A 520k (18 900, 17 010) Array Dispersion LDPC Decoder Architectures through NAND Flash Memory

Shaik.Saiyeeduddin, Jabeena shaik, M.Tech

Department of Electronics and communication Engineering
Quba College of Engineering & Technology,Venkatachalam
(JNTU Anantapur)

Abstract— *Although Latin square is a well-known algorithm to construct low-density parity-check (LDPC) codes for satisfying long code length, high code-rate, good correcting capability, and low error floor, it has a drawback of large sub matrix that the hardware implementation will be suffered from large barrel shifter and worse routing congestion in fitting NAND flash applications. In this paper, a top-down design methodology, which not only goes through code construction and optimization, but also hardware implementation to meet all the critical requirements, is presented. A two-step array dispersion algorithm is proposed to construct long LDPC codes with a small sub matrix size. Then, the constructed LDPC code is optimized by masking matrix to obtain better bit-error rate (BER) performance and lower error-floor. In addition, our LDPC codes have a diagonal-like structure in the parity-check matrix leading to a proposed hybrid storage architecture, which has the advantages of better area efficiency and large enough data bandwidth for high decoding throughput. To be adopted for NAND flash applications, an (18 900, 17 010) LDPC code with a code-rate of 0.9 and sub matrix size of 63 is constructed and the field-programmable gate array simulations show that the error floor is successfully suppressed down to BER of* 10^{12}*. An LDPC decoder using normalized min-sum variable-node-centric sequential scheduling decoding algorithm is implemented in UMC 90-nm CMOS process. The post layout result shows that the proposed LDPC decoder can achieve a throughput of 1.58 Gb/s at six iterations with a gate count of 520k under a clock frequency of 166.6 MHz It meets the throughput requirement of both NAND flash memories with Toggle double data rate 1.0 and open NAND flash interface 2.3 NAND interfaces.*

Index Terms— **Channel coding, low-density parity-check (LDPC) codes, NAND flash memory, sequential scheduling**

I. INTRODUCTION

NOWADAYS, the NAND flash memory is the main storage component in mobile applications, such as mobile phones, tablet, flash drive, and solid-state disk. Its advantages include high program/read speed, shock-resistance, lightweight, and small form-factor. As the applications of multimedia grow explosively, the demand of larger storage capacity keeps increasing. To satisfy the demand, the memory vendors continue boosting the capacity by technology scaling and storing the multiple-bit-per-cell technique. Several state-of-the-art works [1], [2] achieve 128-Gb storage capacity by scaling technology toward 2Xnm and storing 3-bit data in one memory cell. However, these aggressive techniques come along with unavoidable drawbacks. Due to the physical limita-tion and reduced noise margin, the raw bit-error-rate (RBER) keeps increasing and this severely degrades the reliability of NAND flash memory. Error correction code (ECC) is an efficient approach to recover the user data from errors. In current applications, BCH code [3], [4] is the most commonly used ECC which can guarantee t-error correcting capability. For a BCH code, the size of parity bits is almost linearly proportional to its t-error correcting capability. As the storage space for parity bits in flash memory is limited, the correcting capability of BCH code cannot be increased unlimitedly. To overcome the increasing RBER, advanced ECC, such as low-density parity-check (LDPC) codes [5], is widely researched and considered as the next-generation ECC for flash memory application due to its excellent correcting capability.

Manuscript received November 16, 2014; revised February 13, 2015, April 24, 2015, and June 27, 2015; accepted July 11, 2015. Date of publication August 18, 2015; date of current version March 18, 2016. This work was supported by the National Science Council of Taiwan under Project NSC 101-2628-E-009-013-MY3.

The authors are with the Department of Electronics Engineering, Institute of Electronics, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: chu.ee97g@nctu.edu.tw; hcchang@mail.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

LDPC codes were first proposed by Gallager in 1962 and then rediscovered in the late 1990s [6], [7]. An LDPC code is defined by an $m \times n$ parity-check matrix H with m check nodes (CNs) and n variable nodes (VNs). The parameters column degree *(d_v)* and row degree *(d_c)* represent the number of CNs/VNs connected to each VN/CN. LDPC codes can be decoded using belief propagation (BP) algorithm, which iteratively exchanges the messages between the CNs and the VNs. The two most famous BP decoding algorithms are sum–product algorithm [7] and min-sum algorithm (MSA) [8]. Chen and Fossorier [9] presented a normalized MSA, which compensates the performance loss of MSA by multiplying a scaling factor in CN update. In these few years, many researchers and industrial companies are working on how to employ LDPC codes in flash memory applications [10]–[12]. The research topics mainly focus on the code construction, hardware implementation, and the methods of getting soft information from flash memory.

To be adopted for NAND flash application, the LDPC codes must satisfy the critical requirements, such as long code length, high code-rate, good correcting capability, and low error-floor. Latin square [13], [14] is one of the most famous algebraic algorithms [13]–[16] to construct LDPC codes for satisfying the requirements above. In addition, Latin codes are quasi-cyclic (QC) LDPC codes, which are famous for simply controls in the routing network. However, it has critical drawbacks in the aspect of hardware implementa-tion. For NAND flash applications, the code length and the code-rate of ECC is suggested to be 1 or 2 kB, and 0.9, respectively [17]. Such Latin codes come along with large row degree and large submatrix size. In general, VN-centric sequential scheduling [18] (VSS; also known as

The remainder of this paper is organized as follows.
In Section II, we present the VSS and its challenges of hardware implementation for flash memory application. In Section III, we will present the two-step array dispersion code construction algorithm and the code optimizations. In Section IV, we present the overall LDPC decoder architecture with the proposed hybrid register-memory-based architecture, implementation results, and comparison. Finally, the performance analysis of the proposed decoder on real NAND flash memory and the conclusions are given in Sections V and VI, respectively.

II. VARIABLE-NODE-CENTRIC SEQUENTIAL SCHEDULING AND ITS CHALLENGES

In hardware implementation, we adopt the normalized min-sum VSS [18] (NMS-VSS; also known as shuffled decoding [19]) in implementation due to large row degree. The NMS-VSS algorithm not only reduces the hardware overhead of CNU by completing its update process in several subgroups, but also provides faster convergence speed of decoding process. In this section, we will present the NMS-VSS algorithm and its challenges of hardware implementation for flash memory application in detail.

A. Review of NMS-VSS Algorithm

An LDPC code is defined by an $m \times n$ parity-check matrix H with m CNs and n VNs. Two parameters column degree *(d_v)* and row degree *(d_c)* represent the number of CNs/VNs connected to each VN/CN. Notice that L^{w}_{cv} denotes the check-to-variable (CV) message from CN c to VN v , and L^{w}_{vc} denotes the variable-to-check (VC) message from VN v to CN c at the wth decoding iteration. Let $N(c)$ be the set of VNs connected to CN c and let $M(v$ $)$ be the set of CNs connected to VN v . The VSS divides n VNs into G groups that each group contains $n/ G = N_G$ VNs. Let N_g be the set of VNs located in the gth groups for $g = 0, 1, \ldots, G \quad 1$. The following shows the updating process in gth group at wth decoding iteration using NMS-VSS.

$$H_{B,\text{binary}} = \begin{bmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,s-1} \\ A_{1,0} & & & \\ \vdots & \vdots & \ddots & \vdots \\ A_{r-1,0} & \cdots & \cdots & A_{r-1,s-1} \end{bmatrix}$$

of N_g , N_{g+1}, . . . , $N_{G\ 1}$. One complete iteration represents that G groups of VNs are updated in a sequential order. At the end of each iteration, the *a posteriori* probability $L_{\text{app},v}$ value of each VN v is computed by

$$L\text{app},v \;\; L \qquad {}_{L}w \;\; 1 \qquad (5)$$

$$= \qquad \text{init},v \; + \qquad c\;v$$

$c \in M(v\,)$

B. Hardware Implementation Challenges

As mentioned before, there are several challenges of implementing large LDPC codes with NMS-VSS algorithm for flash memory application. They include large barrel shifters and severe routing congestion due to large sub matrix size, and larger storage elements due to increase in data bandwidth.

Chen *et al.* [18] propose a register-based architecture to achieve high throughput performance. CNUs are implemented in fully parallel and all CV messages L^{w}_{cv} are stored in registers. Although registers can provide high data bandwidth, they consume larger area than SRAMs. A 2-kB-length LDPC code requires a huge amount of registers, which will consume too large silicon area and is very power consuming.
size (z), d_c, and k to be 128, 50, and 2, respectively, the bandwidth of memory is equal to $z \times k \times ($6-bit min. value + $\log_2(d_c)\ ) = 128 \times 2 \times (6 + 6) = 3072$ bits. An SRAM with such large data bandwidth will have inefficient aspect ratio, and therefore occupies large silicon area.

Ueng *et al.* [26] present a first-input first-output (FIFO)-based CN architecture supporting multiple

application standards.

III. TWO-STEP CODE CONSTRUCTION ALGORITHM

In this section, we introduce the proposed two-level array dispersion code construction algorithm and advantages on hardware implementation. The goal of presented algorithm is to construct large LDPC codes with good correcting capability, low error floor, and small submatrix size. The two-step algorithm combines two algorithms, namely, Latin square [13], [14] and array dispersion [15]. First, it constructs a small base matrix by Latin square algorithm. Then, we apply the array dispersion to expand the base matrix into a 2-kB-length LDPC code. The expanded LDPC codes have a diagonal-like structure of nonzero submatrices with the same submatrix size as that of the base matrix.

A. Code Construction Algorithm

In the first step of code construction, we adopt the Latin square algorithm [13], [14] to construct a base matrix based on Galois fields (GFs) of 2^q . The Latin square algorithms will construct a large Latin square matrix with size of $2^q \times 2^q$. The entries of matrix are the zero or nonzero elements in GF (2^q). Then, we build a small base matrix by selecting $r \times s$ entries from the large matrix, where $r \leq 2^q$ and $s \leq 2^{\ q}$. LDPC codes which are constructed from different selections of entries have average good performance [13]. In this paper, we select the upper left corner of large matrix for simplicity. Each entry is then mapped to a vector having only one 1 at corresponding location over GF (2). The size Fig. 1. Base parity-check matrix $H_{B,\text{binary}}$ constructed by Latin square algorithm
of vector is $2^q \quad 1$ relating to the number of elements in GF (2^q) without zero element. Furthermore, each binary vector is expanded to a $(2^q \quad 1) \times (2^q \quad 1)$ cyclic sub matrix where each row is a right cyclic-shift of the row above it. Notice that the detailed explanation and the property analysis of Latin square code can be referenced from [13]. Fig. 1 shows the binary base

sub matrices. represent
matrix $H_{B,\text{binary}}$ consisting of r A $i,$ s
$\times s$ q j
a sub matrix
with a size z of 2 1, for 0 $\leq i < r$
and $0 \leq j < s$. $A_{i,j}$

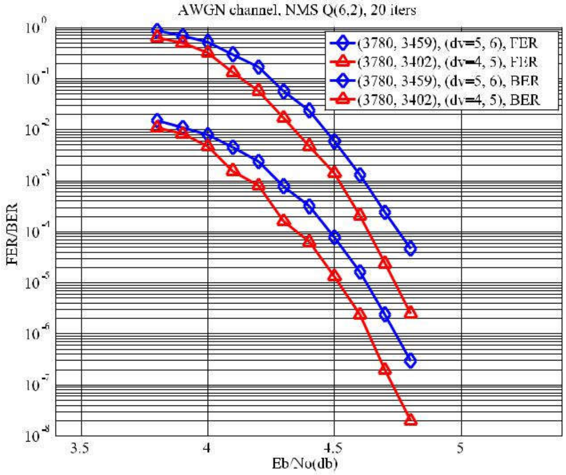
can be a zero matrix or a cyclic-shift of an identity matrix. If the $H_{B,\text{binary}}$ contains no zero sub matrix, the column degree d_v of $H_{B,\text{binary}}$ will be r . Since the Latin square matrix has a diagonal of zero sub matrix, the column degree of $H_{B\ ,\text{binary}}$ will be r or $r \quad 1$. The null space of $H_{B,\text{binary}}$ gives a QC-LDPC code with the guarantee of no cycle-4.

The, we apply the masking technique [27] to the base matrix before going to the second step of code construction. Masking refers to the replacement of a set of sub matrices by zero sub matrices. Since short cycles and small trapping sets [30] are the root cause of error-floor, using the masking technique can efficiently reduce the short cycles and small trapping sets to further improve the BER performance and suppress the error-floor. As the correcting capability of LDPC codes depends on several factors, such as code length, degree distribution, length of girth, and connection of nodes, it is hard to come out an optimized methodology for searching masking matrix. The research of the masking technique is still open and challenging [31]–[33]. In this paper, we apply two rules on the random searching process. One rule is to suppress error floor by generating masking matrices, which guarantee the masked parity-check matrix has no column degree less than 4. Another one is to keep column degrees as close as possible for better hardware utilization of VNU design. To speed up the process of searching the masking matrices, we build up an FPGA platform for fast emulation and over hundreds of masked LDPC codes are simulated. We evaluate and compare the BER of each masked code and the one with the best BER performance is chosen.

In the second step of code construction, we adopt the array dispersion [15] to expand the masked base LDPC code to a large one. LDPC codes constructed by this dispersion technique not only have good erasure-burst-correction capabilities, but also perform well over the additive white Gaussian noise (AWGN) and binary random erasure channels [15]. Fig. 2 shows the array dispersion. A matrix $A_{(t,t\ )}$ with t× t sub matrices is divided into an upper matrix $A_{U\ (t,t\ )}$ and a lower matrix $A_{L(t,t\ )}$. Then, we construct a large matrix by combining the $A_{U\ (t,t\ )}$ and $A_{L(t,t\ )}$ in a diagonal structure with a expand factor of l. Finally, we have a large matrix $H_{(lt,lt\ )}$ which consists of lt × lt sub matrices. The column degree

B. Proposed LDPC Code and Simulation Results

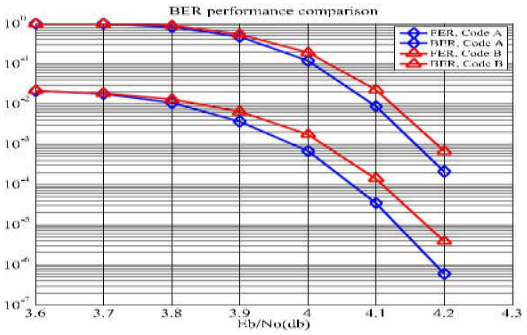
In this paper, we propose an (18 900, 17 010) LDPC code with a code-rate of 0.9 for NAND flash memory applications. In the first step, we construct a $H_{B,\text{binary}}$ with 6 × 60 sub-matrices based on GFs of 2^6. The null space of $H_{B,\text{binary}}$ g distribution and the sub matrix size of $H_{(lt,lt\ )}$ are the same as those of the matrix $A_{(t,t\ )}$.



It is obvious that the matrix hasLDPC codes satisfying row-column-constraint [27], whichdefines that no two rows (columns) that have more than one place where they both have 1 in a matrix, are guaranteed that the parity-check matrix contains no cycle-4. Latin code is one of them and this property is preserved after array dispersion. As shown in Fig. 2, there is only one upper triangle A_U and one lower triangle A_L in each column or row group in the expanded matrix. Since A_U and A_L are decomposed from the Latin square matrix, grouping them horizontally or vertically will not violate the RC-constraint. Therefore, the property of having no cycle-4 is preserved in the LDPC codes after array dispersion.

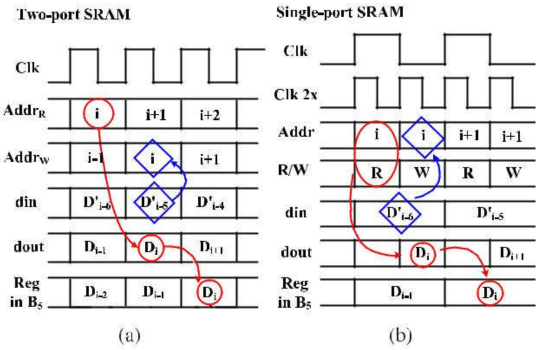
an (3780, 3459) LDPC code with 57 redundant rows and a code-rate of 0.915. The sub matrix size is 63 and the column degree is 5 or 6. Next, a masking matrix is applied to $H_{B,\text{binary}}$ to improve its performance. To preserve the column degree of each column group as close as possible, only one submatrix in each column group will be masked, leading to the column degree of the masked matrix to be 4 or 5.

After generating about 500 sets of masking matrices ran-domly with the above criteria, we evaluate and compare the BER of each masked code at a specified SNR point around BER of 10 ^{7} and then the code with the lowest BER will be chosen. The FPGA simulations of about 500 codes take about two weeks. The null space of the masked parity-check matrix gives an (3780, 3402) LDPC code with no redundant rows and a code-rate of 0.9. Fig. 3 shows the frame error rate (FER) and BER performances of both masked and unmasked base LDPC codes.



In the second step, the base matrix will be divided into multiple sub matrices and expanded by array dispersion. Note that using the masking technique will affect the number of redundant rows, but the existence of redundant rows does not affect the procedure of array dispersion. The base matrix is always consisted of 6 × 60 sub matrices and no row redundant elimination is performed. The masked base matrix $H_{B,\text{binary}}$ is then divided into ten 6× 6 matrices $H_{B,\text{binary},q}$ for $0 \leq q < 10$ by dividing columns into ten groups and it can be presented

as ^{H}B,binary = { ^{H}B,binary,0, . . . , ^{H}B,binary,9}. Each ^{H}B,binary,q consisting of 6 × 6 sub matrices is expanded with a factor of 5 separately by array dispersion and then a larger matrix.



As shown in Fig. 6, the proposed hybrid storage architecture consists of six blocks of registers and a large two-port SRAM. The registers in each block store the messages, which are going to be updated by VNUs and CNUs at each cycle. After the update, the messages that will be processed in the next decoding cycle are stored in the registers again. This kind of messages is denoted as the immediate-use data. At the same time, the messages that are not needed in the next decoding cycle are stored in the SRAM for later use. This kind of messages is denoted as the no immediate-use data. Fig. 6 shows more details of the access behavior of the

.

Let us recall the CNU from [18], there is a local sorter to get the minimum value from VNs located in one column group. Since our parity-check matrix is divided into 300 groups, there are 63 VNs in one column group. Each CNU out of 63 CNUs in one CNU block will receive the messages from one VN accordingly. In the other words, there is only one new message taken as an input to the sorter in each CNU. We can remove the local sorter from the CNU. It is not surprised that the CNU is now equivalent to the CNU with $k = 2$ in [25]. For each CNU, the Since the decoder updates 63 VNs of one column block at each cycle, the data bandwidth of SRAM storing the channel values is *(*63 × 2 bit*)* = 126 bits. All the channel values are stored in a 300 × 126 single-port SRAM. In the beginning of the decoding process, the hard or soft-2-bit channel values are mapped to predefined log-likelihood ratio (LLR). The hard decision of VNU is written to a 300 × 63 single-port SRAM at each cycle. Finally, the two-port SRAMs storing min-and-index messages and global sign bits are replaced by single-port SRAMs to reduce silicon area by operating the SRAMs at double frequency. Note that the ECC protection for SRAM is not adopted in this paper. The NAND flash controller usually adopts the read retry technique [12] to reread and redecode the data multiple times if correction fails. In addition, the failure of decoding due to soft error in SRAM will not lead to a system crash. In general, the NAND flash controller will report a read failure to the OS system and the system will handle the failure as an exception case.

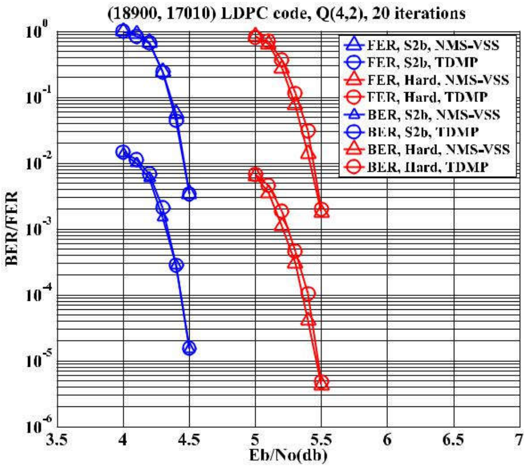
Registers store the min-and-index messages and 1-bit global sign bits. Meanwhile, the min-and-index message refers to the first and second minimum values and their indices.
The number of quantization bits q in our proposed decoder is assumed as 4 and the row degree d_c is around 50. The min-and-index message is $2 \times ((q \quad 1) + \log_2(d_c)\ ) = 2 \times (3 + 6) = 18$ bits in total. The data bandwidth of one

B. Overall Architecture

Fig. 8 shows the overall decoder architecture and the details of one CNU block. There are one VNU block and six CNU blocks, where a VNU block includes 63 VNUs and each CNU block includes 63 sorters, 63 sign operation units, and two barrel shifters for messages and global sign bit. The data flows start from registers, then go through VNUs, CNUs, and barrel shifters. Finally, the data are written back to registers or SRAMs. As presented in Section II, the CV message includes the min-and-index message and global sign bit. $M_{\mathrm{CNU},\ B k}$ represents the 63 min-and-index messages stored in the registers of block B_k for $0 \leq k < 6$. It is sent to the sorters in block B_k . Then, the min message $L_{C\ X,m,\ BK}$ is selected from $M_{\mathrm{CNU},\ Bk}$ by the first/second min selector and sent to VNU

C. Implementation Results and Comparison

Finally, we implement the proposed decoder architecture using UMC 90 nm. The input quantization and message quantization in decoder are soft-2-bit and 4-bit, respectively. The maximum number of iteration is 20 and early termi-nation is adopted. The postlayout simulation shows that the presented decoder can achieve a throughput of 1.58 Gb/s with



flash memory. The gate count is reported as 520k gates as we resynthesis the postlayout net-list (including SRAMs). Fig. 13 shows the place-and-route result and gate count of major modules. The FER and BER performances over AWGN channel are provided in Fig. 11.

Table I lists the comparisons with several state-of-the-art works implementing NMS-VSS algorithm [18], [25], [26] and they are designed for wireless communication application. Meanwhile, Kim and Sung [34] present an LDPC decoder designed for NAND flash application with a very long code length, but the decoding algorithm is not NMS-VSS. In addi-tion, we compare our work with two designs [23], [24] implementing row-based shuffle decoding algorithm for IEEE 802.3an [36]. The LDPC codes adopted in IEEE 802.3an have similar properties, such as high code-rate and low error floor, with the codes adopted for NAND flash application. For fair comparison, the area of all LDPC decoders is normalized to 90 nm. The throughput-to-area ratio (TAR) [35] is computed as TAR = throughput (in Gb/s)/scaled are taken.

VI. CONCLUSION

We have demonstrated the proposed design methodology consisting of code construction, optimization, and hardware implementation. The presented two-step approach can con-struct LDPC codes not only satisfy long code length, high code-rate, good correcting capability, and low error floor, but also have hardware-friendly features, such as small submatrix size and diagonal-like structure of nonzero elements. The code optimization adopts masking to further improve the BER.

REFERENCES

1.Y. Li et al., "128 Gb 3 b/cell NAND flash memory in 19 nm technology with 18 Mb/s write rate and 400 Mb/s toggle mode," in IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC), Feb. 2012, pp. 436–437.

2. G. Naso et al., "A 128 Gb 3 b/cell NAND flash design using 20 nm planar-cell technology," in IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC), Feb. 2013, pp. 218–219.

3. R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," Inf. Control, vol. 3, no. 1, pp. 68–79, 1960.

4. A. Hocquenghem, "Codes correcterus d'erreurs," Chiffres, vol. 2, Pp.117–156, Sep. 1959.

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

6. D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in Proc. 5th IMA Conf. Cryptogr. Coding, Oct. 1995,pp.100–111.

7. D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," IEEE Trans. Inf. Theory, vol. 45, no. 2, pp. 399–431, Mar. 1999.

8. 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.

9. J. Chen and M. P. C. Fossorier, "Density evolution for two improved BP-based decoding algorithms of LDPC codes," IEEE Commun. Lett., vol. 6, no. 5, pp. 208–210, May 2002.

10. J. Yang, "High-efficiency SSD for reliable data storage systems," presented at the Flash Memory Summit, 2011.

11. S. Tanakamaru, Y. Yanagihara, and K. Takeuchi, "Over-10×-extended-lifetime 76%-reduced-error solid-state drives (SSDs) with error-prediction LDPC architecture and error-recovery scheme," in IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC), Feb. 2012, pp. 424–426.

12. K.-C. Ho, P.-C. Fang, H.-P. Li, C.-Y. M. Wang, and H.-C. Chang, "A 45 nm 6 b/cell charge-trapping flash memory using LDPC-based ECC and drift-immune soft-sensing engine," in IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC), Feb. 2013, pp. 222–223.

13. L. Zhang, Q. Huang, S. Lin, K. Abdel-Ghaffar, and I. F. Blake, "Quasi-cyclic LDPC codes: An algebraic construction, rank analysis, and codes on latin squares," IEEE Trans. Commun., vol. 58, no. 11, pp.3126–3139, Nov. 2010.

.