A Low Complexity LDPC-BCH Concatenated Decoder for NAND Flash Memory

Zhongjie Chen\textsuperscript{1a)}, Jin Sha\textsuperscript{1b)}, Chuan Zhang\textsuperscript{2c)}, and Feng Yan\textsuperscript{1}
\textsuperscript{1} Electronic Science and Engineering School, Nanjing University, Nanjing, China.
\textsuperscript{2} National Mobile Communications Research Laboratory, Southeast University, Nanjing, China.

Abstract: Low-density parity-check (LDPC) codes are widely used in NAND flash memory as an advanced error correction method due to their excellent correcting capability. The major challenge is the error floor problem. Dispersed array LDPC (DA-LDPC) code is highly structured and provides implementation convenience due to its regularity. In this paper, it is shown that the constructed (18289, 16384) DA-LDPC code suffers the error floor at BER of $10^{-9}$, which is far from the demand of flash memory error control. Carefully observing the error patterns in the error floor region, we propose a concatenation of BCH code to alleviate this issue. The error floor has been successfully brought down to BER of $10^{-14}$ by concatenating a BCH code with correcting capability of 14 bits. Compared to the standalone LDPC decoder, the concatenated decoder only consumes 7\% extra hardware and the code rate penalty is less than 1\%. Meanwhile, hardware implementation has shown that the throughput can achieve 3.52 Gbps with 6 iterations under a clock frequency of 200 MHz.

Keywords: Dispersed array LDPC code, BCH code, concatenation, error floor, NAND flash.

Classification: Integrated Circuits (logic)

References

[1] Neal R. Mielke, \textit{et al.:} “Reliability of Solid-State Drives Based on NAND Flash Memory,” Proc. IEEE \textbf{105} (2017) 1725 (DOI: 10.1109/JPROC.2017.2725738).
[2] R. C. Bose and D. K. Ray-Chaudhuri: “On a class of error correcting binary group codes,” Information & Control \textbf{3} (1960) 68.
[3] J.L. Massey: “Theory and practice of error control codes,” Proc. IEEE \textbf{74} (1986) 1293 (DOI: 10.1109/PROC.1986.13626).
[4] H. Burton: “Inversionless decoding of binary BCH codes,” IEEE Trans. Inf. Theory \textbf{17} (1971) 464 (DOI: 10.1109/TIT.1971.1054655).
1 Introduction

For NAND flash, it is important to ensure the data reliability. Bit errors will appear due to several factors such as cell-to-cell interference and retention.
errors [1]. To this end, error correction code (ECC) has been introduced for recovery purpose. For single-level cell (SLC) flash memory, Bose-Chaydhuri-Hocquenghem (BCH) [2] codes are commonly employed. Though providing good performance, Berlekamp-Massey (BM) [3] algorithm is of high space and time complexity. Therefore, the inversionless decoding algorithm (IBM) is considered for NAND flash instead, whose algorithm and implementation can be found in [4] and [5], respectively.

Nowadays, with the information development, the demand for larger storage capacity grows drastically. To satisfy the demand, multi-level cell (MLC) and trinary-level cell (TLC) are introduced. However, it also brings higher raw bit error rate (RBER) due to the margin reduction between neighboring states. Since BCH code’s t-error correcting capability is almost linearly proportional to its parity length, long code has to be considered for higher-level cell if we would like to keep with BCH code. However, long BCH code is no longer suitable due to the parity limit. To overcome the increasing RBER, low-density parity-check (LDPC) [6] codes are used instead. Among all LDPC codes, quasi-cyclic (QC) LDPC code [7] is the most promising one for practical application due to its good balance of performance and implementation. Several state-of-art works make contributions to the construction and implementation of QC-LDPC codes for NAND flash [8, 9, 10], among which Latin square is a well-known algorithm to construct QC-LDPC codes with long code length and high code rate. However, the implementation of QC-LDPC code constructed from Latin square usually suffers from large scale of barrel shifters. To alleviate the large barrel shifters and routing congestion, array dispersion [11], which is an algebraic method is employed. Ho et al. proposed a dispersed Latin LDPC code combining Latin square with array dispersion [12]. In this paper, we adopt the dispersed array LDPC (DA-LDPC) code which needs no barrel shifter according to existing research [13].

Although LDPC codes have outstanding correcting capability, the error floor [14] of LDPC codes might not meet the extreme low error rate requirement of flash memory. Urard et al. proposed a concatenation of BCH and LDPC codes to lower the error floor based on the DVB-S2 standard [15]. Chen et al. proposed a concatenated coding system which divided the long BCH code into several short BCH codes with an iterative decoding algorithm between inner QC-LDPC and outer BCH codes [16]. However, code penalty of this scheme is too high and the decoding algorithm is time consuming. Therefore, this scheme is not suitable for flash memory, which requires high code rate and throughput. In this paper, a new concatenation scheme based on the proposed (18289, 16384) DA-LDPC code is presented and implemented based on column-based shuffle decoding (CBSD) [17].

The remainder of the paper is organized as follows. Section II introduces the construction of the DA-LDPC code and the (18499, 16384) concatenated code. Section III presents the architecture of the concatenated decoder. Section IV shows the implementation results. Section V concludes this paper.
2 Code Construction

Hardware implementation of LDPC decoder has attracted lots of research attentions in the past a dozen years. Fully parallel architecture directly maps standard belief propagation (BP) algorithm into hardware by specifying connections between check nodes and variable nodes, which is the fastest but the routing congestion it induced is not tolerable [18]. Partially parallel decoder architecture achieves a good tradeoff between decoding speed and hardware complexity [19]. The biggest challenge is the large message memory consumption it introduced. Compressed message storage together with min-sum decoding algorithm (MSA) and CBSD decoding schedule is a smart way to relieve this problem. Now the tricky part becomes the huge register-file requirement and the barrel shifters to support the circulant permutation matrix (CPM) indexes. Further two smart moves have been proposed to relieve these two problems. Firstly, the array dispersion skill can make the parity check matrix very sparse and most of the messages can be stored in SRAM instead of registers. Secondly, a very regular array code instead of random QC-LDPC code construction can be applied to relieve the decoder from the resource hungry barrel shifters. By combining all the techniques mentioned above, an ad-hoc code construction called dispersed array LDPC code (DA-LDPC) can be achieved which induces the most efficient decoder hardware implementation so far.

In this section, a (18289, 16384) DA-LDPC code will be constructed first. After performance simulation of the DA-LDPC code, we select a suitable BCH code to make up the whole concatenated code.

2.1 Construction of the DA-LDPC code

An \((N, N-M)\) LDPC code can typically be defined by a parity-check matrix of size \(M \times N\). An array LDPC square \(W^a\) of size \(z \times z\) will be applied to carry out the array dispersion. Eq. (1) shows the array square \(W^a\). The size of the square of \(z\) must be a prime number, where \(z = 127\) in this paper. The entries in \(W^a\) derive from galois field \(GF(q)\), where \(q = z + 1\). We construct the dispersed array LDPC code as following steps.

\[
W^a = \begin{pmatrix}
\alpha^0 & \alpha^0 & \alpha^0 & \ldots & \alpha^0 \\
\alpha^0 & \alpha^1 & \alpha^2 & \ldots & \alpha^{z-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\alpha^0 & \alpha^{z-1} & \alpha^{z-2} & \ldots & \alpha^1
\end{pmatrix}
\]

(1)

2.1.1 Base Matrix

Selecting \(r \times s\) entries from array square \(W^a\) to build a base matrix \(H^b\). In this paper, we select the upper left corner of the check matrix for simplicity and the size of \(H^b\) is \(5 \times 50\).

2.1.2 Array Dispersion

Fig. 1 shows the process of array dispersion. Firstly, we separate the base matrix into ten subtrices, where \(H^b = [H^b_0, \ldots, H^b_9]\), and \(H^b_0\) is shown in Eq.
(2) as example. Then, we transfer each $H_i^b$, where $i = 0, 1, 2, \ldots, 9$, into a dislocated form, shown in upper right corner of Fig. 1. The matrix $H_i^b$ is decomposed into a upper triangular matrix $H_{ui}^b$ and a lower triangular matrix $H_{li}^b$. Afterwards, we integrate $H_{ui}^b$ and $H_{li}^b$ together and repeat it for $t$ times to obtain matrix $H_i^b$. In addition, the matrix $H_{ui1}^b$ in $H_{li0}^b$ has the similar format as the Eq. (2). Finally, we combine ten matrices $H_i^b$ to obtain the parity-check matrix $H^D$ of the DA-LDPC code. In this paper, we set the value of $t$ as 3. So the size of matrix $H^D$ is $(3 \times 5) \times (3 \times 50)$. All entries of matrix $H^D$ derive from array square $W^a$ as well as galois field $GF(128)$. The array dispersion operation in this paper is slightly different from that in [12]. This method of array dispersion can retain the row entries as arithmetic progressions so that it can relieve the decoder from barrel shifters [13].

$$H_i^b = \begin{pmatrix}
(a^0)^{5i} & (a^0)^{5i+1} & (a^0)^{5i+2} & (a^0)^{5i+3} & (a^0)^{5i+4} \\
(a^1)^{5i} & (a^1)^{5i+1} & (a^1)^{5i+2} & (a^1)^{5i+3} & (a^1)^{5i+4} \\
(a^2)^{5i} & (a^2)^{5i+1} & (a^2)^{5i+2} & (a^2)^{5i+3} & (a^2)^{5i+4} \\
(a^3)^{5i} & (a^3)^{5i+1} & (a^3)^{5i+2} & (a^3)^{5i+3} & (a^3)^{5i+4} \\
(a^4)^{5i} & (a^4)^{5i+1} & (a^4)^{5i+2} & (a^4)^{5i+3} & (a^4)^{5i+4}
\end{pmatrix}$$

2.1.3 CPM Dispersion

Every entry in $H^D$ is an element in galois field $GF(q)$. The CPM dispersion is to transfer $\alpha^i$ into a $(q-1) \times (q-1)$ square matrix. Specifically, 0 represents a zero matrix, $\alpha^0$ represents an identity matrix and $\alpha^i$ represents a permutation matrix with $i$ cyclic right (or left) shift. The row and column of the matrix will be larger, however their degree will remain unchanged. After CPM dispersion, we obtain the final parity-check matrix $H^{AD}$ with size of $1905 \times 19050$.

$$H^t = \begin{bmatrix}
A_0 & A_1 & A_2 & A_3 & A_4 \\
A_0 & A_1 & A_2 & A_3 & A_4 \\
A_0 & A_1 & A_2 & A_3 & A_4 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
A_0 & A_1 & A_2 & A_3 & A_4
\end{bmatrix} \rightarrow \begin{bmatrix}
A_0 & A_1 & A_2 & A_3 & A_4 \\
A_0 & A_1 & A_2 & A_3 & A_4 \\
A_0 & A_1 & A_2 & A_3 & A_4 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
A_0 & A_1 & A_2 & A_3 & A_4
\end{bmatrix}$$

$$H^t = \begin{bmatrix}
0 & A_0 & A_1 & A_2 & A_3 \\
0 & 0 & A_0 & A_1 & A_2 \\
0 & 0 & 0 & A_0 & A_1 \\
0 & 0 & 0 & 0 & A_0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix} \rightarrow \begin{bmatrix}
A_0 & 0 & 0 & 0 & 0 \\
A_1 & A_0 & 0 & 0 & 0 \\
A_2 & A_1 & A_0 & 0 & 0 \\
A_3 & A_2 & A_1 & A_0 & 0 \\
A_4 & A_3 & A_2 & A_1 & A_0
\end{bmatrix}$$

$$H^t = \begin{bmatrix}
H^t & 0 & H^{t+1}_n \\
H^t & H^t & 0 \\
0 & H^t & H^t
\end{bmatrix}$$

Fig. 1. Illustration of array dispersion.
Table I. Number of Error Frames with Different Error Bits

| EF σ | EB    | 1 to 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | >14 |
|------|-------|--------|---|---|---|---|----|----|----|----|----|-----|
| 0.41 | 0     | 2      | 78| 0 | 1 | 0 | 18 | 1  | 0  | 0  | 0  | 0   |
| 0.40 | 0     | 1      | 87| 0 | 0 | 1 | 9  | 0  | 1  | 0  | 0  |     |
| 0.39 | 0     | 0      | 91| 0 | 1 | 1 | 7  | 0  | 0  | 0  | 0  |     |
| 0.38 | 0     | 0      | 90| 0 | 1 | 0 | 8  | 1  | 0  | 0  | 0  |     |

*EF = Error frames, *EB = Error bits per frame.

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 [1]. In this paper, the length of information bits is 2 kB. We need add 761 zeros, part of which serves as BCH parity bits, after 2 kB-length information bits when encoding. Irrespective of compensatory zeros, the LDPC code length is 18289 (n = 18289), parity check bit length is 1905 (m = 1905), information bit length is 16384 (k = 16384), code rate is 0.896 (r = 0.896), column degree is 5 and row degree is 50.

2.2 Concatenated Code

Through extensive simulations, we find that our DA-LDPC code cannot lower the error floor enough by masking [12]. Hence, concatenation is the suitable scheme for our code. The role of the BCH code is to help the LDPC code overcome its error floor, so the selection of BCH code is decided by the correcting performance of the LDPC code. The performance of our DA-LDPC code is simulated over AWGN channel with GPU [20]. The statistics of error frame number with different error bits in error floor region are shown in Table I. The total error frame numbers are all set as 100 and σ denotes the standard deviation of AWGN channel. By observing the error patterns in the error floor region, we find that overwhelming majority of error frames has 8 error bits and 12 error bits as well as that all 100 frames have less than 14 error bits. The error floor phenomenon appears due to trapping sets of the LDPC code [14]. Hence, we set the correcting capability of BCH code as 14 bits to overcome the trapping sets, what in other words means that the BCH code can guarantee to correct error frames with less than or equal to 14 error bits. The encoding of concatenated code is shown in Fig. 2. The 2 kB information bits are sent to a BCH system encoder with error correcting capability of 14 bits. Then the BCH codewords output is encoded by the LDPC system encoder to generate LDPC parity bits. The length of BCH parity bits and LDPC parity bits are 210 bits and 1905 bits, respectively. Hence, the final concatenated code is a (18499, 16384) code and the code rate drops to 0.886 (r = 0.886).
3 Concatenated Decoding Architecture

Fig. 3 shows the flow chart of the concatenated decoding system. The LDPC decoder reads hard information from memories and make hard decisions first. If hard decisions fail, the LDPC decoder turns to implement soft-decoding. If the LDPC decoder fails again, the LDPC decoder output will be sent to the BCH decoder, then obtain the final decoding result. In this section, hardware architecture of the DA-LDPC decoder based on CBSD algorithm will be introduced first. Then the BCH decoder is connected after the LDPC decoder through some controls to construct the complete concatenated decoder.

3.1 Decoding Architecture of the DA-LDPC Code

Fig. 4 shows the decoding architecture of the DA-LDPC code and the data width is marked on each bus. The data width is based on soft decoding, where the number of quantization bit of intermediate messages is 4. Three memories marked by yellow work as follows. The initial LLR channel messages will first be stored into INIT LLR MEM before decoding starts. C2V MEM is a dual-port RAM that stores C2V messages including 20 bits min-and-index message, 1 bit global sign and 1 bit check of hard decisions. The 20 bits min-
and-index message consists of 3 bits absolute value and 7 bits index of the first minimum as well as the second minimum. There are 15 sets of C2V messages in total, 10 of which not used in the next decoding cycle are stored in C2V MEM and the rest 5 sets are updated in the coming cycle. 5 bits sign message of V2C messages and 1 bit check of hard decision are stored in V2C MEM.

There are one VNU block and five CNU block processing units, where the VNU updates the V2C messages, and the CNUs update the C2V messages. A complete iteration begins and ends in the rightmost storage elements including a C2V MEM and five registers, ranging from $R_0$ to $R_4$. Five sets of C2V messages to be dealt with are stored in five registers. Then the min&index messages are sent to selectors that select the first or the second minimum according to their indexes. Afterwards, the initial LLR messages from INIT LLR MEM and the selected minimum messages are sent to VNUs. VNUs add them together to obtain V2C and APP messages. The VNU output messages are sent to the CNUs to update C2V messages as well as to the V2C MEM to store. The absolute values and signs are updated separately in CNUs. Then the updated C2V messages are stroed back in the storage elements after shifted by five shift registers $SR_i$. Messages from register $R_i$ are shifted by the corresponding shift register $SR_i$. In addition, the shifting parameter of the shift register $SR_i$ equals to $i$, where $i = 0, \ldots, 4$. One set of updated C2V messages will be stroed in MEM C2V and the rest four sets are stroed back in registers. A new set of C2V messages taken from MEM C2V will fill the blank register. With this, the iteration of one column block ends. When overall 150 column block (consume 150 clocks) iterations accomplish, the whole decoding architecture accomplish an iteration. If the codewords can pass the parity check at any decoding cycle, the decoder stops the iteration and outputs the decoded words, which is the function of the Early termination module. Owning to the well-defined structure of the DA-LDPC code, the five shift registers can replace the barrel shifters, which reduces a large area of the decoder.

3.2 BCH Decoder

Whether the LDPC decoder decodes successfully decides the work mode of concatenated decoding system, through the BCH decoder or not. We improve the conventional BCH decoder architecture using IBM algorithm in [5] by reusing multipliers over galois field, which can save approximately 20% area of the BCH decoder. The width of LDPC decoder output is 127 bits, while the width of BCH decoder input is not necessary as high as LDPC decoder. This is because the BCH decoder works only when the LDPC decoder fails, probability far less than 0.1% if the flash memory is in its life time. To lower the complexity of the decoder, we set the width of BCH decoder input as 10 bits. Its throughput is only approximately 1/12 of LDPC decoder in fact. The BCH decoder consumes very few extra hardware resources and brings about negligible throughput loss. Due to the difference of data width between two decoders, a FIFO to convert data width is necessary. In next
Fig. 4. Decoding architecture for the DA-LDPC code.

section, we will compare hardware resources of the concatenated decoder with the LDPC decoder.

4 Implementation Results

To verify the performance of our concatenated decoding system, a register transfer level (RTL) design is implemented in Xilinx VU440 FPGA device. For comparison, a (18900, 17010) LDPC code constructed from Latin square [12] is simulated. Fig. 5 shows the frame error rate (FER) and bit error rate (BER) performances of our dispersed array LDPC code and dispersed latin LDPC code over AWGN channel, both under CBSD decoding algorithm. The input quantization and message quantization of two codes are all soft-2-bit and 4-bit, respectively. We can see both of them suffer from error floor at BER above $10^{-10}$, which cannot meet the demand of flash memory. The performance curves of concatenated codes are also shown in Fig. 5. The concatenated codes include concatenating BCH codes with correcting capability of 10 bits and 14 bits. Thanks to the slight code rate loss, the performance of concatenated code in water-fall area does not suffer the degradation compared to the standalone DA-LDPC code. The error floor is suppressed down to BER of $10^{-14}$ when concatenating a BCH code with correcting capability of 14 bits.

| Table II. Gate Count of the Decoders |
|-------------------------------------|
| Gate Count | Concatenated | BS LDPC | NBS LDPC |
|------------|--------------|---------|----------|
| 650k       | 840k         | 610k    |

Then we will discuss the hardware resources and throughput of the decoder based on implementation using TSMC 90nm library. We first take the area of concatenated decoder, LDPC decoder with barrel shifters (BS LDPC) [12] and non-barrel shifter LDPC decoder (NBS LDPC) for comparison. The
Fig. 5. Error correction performance of LDPC codes and concatenated codes. The (19050, 17145) LDPC code constructed from array square and the (18900, 17010) LDPC code constructed from Latin square are simulated over AWGN channel. The number of quantization bit of intermediate messages is 4, and the normalization factor is 0.75. The maximum iteration number is set as 20. The concatenated codes include concatenating BCH codes with correcting capability of 10 bits and 14 bits. Concatenation is implemented on our dispersed array LDPC code.

BS LDPC decoder and NBS LDPC decoder use dispersed array LDPC codes and dispersed Latin LDPC codes of same size, respectively. Table II lists the gate count of the architectures above. It is observed that NBS LDPC decoder saves about 25% hardware resources compared with the BS LDPC decoder and the concatenated decoder consumes only less than 7% extra gates compared with the NBS LDPC decoder. So, our concatenated decoder has advantages over BS LDPC decoder in hardware resources.

When decoding, the average iteration number is a variable on the RBER. So the the normalized throughput (NT), which means normalizing the throughput with the iteration number, is introduced to represent the processing speed of decoder. Meanwhile, there is a positive correlation between throughput and hardware resource cost. So, it is fair to compare ratio between normalized throughput (NT) and gate count with related works. Table III shows the implementation results of the proposed concatenated decoder, in comparison with related works. The postlayout simulation shows the proposed concatenated decoder can achieve a throughput of 3.52 Gbps with 6 iterations under a clock frequency of 200 MHz. Concatenation lower the code rate from previous 0.895 to 0.885 to get a better correction performance. Thanks to the non-barrel shifter LDPC decoder architecture, our concatenated decoder achieve the highest NTGR among these works with good error correction performance.
Table III. Comparison with Related Works

|                  | This work [Proposed] | [8] M. Li [ISCAS’11] | [12] K. Ho [TVLSI’11] | [9] H. Lee [TCAS-I’11] |
|------------------|----------------------|----------------------|----------------------|----------------------|
| Impl. Schedule   | Post-layout          | Synthesis            | Post-layout          | Post-layout          |
| Iterations       | 6                    | 8                    | 6                    | 8                    |
| Quant. bits      | 4                    | 5                    | 4                    | 5                    |
| Technology       | 90 nm                | 90 nm                | 90 nm                | 90 nm                |
| Code length      | 18,499               | 18,624               | 18,900               | 18,396               |
| Code rate        | 0.885                | 0.896                | 0.9                  | 0.905                |
| Freq. (MHz)      | 200                  | 166                  | 166                  | 200                  |
| Gate count       | 650k                 | 620k                 | 520k                 | 926k                 |
| Throughput (Gbps)| 3.52                 | 0.77                 | 1.58                 | 4.25                 |
| *NT (Gbps)       | 0.587                | 0.096                | 0.263                | 0.531                |
| *NTGR (Mbps/G)   | 0.882                | 0.155                | 0.506                | 0.574                |

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

5 Conclusion

In this paper, we propose the code construction and decoder architecture of a (18499, 16384) DA-LDPC and BCH concatenated code. The concatenated code has long code length, high code rate and good correction performance. Meanwhile, the decoder architecture can achieve high throughput by using relatively few hardware resources due to the codesign of code construction and decoder architecture. These features make the concatenated code suitable for NAND flash applications. Furthermore, adjusting the BCH correcting capability can make the concatenated system support multi-code rate, which has the prospect of research and application.

Acknowledgments

This work was supported by National Key R&D Program of China (No. 2016YFA0202102), National Nature Science Foundation of China under Grant No. 61370040, the project on the Industry key technologies of Jiangsu Province BE2017153; A Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions PAPD