Dispersed Array LDPC Codes and Decoder Architecture for NAND Flash Memory

Wei Shao, Jin Sha, *Senior Member, IEEE*, and Chuan Zhang, *Member, IEEE*

***Abstract*—Quasi-cyclic (QC) low-density parity-check (LDPC) codes have become popular in NAND flash memories, owing to their excellent error correction performance and hardware-friendly structures. However, the large scale of barrel shifters result in prohibitive routing complexity. Array LDPC code is a kind of highly-structured QC-LDPC code, which provides a good balance of performance, complexity and throughput. In this paper, a construction method of dispersed array LDPC (DA-LDPC) codes based on array square is proposed. DA-LDPC codes do not only benefit from the array property, but also a hybrid and efficient storage architecture due to their stair-like structure. For NAND flash applications, the code construction and decoder architecture of a $(18300, 16470)$ DA-LDPC code is illustrated in this paper, where a 2-level decision of LDPC decoding strategy is employed. The numerical results based on an FPGA emulation platform have shown that the error floor of the $(18300, 16470)$ DA-LDPC code is under 10^{-11} in term of bit error rate (BER). Thanks to the well-structured DA-LDPC codes, we can conveniently apply column-based shuffle decoding (CBSD) algorithm for ease of implementation. The corresponding ASIC implementation results have proved that the decoder architecture of DA-LDPC codes can achieve higher normalized-throughput-gate-count-ratio (NTGR) compared to state-of-art works.**

***Index Terms*—DA-LDPC codes, NAND flash memory, column-based shuffle decoding, ASIC.**

I. Introduction

In recent years, more and more NAND flash memories have been used as the number of mobile devices increases. With the growing demand of storage capacity, multi-level cell (MLC) and trinary-level cell (TLC) techniques are introduced to take place of single-level cell (SLC) technique in high-capcity storage applications. The increasing bit density is always accompanied with higher raw bit error rate (RBER) for several factors such as cell-to-cell interference [1] and retention time errors [2]. As a result, more efficient LDPC codes [3] have to be considered. For LDPC decoders, the complex routing, large memory requirement, and limited throughput are severe problems due to the iterative decoding algorithm, which is known as the sum-product algorithm (SPA), and its variants such as min-sum algorithm (MSA) [4]. A 2-level decision of LDPC decoding strategy for NAND flash memories was proposed for higher throughput and reduced read latency penalty [5]. The first level is iterative decoding with the hard decision initial log-likelihood ratio (LLR), and the second level is iterative decoding with the soft one. To improve P/E cycling endurance at minimal read latency overhead, a design strategy uses unequal error correction was proposed in [6].

QC-LDPC code is the most promising class of structured LDPC codes due to its ease of implementation and excellent error correction performance. Several state-of-art works make contributions to the implementation of QC-LDPC codes. Li *et al.* proposed a deep quantization technique to meet a strict cost requirement imposed by NAND flash memories [7]. Ho *et al.* proposed a top-down design methodology, which not only includes code construction but also hardware implementation, in order to design an efficient QC-LDPC decoder [8]. Lee *et al.* proposed a reordered layered decoding processing to save the FIFO buffer at the cost of increased data dependency [9]. QC-LDPC codes constructed from Latin square are popular due to their flexible code constrction method for long code length and high code rate [10]. However, QC-LDPC codes constructed from Latin square always suffer from the large scale of barrel shifters due to the large submatrix size. To solve the problem, the construction method and decoder architecture of the DA-LDPC codes are proposed in this paper. We modify the data flow scheduling of the hybrid storage architecture proposed in [8] for the DA-LDPC codes, and, hence, the barrel shifters are not required any more. CBSD algorithm is applied in our design, and the degree of parallelism (DOP) of it equals to the submatrix size. Thanks to the structure of DA-LDPC codes, the DOP can flexibly be increased to a higher degree without introducing more routing congestion.

The remainder of this paper is organized as follows. Section II reviews the background of CBSD algorithm. In Section III, the construction method of a $(18300, 16470)$ DA-LDPC code and the corresponding girth analysis is proposed. In Section IV, a modified data flow scheduling and improved decoding architecture of the derived DA-LDPC code are illustrated. Implementation results are shown in Section V. Finally, we conclude the paper in Section VI.

II. Column-Based Shuffle Decoding

QC-LDPC codes applied to NAND flash memories are always of long code length and high code rate. As a result, CBSD algorithm [11] is a reasonable choice for them. In this section, we will take a review of the decoding algorithm first, and then analyze the challenges in hardware implementation.

A. Review of CBSD Algorithm

An $(N, N-M)$ LDPC code can typically be defined by a parity-check matrix of size $M \times N$. Notice that L_{cv}^{ω} denotes

Manuscript received July 24, 2017; revised October 12, 2017; minor revised November 28; accepted December 07, 2017. This work was supported by the National Key R&D Program of China (No. 2016YFA0202102), the National Natural Science Foundation of China (No. 61370040 and No. 61501116), the Project on the Industry Key Technologies of Jiangsu Province (BE2017153), Jiangsu Provincial NSF (No. BK20140636), State Key Laboratory of ASIC & System (No. 2016KF007), and PAPD. (Corresponding author: Jin Sha and Chuan Zhang.)

Jin Sha and Wei Shao are with the Electronic Science and Engineering School, Nanjing University, Nanjing, China. Email: shajin@nju.edu.cn, kaitoukito@foxmail.com. Chuan Zhang is with the National Mobile Communications Research Laboratory, Southeast University, Nanjing, China. Email: chzhang@seu.edu.cn.

the check-to-variable (C2V) message from check node (CN) c to variable node (VN) v, and L_{vc}^{ω} denotes the variable-to-check (V2C) message from VN v to CN c at the ωth iteration, where $c = 0, ..., M-1$, and $v = 0, ..., N-1$. Let $L_{init,v}$ be the initial LLR of VN v. Notice that $\mathcal{N}(c)$ is the set of VNs connected to CN c, and $\mathcal{M}(v)$ is the set of CNs connected to VN v. The VNs are divided into G column-groups, and each column-group N_g contains N/G VNs, where $g = 0, ..., G-1$. In addition, for a QC-LDPC code, the column-group size N/G equals to the submatrix size z in this paper. Notice that $\mathcal{N}_L$, ranging from N_0 to $N_{(g-1)}$, contains the column-groups on the left of N_g, while $\mathcal{N}_R$, ranging from N_g to N_{G-1}, contains the column-groups on the right of N_g as well as itself. The bidirectional messages passing between VNs and CNs are calculated as follows.

1) Vertical Step: The V2C message is updated as Eq. (1), and the a posteriori probability (APP) is computed as Eq. (2).

$$L_{vc}^{\omega} = L_{init,v} + \sum_{c' \in \mathcal{M}(v) \backslash c} L_{c'v}^{\omega-1} \tag{1}$$

$$L_{app,v} = L_{init,v} + \sum_{c' \in \mathcal{M}(v)} L_{c'v}^{\omega-1} \tag{2}$$

2) Horizontal Step: The C2V message is updated as Eq. (3), where a represents the normalization factor.

$$\begin{aligned} L_{cv}^{\omega} =& a \times \prod_{v' \in \mathcal{N}(c) \cap \mathcal{N}_L} \operatorname{sgn}(L_{v'c}^{\omega}) \prod_{v' \in \mathcal{N}(c) \cap \mathcal{N}_R \backslash v} \operatorname{sgn}(L_{v'c}^{\omega-1}) \\ & \times \min\left(\min_{v' \in \mathcal{N}(c) \cap \mathcal{N}_L} |L_{v'c}^{\omega}|, \min_{v' \in \mathcal{N}(c) \cap \mathcal{N}_R \backslash v} |L_{v'c}^{\omega-1}| \right) \end{aligned} \tag{3}$$

B. Challenges in the Implementation of CBSD Algorithm

QC-LDPC codes are the most promising class of structured LDPC codes due to their ease of implementation and excellent error correction performance. Barrel shifters are widely used in QC-LDPC decoders, and the function of them is to bring the C2V messages of two adjacent column-groups into alignment while using CBSD algorithm. For example, Fig. 1 shows two adjacent entries of a QC-LDPC code, which are located in (x_1, y_1) and (x_1, y_1+1), respectively. The primitive element α is shown in Eq. (4), which is a permutation matrix with a single cyclic right (or left) shift. Notice that each entry in $\mathbf{H^{QC}}$ represents a circulant permutation matrix (CPM) of size $z \times z$. For example, α^A means α to the power of A. Particularly, α^0 represents an identity matrix, and $\alpha^{-\infty}$ represents a zero matrix. In order to bring α^A and α^B into alignment, the shifting parameter of the barrel shifter should be set as $(B-A) \bmod z$. According to the above rule, the barrel shifters have to provide support for z different shifting parameters, ranging from 0 to $z-1$, which leads to the large scale of barrel shifters. QC-LDPC codes constructed from Latin square always suffer from the large scale of barrel shifters. A code construction method was proposed in [8] to construct long QC-LDPC codes with a small submatrix size, which efficiently reduce the scale of barrel shifters. In addition, a corresponding decoder architecture was also proposed for their derived codes.

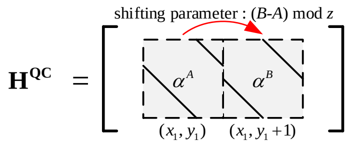

Fig. 1. Illustration of shifting parameter of a QC-LDPC code.

$$\alpha = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix} \tag{4}$$

Array LDPC code [12], which was first proposed in 2002, is a highly structured ensemble of QC-LDPC codes. In this paper, a joint design of code construction and decoder architecture is proposed. Thanks to the structure of DA-LDPC codes, barrel shifters are saved, and the simulation results have shown no degradation of error correction performance.

III. Code Construction and Girth Analysis

For NAND flash applications, the construction method of a $(18300, 16470)$ DA-LDPC code is illustrated in this section. Furthermore, the girth analysis shows that the DA-LDPC codes have girth 6, which indicates that there is no 4-cycle in the parity-check matrix.

A. Construction of a DA-LDPC Code

An array LDPC square $\mathbf{W^a}$ of size $z \times z$, where z is a prime number, is shown in Eq. (5). Given an array square with submatrix size $z = 61$, we can construct a $(18300, 16470)$ DA-LDPC code as following steps.

$$\mathbf{W^a} = \begin{pmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \dots & \alpha^0 \\ \alpha^0 & \alpha^1 & \alpha^2 & \dots & \alpha^{z-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha^0 & \alpha^{z-1} & \alpha^{z-2} & \dots & \alpha^1 \end{pmatrix} \tag{5}$$

1) Step 1: Select a base matrix $\mathbf{H^b}$ from the array square $\mathbf{W^a}$. In this paper, we select the upper left corner of $\mathbf{W^a}$ for simplicity. The size of $\mathbf{H^b}$ is $r \times s$, and here we set $r = 6$ and $s = 60$.

2) Step 2: Separate the base matrix into ten square submatrices, where $\mathbf{H^b} = [\mathbf{H_0^b}, ..., \mathbf{H_9^b}]$. $\mathbf{H_0^b}$ is shown in Eq. (6) as an example, and the rest is similar to it.

$$\mathbf{H_0^b} = \begin{pmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha^0 & \alpha^5 & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} \end{pmatrix} \tag{6}$$

3) Step 3: Transfer each $\mathbf{H_i^b}$ into corresponding dislocated matrix $\mathbf{H_i^d}$. $\mathbf{H_0^d}$ is shown in the upper left corner of Fig. 2 as an example, and the rest is similar to it.

4) Step 4: Repeat each $\mathbf{H_i^d}$ for t times in the stair-like structure. Hence, we obtain corresponding repeated matrix $\mathbf{H_i^r}$. Here we set $t = 5$. $\mathbf{H_0^r}$ is shown on the left-most side in Fig. 2 as an example, and the rest is similar to it.

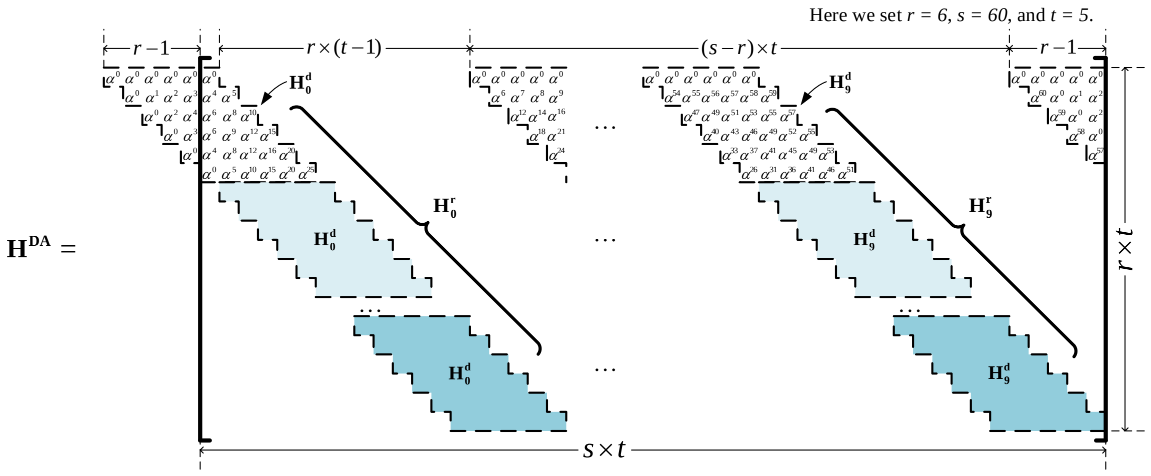

Fig. 2. Illustration of DA-LDPC codes. Here we set $r = 6$, $s = 60$, and $t = 5$.

5) Step 5: Combine the ten repeated matrices $\mathbf{H_i^r}$, where $i = 0, 1, ..., 9$, to obtain the parity-check matrix $\mathbf{H^{DA}}$ of the $(18300, 16470)$ DA-LDPC code. To keep the column-weight $n_v = 6$ and row-weight $n_c = 60$, the left-most column-groups are cut off. To keep the shifting parameter of each row-group as a constant, corresponding entries are added to the upper right corner of the right-most column-groups as is shown in Fig. 2.

B. Girth Analysis for Derived DA-LDPC Code

In this subsection, we will prove that the DA-LDPC code has girth 6, which indicates that there is no 4-cycle in the parity-check matrix. We select two row-groups x_1, x_2 and two column-groups y_1, y_2 from the parity-check matrix of an arbitrary QC-LDPC code of size $M \times N$, whose submatrices are of size $z \times z$. We obtain $\alpha^A, \alpha^B, \alpha^C, \alpha^D$ from the four corresponding entries of the parity-check matrix. Definition 1 is used to define the 4-cycle, and Theorem 1 is proposed to detect whether the four entries are able to make up z 4-cycles. The girth analysis of a QC-LDPC code is illustrated in Fig. 3.

Definition 1. *$\forall i_1, i_2 \in [0 : M - 1]$, and $j_1, j_2 \in [0 : N - 1]$, where $i_1 \neq i_2$, and $j_1 \neq j_2$, $h_{(i_1,j_1)}, h_{(i_1,j_2)}, h_{(i_2,j_1)}, h_{(i_2,j_2)}$ are four elements of the parity-check matrix. The definition for the four elements of making up a 4-cycle is that the value of all the four elements is 1.*

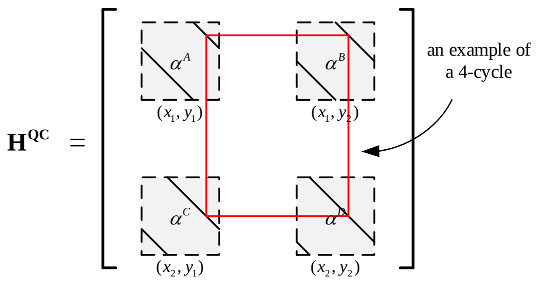

Fig. 3. Illustration of girth analysis of a QC-LDPC code.

Theorem 1. *$\forall A, B, C, D \in [0 : z - 1], (B - A) \mod z = (D - C) \mod z \iff \alpha^A, \alpha^B, \alpha^C, \alpha^D$ are able to make up z 4-cycles.*

Proof. Notice that $h_{i,j}^A$ represents the element of entry α^A of a QC-LDPC code, where $i = 0, ..., z - 1$, and $j = 0, ..., z - 1$. In a CPM α^A, it is obvious that the non-zero elements are located in $h_{i,(i+A) \mod z}^A$, which is the same for the rest three entries $\alpha^B, \alpha^C, \alpha^D$. The necessary and sufficient condition for the z 4-tuples, which are determined by $\{h_{i,(i+A) \mod z}^A, h_{i,(i+B) \mod z}^B, h_{(i+A-C) \mod z,(i+A) \mod z}^C, h_{(i+B-D) \mod z,(i+B) \mod z}^D\}$, are able to make up z 4-cycles is that $(B - A) \mod z = (D - C) \mod z$. □

On the basis of Theorem 1, it can be proved that the DA-LDPC codes has girth 6. We select two row-groups x_1, x_2 and two column-groups y_1, y_2 from the parity-check matrix of the derived DA-LDPC code, and we obtain four corresponding entries. On the one side, if $|x_1 - x_2| < r$, $\alpha^A, \alpha^B, \alpha^C, \alpha^D$ are unable to make up 4-cycles due to the array property. On the other side, if $|x_1 - x_2| \geqslant r$, $\alpha^A, \alpha^B, \alpha^C, \alpha^D$ are also unable to make up 4-cycles beacuse of that at least one entry among the four entries represents a zero matrix.

IV. Decoding Architecture

In this section, we will introduce the decoding architecture of the $(18300, 16470)$ DA-LDPC code. In order to take advanteges of the array property of the DA-LDPC code, we modify the data flow scheduling and improve the decoding architecture proposed in [8].

A. Decoding Architecture of the DA-LDPC Code

Fig. 4 shows the decoder architecture of the DA-LDPC code, and the data width is marked on each bus. The initial LLR channel messages will first be stored into a single-port RAM before decoding starts. There are one VNU block and six CNU block processing units, where the VNU updates the V2C messages, and the CNUs update the C2V messages. A

V2C message includes 3 bits absolute value, 1 bit sign, and 1 bit hard decision of VN, while a C2V message includes 18 bits min-and-index message, 1 bit global sign, and 1 bit check of hard decisions. The so-called min-and-index message consists of 3 bits absolute value and 6 bits index of both the first and the second minimum. The sign messages and hard decisions of VNs calculated by the VNU will not only be passed to CNUs, but also to a dual-port RAM, which is called sign-and-check RAM. A successfully decoded codeword will finally be stored into a single-port RAM named as hard decision RAM. It is worth mentioning that masking technique [13] is widely used to suppress the error floor [14] of LDPC codes. As a result, a single-port ROM is used to store masking matrix, where 0 indicates that the corresponding entry is masked, and vice versa. At each decoding cycle, data flow starts from storage elements, and pass through a selector, a VNU, CNUs, and finally be written back to the storage elements according to multiplexing logics. After all column-groups are updated as the aforementioned process, a decoding iteration is finished. Table I lists the type, width and depth of the ROMs and RAMs. Further illustrations of the hybrid storage architecture and corresponding data flow will be shown in the next subsection.

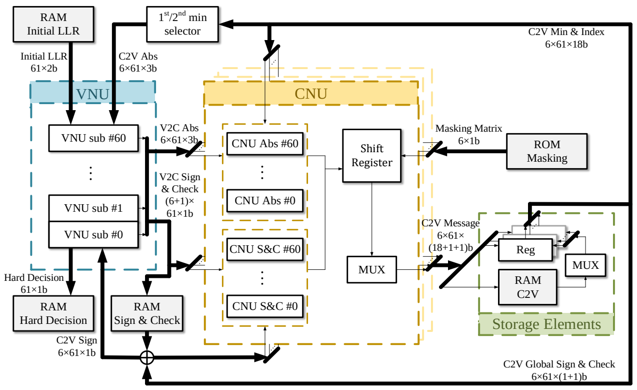

Fig. 4. Decoder architecture for the derived DA-LDPC code.

TABLE I
TYPE, WIDTH AND DEPTH OF THE ROMS AND RAMS

	Type	Width (bit)	Depth
Initial LLR RAM	single-port RAM	122	300
C2V RAM	dual-port RAM	1220	24
Sign & Check RAM	dual-port RAM	427	300
Hard Decision RAM	single-port RAM	61	300
Masking ROM	single-port ROM	6	300

B. Scheduling of CNUs and Storage Elements

Fig. 5 shows the data flow scheduling of the hybrid storage arhitecture, which consits of six registers, ranging from R_0 to R_5, and a C2V RAM. There are six non-zero submatrices in each column-group, and the C2V message corresponding to each submatrix is processed by a CNU block processing unit. The CNUs, ranging from C_0 to C_5, are shown on the left-most side in Fig. 5. Let us define m_i as the C2V message of the ith non-zero submatrix, where $i = 0, ..., 5$. At the start of each decoding cycle, messages are stored in six registers. As is shown in Fig. 5, m_0 will not be used in the next decoding cycle, as a reuslt, it will be stored into the C2V RAM. This kind of messages are denoted as nonimmediate-use data, whose data flow is represented by red arrows. At the same time, the rest five C2V messages, ranging from m_1 to m_5, will be used in the next decoding cycle immediately, as a result, they will be stored back into the corresponding registers. This kind of messages are denoted as the immediate-use data, whose data flow is represented by blue arrows. After $r \times (t-1) = 24$ decoding cycles, the nonimmediate-use data will be read from the C2V RAM, and then be written to the corresponding registers for further use. It deserves to be mentioned that we can use six shift registers instead of the same amount of barrel shifters, owning to the well-defined structure of the DA-LDPC code. In addition, the shifting parameter of the shift register R_i in CNU C_i equals to i, where $i = 0, ..., 5$.

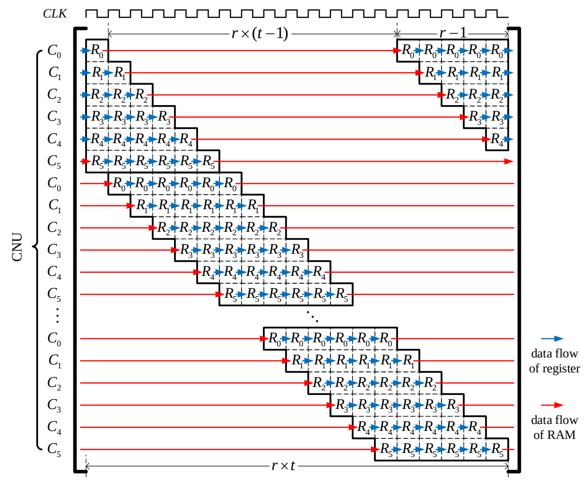

Fig. 5. Data flow of the hybrid storage architecture.

V. IMPLEMENTATION RESULTS

Finally, we implement a register transfer level (RTL) design of the proposed decoder using Xilinx VU440 FPGA device. The maximum iteration number is set as 20. The average iteration number is a variable on the RBER. It is fair to compare the normalized throughput (NT), which means normalizing the throughput with the iteration number, with related works.

For comparison, a $(18900, 17010)$ LDPC code constructed from Latin square [8] is simulated. The error correction performance of both hard-1-bit and soft-2-bit decision of initial LLR over AWGN channel is shown in Fig. 6, and the simulation results have shown no degradation of the LDPC code constructed from array square compared to that constructed from Latin square. It deserves to be mentioned that both codes remain unmasked. The equivalent RBER P_{eq} of AWGN channel is calculated as Eq. (7), where σ denotes the standard deviation. The simulation results have shown that the error floor of the derived DA-LDPC code is under 10^{-11} of both hard and soft sensing in term of BER. In addition, soft-2-bit initial LLR messages are quantized using mutual-information optimized algorithm [15].

$$P_{eq} = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{-1} e^{\frac{-u^2}{2\sigma^2}} du \qquad (7)$$

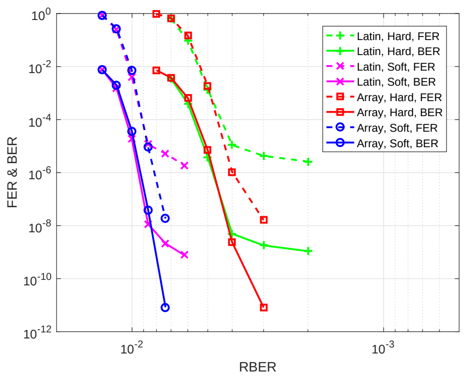

Fig. 6. Comparison between LDPC codes constructed from array square and Latin square. Both codes remain unmasked. The $(18300, 16470)$ LDPC code constructed from array square and the $(18900, 17010)$ LDPC code constructed from Latin square of both hard and soft sensing are simulated over AWGN channel. The number of quantization bit of intermediate messages is 4, and the normalization factor is 0.75.

Table II shows the implementation results of the proposed DA-LDPC docoder, in comparison with related works. Finally, this work is implemented using TSMC 90nm library. The post-layout simulation results have shown that the proposed DA-LDPC decoder is able to achieve a throughput of 1.83Gbps with 6 iterations under a clock frequency of 200MHz. Because of its non-full-rank property while using the encoder architecture proposed in [16], the code rate of the DA-LDPC code is slightly lower than 0.9. It is more justified to compare the NTGR, rather than TAR (throughput-area-ratio [17]), with related works due to the variant number of iterations. The comparison results in Table II have shown that this work achieves the highest NTGR among these works with good error correction performance.

TABLE II
Comparison with Related Works

	This work [Proposed]	[7] M. Li [ISCAS'14]	[8] K. Ho [TVLSI'16]	[9] H. Lee [TCAS-I'17]
Impl.	Post-layout	Synthesis	Post-layout	Post-layout
Schedule	Shuffled	Layered	Shuffled	Layered
Iterations	6	8	6	8
Quant. bits	4	5	4	5
Technology	90 nm	90 nm	90 nm	90 nm
Code length	$18,300$	$18,624$	$18,900$	$18,396$
Code rate	0.897	0.896	0.9	0.905
Freq. (MHz)	200	166	166	200
Gate count	$410k$	$620k$	$520k$	$926k$
Area (mm^2)	1.44	1.75	2.56	4.19
Throughput (Gbps)	1.83	0.77	1.58	4.25
*NT (Gbps)	0.305	0.096	0.263	0.531
*NTGR (Mbps)	0.744	0.155	0.506	0.574

*NT = Throughput/Iterations, *NTGR = NT/Gate count.

VI. Conclusion

In this paper, we propose the code construction method and decoder architecture of a $(18300, 16470)$ DA-LDPC code. The code construction method can flexibly be applied to NAND flash applications of long code length and high code rate. Ultilizing the structure of DA-LDPC codes, the barrel shifters are not required any more, and, hence, the issue of routing congestion is solved without sacrificing error correction performance. Furthermore, masking techniques [10] and postprocessing algorithms of VNU calculation [7], [18] can be adopted to eliminate trapping sets and suppress error floor in future works.

References

[1] G. Dong, S. Li, and T. Zhang, "Using data postcompensation and predistortion to tolerate cell-to-cell interference in MLC NAND flash memory," *IEEE Trans. Circuits Syst. I*, vol. 57, no. 10, pp. 2718–2728, Oct. 2010.

[2] K. Kim, "Technology for sub-50nm DRAM and NAND flash manufacturing," in *Proc. IEDM Tech. Dig.*, Dec. 2005, pp. 323–326.

[3] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[4] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.

[5] W. Zhao, G. Dong, H. Sun, T. Zhang, and N. Zheng, "Hybrid hard-/soft-decision LDPC decoding strategy for NAND flash memory," *China Science Bulletin*, vol. 59, no. 28, pp. 3554–3561, 2014.

[6] J. Li, K. Zhao, and T. Zhang, "Realizing unequal error correction for NAND flash memory at minimal read latency overhead," *IEEE Trans. Circuits Syst. II*, vol. 61, no. 5, pp. 354–358, May 2014.

[7] M.-R. Li, H.-C. Chou, Y.-L.Ueng, and Y. Chen, "A low-complexity LDPC decoder for NAND flash applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 213–216.

[8] K.-C. Ho, C.-L. Chen, and H.-C. Chang, "A 520k (18900, 17010) array dispersion LDPC decoder architectures for NAND flash memory," *IEEE Trans. VLSI Syst.*, vol. 24, no. 4, pp. 1293–1304, Apr. 2016.

[9] H.-C. Lee, M.-R. Li, J.-K. Hu, P.-C. Chou, and Y.-L. Ueng, "Optimization techniques for the efficient implementation of high-rate layered QC-LDPC decoders," *IEEE Trans. Circuits Syst. I*, vol. 64, no. 2, pp. 457–470, Feb. 2017.

[10] J. Li, K. Liu, S. Lin, and K. Abdel-Ghaffar, "Algebraic quasi-cyclic LDPC codes: Construction, low error-floor, large girth and a reduced-complexity decoding scheme," *IEEE Trans. Commun.*, vol. 62, no. 8, pp. 2626–2637, Aug. 2014.

[11] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, Feb. 2005.

[12] E. Eleftheriou and S. Olcer, "Low-density parity-check codes for digital subscriber lines," in *Proc. International Conf. on Commun. (ICC)*, New York, NY, 2002, pp. 1752–1757.

[13] W. E. Ryan and S. Lin, *Channel Codes: Classical and Modern*. New York, NY, USA: Cambridge Univ. Press, 2009.

[14] T. Richardson, "Error floors of LDPC codes," in *Proc. Annu. Allerton Conf. Commun., Control, Comput.*, Oct. 2003, pp. 1426–1435.

[15] J. Wang, T. Courtade, H. Shankar, and R. D. Wesel, "Soft information for LDPC decoding in flash: Mutual-information optimized quantization," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Dec. 2011, pp. 1–6.

[16] C.-F. Tseng and J.-H. Tarng, "Low-complexity and piecewise systematic encoding of non-full-rank QC-LDPC codes," *IEEE Commun. Lett.*, vol. 19, no. 6, pp. 897–900, Jun. 2015.

[17] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Trans. Circuits Syst. II*, vol. 54, no. 6, pp. 542–546, Jun. 2007.

[18] F. Cai, X. Zhang, D. Declercq, S. K. Planjery, and B. Vasić, "Finite alphabet iterative decoders for LDPC codes: Optimization, architecture and analysis," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 5, pp. 1366–1375, May 2014.