VLSI Implementation-Oriented $(3, k)$-Regular Low-Density Parity-Check Codes

Tong Zhang and Keshab K. Parhi
Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis, MN, USA
E-mail:{tzhang, parhi}@ece.umn.edu *

Abstract —**In the past few years, Gallager's Low-Density Parity-Check (LDPC) codes received a lot of attention and many efforts have been devoted to analyze and improve their error-correcting performance. However, little consideration has been given to the LDPC decoder VLSI implementation. The straightforward fully parallel decoder architecture usually incurs too high complexity for many practical purposes and should be transformed to a partly parallel realization. Unfortunately, due to the randomness of LDPC codes, it's nearly impossible to develop an effective transformation for an arbitrary given LDPC code. In this paper, we propose a joint code and decoder design approach to construct a class of $(3, k)$-regular LDPC codes which exactly fit to a partly parallel decoder implementation and have a very good performance. Moreover, for such LDPC codes, we propose a systematic efficient encoding scheme by effectively exploiting the sparseness of its parity check matrix.**

1 INTRODUCTION

Gallager's Low-Density Parity-Check (LDPC) codes [1] 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 real applications in telecommunications and magnetic storage.

However, little consideration has been given to the LDPC decoder hardware realization which is required in many applications. Up to the authors' best knowledge, only two papers concerning this issue has been reported in the literatures, *i.e.*, [2][3]. The authors of [2] implemented a 1024bit, rate-1/2 prototype fully parallel LDPC decoder with the maximum throughput of 1 Gbit/s. However, because of its apparent high complexity, such a fully parallel implementation is not suitable for many practical purposes, even short code length (less than 10000 bits) is used, and an effective design approach to reduce complexity is highly desirable.

To reduce the hardware complexity, we have to effectively transform the fully parallel architecture to partly parallel ones. However, due to the randomness of LDPC

* This research was supported by the Army Research Office by grant number DA/DAAG55-98-1-0315.

Proc. of the 2001 IEEE Workshop on Signal Processing Systems (SiPS), © IEEE 2001

codes, it's nearly impossible to find an effective transformation for an arbitrary given LDPC code. To solve this problem, Boutillon *et al.* [3] propose to reverse the code design sequence: Instead of trying to obtain a partly parallel decoder from a given LDPC code, we can use an available partly parallel decoder to define a constrained random LDPC code, which leads to the *decoder-first code design* [3]. However, the decoder obtained from this design approach contains many independent constrained random number generators which will incur much complexity for real implementations and make the entire design process very complicated. Moreover, similar with the fully random LDPC codes scenario, it's difficult to develop an efficient encoding scheme for those LDPC codes obtained from the decoder-first code design.

In this work we consider the efficient partly parallel decoder architecture design for LDPC codes with short block length (less than 10000 bits) and we believe these LDPC codes are of great interest from practical point of view. It's well known that the LDPC decoding algorithm works well if the corresponding *Tanner graph* (as explained later) does not contain too many short cycles. Inspired by the criteria for less short cycles and the decoder-first code design methodology in [3], we propose a joint code and decoder design approach to develop a class of implementation-oriented $(3, k)$-regular LDPC codes which exactly fit to a partly parallel decoder architecture. Compared with decoder-first code design, our proposed joint design approach leads to a much more efficient decoder by eliminating those complicated random number generators. The performance of such implementation-oriented $(3, k)$-regular LDPC code is nearly identical to the fully random LDPC codes as shown in two design examples presented in this paper. Moreover, we propose a systematic efficient encoding scheme for the implementation-oriented $(3, k)$-regular LDPC codes.

2 LOW-DENSITY PARITY-CHECK CODES

A LDPC code is defined as the null space of a very sparse $M \times N$ parity check matrix $\mathbf{H}$, and typically is represented by a bipartite graph, sometimes called Tanner graph, between N nodes on one side called *variable* (or *message*) nodes and M nodes on another side called *check* (or *constraint*) nodes, as illustrated in Fig. 1. We say that a LDPC code is (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 codes is typically random under the constraint that the corresponding Tanner graph is 4-cycle free.

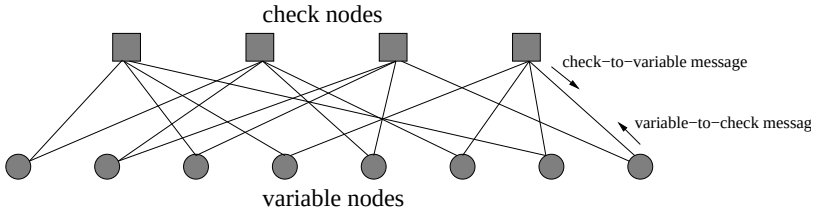

Figure 1: Tanner graph representation of a LDPC code and the decoding message flow.

LDPC codes can be effectively decoded by the iterative belief-propagation (BP) (also known as sum-product) algorithm [4]. The structure of BP decoding algorithm directly matches the Tanner graph: decoding message is computed on each variable node and check node and iteratively communicated through the edges between neighboring nodes. We can summarize BP algorithm as follows:

1. Initialize each variable node with the *intrinsic* (or *channel*) information obtained from the received bit, based on this compute the *variable-to-check message*;
2. Deliver the variable-to-check message from variable nodes to check nodes along the edges of Tanner graph;
3. Each check node generates the *check-to-variable message* based on all incoming message from connected variable nodes;
4. Deliver the check-to-variable message from check nodes to variable nodes along the edges of Tanner graph;
5. Using the incoming message and intrinsic information, each variable node updates the estimate of the corresponding bit and generates the outgoing variable-to-check message;
6. Repeat steps $2-5$ until: (a) current estimated decoded block $\hat{\mathbf{x}}$ is a valid codeword ($\mathbf{H}\cdot\hat{\mathbf{x}}=0$), or (b) a fixed number of iterations has occurred.

In the following, we denote both variable-to-check message and check-to-variable message as *extrinsic* information. In this work we only consider the soft BP decoding in which both intrinsic information and extrinsic information are soft (represented by more than 1 bit).

3 IMPLEMENTATION-ORIENTED $(3,k)$-REGULAR LDPC CODES

In this section, based on a novel high-girth $(2,k)$-regular LDPC code construction approach, we develop a partly parallel $(3,k)$-regular LDPC decoder architecture which defines an implementation-oriented $(3,k)$-regular LDPC code ensemble. As shown later, we may consider that each code in this code ensemble is constructed by letting the decoder insert certain random check nodes into the deterministic high-girth $(2,k)$-regular LDPC code. Thus it is reasonable to expect that the Tanner graph doesn't contain too many short cycles and the corresponding code may assume good performance, which will be further illustrated by two design examples. We call such a design approach as *joint code and decoder design*. Moreover, by exploiting their special structures, we propose a systematic efficient encoding scheme for the implementation-oriented $(3,k)$-regular LDPC codes.

Before presenting the joint design approach, we introduce the definition of *girth average* of a graph G [5]: Let g_u denote the length of 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. As proposed in [5], we can use girth average as an effective criterion for searching good LDPC code over one code ensemble. The joint design approach is briefly described next and the corresponding schematic diagram is shown in Fig. 2.

1. Explicitly construct the two matrices, $\mathbf{H}_0$ 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;
2. Obtain an elaborated $(3, k)$-regular LDPC decoder architecture which defines a random $(3, k)$-regular LDPC code ensemble and each code in this ensemble is a sub-code of C_2;
3. Use the decoder to randomly generate a certain number of $(3, k)$-regular LDPC codes from which we select one code with good performance by girth average comparison and computer simulations;
4. Let $\mathbf{H}$ denote the parity check matrix of the selected code. Introduce an explicit column permutation π_c to generate an approximate upper triangular matrix $\mathbf{H}^{en} = \pi_c(\mathbf{H})$ based on which we can obtain an efficient encoding scheme.

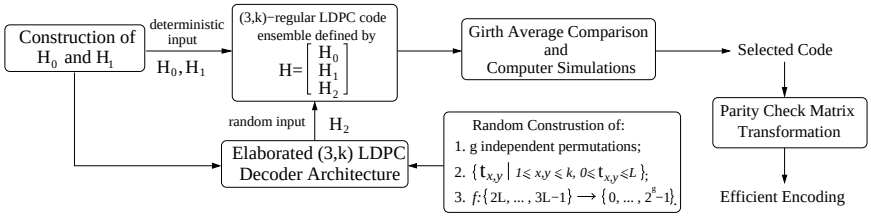

Figure 2: Joint design flow diagram.

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

In the following, we propose a novel method to construct matrix $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ which defines a $(2, k)$-regular LDPC code with girth of 12. Although 12 is not a very large girth value, our simulations show that it is sufficient for generating good implementation-oriented $(3, k)$-regular LDPC codes for short code lengths (less than 10000 bits) which are of interest. More important, such construction method will lead to a very simple decoder architecture and provide 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 $L = a \cdot b, \forall a, b \in \{0, \cdots, k-1\}$.

The structures of $\mathbf{H}_0$ and $\mathbf{H}_1$ are shown in Fig. 3. 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 $T^i(\mathbf{U})$ represents right cyclic shifting matrix $\mathbf{U}$ by i 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. For example, let $L = 5$, $x = 3$ and $y = 4$, we have $u = (x-1) \cdot y \bmod L = 8 \bmod 5 = 3$, then

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

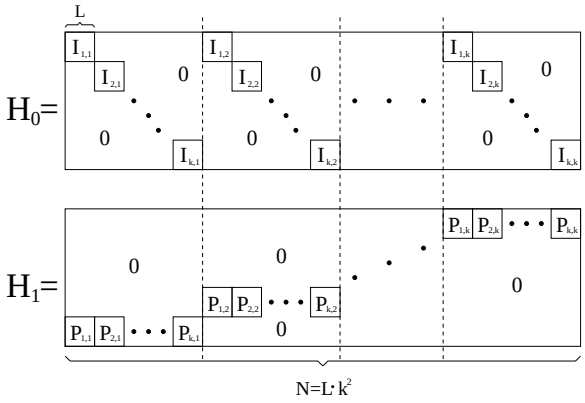

Figure 3: Structure of submatrices $\mathbf{H}_0$ and $\mathbf{H}_1$

Clearly, matrix $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ defines a $(2, k)$-regular LDPC code with $L \cdot k^2$ variable nodes and $2L \cdot k$ check nodes. Let G denote the corresponding Tanner graph, 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.*

3.2 Elaborated $(3, k)$-regular LDPC decoder architecture

Denote the $(2, k)$-regular LDPC code defined by $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ as C_2. In the following, we present an elaborated $(3, k)$-regular LDPC decoder architecture as shown in Fig. 4. It will be shown that 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 a sub-code of C_2.

This decoder contains k^2 memory banks, the i^{th} memory bank is represented as MEM BANK-(x, y), where $x = ((i-1) \bmod k) + 1$ and $y = \lfloor \frac{i-1}{k} \rfloor + 1$, and each one stores all the intrinsic information (in RAM I), extrinsic information (in two-port RAM $E1$, $E2$ and $E3$) and estimated decoded bits (in RAM C) associated with L variable nodes; a 1-layer shuffle network (π_{-1} or Id); a g-layer shuffle network; k *Check Node processor Units* (CNU's) and k^2 *Variable Node processor Units* (VNU's). One Address Generator (AG) is associated with each memory bank to provide the access address. The g-layer shuffle network consists of g 1-layer shuffle networks, each one is 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 $\mathbf{c} = (c_{g-1}, \cdots, c_0)_2$, the overall permutation pattern π is the product of g permutations: $\pi = \pi_{g-1}^{c_{g-1}} \circ \cdots \circ \pi_0^{c_0}$. The control word $\mathbf{c}$ is generated by Random Permutation Generator (RPG). Here we note that L can not be factored as $L = a \cdot b$, where $a, b \in \{0, \cdots, k-1\}$.

In this decoder, the check-to-variable message and variable-to-check message delivered along the same edge in Tanner graph are stored in the same memory location

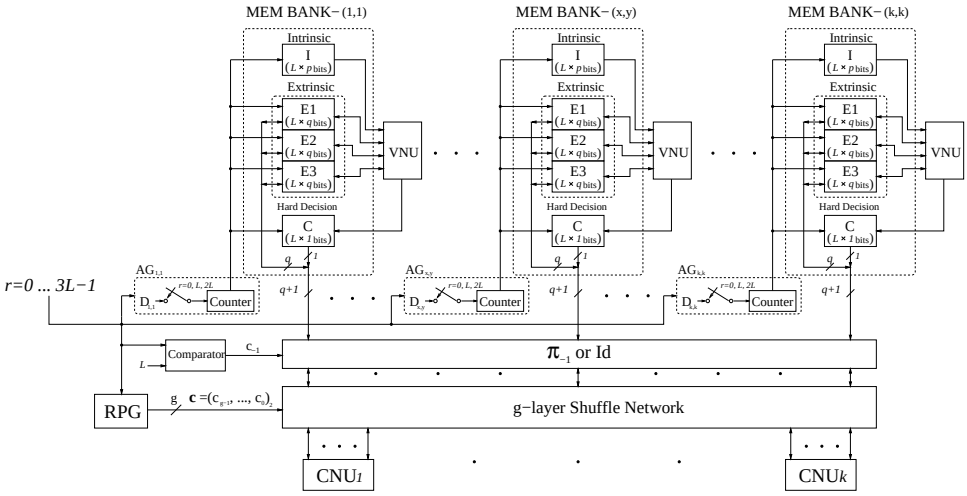

Figure 4: Elaborated $(3, k)$-regular LDPC decoder architecture.

alternatively, and 3 extrinsic information (check-to-variable message or variable-to-check message) associated with the same variable node are stored in the 3 different RAMs, $E1$, $E2$ and $E3$, respectively, with the same address. As shown in Fig. 4, intrinsic and extrinsic information are represented using p and q bits, respectively. This decoder completes each decoding iteration in $3L$ clock cycles, and in each clock cycle it performs:

1. In each memory bank, if all the check-to-variable message associated with one variable node become available after previous clock cycle, then
   (a) Retrieve 1 intrinsic information and 3 check-to-variable message associated with this variable node;
   (b) VNU computes 3 variable-to-check message and updates the corresponding estimated decoded bit;
   (c) Store the 3 variable-to-check message back to RAM $E1$, $E2$ and $E3$ and estimated decoded bit to RAM C.
2. Retrieve k^2 variable-to-check message and the corresponding estimated decoded bits from the k^2 memory banks at the addresses provided by AG_i's;
3. Shuffle the k^2 variable-to-check message and estimated decoded bits according to c_{-1} and $\mathbf{c}$ provided by the comparator and RPG, respectively;
4. Each CNU_i computes k check-to-variable message and performs the parity check on the corresponding k estimated decoded bits;
5. Unshuffle the k^2 check-to-variable message and store them back into the k^2 memory banks at the initial location.

Since $E1, E2, E3$ are two-port RAMs, we can perform the step 1 in the above decoding process in parallel with all other steps. Moreover, this decoder has the following properties:

- Each Address Generator (AG) associated with MEM BANK-(x, y), denoted as $\mathrm{AG}_{x,y}$, is realized by a simple modulo-L binary counter. Each counter is

preset with initial value $D_{x,y}$ every L clock cycles, *i.e.*, at $r = 0, L, 2L$, and

$$D_{x,y} = \begin{cases} 0, & r = 0 \\ ((x-1)\cdot y) \bmod L, & r = L \\ t_{x,y}, & r = 2L \end{cases}, \tag{1}$$

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 $t_{x_1,y} - t_{x_2,y} \not\equiv ((x_1 - x_2)\cdot y) \bmod L$, $\forall x_1, x_2 \in \{1, \cdots, k\}$.

- Provided with the address from each AG, the RAMS $E1$, $E2$ and $E3$ are accessed by the CNU array in the 1^{st}, 2^{nd} and 3^{rd} L clock cycles, respectively. Thus in one iteration, the variable-to-check message will be computed by VNU only in the last (3^{rd}) L clock cycles;
- The 1-bit output of comparator $c_{-1} = 1$ if $r < L$, $c_{-1} = 0$ otherwise ;
- The 1-layer shuffle network performs the permutation $\pi_{-1}^{c_{-1}}$. π_{-1} permutes an input data sequence $\{x_0, \cdots, x_{k^2-1}\}$ to $\{x_{\pi_{-1}(0)}, \cdots, x_{\pi_{-1}(k^2-1)}\}$, where

$$\pi_{-1}(i) = (i \bmod k)\cdot k + \lfloor \frac{i}{k} \rfloor. \tag{2}$$

- During the first $2L$ clock cycles, the output of RPG is a zero vector so that the g-layer shuffle network performs the identity permutation, and during the last L clock cycles, RPG performs as a hash function f: $\{2L, \cdots, 3L-1\} \longrightarrow \{0, \cdots, 2^g - 1\}$.

We can easily verify that the above presented decoder architecture defines a $(3, k)$-regular LDPC code ensemble in which each code has a Tanner graph with $L \cdot k^2$ variable nodes and $3k \cdot L$ check nodes and the corresponding parity check matrix can be divided into 3 submatrices: each one is $L \cdot k$ by $L \cdot k^2$ and corresponds to the interconnections among all the $L \cdot k^2$ variable and $k \cdot L$ check nodes realized by this decoder in the 1^{st}, 2^{nd} or 3^{rd} L clock cycles in each decoding iteration.

It can be proved that the first two submatrices of each code are always identical to matrix $\mathbf{H}_0$ and $\mathbf{H}_1$ presented in last section. The third submatrix, denoted as $\mathbf{H}_2$, of each code is jointly specified by all $t_{x,y}$'s, the hash function f and the g-layer shuffle network. Recall that we denote the high-girth $(2, k)$-regular LDPC code specified by $\tilde{\mathbf{H}} = [\mathbf{H}_0^T, \mathbf{H}_1^T]^T$ as C_2, we know that each code in this code ensemble is actually a sub-code of C_2. Moreover, we may consider that each code is constructed by using the decoder to introduce extra $L \cdot k$ check nodes into C_2.

We can prove that if we construct the hash function f and the g-layer shuffle network in a fully random manner and generate the value of each $t_{x,y}$ in random with the constraints as described above, the code ensemble defined by this decoder only contains 4-cycle free codes which is desirable in practice. Furthermore, from its special structure, it can be proved that the parity check matrix of each code at least contains 2 redundant checks, which just means that the actual code rate may be slightly higher than what the parity check matrix indicates, *i.e.*, $1 - 3/k$.

For real applications, we must select a good code from the implementation-oriented $(3, k)$-regular LDPC code ensemble. In this work, we propose to combine

the girth average comparison and computer simulations together to find a good code: first randomly generate a certain number of implementation-oriented $(3, k)$-regular LDPC codes, then pick several codes with high girth averages and select the one leading to the best simulation result in extensive computer simulations.

3.3 Design Examples

Before presenting the efficient encoding scheme for the implementation-oriented $(3, k)$-regular LDPC codes, to illustrate the above design methodology, we develop two implementation-oriented $(3, 6)$-regular LDPC codes with different code length.

Let $L_1 = 64$ and $L_2 = 128$. Then, using the above presented decoder architecture, 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 $g = 3$ and independently generate 500 groups of hash function f, 3-layer shuffle network and all $t_{x,y}$ in random with the above two constraints on $t_{x,y}$. Then we feed these parameters to the $(3, 6)$-regular LDPC decoder as shown in Fig. 4 and obtain two code ensembles, each one contains 500 codes. The histograms of the girth averages of these two code ensembles are shown in Fig. 5 (b) and (d). In each ensemble, we choose 5 codes with relatively high girth averages and select the one leading to the best performance based on the extensive computer simulations. In the computer simulation, we assume that the LDPC codes are modulated by BPSK and transmitted over AWGN channel.

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, respectively. Moreover, we randomly generate two fully random 4-cycle free $(3, 6)$-regular LDPC code ensembles with code length $N_1 = 2304$ and $N_2 = 4608$, respectively. Each code ensemble contains 500 codes and the histogram of the girth averages is shown in Fig. 5. In each ensemble, we also choose 5 codes with relatively high girth averages and select the one leading to the best performance. We denote the selected fully random LDPC codes with $N_1 = 2304$ and $N_2 = 4608$ as C_R^1 and C_R^2, respectively. In this work, C_R^1 and C_R^2 are $(2304, 1152)$ and $(4608, 2304)$ code, respectively.

The finite precision simulation results of each C_I^i and C_R^i are shown in Fig. 6. In the finite precision simulations, we adopt the quantization scheme developed in [6]: received data is quantized with 4 bits and all intrinsic and extrinsic information are represented with 6 bits. In order to guarantee the simulation accuracy, especially at high SNR, each point in the simulation results is obtained under the condition that the block error number at least exceeds 100. As shown in Fig. 6, the performance of the $(3, k)$-regular LDPC codes developed by these two different approaches are almost identical, but we note that it's nearly impossible to develop a partly parallel decoder for those fully random LDPC codes.

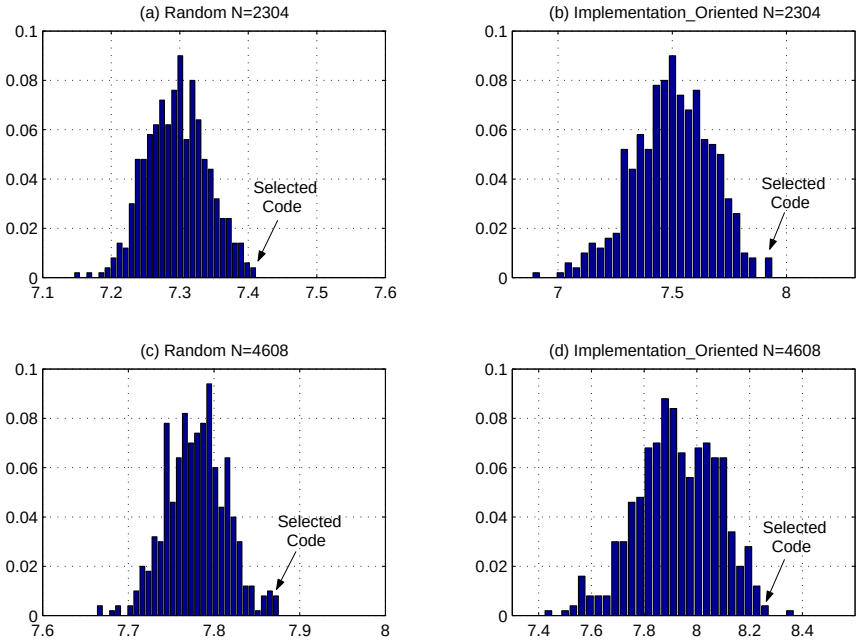

Figure 5: Histograms of girth average for (a)(c) fully random codes, and (b)(d) implementation-oriented codes.

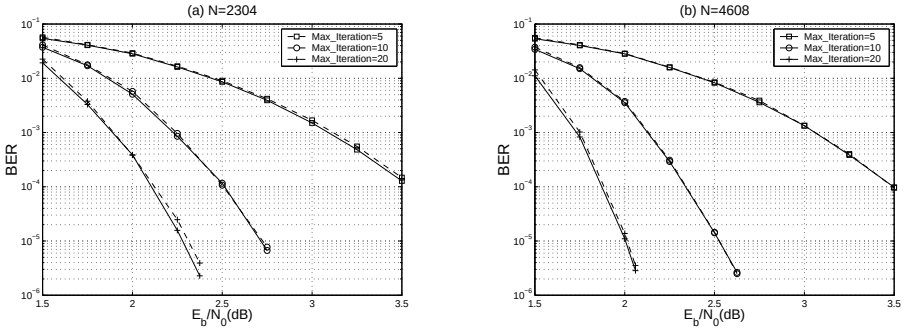

Figure 6: Finite precision simulation results where solid lines and dash lines correspond to C_R^i and C_I^i, respectively.

3.4 Efficient Encoding Scheme

The straightforward encoding scheme for LDPC codes, using the generator matrix which is usually a dense matrix, has quadratic complexity in the block length. It is suggested in [7] and [8] that using an approximate upper triangular parity check matrix to construct LDPC code can reduce the encoding complexity significantly without performance degradation. In this section, we show that the above idea can be easily applied to develop an efficient encoding scheme for our proposed implementation-oriented $(3, k)$-regular LDPC codes.

In [8] the greedy algorithms are used to construct approximate upper triangular LDPC parity check matrix for efficient encoding. Different with that approach, based on the specific structure of the parity check matrix of implementation-oriented $(3, k)$-regular LDPC codes, we propose a systematic approach for its efficient encoding. The basic idea is: First we obtain an approximate upper triangular matrix $\mathbf{H}^{en} = \pi_c(\mathbf{H})$ by introducing an explicit 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 $\mathbf{H}^{en}$: We have known that the parity check matrix of implementation-oriented $(3, k)$-regular LDPC code has the form: $\mathbf{H} = [\mathbf{H}_0^T, \mathbf{H}_1^T, \mathbf{H}_2^T]^T$, where $\mathbf{H}_0$ and $\mathbf{H}_1$ are shown in Fig. 3. We denote the submatrix consisting of 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. 7. We introduce a column permutation π_c to move each $\mathbf{H}^{(1,x)}$ forward to the position just right to $\mathbf{H}^{(k,1)}$ where x increases from 3 to k successively. From the construction of $\mathbf{P}_{x,y}$ presented in section 3.1, we know that each $\mathbf{P}_{1,y}$ is actually an identity matrix. Therefore we know that the matrix $\mathbf{H}^{en} = \pi_c(\mathbf{H})$ has the structure as shown in Fig. 7, based on which we can write matrix $\mathbf{H}^{en}$ in block matrix form as:

$$\mathbf{H}^{en} = \begin{bmatrix} \mathbf{T} & \mathbf{B} & \mathbf{D} \\ \mathbf{A} & \mathbf{C} & \mathbf{E} \end{bmatrix}. \tag{3}$$

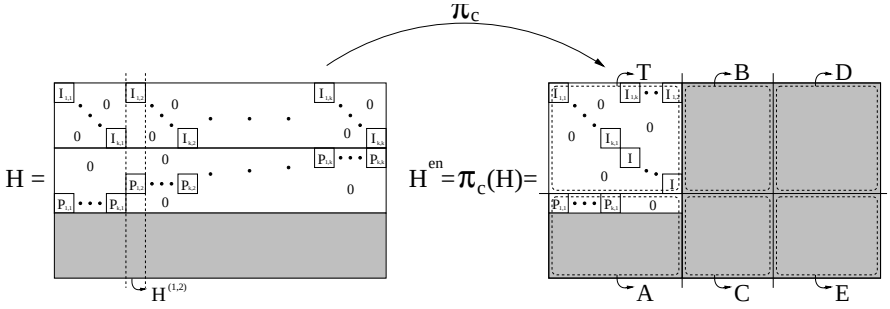

Figure 7: Structure of matrix $\mathbf{H}^{en}$.

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

$$\begin{bmatrix} \mathbf{T} & \mathbf{B} \\ \mathbf{A} & \mathbf{C} \end{bmatrix}, \tag{4}$$

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

Encoding Process: In the following, we describe how the efficient encoding is actually 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 (3), 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. 7, 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* $\mathbf{A}$ *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 with that in [8]. However, in [8], $\hat{\mathbf{x}}'_a$ and $\hat{\mathbf{x}}_a$ have to be solved using the very inefficient *back-substitution* method since $\mathbf{T}^{-1} \neq \mathbf{T}$ in general cases. Finally, we obtain the real codeword $\mathbf{x} = \pi_c^{-1}(\hat{\mathbf{x}})$. It's clear 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 CONCLUSIONS

In this paper, based on a novel method of constructing deterministic high-girth $(2,k)$-regular LDPC code, we present a partly parallel $(3,k)$-regular LDPC decoder architecture which defines an implementation-oriented $(3,k)$-regular LDPC code ensemble. Each code in this ensemble is actually constructed by inserting certain check nodes into the high-girth $(2,k)$-regular LDPC code, thus it's reasonable to expect a good performance for such codes which is illustrated by the two design examples. Compared with the decoder-first code design approach, this joint design approach eliminates the implementations of those random number generators in the decoder so that the complexity is much lower and the entire design process is more simple. Moreover, we present how to effectively exploit the sparseness of the parity check matrix to achieve an efficient encoding scheme for such LDPC codes. We believe such joint design approach should be a key for practical LDPC coding system implementations and future research work will be directed towards extending

this joint design methodology to the more general(j, k)-regular LDPC codes and irregular LDPC codes.

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] 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.

[3] 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.

[4] 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.

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

[6] 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/.

[7] D. J. C. MacKay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular gallager codes", *IEEE Transactions on Communications*, vol. 47, pp. 1449–1454, Oct. 1999.

[8] 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.