_i = L \cdot k \cdot (x + i) + y + 1$ for $i = 0, \cdots, k-1$ and each c_i^{th} column goes through the block matrix $\mathbf{P}_{x+1,i+1}$ in $\mathbf{H}_1$. Node $p_{x,y}$ connects to k nodes in $\mathcal{Q}$ that correspond to k distinct rows of $\mathbf{H}_1$ containing a 1 at column index c_i for $0 \leq i \leq k-1$. Because each block matrix $\mathbf{P}_{x+1,i+1}$ is obtained through right cyclic shifting an identity matrix by $(x \cdot (i+1) \bmod L)$ columns, it can be shown that *iff*

$$r_i = i \cdot L + 1 + (y - x \cdot (i+1)) \bmod L, \tag{A-1}$$

the entry at the position (r_i, c_i) in submatrix $\mathbf{H}_1$ is 1. Moreover, the r_i^{th} row in submatrix $\mathbf{H}_1$ corresponds to node q_{u_i,v_i}, where

$$\begin{aligned} u_i &= \lfloor \frac{r_i - 1}{L} \rfloor = i, \\ v_i &= (r_i - 1) \bmod L = (y - x \cdot (i+1)) \bmod L. \end{aligned} \tag{A-2}$$

Therefore, let $p_{x,y} * q_{u,v}$ denote the node $p_{x,y}$ connects to node $q_{u,v}$, we have

$$p_{x,y} * q_{u,v} \Longleftrightarrow v = (y - x \cdot (u+1)) \bmod L. \tag{A-3}$$

This completes the step 1.

From the construction of $\mathbf{H}_0$ and $\mathbf{H}_1$ in Section 3.1, it is clear that there is no 4-cycles in graph G, or there is no 2-cycles in graph $\tilde{G}$. In the following, based on (A-3), we will prove that $\tilde{G}$ is 4-cycle free by *reduction to*

contradiction. Suppose there is a 4-cycle in $\tilde{G}$. Since $\tilde{G}$ is a bipartite graph, this 4-cycle must pass two nodes in $\mathcal{P}$, denoted as p_{x_1,y_1}, p_{x_2,y_2}, and two nodes in $\mathcal{Q}$, denoted as q_{u_1,v_1} and q_{u_2,v_2}. According to (A-3), we have

$$v_1 = (y_1 - (u_1 + 1) \cdot x_1) \bmod L, \tag{A-4}$$

$$v_2 = (y_1 - (u_2 + 1) \cdot x_1) \bmod L, \tag{A-5}$$

and

$$v_1 = (y_2 - (u_1 + 1) \cdot x_2) \bmod L, \tag{A-6}$$

$$v_2 = (y_2 - (u_2 + 1) \cdot x_2) \bmod L. \tag{A-7}$$

First we show that $x_1 \neq x_2$ and $u_1 \neq u_2$. Suppose $x_1 = x_2$, then $y_1 \neq y_2$ (otherwise p_{x_1,y_1} and p_{x_2,y_2} will be the same node), from (A-4) and (A-6), we have

$$0 = (y_1 - y_2) \bmod L. \tag{A-8}$$

However, from the condition that L can not be factored as $L = a \cdot b$ for $a, b \in \{0, \cdots, k-1\}$, we know that $L \geq k$. Since $|y_1 - y_2| < k$, we may conclude that the equation (A-8) can not hold for any $y_1 \neq y_2$. So we have $x_1 \neq x_2$. Similarly, we also have $u_1 \neq u_2$.

From (A-4), (A-5), (A-6) and (A-7) we get

$$\left.\begin{aligned} v_1 - v_2 &= (u_2 - u_1) \cdot x_1 \bmod L \\ v_1 - v_2 &= (u_2 - u_1) \cdot x_2 \bmod L \end{aligned}\right\} \Longrightarrow 0 = (x_1 - x_2) \cdot (u_2 - u_1) \bmod L. \tag{A-9}$$

Since $x_1, x_2, u_1, u_2 \in \{0, \cdots, k-1\}$, $x_1 \neq x_2$ and $u_1 \neq u_2$, both $|x_1 - x_2|$ and $|u_2 - u_1|$ will be less than k. Again, from the condition that L can not be factored as $L = a \cdot b$ for $a, b \in \{0, \cdots, k-1\}$, we have $(x_1 - x_2) \cdot (u_2 - u_1) \bmod L \neq 0$ for all $x_1, x_2, u_1, u_2 \in \{0, \cdots, k-1\}$, which is contradictory to (A-9). Thus we conclude that there is no 4-cycles in graph $\tilde{G}$. This completes the step 2.

Finally we show that there is at least one 6-cycle passing each node in graph $\tilde{G}$ in a constructive approach. Notice that $k - 3 > 0$ for any $(3, k)$ LDPC codes. $\forall\, p_{x_1,y_1} \in \mathcal{P}$, if $x_1 \leq k - 3$, we have $x_1 + 2 \leq k - 1$ and define

$$\begin{aligned} &(x_2, y_2) = (x_1 + 1, (y_1 + 1) \bmod L), \quad (x_3, y_3) = (x_1 + 2, (y_1 + 4) \bmod L), \\ &(u_1, v_1) = (0, (y_1 - x_1) \bmod L), \qquad (u_2, v_2) = (2, (y_1 - 3x_1 - 2) \bmod L), \\ &(u_3, v_3) = (1, (y_1 - 2x_1) \bmod L), \end{aligned}$$

if $x_1 > k-3$, we have $x_1 \geq 2$ and define

$$\begin{aligned}
&(x_2, y_2) = (x_1 - 1, (y_1 - 1) \bmod L), \quad (x_3, y_3) = (x_1 - 2, (y_1 - 4) \bmod L),\\
&(u_1, v_1) = (0, (y_1 - x_1) \bmod L), \qquad (u_2, v_2) = (2, (y_1 - 3x_1 + 2) \bmod L),\\
&(u_3, v_3) = (1, (y_1 - 2x_1) \bmod L).
\end{aligned}$$

Applying (A-3), we can easily verify that

$$p_{x_1,y_1} * q_{u_1,v_1} * p_{x_2,y_2} * q_{u_2,v_2} * p_{x_3,y_3} * q_{u_3,v_3} * p_{x_1,y_1}.$$

The sequence $(p_{x_1,y_1}, q_{u_1,v_1}, p_{x_2,y_2}, q_{u_2,v_2}, p_{x_3,y_3}, q_{u_3,v_3}, p_{x_1,y_1})$ forms a 6-cycle in graph $\tilde{G}$ passing node p_{x_1,y_1}.

Similarly, $\forall\ q_{u_1,v_1} \in \mathcal{Q}$, if $u_1 \leq k-3$, define

$$\begin{aligned}
&(u_2, v_2) = (u_1 + 1, v_1), \qquad (u_3, v_3) = (u_1 + 2, v_1 - 2),\\
&(x_1, y_1) = (0, v_1), \qquad (x_2, y_2) = (2, (v_1 + 2u_1 + 4) \bmod L),\\
&(x_3, y_3) = (1, (v_1 + 2u_1) \bmod L),
\end{aligned}$$

else define

$$\begin{aligned}
&(u_2, v_2) = (u_1 - 1, v_1), \qquad (u_3, v_3) = (u_1 - 2, v_1 + 2),\\
&(x_1, y_1) = (0, v_1), \qquad (x_2, y_2) = (2, (v_1 + 2u_1) \bmod L),\\
&(x_3, y_3) = (1, (v_1 + u_1 + 1) \bmod L).
\end{aligned}$$

Applying (A-3), we also can easily verify that

$$q_{u_1,v_1} * p_{x_1,y_1} * q_{u_2,v_2} * p_{x_2,y_2} * q_{u_3,v_3} * p_{x_3,y_3} * q_{u_1,v_1}.$$

The sequence $(q_{u_1,v_1}, p_{x_1,y_1}, q_{u_2,v_2}, p_{x_2,y_2}, q_{u_3,v_3}, p_{x_3,y_3}, q_{u_1,v_1})$ forms a 6-cycle in graph $\tilde{G}$ passing node q_{u_1,v_1}.

This completes the proof.

References

[1] R. G. Gallager, *Low-Density Parity-Check Codes*, M.I.T Press, 1963. available at http://justice.mit.edu/people/gallager.html.

[2] R. G. Gallager, "Low-density parity-check codes", *IRE Transactions on Information Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[3] V. Zyablov and M. Pinsker, "Estimation of the error-crrection complexity of Gallager low-density codes", *Problemy Peredachi Informatsii*, vol. 11, pp. 23–26, Jan. 1975.

[4] R. M. Tanner, "A recursive approach to low complexity codes", *IEEE Transactions on Information Theory*, vol. IT-27, pp. 533–547, Sept. 1981. available at http://www.cse.ucsc.edu/˜tanner/pubs.html.

[5] G. A. Margulis, "Explicit constructions of graphs without short cycles and low density codes", *Combinatorica*, vol. 2, pp. 71–78, 1982.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes", *in Proc. of ICC'93*, pp. 1064–1070, Geneve, Switzerland, May 1993.

[7] N. Wiberg, "Codes and decoding on general graphs", Ph.D. Dissertation, Linkoping University, Sweden, 1996. available at http://www.essrl.wustl.edu/~jao/itrg2000/.

[8] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes", *Electronics Letters*, vol. 32, pp. 1645–1646, Aug. 1996.

[9] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices", *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, Mar. 1999.

[10] M. C. Davey and D. J. C. MacKay, "Low-density parity check codes over $GF(q)$", *IEEE Commun. Letters*, vol. 2, pp. 165–167, June 1998.

[11] M. G. Luby, M. Shokrolloahi, M. A. Mizenmacher, and D. Spielman, "Improved low-density parity-check codes using irregular graphs and belief propagation", *in Proc. 1998 IEEE Intl. Symp. Inform. Theory*, p. 117, Cambridge, MA, Aug. 1998. availabel at http://http.icsi.berkeley.edu/~luby/.

[12] T. Richardson and R. Urbanke, "The capacity of low-density parity check codes under message-passing decoding", *IEEE Transactions on Information Theory*, vol. 47, pp. 599–618, Feb. 2001. available at http://lthcwww.epfl.ch/publications.html.

[13] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching low-density parity check codes", *IEEE Transactions on Information Theory*, vol. 47, pp. 619–637, Feb.

[14] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation", *IEEE Transactions on Information Theory*, vol. 47, pp. 657–670, Feb. 2001.

[15] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs", *IEEE Transactions on Information Theory*, vol. 47, pp. 585–598, Feb. 2001.

[16] S. Chung, G. D. Forney, T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the shannon limit", *IEEE Communications Letters*, vol. 5, pp. 58–60, Feb. 2001. available at http://lthcwww.epfl.ch/publications.html.

[17] J. Hou, P. H. Siegel, and L. B. Milstein, "Performance analysis and code optimization of low density parity-check codes on Rayleigh fading channels", *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 924–934, May 2001.

[18] G. Miller and D. Burshtein, "Bounds on the maximum-likelihood decoding error probability of low-density parity-check codes", *IEEE Transactions on Information Theory*, vol. 47, pp. 2696–2710, Nov. 2001.

[19] C. Howland and A. Blanksby, "Parallel decoding architectures for low density parity check codes", *in Proc. of 2001 IEEE Int. Symp. on Circuits and Systems*, Sydney, May 2001.

[20] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons, 1999.

[21] E. Boutillon, J. Castura, and F. R. Kschischang, "Decoder-first code design", *in Proceedings of the 2nd International Symposium on Turbo Codes and Related Topics*, pp. 459–462, Brest, France, Sept. 2000. available at http://lester.univ-ubs.fr:8080/~boutillon/publications.html.

[22] Y. Kou, S. Lin, and M. Fossorier, "Construction of low density parity check code: a geometric approach", *in Proceedings of the 2nd International Symposium on Turbo Codes and Related Topics*, pp. 137–140, Brest, France, Sept. 2000.

[23] T. Mittelholzer, "Construction of Steiner systems and high-rate low-density parity-check codes", *IBM Research Report*, 2000.

[24] J. Rosenthal and P. O. Vontobel, "Construction of LDPC codes using Ramanujan graphs and ideas from Margulis", *in Proc. of 38th Allerton Conference on Communication, Control and Computing*, Monticello, Illinois, Oct. 2000. available at http://www.nd.edu/˜eecoding/pub1.html.

[25] J. Lafferty and D. Rockmore, "Codes and iterative decoding on algebraic expander graphs", *in International Symposium on Information Theory and its Application*, Honolulu, Hawaii, 2000. availabel at http://www.cs.cmu.edu/˜lafferty/.

[26] Y. Mao and A. Banihashemi, "Design of good LDPC codes using girth distribution", *in IEEE International Symposium on Information Theory*, Italy, June 2000.

[27] T. Zhang and K. K. Parhi, "VLSI implementation-oriented $(3, k)$-regular low-density parity-check codes", pp. 25–36, IEEE Workshop on Signal Processing Systems (SiPS), Sept. 2001. available at http://www.ece.umn.edu/groups/ddp/turbo/.

[28] M. Chiani, A. Conti, and A. Ventura, "Evaluation of low-density parity-check codes over block fading channels", *in 2000 IEEE International Conference on Communications*, pp. 1183–1187, 2000.

[29] David J. C. MacKay, Simon T. Wilson, and Matthew C. Davey, "Comparison of constructions of irregular Gallager codes", *IEEE Transactions on Communications*, vol. 47, pp. 1449–1454, Oct. 1999.

[30] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes", *IEEE Transactions on Information Theory*, vol. 47, pp. 638–656, Feb. 2001.

[31] T. Zhang and K. K. Parhi, "A class of efficient-encoding generalized low-density parity-check codes", *in Proc. of 2001 IEEE Int. Conference on Acoustics, Speech, and Signal Processing*, Salt Lake City, Utah, May 2001. available at http://www.ece.umn.edu/groups/ddp/turbo/.

[32] P. J. Cameron and J. H. Van Lint, *Graph Theory, Coding Theory and Block Designs*, Cambridge: Cambridge Univ. Press, 1975.

[33] H. V. Maldeghem, *Generalized Polygons*, Birkhauser Verlag, 1998.

[34] A. Lubotzky, R. Philips, and P. Sarnak, "Ramanujan graphs", *Combinatorica*, vol. 8, pp. 261–277, 1988.

[35] T. Zhang, Z. Wang, and K. K. Parhi, "On finite precision implementation of low-density parity-check codes decoder", *in Proc. of 2001 IEEE Int. Symp. on Circuits and Systems*, Sydney, May 2001. available at http://www.ece.umn.edu/groups/ddp/turbo/.