JOINT CODE-ENCODER-DECODER DESIGN FOR LDPC CODING SYSTEM VLSI IMPLEMENTATION

Hao Zhong and Tong Zhang

Electrical, Computer and Systems Engineering Department
Rensselaer Polytechnic Institute, USA

ABSTRACT

This paper presents a design approach for low-density parity-check (LDPC) coding system hardware implementation by jointly conceiving irregular LDPC code construction and VLSI implementations of encoder and decoder. The key idea is to construct good irregular LDPC codes subject to two constraints that ensure the effective LDPC encoder and decoder hardware implementations. We propose a heuristic algorithm to construct such implementation-aware irregular LDPC codes that can achieve very good error correction performance. The encoder and decoder hardware architectures are correspondingly presented.

1. INTRODUCTION

Low-density parity-check (LDPC) codes have received much attention because of their excellent error-correcting performance and highly parallelizable decoding algorithm. However, the effective VLSI implementations of the LDPC encoder and decoder remain a big challenge and a crucial issue in determining how well we can exploit the attractive merits of LDPC codes in real applications.

It has been well recognized that the conventional code to encoder/decoder design strategy (i.e., first construct a code exclusively optimized for error-correcting performance, then implement the encoder and decoder for that code) is not applicable to LDPC coding system implementations. Consequently, joint design becomes a key in most recent work [1–5]. However, two challenges still remain largely unsolved: (1) Complexity reduction and effective VLSI architecture design for LDPC encoder remain **largely unexplored**; (2) Given the desired node degree distribution, no systematic method has ever been proposed to construct the code for hardware implementation. The current practice largely relies on handcraft, e.g., the code template presented in [2].

In this paper, we propose a **joint code-encoder-decoder** design for irregular LDPC codes to tackle the above two challenges. The key is implementation-aware irregular LDPC code construction subject to two constraints that ensure effective encoder and decoder hardware implementation. A heuristic algorithm inspired by rules of thumb for constructing good LDPC code is proposed to construct the code. Encoder and decoder hardware architectures are correspondingly presented. To the best of our knowledge, this is the first complete solution for LDPC coding system implementation in the open literature.

2. BACKGROUND

In this section, we summarize some important facts and state of the art in LDPC code construction and encoder/decoder design, which directly inspired the joint design solution proposed in this paper.

LDPC Code Construction: To achieve good performance, LDPC codes should have the following properties: (a) *Large code length*: The performance improves as the code length increases, and the code length cannot be too small (at least 1K); (b) *Not too many small cycles*: Too many small cycles in the code bipartite graph will seriously degrade the error-correcting performance; (c) *Irregular node degree distribution*: It has been well demonstrated that carefully designed LDPC codes with irregular node degree distributions remarkably outperform regular ones.

LDPC Encoder: The straightforward encoding process using the generator matrix results in prohibitive VLSI implementation complexity. Richardson and Urbanke [6] demonstrated that, if the parity check matrix is approximate upper triangular, the encoding complexity can be significantly reduced. However, the encoding algorithm in [6] suffers from extensive usage of back-substitution operations that will increase the encoding latency and make effective hardware implementation problematic. The authors of [4] showed that all the back-substitution operations can be replaced by a few matrix-vector multiplications if the approximate upper triangular parity check matrix has the form as shown in Fig. 1, where $\mathbf{I}_1$ and $\mathbf{I}_2$ are identity matrices and $\mathbf{O}$ is a zero matrix.

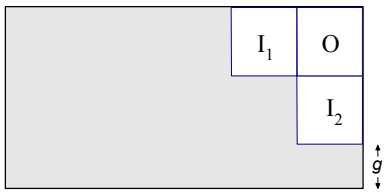

Fig. 1. The encoder-aware parity check matrix structure.

LDPC Decoder: Most recently proposed LDPC decoder design schemes share the same property: The parity check matrix is a block structured matrix that can be partitioned into an array of square block matrices, each one is either a zero matrix or a cyclic shift of an identity matrix. Such block structured parity check matrix directly leads to effective decoder hardware implementations.

3. PROPOSED JOINT DESIGN APPROACH

Motivated by the above summarized state of the art, we propose a joint code-encoder-decoder design as a complete solution for LDPC coding system implementations. In the following, we first present an implementation-aware code construction approach, then present the corresponding encoder and decoder design and hardware architectures.

3.1. Implementation-Aware Irregular Code Construction

The basic idea is to build the parity check matrix of irregular LDPC code subject to two constraints: (1) It has an approximate upper triangular form as shown in Fig. 1 with g as small as possible; (2) It is a block structured matrix. These two constraints ensure the effective encoder and decoder hardware implementations.

The design challenge is how to, under the above two constraints, construct good LDPC codes. This can be formulated as: *Given the code construction parameters, i.e., size of parity check matrix, size of each block matrix, node degree distribution*[1]*, and expected value of g, how to construct a good LDPC code?* We present an approach to tackle this design challenge as follows.

Firstly, we note that, for irregular LDPC codes, the variable nodes with high degree tend to converge more quickly than those with low degree. Therefore, with finite number of decoding iterations, not all the small cycles in the code bipartite graph are equally harmful, i.e., those small cycles passing too many low-degree variable nodes degrade the performance more seriously than the others. Thus, it is intuitive that we should prevent small cycles from passing too many low-degree variable nodes. To this end, we introduce a concept of cycle degree:

Definition 3.1 *We define the sum of degrees of all the variable nodes on a cycle as the* ***cycle degree*** *of this cycle.*

It is intuitively desirable to make the cycle degree as large as possible for those unavoidable small cycles. Motivated by such intuition, we propose an algorithm, called Heuristic Block Padding (HBP), to construct LDPC codes subject to above two structural constraints, i.e., the parity check matrix has the the structure as shown in Fig. 2. The algorithm is described as follows:

Code construction parameters: The size of each block matrix is $p \times p$, the size of parity check matrix is $(m \cdot p) \times (n \cdot p)$, and $g = \gamma \cdot p$. The row and column weight distributions are $\{w_1^{(r)}, w_2^{(r)}, \cdots, w_m^{(r)}\}$ and $\{w_1^{(c)}, w_2^{(c)}, \cdots, w_n^{(c)}\}$, where $w_i^{(r)}$ and $w_j^{(c)}$ represent the weight of i-th block rows and j-th block columns, respectively.

Output: $(m \cdot p) \times (n \cdot p)$ parity check matrix $\mathbf{H}$ with the structure as shown in Fig. 2 , in which each $p \times p$ block matrix $\mathbf{H}_{i,j}$ is either a zero matrix or a right cyclic shift of an identity matrix.

Procedure:

1. *Generate an $(m \cdot p) \times (n \cdot p)$ matrix with the structure as shown in Fig. 2, where $\mathbf{I}_1$ and $\mathbf{I}_2$ are identity matrices with roughly the same size and $\mathbf{O}$ is a zero matrix. All the blocks in the un-shaded region are initially set as NULL blocks.*
2. *According to the column weight distribution, generate a set $\{a_1, a_2, \cdots, a_n\}$, in which $a_j = w_j^{(c)}$ if $1 \le j \le n - m + \gamma$, and $a_j = w_j^{(c)} - 1$ if $n - m + \gamma + 1 \le j \le n$.*
3. *According to the row weight distribution, generate a set $\{b_1, b_2, \cdots, b_m\}$, in which $b_i = w_i^{(r)} - 1$ if $1 \le i \le m - \gamma$, and $b_i = w_i^{(r)}$ if $m - \gamma + 1 \le i \le m$.*
4. *Initialize the cycle degree constraint $d = d_{init}$.*
5. *For $j = 1$ to n, replace a_j NULL blocks on the j-th block column with a_j right cyclic shifted identity matrices:*
   (a) *Randomly pick $i \in \{1, 2, \cdots, m\}$ such that $b_i > 0$ and $\mathbf{H}_{i,j}$ is a NULL block. Replace $\mathbf{H}_{i,j}$ with a right cyclic shift of a $p \times p$ identity matrix with randomly generated shift value.*
   (b) *Let $f(\mathbf{H})$ denote the minimum cycle degree in the bipartite graph corresponding to the current matrix $\mathbf{H}$. If $f(\mathbf{H}) < d$ or the bipartite graph contains 4-cycles, reject the replacement and go back to (a). If $f(\mathbf{H})$ remains less than d after a certain number of iterations, decrease d by one before go back to (a).*
   (c) *$b_i = b_i - 1$.*
   (d) *Terminate and restart the procedure if $d < d_{min}$, where d_{min} is the minimum allowable cycle degree.*
6. *Replace all the remaining NULL blocks with zero matrices and output the matrix $\mathbf{H}$.*

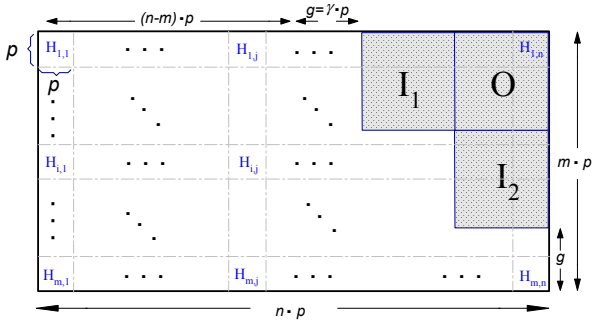

Fig. 2. The parity check matrix $\mathbf{H}$.

3.2. LDPC Encoder Design

In the following, we present an encoder design by exploiting the structural property of the code parity check matrix. We first describe an encoding process, which is similar to that presented in [6] but does not contain any back-substitution operations. Then we present the encoder hardware architecture design.

Encoding Process: According to Fig. 2, we can write the parity check matrix[2] as

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix}, \tag{1}$$

where $\mathbf{A}$ is $(m \cdot p - g) \times ((n - m) \cdot p)$, $\mathbf{B}$ is $(m \cdot p - g) \times g$, the upper triangular matrix $\mathbf{T}$ is $(m \cdot p - g) \times (m \cdot p - g)$, $\mathbf{C}$ is $g \times ((n - m) \cdot p)$, $\mathbf{D}$ is $g \times g$, and $\mathbf{E}$ is $g \times (m \cdot p - g)$. Let $[z_1, z_2, z_3]$ be a codeword decomposed according to (1), where z_1 is the information bit vector with the length of $(n - m) \cdot p$, redundant parity check bit vector z_2 and z_3 have the length of g and $m \cdot p - g$, respectively. Because of the structural property of the binary upper triangular matrix $\mathbf{T}$, we can prove $\mathbf{T}$=$\mathbf{T}^{-1}$. Fig. 3 shows the encoding flow diagram, where $\mathbf{\Phi} = -\mathbf{ETB} + \mathbf{D}$.

In the encoding process, except the step of multiply by $\mathbf{\Phi}^{-1}$, all the other steps perform multiplication between a **sparse** matrix and a vector. Although the complexity of multiply by $\mathbf{\Phi}^{-1}$ scales with g^2, the value of g can be very small compared to the matrix size. Thus the overall computational complexity of the encoding is much less than that of the encoding based on generator matrix.

[1]Notice that the node degree distribution is equivalent to parity check matrix row and column weight distribution. The good distributions can be obtained using density evolution [7].

[2]We assume that the parity check matrix is full rank, i.e., the $m \cdot p$ rows are linearly independent. In our computer simulation, all the matrices constructed using the above HBP algorithm are full rank.

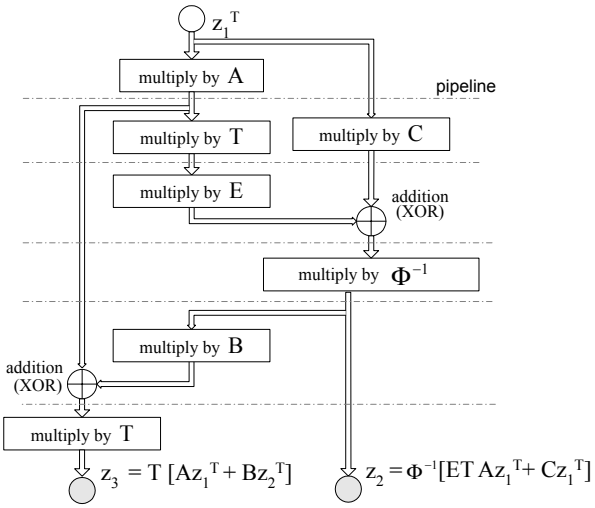

Fig. 3. Flow diagram of encoding process.

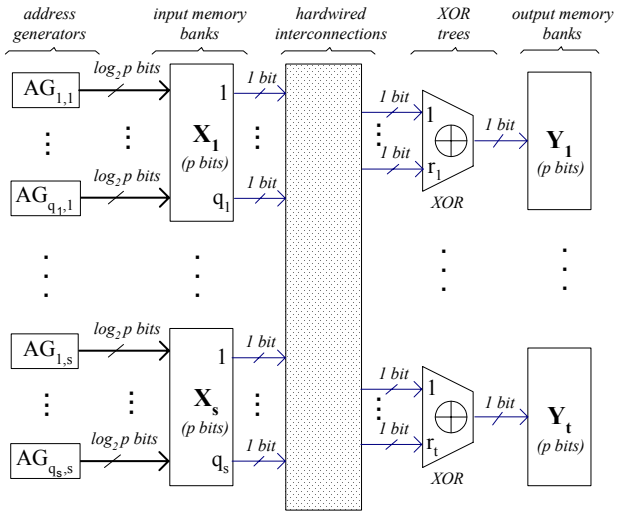

Fig. 4. Hardware design for sparse matrix-vector multiplication.

Encoder Architecture: The above encoding process mainly consists of six large sparse matrix-vector multiplications and one small dense matrix-vector multiplication. Directly mapping these large sparse matrix-vector multiplications to silicon can achieve very high speed but will suffer from significant logic gate and interconnection complexities.

Leveraging the structural property of the parity check matrix, we propose an approach to trade the speed for complexity reduction in the implementation of such large sparse matrix-vector multiplications. Since each large sparse matrix is block structured, the matrix-vector multiplications can be written as:

$$\begin{bmatrix} \mathbf{U}_{1,1} & \mathbf{U}_{1,2} & \dots & \mathbf{U}_{1,s} \\ \mathbf{U}_{2,1} & \mathbf{U}_{2,2} & \dots & \mathbf{U}_{1,s} \\ \vdots & \vdots & \dots & \vdots \\ \mathbf{U}_{t,1} & \mathbf{U}_{t,2} & \dots & \mathbf{U}_{t,s} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_s \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_t \end{bmatrix}, \quad (2)$$

where each $p \times p$ block matrix $\mathbf{U}_{i,j}$ is either a zero matrix or a right cyclic shift of an identity matrix, and each $\mathbf{x}_j$ and $\mathbf{y}_i$ are $p \times 1$ vectors. Let the column and row weight distributions of matrix $\mathbf{U}$ be $\{q_1, q_2, \cdots, q_s\}$ and $\{r_1, r_2, \cdots, r_t\}$, where q_j and r_i represent the weights of j-th block columns and i-th block rows.

To trade the speed for complexity reduction, we propose to perform such large sparse matrix-vector multiplication in a *inter-vector-parallel/intra-vector-serial* fashion: compute all the t vectors $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_t$ in parallel, but only 1 bit of each vector is computed at once. Define a set $\mathcal{P} = \{(i,j) | \forall\, \mathbf{U}_{i,j} \text{ is non-zero.}\}$. Since each non-zero $\mathbf{U}_{i,j}$ is a right cyclic shift of an identity matrix, we have $\mathbf{y}_i = \sum_{(i,j)\in\mathcal{P}} \mathbf{x}_j[\uparrow\ d_{i,j}]$, where $d_{i,j}$ is the right cyclic shift value of $\mathbf{U}_{i,j}$ and $\mathbf{x}_j[\uparrow\ d_{i,j}]$ represents cyclic shifting up the vector $\mathbf{x}_j$ by $d_{i,j}$ positions. To reduce the implementation complexity, we compute each vector $\mathbf{y}_i$ bit by bit via sharing the same computational resource, i.e., an r_i-input XOR tree.

Fig. 4 shows a hardware architecture to implement the sparse matrix-vector multiplication in such inter-vector-parallel/intra-vector-serial fashion. Each input vector $\mathbf{x}_j$ and output vector $\mathbf{y}_i$ are stored in memory X_j and Y_i, respectively. The entire matrix-vector multiplication is completed in p clock cycles, each clock cycle it computes t bits at the same position in the t vectors $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_t$. This demands that the memory banks X_j's should provide $|\mathcal{P}|$ bits at the same position in the $|\mathcal{P}|$ vectors $\{\mathbf{x}_j[\uparrow\ d_{i,j}] | \forall (i,j) \in \mathcal{P}\}$. To fulfill this requirement, each X_j provides q_j 1-bit outputs with addresses generated by q_j address generators $\mathrm{AG}_{1,j}, \cdots, \mathrm{AG}_{q_j,j}$. Each address generator $\mathrm{AG}_{k,j}$ is simply a binary counter which is initialized with a distinct value in $\{d_{i,j} | \forall (i,j) \in \mathcal{P}\}$.

As illustrated in Fig.3, the encoding is realized with 6-stage pipelining and the encoder contains six inter-vector-parallel/intra-vector-serial sparse matrix-vector multiplication blocks and one dense matrix-vector multiplication block that is directly mapped to silicon after logic minimization. To support the pipelining, we should double the size of input memory banks in each sparse matrix-vector multiplication block, i.e., two sets of input memory banks alternatively receive the output from the previous stage and provide the data for current computation.

To estimate the encoder logic gate complexity in terms of the number of 2-input NAND gates, we count each 2-input XOR gate as three 2-input NAND gates and each l-bit binary counter as $8l$ 2-input NAND gates. Assume the number of non-zero block matrices in sub-matrix $\mathbf{T}$ is $2m$ and the small dense matrix-vector multiplication can be realized using $g^2/6$ 2-input XOR gates. Let f_E denote the clock frequency of the encoder. We estimate the key metrics of this 6-stage pipelined encoder as follows:

User Data Rate	Memory (bits)	# of Gates
$(n-m) \cdot f_E$	$(2n+m) \cdot p + 3g$	$3 \cdot \lvert\mathcal{P}\rvert + g^2/2 + 8 \cdot \lceil \log_2 p \rceil \cdot \lvert\mathcal{P}\rvert$

3.3. LDPC Decoder Design

The LDPC code constructed above, whose parity check matrix has the structure as shown in Fig. 2, directly fits to a decoder architecture as illustrated in Fig. 5. It contains m check node computation units (CNUs) and n variable node computation units (VNUs), which perform all the node computation in time-division multiplexing fashion. The decoder uses n memory blocks to store the $n \cdot p$ channel input message and $|\mathcal{P}|$ memory blocks to store all the decoding message, recall that $|\mathcal{P}|$ is the total number of non-zero block matrices.

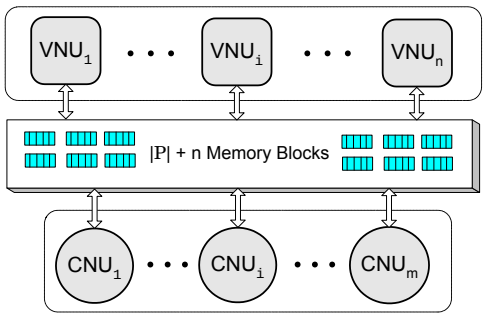

Fig. 5. Decoder architecture.

The message passing between variable and check nodes is jointly realized by memory addressing and hardwired interconnection between memory blocks and node computation units. Since each non-zero block matrix is a right cyclic shift of an identity matrix, the access address for each memory block can be simply generated by a binary counter. We note that this design strategy shares the same basic idea with the state of the art decoder design [1–3].

Given each decoding message quantized to q bits, we estimate that each CNU and VNU require $320 \cdot q$ and $250 \cdot q$ gates (in terms of 2-input NAND gate), respectively. Let f_D denote the clock frequency of the decoder and the average number of iterations is D_{avg}. We estimate the key metrics of the decoder as:

User Data Rate	Memory (bits)	# of Gates
$\frac{(n-m)\cdot f_D}{2D_{avg}}$	$(n+\|\mathcal{P}\|)\cdot p\cdot q$	$(320m+250n)\cdot q$

4. AN EXAMPLE

Applying our proposed HBP algorithm, we constructed a rate-1/2, 8K irregular LDPC code. The column weights are 2, 3, 4, and 5, and the row weights are 6 and 7. Let $m = 64$, $n = 128$, $p = 64$, and $\gamma = 3$. We have each block matrix is 64×64 and $g = \gamma \cdot p = 192$. When constructing the code using HBP algorithm, we set the minimum allowable cycle degree $d_{min} = 8$. We simulate the code error-correcting performance by assuming the code is modulated by BPSK and transmitted over AWGN channel.

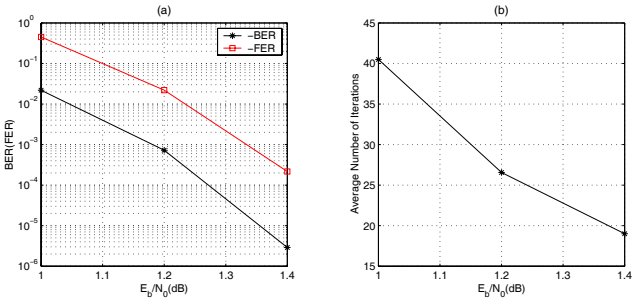

Fig. 6. Simulation results.

Fig. 6 shows the simulated bit error rate (BER), frame error rate (FER) and the average number of iterations. We note that such error-correcting performance is better or comparable to the published results in the open literature.

The parity check matrix of the constructed rate-1/2, 8K code contains 404 non-zero block matrices. Denote the clock frequencies of encoder and decoder as f_E and f_D, respectively. Suppose each decoding message is quantized to 4 bits and the average number of iterations is 20. Based on the key metrics estimation of the encoder and decoder listed in Sections 3.2 and 3.3, we have the following estimated key metrics of the coding system implementations for this rate-1/2, 8K code:

LDPC	User Data Rate	Memory (bits)	# of Gates
Encoder	$64\cdot f_E$	21K	38K
Decoder	$1.6\cdot f_D$	133K	205K

5. CONCLUSION

In this paper, we presented a joint code-encoder-decoder design approach for practical LDPC coding system hardware implementations. The basic idea is implementation-aware LDPC code design, which constructs irregular LDPC code subject to to two constraints that ensure the effective LDPC encoder and decoder hardware implementations. A heuristic algorithm has been developed to perform the code construction aiming to optimize the error correction performance. The efficient encoding process was described and a pipelined encoder hardware architecture was developed. The decoder hardware architecture is also presented. This proposed approach for the first time provides a complete systematic solution for LDPC coding system hardware implementation.

6. REFERENCES

[1] M. M. Mansour, M. M. Mansour, and N. R. Shanbhag, "A novel design methodology for high-performance programmable decoder cores for AA-LDPC codes," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Seoul, Korea, August 2003.

[2] D. E. Hocevar, "LDPC code construction with flexible hardware implementation," in *IEEE International Conference on Communications*, 2003, pp. 2708 –2712.

[3] Y. Chen and D. Hocevar, "An FPGA and ASIC implementation of rate 1/2 8088-b irregular low density parity check decoder," in *Proc. of Globecom*, 2003.

[4] T. Zhang and K. K. Parhi, "Joint $(3, k)$-regular LDPC code and decoder/encoder design," *to appear IEEE Transactions on Signal Processing*, 2003.

[5] E. Yeo, B. Nikolic, and V. Anantharam, "Architectures and implementation of low-density parity-check decoding algorithms," in *45th IEEE Midwest Symposium on Circuits and Systems*, August 2002, pp. 437–440.

[6] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[7] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.