Loosely Coupled Memory-Based Decoding Architecture for Low Density Parity Check Codes

Se-Hyeon Kang and In-Cheol Park
Department of EECS, Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
Email: shkang@ics.kaist.ac.kr, icpark@ee.kaist.ac.kr

***Abstract*— Parallel decoding is required for low density parity check (LDPC) codes to achieve high decoding throughput, but it suffers from a large set of registers and complex interconnections due to randomly located 1's in the sparse parity check matrix. This paper proposes a new LDPC decoding architecture to reduce registers and alleviate complex interconnections. To reduce the number of messages to be exchanged among processing units, two data flows that can be loosely coupled are developed by allowing duplicated operations. In addition, a partially parallel architecture is proposed to promote the memory usage and an efficient algorithm that schedules the processing order of the partially parallel architecture is also proposed to reduce the overall processing time by overlapping operations. To verify the proposed architecture, a 1024 bit rate-1/2 LDPC decoder is designed using a 0.18 um CMOS process. The decoder occupies an area of 10.08mm^2 and provides almost 1Gbps decoding throughput at the frequency of 200MHz.**

I. INTRODUCTION

Low density parity check (LDPC) codes are originally devised to exploit low decoding complexity by constructing sparse parity check matrices. Though the LDPC code does not have a maximized minimum distance due to the randomly generated sparse parity check matrix, the typical minimum distance increases linearly as the block length increases. Moreover, the error probability decreases exponentially as SNR increases when the code length is sufficiently long, whereas the decoding complexity is linearly proportional to the code length. Recent simulation results show that the LDPC code can achieve a performance that is within 0.04 dB of Shannon limit [2] and the performance of the LDPC code is close to that of the turbo code if the block length is larger than 1000 bits [3].

Despite of these advantages, when the LDPC code was first introduced, it made little impact on the information theory community because enormous storage is required in encoding and the large computational complexity in decoding. Modern VLSI technology, however, is so advanced that it enables parallel architectures exploiting the benefit of inherently parallel LDPC decoding algorithms. Blanksby et al. implemented an 1-Gb/s fully parallel decoder in which the message passing algorithm is directly mapped [4]. This architecture, however, requires a large number of complex routings between concurrent processing units (*PU*s) each of which corresponds to a node of the factor graph of the code, leading to the average net length of 3mm and the total die size of 52.5mm^2. On the other side, Yeo et al. proposed an area-efficient architecture that serializes the computations by sharing *PU*s [5]. Consequently, one iteration takes about ten-thousand cycles and wide-input multiplexers are required to select one out of several thousand intermediate values to be fed into the shared *PU*s. These two counter examples show that high throughput LDPC decoding architectures should exploit the benefit of parallel decoding algorithms while reducing the interconnection complexity.

This paper proposes a new architecture to reduce registers and alleviate complex interconnections. The proposed architecture consists of two data flows to minimize the number of messages to be exchanged, leading to an area-efficient decoder. Instead of sending all the messages, only the minimal information is exchanged between the two data flows. Then each data flow reconstructs the original messages using the minimal information. Furthermore, the intermediate values are stored into local memories each of which is accessed by only one *PU*.

The rest of this paper is organized as follows. Section II briefly introduces the low density parity check code and the decoding algorithm. Section III proposes a new LDPC decoding architecture based on loosely coupled two data flows and an efficient scheduling algorithm. The performance of the proposed LDPC decoder is summarized in Section IV. Finally, Section V addresses some concluding remarks.

II. LOW DENSITY PARITY CHECK CODES

The LDPC code, which was first introduced by Gallager in 1962 [1], is a kind of binary linear block codes. A (n, γ, ρ) LDPC code means that its block length is n and the column and row weight of its parity check matrix are γ and ρ, which represent the number of 1's in a column and a row, respectively. The column and row weight are much smaller than n to achieve a sparse matrix **H**. An LDPC code associated with fixed row and column weights is called a regular LDPC code, and an irregular LDPC code allows some variations in row and column weights.

A. Factor Graph

The LDPC code is often represented by a factor graph to make it easy to understand the message passing decoding algorithm, which is a bipartite graph that expresses how a global function of many variables is factored into a product of local functions [7].

Fig. 1 shows the factor graph of a (12, 3, 6) LDPC code, which consists of two sets of nodes: i.e. variable nodes, $\{v_j\}$, and check nodes, $\{c_i\}$. A column of the parity check matrix corresponds to a variable node represented as a circle in the left side of the factor graph and a row corresponds to a check node represented as a square. The edge between a variable node v_j and a check node c_i is constructed if there is 1 at (i, j) in the parity check matrix. Therefore each check node represents a check equation used in generating parity check bits and each variable node represents one bit in the codeword. The variable nodes that are connected to a check node are called the neighbor variable nodes of the check node. For a variable node, the neighbor check nodes can be defined similarly.

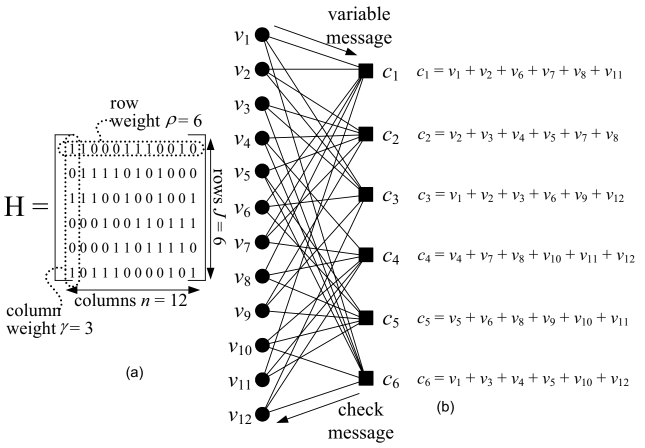


Fig. 1. a (12, 3, 6) LDPC code. (a) parity check matrix. (b) factor graph.

B. *Message Passing Algorithm*

The message passing algorithm to be described below is widely used, though it is optimal only when there is no cycle in the factor graph. In the following equations, an antipodal bit-to-symbol mapping is used; i.e., 0 → -1, 1 → 1.

1) Initialize each variable node v_j to the probability ratio of the corresponding received bit.

$$\Delta_j = \frac{p(r_j \mid x_j = +1)}{P(r_j \mid x_j = -1)} = \exp\left(\frac{2r_j}{\sigma^2}\right) \tag{1}$$

2) Variable-to-check (*VTC*) step: Each check node c_i computes the likelihood ratio, Λ_{ij}, by using the following equation and sends it to variable node v_j.

$$\Lambda_{ij} = \prod_{j' \in N(i) \setminus j} \frac{1-\Delta_{j'i}}{1+\Delta_{j'i}} \tag{2}$$

3) Check-to-variable (*CTV*) step: Each variable node v_j computes probability ratios, denoted by Δ_{ji}, by using the following equation and sends it to check node c_i.

$$\Delta_{ji} = \frac{p(r_j \mid +1)}{P(r_j \mid -1)} \prod_{i' \in M(j) \setminus i} \frac{1-\Lambda_{i'j}}{1+\Lambda_{i'j}} \tag{3}$$

4) For each variable node, calculate a tentative bit-by-bit decoding $\hat{x}_j$ using the pseudo-posteriori probability Δ_j.

$$\Delta_j = \frac{p(r_j \mid +1)}{P(r_j \mid -1)} \prod_{i \in M(j)} \frac{1-\Lambda_{ij}}{1+\Lambda_{ij}} \tag{4}$$

$$\hat{x}_j = \begin{cases} 0 & when\ \Delta_j \le 1 \\ 1 & when\ \Delta_j > 1 \end{cases} \tag{5}$$

5) Check if $\hat{\mathbf{x}} \times \mathbf{H}^T = 0$ is satisfied. If it is satisfied or the maximum number of iterations is reached, the decoding algorithm finishes. Otherwise, the algorithm repeats from step 2).

The symbols, Δ and Λ, correspond to the outgoing messages of the variable and check nodes, respectively, and $N(i)$ and $M(j)$ represent the set of neighbor nodes of check node c_i and variable node v_j, respectively. The not-notation (') means that the product is calculated over the indices excluding its own index. A detailed explanation of the algorithm can be found in [6].

To reduce the hardware cost, the product operations are transformed into summations by using log-likelihood probability, and the probability ratio is expressed in terms of tanh and $\tanh^{-1}$ functions by applying

$$\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}, \quad \tanh^{-1}(x) = -\frac{1}{2}\log\left(\frac{1+x}{1-x}\right) \tag{6}$$

Therefore equations (2) and (3) are transformed into the following equations (7) and (8). In these equations, Δ' and Λ' represents log probability ratio; i.e. $\Delta' = \log \Delta$ and $\Lambda' = \log \Lambda$. For the sake of clear representation, we redefine and use Δ and Λ as log probability ratios hereafter. The *a posteriori* probability expressed in log domain is robust and can be represented with a small number of bits while maintaining coding gain, leading to an area-efficient implementation. Detailed explanation about the log-likelihood ratio can be found in [9].

$$\Lambda'_{ij} = \sum_{j' \in N(i) \setminus j} \log\left(\tanh\left(-\frac{\Delta'_{j'i}}{2}\right)\right) \tag{7}$$

$$\Delta'_{ji} = \log\left(\frac{p(r_j \mid +1)}{p(r_j \mid -1)}\right) - \sum_{i' \in M(j) \setminus i} 2 \cdot \tanh^{-1}(\exp(\Lambda'_{i'j})) \tag{8}$$

III. Proposed LDPC Decoding Architecture

Since the functions of variable nodes and check nodes are not complex compared to other elaborate codes such as turbo codes, the main issue in implementation is how to exchange the large number of messages with other nodes. As opposed to the previous implementations that solve the problem by providing complex wire interconnections, this paper proposes a new architecture based on loosely coupled data flows to reduce the interconnection complexity.

A. *Previous Architecture*

Although the fully parallel implementation of the message-passing algorithm is straightforward and results in a high throughput decoder, it faces with complex interconnections caused by a quite large number of irregular edges. Complex interconnections are required to sum up the Λ_{ij} and Δ_{ji} messages that are calculated in the *PU*s spread over the chip area. The complexity of the interconnection can be easily inferred from the number of edges in Fig. 1 by expanding the factor graph for more than one thousand nodes.

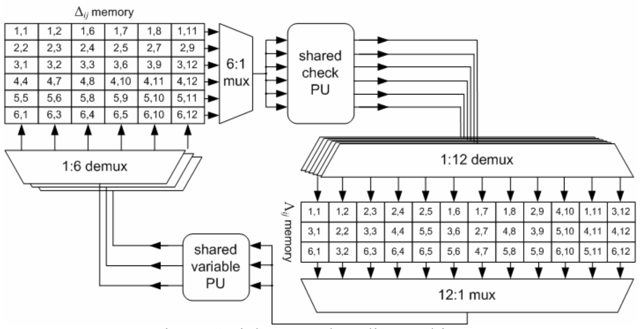


Fig. 2. Serial LDPC decoding architecture.

The interconnection complexity can be reduced by employing the serial architecture in which a shared *PU* computes all the rows or columns one after another as shown in Fig. 2. Whereas the fully parallel architecture computes all the messages simultaneously, the serial architecture computes messages row-by-row or column-by-column because there is only one *PU* for each step. In this architecture, the Λ_{ij} or Δ_{ji} messages are stored in a memory, but aligned in different ways. The read operation is relatively simple because a *PU* can read a corresponding row or column through multiplexers. The write operation, however, is not so simple because computed messages are written to the other memory aligned in a different manner. Many multiplexers are required for the write access, which are as many as the number of messages computed at a time. Thus the complex interconnection problem is transformed to how to read and write the messages aligned in different ways, requiring complex index generation to access the storage elements.

B. Proposed Loosely Coupled Processing

To resolve the complex interconnection problem, the proposed architecture delivers only the row and column summation values, Δ_j and Λ_i defined in equation (10) and (14), to the neighbor nodes as shown in Fig. 3.

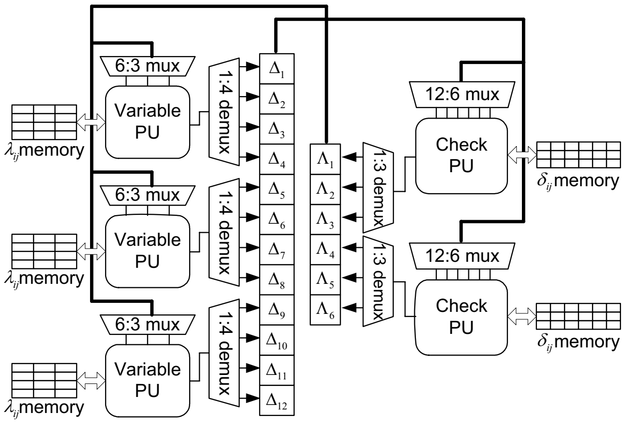

6:3 mux
Variable PU
1:4 demux
λ_{ij}memory
12:6 mux
Check PU
1:3 demux
δ_{ij}memory
Δ_1 Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 Δ_7 Δ_8 Δ_9 Δ_{10} Δ_{11} Δ_{12}
Λ_1 Λ_2 Λ_3 Λ_4 Λ_5 Λ_6

Fig. 3. Partially parallel LDPC decoding architecture.

To derive the individual messages from the summation value, a *PU* has to include additional operations computed in the neighbor *PU*s in the previous architecture. Equations (7) and (8) are restructured to reflect the additional operations. Given the column summation values, Δ_j, a check *PU* recovers individual messages and generates the next row summation value Λ_i to be delivered to the variable *PU*s. And then it computes the intermediate values, δ_{ij}, to be used to recover the individual messages from the next column summation values and stores them into the local flip-flops. The following equations stand for the detailed operation of the check *PU*.

$$\Delta_{ji} = \Delta_j - \delta_{ij} \tag{9}$$

$$\Lambda_i = \sum_{j\in N(i)} \log(\tanh(-\frac{\Delta_{ji}}{2})) \tag{10}$$

$$\Lambda_{ij} = \Lambda_i - \log(\tanh(-\frac{\Delta_{ji}}{2})) \tag{11}$$

$$\delta_{ij} = -2\cdot\tanh^{-1}(\exp(\Lambda_{ij})) \tag{12}$$

Similarly, the variable *PU* recovers individual messages and generates column summation value Λ_i to be delivered to the check *PU*s. In this case, λ_{ji} is the intermediate value.

$$\Lambda_{ij} = \Lambda_i - \lambda_{ij} \tag{13}$$

$$\Delta_j = \log(\frac{p(r_j\,|+1)}{p(r_j\,|-1)}) - \sum_{i\in M(j)} 2\cdot\tanh^{-1}(\exp(\Lambda_{ij})) \tag{14}$$

$$\Delta_{ji} = \Delta_j + 2\cdot\tanh^{-1}(\exp(\Lambda_{ij})) \tag{15}$$

$$\lambda_{ij} = \log(\tanh(-\frac{\Delta_{ji}}{2})) \tag{16}$$

Detailed block diagrams of the variable *PU* and check *PU* are depicted in Fig. 4, where the functions of *LT* and *AE* are $-\log(\tanh(x/2))$ and $2\cdot\tanh^{-1}(\exp(-x))$, respectively. Compared to the previous architecture, the *PU*s in the proposed architecture have both *LT* and *AE* look-up tables and additional subtractions for the added equations (9), (12), (13) and (16). As δ_{ij} and λ_{ji} are not delivered to other type of *PU*s, the number of interconnections is reduced significantly. This loosely coupled architecture alleviates the interconnections required for exchanging Λ and Δ messages aligned in different manners. The duplicated operations increase the hardware complexity of *PU*s. This overhead is inevitable to reduce the interconnection complexity that is more serious than the logic complexity in today's deep submicron technology.

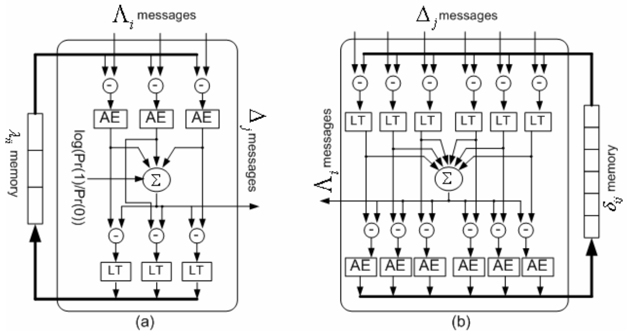


Fig. 4. Processing units for the proposed LDPC decoder. (a) Variable *PU*. (b) Check *PU*.

To reduce the overhead, the proposed architecture can exploit the partially parallel architecture in which each *PU* takes in charge of several rows or columns. As a *PU* is shared for a number of rows or columns, the number of *PU*s becomes much smaller than that of the fully parallel architecture. In addition, since a *PU* processes a row or column at a time, the intermediate values, $\{\delta_{ij}\}$ and $\{\lambda_{ji}\}$, processed by a *PU* can be grouped and stored into a local memory instead of flip-flops to save area. This architecture can reduce the number of interconnections further by sharing the multiplexers required to access the row and column summation values.

C. Overlapped Processing

Traditionally, the *CTV* operations start only after the entire *VTC* step finishes completely, and vice versa. A well-scheduled sequence of the message calculations can reduce the latency, because the *CTV* step can start in the course of the *VTC* step if the corresponding row summation values are available, and vice versa. The overlapped processing of the *VTC* and *CTV* steps results in a reduced number of cycles.

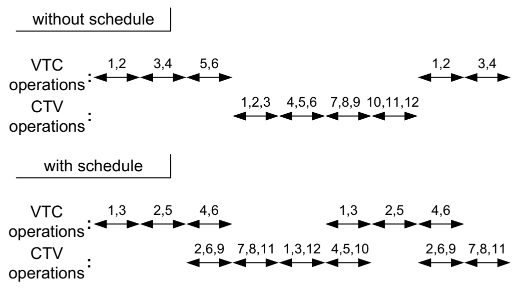
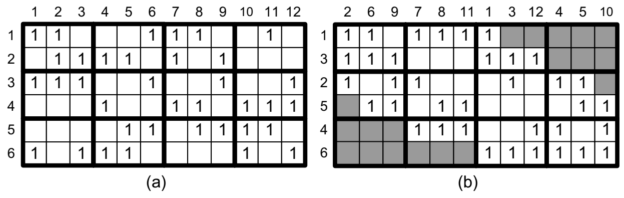


Fig. 5. An example of overlapped processing.

Fig. 5 shows an example scheduling for the (12, 3, 6) LDPC code, assuming that the numbers of check *PU*s and variable *PU*s are two and three, respectively. Each arrow denotes a clock cycle and the numbers above an arrow indicate the row or column indices processed at that cycle. If the *VTC* and *CTV* steps are performed according to the increasing order of indices without scheduling, no overlapped processing is possible. Though two *CTV* operations for variable nodes of index 2 and 7 can start at the last cycle of the *VTC* step, *CTV* step does not start until three *CTV* operations are enabled for easy control design. If the *VTC* operations are weill-scheduled as shown in Fig. 5, three *CTV* operations for variable nodes of index 2, 6 and 9 can start processing at the last cycle of the *VTC* step. Furthermore, the *CTV* operations can be scheduled to enable the *VTC* operations of the next iteration to start earlier. In this example, the scheduling of the *VTC* and *CTV* operations saves two cycles.



(a)

	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1				1	1	1			1	
2		1	1	1	1		1		1			
3	1	1	1			1			1			1
4				1			1	1		1	1	1
5					1	1		1	1	1	1	
6	1		1	1	1					1		1

(b)

	2	6	9	7	8	11	1	3	12	4	5	10
1	1	1		1	1	1	1	▨	▨	▨	▨	▨
3	1	1	1				1	1	1	▨	▨	▨
2	1		1	1				1		1	1	▨
5	▨	1	1		1	1					1	1
4	▨	▨	▨	1	1	1			1	1		1
6	▨	▨	▨	▨	▨	▨	1	1	1	1	1	1

Fig. 6. Scheduling by the permutation of matrix H. (a) Matrix H. (b) Permuted matrix.

It takes a considerable amount of time to find an optimum schedule that minimizes the overall cycles because of the large number of rows and columns. We proposed a new scheduling algorithm developed for the partially parallel LDPC decoding architecture based on the concept of the matrix permutation. The algorithm makes the row sequence and column sequence that can result in empty spaces in the lower left and upper right corners of the permuted matrix when the matrix **H** is rearranged according to the sequences. The well-scheduled sequence in Fig. 5 can be obtained using the proposed algorithm as shown in Fig. 6 (b). For various LDPC codes, the proposed algorithm saves more than 25% cycles on the average. More details about the scheduling algorithm and its performance results can be found in [8].

IV. Experimental Results

We designed a (1024, 3, 6) LDPC decoder based on the proposed partially parallel architecture using a 0.18 um 4-Metal CMOS process. The performances of the decoders which has 32 check *PU*s and 64 variable *PU*s are summarized in TABLE I. The decoder occupies an area of 10.08 mm^2 and provides almost 1Gbps decoding throughput at the frequency of 200MHz. The performance of the second decoder is comparable to the fully parallel architecture proposed by Blanksby et al., but the proposed decoders achieve significant area reduction. Fig. 7 shows the layout of the proposed LDPC decoder. Although the iteration number in Blanksby's implementation is fixed to 64 due to the scan-chain like I/O mechanism, there is no restriction in the proposed architecture. We chose 8 iterations to provide sufficient BER performance.

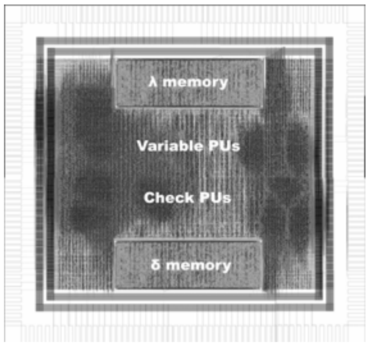


Fig. 7. Layout of the proposed LDPC decoder.

TABLE I
COMPARISON OF LDPC DECODERS

	Blanksby [4]	Proposed
Technology	0.16 um	0.18 um
Bit rate	1 Gbps	985 Mbps
Frequency	64MHz	200MHz
Gate counts	1750K	543K
Area	52.5 mm^2	10.08 mm^2

V. Conclusion

This paper has presented a new LDPC decoding architecture proposed to relax the interconnection complexity and reduce area. In the proposed architecture, only the row and column summation values are exchanged among check and variable *PU*s to reduce the interconnection complexity. To recover the individual messages from the summation values, the function of a *PU* is restructured to include some operations that are traditionally performed in the neighbor *PU*s. In addition, intermediate values accessed by a *PU* are grouped and stored into a local memory instead of registers, since other *PU*s do not access them. The memory-based architecture is area-efficient in the sense that the memory takes much less area compared to the register if both have the same size. We have extended the proposed architecture for the partially parallel architecture to further reduce area by increasing memory usage, and have proposed an efficient algorithm that schedules the processing order of the partially parallel architecture to save the overall processing cycles by overlapping *CTV* and *VTC* steps.

Acknowledgment

This work was supported by Institute of Information Technology Assessment through the ITRC, by the Korea Science and Engineering Foundation through MICROS center and by IC Design Education Center (IDEC).

References

[1] R. G. Gallager, "Low density parity check codes," IRE Trans. Info. Theory, vol. IT-8, pp. 533-547, Jan. 1962.
[2] S. Chung, 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 Comm. Letters, vol. 5, pp. 58-60, Feb. 2001.
[3] T. Richardson, M. Shokrollahi and R. Urbanke, "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes," IEEE Trans. Info. Theory, vol. 47, pp. 619-637.
[4] A. Blanksby and C. Howland, "A 690-mW 1-Gb/s, rate-1/2 low-density parity-check code decoder," IEEE J. of Solid-State Circuits, vol. 37, pp. 404-412, Mar. 2002.
[5] E. Yeo, P. Pakzad, B. Nikolić and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," IEEE Trans. Magnetics, vol. 37, pp. 748-755, Mar. 2001.
[6] K. Lo, Layered space time structures with low density parity check and convolutional codes, MS Thesis, Univ. of Sydney, 2001.
[7] F. R. Kschischang, B. J. Frey, and H. Loeliger, "Factor graphs and the sum-product algorithm," IEEE Trans. Info. Theory, vol. 47, pp. 498-519, Feb. 2001.
[8] I. C. Park and S. H. Kang, "Scheduling Algorithm for Partially Parallel Architecture of LDPC Decoder by Matrix Permutation," *IEEE International Conference on Circuits and Systems*, May 2005.
[9] J. Barry, Low-Density Parity-Check Codes, Tech. Report, Georgia Institute of Technology, 2001