Ph.D. DISSERTATION

High-Performance LDPC Decoding for Error Correction of NAND Flash Memory

낸드플래시 메모리 오류정정을 위한 고성능 LDPC 복호방법 연구

BY

JONGHONG KIM

AUGUST 2013

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

High-Performance LDPC Decoding for Error Correction of NAND Flash Memory

낸드플래시 메모리 오류정정을 위한 고성능 LDPC 복호방법 연구

지도교수 성 원 용

이 논문을 공학박사 학위논문으로 제출함

2013년 8월

서울대학교 대학원

전기·컴퓨터 공학부

김 종 홍

김종홍의 공학박사 학위논문을 인준함

2013년 8월

위 원 장 :	노 종 선
부위원장 :	성 원 용
위 원 :	채 수 익
위 원 :	이 정 우
위 원 :	조 준 호

Abstract

High-performance error correction for NAND flash memory is greatly needed because the raw bit error rate increases as the semiconductor geometry shrinks for high density. Soft-decision error correction, such as low-density parity-check (LDPC) codes, offers high performance but their implementation complexity hinders wide adoption to consumer products. This dissertation proposes two high-performance message-passing schedules and a low-complexity decoding algorithm for LDPC codes. In particular, an efficient decoder architecture for finite geometry (FG) LDPC codes is proposed, and the energy consumption of soft-decision decoding for NAND flash memory is analyzed.

The first part of this dissertation is devoted to improving the informed dynamic scheduling (IDS) algorithms. We analyze the behavior of the residual belief propagation (RBP), which is the fastest IDS algorithm, and develop an improved RBP (iRBP) by avoiding the concentration of message updates at a particular node. We also study the syndrome-based mixed scheduling of the iRBP and the node-wise scheduling (NS). The proposed mixed scheduling outperforms all other scheduling methods tested in this work.

The next part of this dissertation is to develop a conditional variable node update scheme for the *a posteriori* probability (APP) algorithm. The developed algorithm is robust to decoding failures and can reduce the dynamic power consumption by lowering switching activities in the LDPC decoder. To implement the developed al-

gorithm, we propose a memory-efficient pipelined parallel architecture for LDPC decoding. The architecture employs FG-LDPC codes that not only show fast convergence speed and good error-floor performance but also perform well with iterative decoding algorithms, which is especially suitable for data storage devices. We also developed a rate-0.96 (68254, 65536) Euclidean geometry LDPC code and implemented the proposed architecture in 0.13-μm CMOS technology.

This dissertation also covers low-energy error correction of NAND flash memory through soft-decision decoding. The soft-decision-based error correction algorithms show high performance, but they demand an increased number of flash memory sensing operations and consume more energy for memory access. We examine the energy consumption of a NAND flash memory system equipping an LDPC code-based soft-decision error correction circuit. The sum of energy consumed at NAND flash memory and the LDPC decoder is minimized. In addition, the chip size and energy consumption of the decoder were compared with those of two Bose-Chaudhuri-Hocquenghem (BCH) decoding circuits showing the comparable error performance and the throughput. We also propose an LDPC decoder-assisted precision selection method that needs virtually no overhead. This dissertation is intended to develop high-performance and low-power error correction circuits for NAND flash memory by studying improved decoding and scheduling algorithms, VLSI architecture, and a read precision selection method.

Keywords : Dynamic scheduling, low-density parity-check (LDPC) codes, NAND flash memory, soft-decision error correction, soft-decision sensing operation

Student Number : 2009-30185

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 NAND Flash Memory

NAND flash memory is widely used in many mobile devices, such as cellular phones, digital cameras, and smart-pads because of high capacity, fast access speed, and low power consumption. In particular, solid-state drives (SSDs) for notebook computers become popular as the density of NAND flash memory increases rapidly.

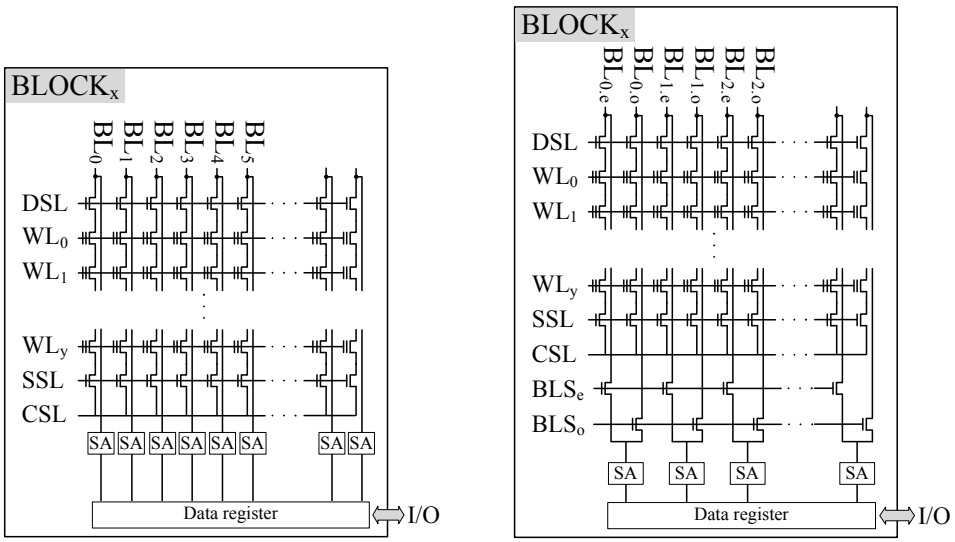
A NAND flash memory device contains thousands of cell blocks that can independently be erased. Each cell block consists of rows and columns of cells. The cells in the same row and those in the same column are controlled by the same word-line (WL) and the same bit-line (BL), respectively. Each flash memory cell is a floating gate NMOS transistor in which the gate stores charges to control the threshold voltage of the transistor. Because of the process variation, program inaccuracy, charge leakage, and noise, the threshold voltage of NAND flash memory has a Gaussian-like distribution, which can cause bit errors when reading the cell. Hence, traditionally,

NAND flash memory systems equip error correction circuits that employ Hamming, Bose-Chaudhuri-Hocquenghem (BCH), or Reed-Solomon (RS) codes [1, 2, 3, 4, 5].

Conventional NAND flash memory devices adopt either all-BL or even/odd-BL structure. Figure 1.1(a) shows the all-BL structure in which all the cells in the same WL can be read or programmed simultaneously, where DSL, SSL, and CSL denote drain-select, source-select, and common-source lines, respectively. Because the unit of read and write operations is called a *page*, the number of BLs in the all-BL structure equals to the number of bits in a page. Note that the typical page size of the current generation of NAND flash memory is 64 kbits (8 kbytes) besides the parity data. The even/odd-BL structure is illustrated in Fig. 1.1(b) in which the cells in even BLs and those in odd BLs are independently selected, thus the same peripherals can be shared by two adjacent BLs, and two pages are mapped to an WL. Although the even/odd-BL structure reduces the overhead of peripheral circuits by sharing data latches and sense amplifiers (SAs), this one incurs larger cell-to-cell interference (CCI) when compared to the all-BL structure.

Today's NAND flash memory adopts the multi-level cell (MLC) technology that stores more than one bit per memory cell to increase the density. The organization of a 128-Gbit NAND flash memory device with 2-bit MLC technology is shown in Table 1.1 [6]. Note that in 2-bit MLC NAND flash memory, two and four pages are mapped to an WL in the all-BL and the even/odd-BL structures, respectively.

The MLC technology, however, reduces the gap between adjacent threshold voltage levels, which significantly increases the bit error rate (BER). Moreover, as the feature size of NAND flash memory shrinks, the number of electrons in the floating gate of a transistor also decreases, and as a result, the memory is very prone to charge loss caused by long data retention [7]. The CCI also increasingly deteriorates



(a) All-bit-line structure (b) Even/odd-bit-line structure

Figure 1.1: Two bit-line structures of NAND flash memory

Table 1.1: The features of 34-nm 2-bit MLC NAND flash memory

Capacity	128 Gbits
MLC tech.	2 bits/cell
Device size	8,192 blocks
Block size	256 pages
Page size	8,192 + 448 bytes

the reliability of information stored at the floating gates [8, 9]. It is also well known that SSD applications usually demand high program-and-erase cycles, which greatly affects the reliability of NAND flash memory [10].

NAND flash memory devices have a spare region at each page to store parity bits for error correction. Traditionally, Hamming and BCH codes have been widely used for NAND flash memory error correction. However, as the process technology scales down continuously, more advanced error-correcting codes are needed to keep NAND flash memory reliable. Soft-decision error-correcting methods can increase

the error-correcting performance because the reliability of stored information can also be utilized. In this dissertation, we consider LDPC codes as error correction of NAND flash memory because of their excellent error-correcting capability and highly parallelizable decoding scheme.

1.2 LDPC Codes

LDPC codes [11, 12] have received great attention in recent years because of their capacity-approaching performance and fully parallelizable decoding algorithms. In particular, LDPC codes have successfully been applied to many communication systems such as DVB-S2 [13], IEEE 802.3an [14], IEEE 802.11n [15], and IEEE 802.16e [16].

The performance of LDPC decoding can be improved by employing the serial message passing schedule [17, 18, 19, 20, 21]. The serial schedule uses the renewed messages immediately for updating their neighboring nodes and, as a result, shows better error performance than the conventional flooding-based ones. In addition, the informed dynamic scheduling (IDS) algorithms not only increase the convergence of the decoding but also significantly improve the error performance by removing trapping set errors [22, 23] when compared to the static scheduling algorithms such as the serial and the flooding schedules. Although it takes more operations to decode a codeword due to the nature of the IDS algorithms, it can be used for future applications where error performance is critical. Meanwhile, in order to further improve the error performance, mixed scheduling of IDS algorithms has been intensively studied in the past few years [23, 24, 25, 26]. In this dissertation, we propose an improved IDS algorithm to increase the convergence speed. We also propose a mixed IDS strategy

that adopts a different approach to improve the error performance of the algorithm compared to the adaptive mixed scheduling algorithms [23, 24, 25, 26].

With the advances in semiconductor technology, there have been many works to implement LDPC decoders in VLSI. The early stage of the study featured fully parallel LDPC decoders with the belief propagation (BP) algorithm [27, 28]. However, in order to reduce the implementation cost, most high-throughput LDPC decoders [29, 30, 31] usually employ partially parallel architectures with the min-sum (MS) algorithm, an approximate BP algorithm [32, 33]. Moreover, since LDPC codes were chosen in many communication standards, multi-rate LDPC decoders have been extensively studied [34, 35, 36]. Nevertheless, only little work has been conducted on the implementation of LDPC decoders with a large code length [37, 38, 39].

LDPC codes have been considered for error correction of NAND flash memory [40, 41, 42] because of severe performance degradation of recent NAND flash memory devices. The threshold voltage signal of high-density NAND flash memory contains a large amount of noise because of aggressive scaling down of memory cells, CCI, program-and-erase (PE) cycling, data retention, and MLC technology. Hard-decision error correction algorithms, such as BCH or RS, are no more sufficient for high-density NAND flash memory. In NAND flash memory, the read and write operations are performed by the unit of a page that has been recently increased to 8 kB. In addition, the empirical performance of LDPC codes converges to its expected behavior as the code length increases [43]. Therefore, LDPC codes with fairly large code lengths need to be studied for the application to NAND flash memory. Soft-decision decoding of LDPC codes shows much better error correcting performance than hard-decision decoding, however it demands multiple memory sensing operations. Multiple sensing operations and delivering soft-decision data obviously

increase the energy consumption of NAND flash memory. In this dissertation, we implement a high-rate LDPC decoding circuit for NAND flash memory and analyze the energy consumption of a NAND flash memory system with soft-decision LDPC decoding.

1.3 Outline of the Dissertation

This dissertation is organized as follows. Chapter 2 describes the background of LDPC codes, decoding algorithms, and message-passing schedules. In Chapter 3, two improved dynamic scheduling algorithms are proposed. The improved residual BP (RBP) algorithm that increases the convergence of the conventional RBP algorithm is proposed in Section 3.2, and mixed scheduling of two IDS algorithms is developed and presented in Section 3.3. Chapter 4 proposes a pipelined parallel architecture for decoding of finite geometry (FG) LDPC codes. The conditional variable node update algorithm that makes the conventional normalized *a posteriori* probability (APP) algorithm resilient to decoding failure is proposed in Section 4.2. The proposed decoder architecture and optimization strategies are described in Section 4.3. Section 4.4 provides the implementation results. Chapter 5 analyzes the energy consumption for read operation of NAND flash memory with soft-decision error correction. The performance of LDPC decoding for NAND flash memory is presented in Section 5.3. The energy consumption for NAND flash memory access and that of the LDPC decoder implemented in Chapter 4 are examined in Section 5.2 and Section 5.4, respectively. Section 5.5 optimizes the total energy consumption for accessing NAND flash memory with soft-decision error correction and proposes an LDPC decoder-assisted precision selection method. Finally, Chapter 6 concludes this dissertation.

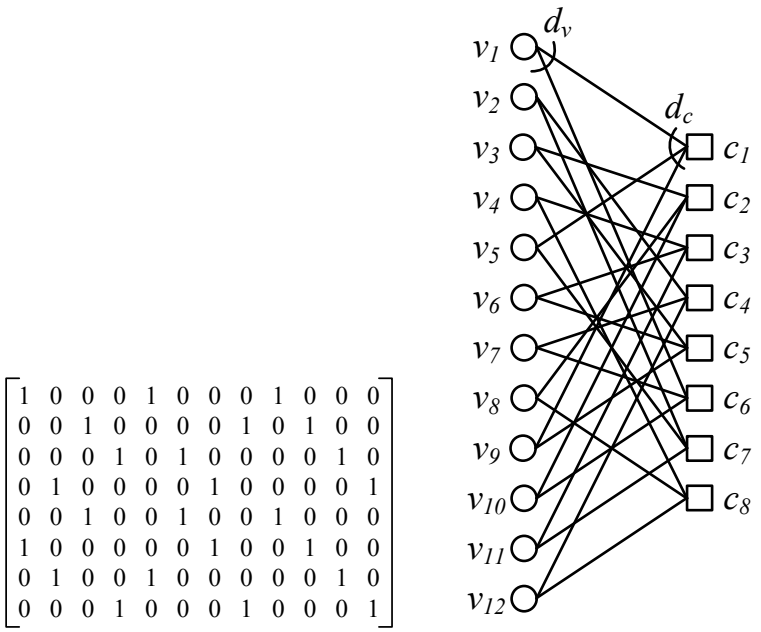
The material in this dissertation was presented in [44, 45, 46, 47, 48].

Chapter 2

LDPC Decoding and Scheduling Algorithms

2.1 Introduction

LDPC codes are linear codes with parity-check matrices having few non-zero elements, which allows low decoding complexity, and show good error performance when decoded with soft-decision information. A (d_v, d_c)-regular (N, K) LDPC code is defined by an $M \times N$ parity-check matrix $\boldsymbol{H}$ with the column weight d_v and the row weight d_c, where N and K denote the code length and the number of information bits, respectively, and $M \geq N - K$. Note that d_v and d_c are also referred to as the degree of a variable node and that of a check node, respectively. If $\boldsymbol{H}$ is full rank, $M = N - K$. Each column of the parity-check matrix corresponds to a codeword bit, and each row of the matrix represents a parity-check constraint that defines a code; i.e., $\sum_{n \in \mathcal{N}(m)} \bigoplus c_n = 0$ for m-th row, where $\bigoplus$ denotes the exclusive OR (XOR) operation. The number of parity bits, $N - K$, of an LDPC code corresponds to the rank of



$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) The parity-check matrix

(b) The bipartite graph

Figure 2.1: A (2, 3)-regular (12, 5) RS-LDPC code

the parity-check matrix of the code.

The LDPC code can be represented by a bipartite graph that has N variable nodes and M check nodes as well as edges. Each variable node corresponds to a codeword bit, or equivalently a column of the parity-check matrix, and each check node corresponds to a parity-check constraint, or equivalently a row of the matrix. Since there is an edge between the n-th variable node and the m-the check node if and only if $H_{m,n} = 1$, every variable (check) node is connected to d_v check (d_c variable) nodes. Note that for irregular LDPC codes, d_v or d_c are not constant. Figure 2.1 shows the parity-check matrix of a (2, 3)-regular (12, 5) RS-LDPC code and the corresponding bipartite graph. Every variable node has d_v ($=2$) neighboring check nodes and every check node are connected to d_c ($=3$) variable nodes. Since the parity-check matrix

contains one redundant row, $M \neq N - K$.

In the bipartite graph, each edge has a variable-to-check (VTC) and a check-to-variable (CTV) messages that can be represented as either probabilities or log-likelihood ratios (LLRs). However, in practice, it is more convenient to use LLRs [11, 12]. Each node receives messages from neighboring nodes, updates the outgoing messages, and propagates the messages back to its neighboring nodes. The message update rules for variable and check nodes are given by decoding algorithms, whereas the order of message updates is determined by scheduling algorithms. Therefore, decoding of LDPC codes can be configured in many ways according to decoding and scheduling algorithms. In the following section, decoding and scheduling algorithms are introduced.

2.2 Decoding Algorithms for LDPC Codes

This section contains a brief review of decoding algorithms for LDPC codes. The belief propagation (BP) algorithm that provides the best error performance is described in Section 2.2.1, and simplified BP algorithms that approximate the variable and check node update operations are explained in Section 2.2.2.

2.2.1 Belief Propagation Algorithm

The probabilistic decoding was devised by Gallager [11, 12] and later generalized by Tanner [49] and Wiberg [32], whereas the BP algorithm, also known as the sum-product algorithm [50], was first proposed by Pearl [51]. However, it turned out that the probabilistic decoding is a special version of the BP algorithm [52, 53, 50].

In order to describe the BP decoding algorithm, the following notations are first

introduced. Let $\boldsymbol{c} = \{c_1, c_2, \cdots, c_N\}$, $\boldsymbol{x} = \{x_1, x_2, \cdots, x_N\}$, and $\boldsymbol{y} = \{y_1, y_2, \cdots, y_N\}$ be an N-bit cordword, the transmitted bipolar sequence, and the corresponding received word, respectively, where $c_n \in \{0,1\}$ and $x_n \in \{\pm 1\}$. Let I_n be the channel LLR of the n-th received symbol, and let Z_n be the *a posteriori* LLR of the n-th variable node. Let L_{mn} denote a CTV message sent from the check node m to the variable node n. Similarly, let Z_{nm} be a VTC message sent from the variable node n to the check node m. Define $\mathcal{N}(m)$ and $\mathcal{M}(n)$ as the set of variable nodes connected to the check node m and that of check nodes connected to the variable node n, respectively. Then, the variable and check node update rules are given by

$$Z_{nm} = I_n + \sum_{m' \in \mathcal{M}(n) \setminus m} L_{m'n} \tag{2.1}$$

and

$$L_{mn} = \prod_{n' \in \mathcal{N}(m) \setminus n} \operatorname{sign}(Z_{n'm}) \times 2 \tanh^{-1} \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{|Z_{n'm}|}{2} \right) \right), \tag{2.2}$$

respectively, and the *a posteriori* LLR is computed by

$$Z_n = I_n + \sum_{m \in \mathcal{M}(n)} L_{mn}. \tag{2.3}$$

For a binary-input memoryless channel, the channel LLR is given by

$$I_n = \ln \frac{p(c_n = 0 | y_n)}{p(c_n = 1 | y_n)}. \tag{2.4}$$

If a codeword $\boldsymbol{c}$ is transmitted over an additive white Gaussian noise (AWGN) channel

with zero mean and variance σ^2,

$$I_n = \ln \frac{p(c_n = 0|y_n)}{p(c_n = 1|y_n)} = \begin{cases} -2y_n/\sigma^2, & x_n = 2c_n - 1 \\ 2y_n/\sigma^2, & x_n = (-1)^{c_n}. \end{cases} \tag{2.5}$$

Note that the channel LLR is also called the intrinsic information, whereas the VTC and CTV messages are also referred to as the extrinsic information. Note also that the check node update rule of Gallager's probabilistic decoding is

$$L_{mn} = \prod_{n' \in \mathcal{N}(m) \setminus n} \operatorname{sign}(Z_{n'm}) \times f\left(\sum_{n' \in \mathcal{N}(m) \setminus n} f(|Z_{n'm}|) \right), \tag{2.6}$$

where $f(x) = \ln(e^x + 1)/(e^x - 1)$.

2.2.2 Simplified Belief Propagation Algorithms

The variable and check node update rules of the BP decoding can be simplified using the *a posteriori* probability (APP) [33] and the min-sum (MS) algorithms [32], respectively. These simplified algorithms show degraded error performance when compared to the BP decoding but greatly reduce the implementation complexity, especially for LDPC codes with high node degrees. The Max-Log-MAP algorithm presented in [54] and the max-product algorithm [55] are equivalent to the MS algorithm, where the performance of the Max-Log-MAP and max-product algorithms were evaluated using the density evolution [43]. In addition, the reduced-complexity decoding algorithms based on the forward-backward algorithm [50] that operates in the LLR domain was proposed [56] and later elaborated [57] in which piecewise linear approximation of the check node update operation was also proposed.

Two modified versions of the MS algorithm improve the error performance using a normalization factor [58] or a single correction term [56, 59]. The MS algorithm with a normalization factor is called the normalized MS (NMS) algorithm, whereas that with a single correction term is referred to as the offset MS decoding (OMS) [60]. The modified MS algorithms incur only negligible degradation in error performance when compared to the BP algorithm. The normalized APP-based algorithm that combines the APP and NMS algorithms was also proposed to improve the error performance of the APP algorithm [61]. After that, the MS algorithms has been extensively studied in the last decade, which includes the λ-min algorithm [62], the MS with conditional correction [63], the MS with the degree-matched approximation [64], the adaptive OMS [65], the two-dimensional NMS [66], the transformed MS [67], the self-corrected MS [68], and the MS with two normalization factors [69].

The asymptotic performance of the modified MS algorithms were analyzed using DE in [59, 60, 70, 71], and the quantization effects of fixed-point arithmetic in the algorithms were studied in [63, 70, 72, 73, 74, 75].

In the following, the MS and its modified versions are described.

2.2.2.1 Min-Sum Algorithms

The MS algorithm and its two modified versions, the NMS and OMS algorithms, approximate the check node update rule of the BP decoding, while preserving the variable node update rule. In the MS algorithm, the core operation of the check node

update rule can be approximated as [76]

$$\begin{aligned} L(U \oplus V) &= \log \frac{1+e^{L(U)+L(V)}}{e^{L(U)}+e^{L(V)}} \\ &= \operatorname{sign}(L(U)) \operatorname{sign}(L(V)) \cdot \min(|L(U)|,|L(V)|) + s(L(U),L(V)) \\ &\approx \operatorname{sign}(L(U)) \operatorname{sign}(L(V)) \cdot \min(|L(U)|,|L(V)|), \end{aligned} \tag{2.7}$$

where U and V are statistically independent random variables, and the term $s(L(U),L(V)) = \log\left(1+e^{-|L(U)+L(V)|}\right) - \log\left(1+e^{-|L(U)-L(V)|}\right)$ is the correction factor [57]. Then, Eq. (2.2) can be written as

$$L_{mn} = \prod_{n' \in \mathcal{N}(m)\backslash n} \operatorname{sign}(Z_{n'm}) \cdot \min_{n' \in \mathcal{N}(m)\backslash n} |Z_{n'm}|. \tag{2.8}$$

The correction factor in Eq. (2.7) can be approximated to a fixed number $\beta > 0$, then the check node operation Eq. (2.2) becomes

$$L_{mn} = \prod_{n' \in \mathcal{N}(m)\backslash n} \operatorname{sign}(Z_{n'm}) \cdot \max\left(\min_{n' \in \mathcal{N}(m)\backslash n} |Z_{n'm}| - \beta, 0\right), \tag{2.9}$$

which corresponds to the check node update rule of the OMS algorithm. In the NMS algorithm, a scaling factor $\beta < 1$ is introduced to reduce the overestimated CTV messages in the MS algorithm, that is to say $|L(U \oplus V)| \leq \min(|L(U)|,|L(V)|)$ [72]. The check node update rule of the NMS algorithm is given by

$$L_{mn} = \prod_{n' \in \mathcal{N}(m)\backslash n} \operatorname{sign}(Z_{n'm}) \cdot \min_{n' \in \mathcal{N}(m)\backslash n} |Z_{n'm}| \cdot \alpha. \tag{2.10}$$

2.2.2.2 APP Algorithm

The APP algorithm simplifies the variable node operation by substituting the extrinsic outgoing messages from a variable node with the *a posteriori* LLR of the corresponding variable node [33], namely $Z_{nm} = Z_n \ \forall n, m \in \mathcal{M}(n)$, while maintaining the same check node update rule of the BP decoding. The APP algorithm not only reduces the computational complexity of the variable node operation but also saves the memory that stores the extrinsic information Z_{nm}. However, the correlation among the extrinsic outgoing messages significantly degrades the error performance compared to the BP and MS decoding algorithms [33, 72].

2.3 Message-Passing Schedules for Decoding of LDPC Codes

This section describes the scheduling algorithms that determine the order of message updates for decoding of LDPC codes. The scheduling algorithms are categorized into static and dynamic schedules; the former updates messages in a predetermined order, whereas the latter dynamically updates messages based on a specific metric such as the reliabilities or residuals of the messages propagated. Depending on the scheduling algorithm employed, LDPC decoding shows different performance and complexity characteristics. We start with the static scheduling algorithms that include flooding and the serial schedule.

2.3.1 Static Schedules

Flooding is the most well-known message-passing schedule that first updates all check nodes with VTC messages and then updates all variable nodes with intrinsic information and the CTV messages in every iteration, which is also known as the

two-phase message-passing algorithm. The BP decoding with the flooding schedule is formally described in Algorithm 1.

Algorithm 1 The BP decoding with flooding schedule

1: Initialize $k = 0$
2: Initialize all $Z_{nm}^{(-1)} = I_n$
3: **for** $m = 1$ to M **do**
4: **for** every $n \in \mathcal{N}(m)$ **do**
5:

$$L_{mn}^{(k)} = \prod_{n' \in \mathcal{N}(m) \setminus n} \operatorname{sign}\left(Z_{n'm}^{(k-1)}\right) \times 2\tanh^{-1}\left(\prod_{n' \in \mathcal{N}(m) \setminus n} \tanh\left(\frac{\left|Z_{n'm}^{(k-1)}\right|}{2}\right)\right)$$

6: **end for**
7: **end for**
8: **for** $n = 1$ to N **do**
9: **for** every $m \in \mathcal{M}(n)$ **do**
10: $Z_{nm}^{(k)} = I_n + \sum_{m' \in \mathcal{M}(n) \setminus m} L_{m'n}^{(k)}$
11: $Z_n^{(k)} = I_n + \sum_{m \in \mathcal{M}(n)} L_{mn}^{(k)}$
12: **end for**
13: **end for**
14: Decide a hard-decision vector $\hat{\boldsymbol{w}} = \{\hat{w}_1, \ldots, \hat{w}_N\}$ based on

$$\hat{w}_n = \begin{cases} 0, & \text{if } Z_n^{(k)} \geq 0 \\ 1, & \text{otherwise} \end{cases}$$

15: **if** $\boldsymbol{H}\hat{\boldsymbol{w}}^T = 0$ or the maximum iteration number is reached **then**
16: Output the hard-decision $\hat{\boldsymbol{w}}$
17: **else**
18: $k = k + 1$
19: Go to line 3;
20: **end if**

Another well-known static scheduling algorithm is the serial-C schedule [17] in which a check node is updated with its neighboring variable nodes first, and then the newly updated check node messages are immediately used to partially update the

variable nodes. The layered [18] and turbo decoding [19] algorithms are equivalent to the serial-C schedule. The serial-V schedule is a dual algorithm to the serial-C one, hence a variable node is updated first, and then the neighboring check nodes are updated. The shuffled iterative decoding [20] and the lazy scheduling [21] are equivalent to the serial-V schedule.

The serial schedule is advantageous in convergence speed and hardware implementation. Both the serial-C and serial-V schedules converge almost twice faster than the conventional flooding and save the memory for storing the VTC messages, Z_{nm}'s, because the VTC messages can be recovered from the APP and CTV messages, namely $Z_{nm} = Z_n - L_{mn}$. The BP decoding with the serial-C and that with the serial-V schedule are formally described in Algorithm 2 and 3, respectively, where P_m denotes the *check product* of the m-th check node and $\Delta L_{mn}^{(k)} = L_{mn}^{(k)} - L_{mn}^{(k-1)}$. Note that the BP decoding with the serial-V schedule needs additional memory for M check products.

2.3.2 Dynamic Schedules

This subsection introduces the dynamic scheduling algorithms based on *residuals*, which is called the informed dynamic scheduling (IDS) strategies. The IDS of the BP decoding was first proposed under the name of the residual BP (RBP) algorithm by Elidan *et al*. [77]. Then, Vila Casado *et al*. applied the algorithm to decoding of LDPC codes and also introduced the node-wise scheduling (NS) to relieve the problem caused by the greediness of the RBP [22, 23]. Since then, variants of IDS have been investigated [24, 25, 26, 78, 79, 80].

The RBP is an IDS scheme that schedules message updates according to the *residual* that is defined as the norm of the difference between the messages before and after an update [77]. Since all of the messages are represented as one-dimensional vari-

Algorithm 2 The BP decoding with the serial-C schedule

1: Initialize $k = 0$
2: Initialize all $Z_n = I_n$
3: Initialize all $L_{mn}^{(-1)} = 0$
4: **for** $m = 1$ to M **do**
5: **for** every $n \in \mathcal{N}(m)$ **do**
6: $Z_{nm}^{(k-1)} = Z_n - L_{mn}^{(k-1)}$
7: **end for**
8: **for** every $n \in \mathcal{N}(m)$ **do**
9:

$$L_{mn}^{(k)} = \prod_{n' \in \mathcal{N}(m) \backslash n} \operatorname{sign}\left(Z_{n'm}^{(k-1)}\right) \times 2 \tanh^{-1}\left(\prod_{n' \in \mathcal{N}(m) \backslash n} \tanh\left(\frac{\left|Z_{n'm}^{(k-1)}\right|}{2}\right)\right)$$

10: **end for**
11: **for** every $n \in \mathcal{N}(m)$ **do**
12: $Z_n = Z_{nm}^{(k-1)} + L_{mn}^{(k)}$
13: **end for**
14: **end for**
15: Decide a hard-decision vector $\hat{\boldsymbol{w}} = \{\hat{w}_1, \ldots, \hat{w}_N\}$ based on

$$\hat{w}_n = \begin{cases} 0, & \text{if } Z_n \geq 0 \\ 1, & \text{otherwise} \end{cases}$$

16: **if** $\boldsymbol{H}\hat{\boldsymbol{w}}^T = 0$ or the maximum iteration number is reached **then**
17: Output the hard-decision $\hat{\boldsymbol{w}}$
18: **else**
19: $k = k + 1$
20: Go to line 4;
21: **end if**

Algorithm 3 The BP decoding with the serial-V schedule

1: Initialize $k = 0$
2: Initialize all $Z_n^{(-1)} = I_n$
3: Initialize all $P_m = \prod_{n \in \mathcal{N}(m)} \tanh\left(\frac{Z_n^{(-1)}}{2}\right)$
4: **for** $n = 1$ to N, $\forall m \in \mathcal{M}(n)$ **do**
5: $\quad Z_{nm}^{(k-1)} = Z_n^{(k-1)} - L_{mn}^{(k-1)}$
6: $\quad L_{mn}^{(k)} = 2\tanh^{-1}\left(P_m / \tanh\left(\frac{Z_{nm}^{(k-1)}}{2}\right)\right)$
7: $\quad Z_n^{(k)} = Z_n^{(k-1)} + \sum_{m \in \mathcal{M}(n)} \Delta L_{mn}^{(k)}$
8: $\quad Z_{nm}^{(k)} = Z_n^{(k)} - L_{mn}^{(k)}$
9: $\quad P_m \leftarrow \tanh\left(\frac{L_{mn}^{(k)}}{2}\right) \cdot \tanh\left(\frac{Z_{nm}^{(k)}}{2}\right)$
10: **end for**
11: Decide a hard-decision vector $\hat{\boldsymbol{w}} = \{\hat{w}_1, \ldots, \hat{w}_N\}$ based on

$$\hat{w}_n = \begin{cases} 0, & \text{if } Z_n^{(k)} \geq 0 \\ 1, & \text{otherwise} \end{cases}$$

12: **if** $\boldsymbol{H}\hat{\boldsymbol{w}}^T = 0$ or the maximum iteration number is reached **then**
13: $\quad$ Output the hard-decision $\hat{\boldsymbol{w}}$
14: **else**
15: $\quad k = k + 1$
16: $\quad$ Go to line 4;
17: **end if**

ables in the LLR-BP decoding of LDPC codes, the residual is the absolute value of the difference of LLR values [22]. In particular, the RBP decoding presented in [22] and [23] considers only CTV messages when computing the residuals. Thus, the residual of the message L_{mn} propagated from the m-th check node to the n-th variable node is expressed as

$$r(L_{mn}) = \left| L_{mn}^{(k+1)} - L_{mn}^{(k)} \right|, \tag{2.11}$$

where $L_{mn}^{(k+1)}$ is computed based on the VTC messages, $Z_{nm}^{(k)}$'s. As the BP converges, all of the residuals become zero. Therefore, giving the priority of update to the message that has the largest residual can accelerate the decoding convergence [22]. The RBP is formally described in Algorithm 4, where the decoder checks the stopping rule when the number of message updates reaches the number of edges in the bipartite graph of an LDPC code [22, 23].

Algorithm 4 The residual BP

1: Initialize all $Z_{nm} = I_n$
2: Initialize all $L_{mn} = 0$
3: Compute all $r(L_{mn})$
4: Find $m, n = \underset{\substack{\forall m', 1 \le m' \le M \\ n' \in \mathcal{N}(m')}}{\arg\max}\ r(L_{m'n'})$
5: Generate and propagate L_{mn}
6: Set $r(L_{mn}) = 0$
7: **for** every $m' \in \mathcal{M}(n) \setminus m$ **do**
8: Generate and propagate $Z_{nm'}$
9: **for** every $n' \in \mathcal{N}(m') \setminus n$ **do**
10: Compute $r(L_{m'n'})$
11: **end for**
12: **end for**
13: **if** Stopping rule is not satisfied **then**
14: Go to line 4;
15: **end if**

Algorithm 5 The BP decoding with the node-wise scheduling

1: Initialize all $Z_{nm} = I_n$
2: Initialize all $L_{mn} = 0$
3: Compute all $r(L_{mn})$
4: Find $m = \underset{\substack{\forall m', 1 \le m' \le M \\ n' \in \mathcal{N}(m')}}{\arg\max}\ r(L_{m'n'})$
5: **for** every $n \in \mathcal{N}(m)$ **do**
6: Generate and propagate L_{mn}
7: Set $r(L_{mn}) = 0$
8: **for** every $m' \in \mathcal{M}(n) \setminus m$ **do**
9: Generate and propagate $Z_{nm'}$
10: **for** every $n' \in \mathcal{N}(m') \setminus n$ **do**
11: Compute $r(L_{m'n'})$
12: **end for**
13: **end for**
14: **end for**
15: **if** Stopping rule is not satisfied **then**
16: Go to line 4;
17: **end if**

The RBP shows the fastest convergence speed, thus exhibiting substantially better performance than the BP decoding with the flooding or serial schedules when the number of iterations is small. However, the RBP shows worse error performance for a large number of iterations due to the greediness of the RBP [22, 23]. In order to alleviate the negative effects caused by the greediness, Vila Casado *et al*. proposed the NS algorithm that propagates and generates $L_{mn'}$, $\forall n' \in \mathcal{N}(m)$ such that L_{mn} has the largest residual r^* [22, 23]. The NS algorithm is described in Algorithm 5.

The NS not only shows faster convergence speed than the BP with the flooding and serial schedules but also achieves better performance than the RBP and the BP decoding with static schedules when the number of iterations is large. This is because the NS can correct trapping set errors.

Chapter 3

Improved Dynamic Scheduling Algorithms for Decoding of LDPC Codes

3.1 Introduction

Since the length of an LDPC code is finite and the number of decoding iterations is limited, practical LDPC codes can hardly achieve the asymptotic performance predicted by density evolution [43]. In order to improve the error performance of LDPC codes, several researchers have studied message passing schedules rather than decoding algorithms themselves [17, 18, 19, 20, 21, 81, 82, 83, 84, 85]. Recently, Vila Casado *et al*. have proposed the informed dynamic scheduling (IDS) that determines the order of message passing based on the differences of messages generated in the previous and current updates, which is different from the static scheduling schemes, such as flooding or layered decoding, that update messages in a predetermined order,

as discussed in Chapter 2 [22, 23].

In particular, in order to improve the error performance of IDS, three mixed scheduling methods were proposed in [23, 24, 25]. The two-staged IDS algorithm [24] combines the residual belief propagation (RBP) and the node-wise scheduling (NS), whereas both the adaptive layered BP (LBP)/NS [23] and the adaptive mixed scheduling [25] are the combination of the LBP and the NS. Note that the LBP is equivalent to the serial-C schedule.

In this Chapter, we propose an improved RBP (iRBP) that accelerates the convergence speed of the RBP and also study a syndrome-based mixed scheduling of the iRBP and the NS. While the mixed scheduling strategies proposed in [23, 24, 25] consist of two stages in which the number of decoding iterations of the first stage is fixed [23] or adaptively determined [23, 24, 25], the proposed mixed scheduling performs either the iRBP or the NS according to the syndrome of the check node that propagates the message.

The rest of this Chapter is organized as follows. Section 3.2 explains the IDS of the BP algorithm and proposes the iRBP. Section 3.3 presents the syndrome-based mixed scheduling of the iRBP and the NS. The complexity analysis and the simulation results are provided in Section 3.4, and concluding remarks are given in Section 3.5.

3.2 Improved Residual Belief Propagation Algorithm

The RBP decoding can be considered a greedy algorithm because it finds and updates the message that has the largest residual at every message update. In the RBP decoding, however, different outgoing check-to-variable (CTV) messages from the

same check node can be selected and updated continuously within several message updates. For ease of description, we designate the check node that contains the CTV message with the largest residual as the *selected check* (SC). The output value of a check node is mainly determined by the minimum magnitude among the input messages to the check node as shown in Eq. (2.7). Thus, if two minimum magnitudes among the input to a check node are close, the outgoing messages from the check node have similar magnitudes. This can be more clearly explained by the min-sum approximation of the BP decoding whose check node operation is given by Eq. (2.8). When the incoming variable-to-check (VTC) messages to the SC satisfy the above condition, the residuals of outgoing messages from the SC can be similar in magnitude, which results in *continuous updates* of different CTV messages from the same SC within several message updates. As stated in [22] and [23], the RBP tends to give a high priority of update to the message propagated to the less reliable variable node. In particular, for check nodes that had not been updated up to the previous message update, the RBP always propagates the message to the least reliable variable node.[1] Then, due to the continuous update, the variable nodes with relatively higher reliability as well as those with lower ones are renewed.

The iRBP is proposed to avoid updating variable nodes with high reliability, which is implemented by forcing the residuals of the SC to zeros. In other words, assuming that the CTV message L_{mn} has the largest residual $r(L_{mn}) = r^*$, the proposed iRBP sets the residuals of all CTV messages $r(L_{mn'})$ to zeros as shown in line 19 of Algorithm 6, where $n' \in \mathcal{N}(m)$, while the RBP sets the residual of the tar-

[1] For the check nodes that had been updated at least once before the current message update, some residuals of the outgoing messages from the check node have been set to zeros unless they are updated by other check nodes. In this case, only the CTV messages having non-zero residuals are the candidates for the current message update.

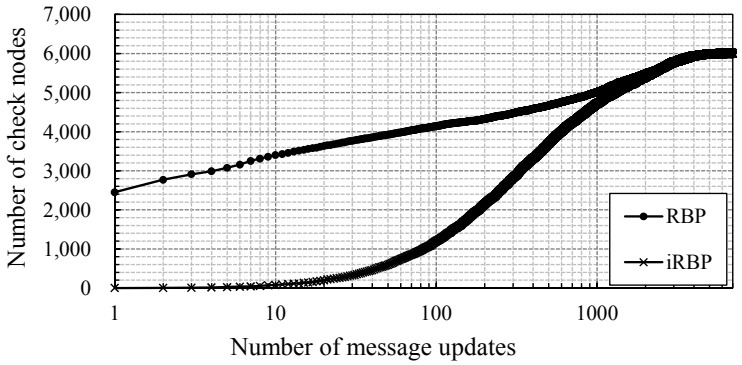


Figure 3.1: Cumulative number of the continuous updates for the rate-1/2 1944-bit LDPC code

get message $r(L_{mn})$ to zero after the propagation of the message L_{mn}. The proposed iRBP corresponds to line 18 to 25 of Algorithm 6.

Figure 3.1 illustrates an example of the cumulative number of continuous updates for the rate-1/2 1944-bit LDPC code defined in IEEE 802.11n standard [15]. The x-axis represents the number of message updates between continuous updates of CTV messages from the same SC, and it is plotted in log scale. The simulation was carried out for 6,966 message updates, which corresponds to one decoding iteration. For example, the value of 2,452 at the message update 1 for the RBP decoding represents that 2,452 check nodes are once again chosen as SCs right after their previous message updates, while the value of 2,771 at $x = 2$ indicates that 2,771 check nodes are selected again after one or two message updates, which includes the value of 2,452 at $x = 1$. Compared to the RBP decoding, the proposed iRBP shows a very small number of continuous updates within several message updates, which leads to the fastest convergence speed among various decoding schedules as demonstrated in Section 3.4. Note that the extreme case of the continuous update is the NS decoding that updates all CTV messages from an SC, thus resulting in slow decoding conver-

gence as already observed in [22] and [23].

3.3 Syndrome-Based Mixed Scheduling of iRBP and NS

When the RBP propagates the message L_{mn} having r^* from an unsatisfied check node m to a correct variable node n with low reliability, the sign of the variable node can be flipped because the propagated message tries to correct the variable node, which incurs an additional bit error [22, 23]. To solve this problem, we propose a mixed scheduling strategy that performs either the iRBP or the NS according to the syndrome. The proposed algorithm is different from the two-staged adaptive scheduling strategies that switch from the first to the second stage after a given number of iterations as introduced in [23, 24, 25]. The syndrome s_m of the m-th check node is defined as the modulo-2 sum of the hard-decision bits of the variable nodes connected to the m-th check node [86]. A check node is said to be satisfied if the syndrome of the check node is zero; otherwise, it is unsatisfied. The proposed mixed scheduling does not require much overhead because it uses the syndromes that are computed for the stopping rule check in the previous iteration rather than those that are generated in each message update. Thus, the mixed scheduling only needs an additional N-bit memory.

The proposed mixed strategy is performed as follows. In the first iteration, the decoder performs the iRBP to improve the convergence of the decoding. In the subsequent iterations, the decoder performs either the iRBP or the NS according to the syndrome. If an SC is unsatisfied, the decoder runs the NS to alleviate the negative effects caused by the greediness of the RBP, thereby improving the error-correcting performance. However, if an SC is satisfied, the decoder performs the iRBP because

Algorithm 6 Syndrome-based mixed scheduling

1: Initialize all $k = 0$
2: Initialize all $Z_{nm} = I_n$
3: Initialize all $L_{mn} = 0$
4: Compute all $r(L_{mn})$
5: Find $m,n = \underset{\substack{\forall m', 1 \le m' \le M \\ n' \in \mathcal{N}(m')}}{\arg\max}\ r(L_{m'n'})$
6: **if** $(k > 0)$ AND $(s_m = 1)$ **then**
7: **for** every $n \in \mathcal{N}(m)$ **do**
8: Generate and propagate L_{mn}
9: Set $r(L_{mn}) = 0$
10: **for** every $m' \in \mathcal{M}(n) \setminus m$ **do**
11: Generate and propagate $Z_{nm'}$
12: **for** every $n' \in \mathcal{N}(m') \setminus n$ **do**
13: Compute $r(L_{m'n'})$
14: **end for**
15: **end for**
16: **end for**
17: **else**
18: Generate and propagate L_{mn}
19: Set $r(L_{mn'}) = 0\ \forall n' \in \mathcal{N}(m)$
20: **for** every $m' \in \mathcal{M}(n) \setminus m$ **do**
21: Generate and propagate $Z_{nm'}$
22: **for** every $n'' \in \mathcal{N}(m') \setminus n$ **do**
23: Compute $r(L_{m'n''})$
24: **end for**
25: **end for**
26: **end if**
27: **if** Stopping rule is not satisfied **then**
28: k++
29: Go to line 4;
30: **end if**

the message propagated from a satisfied check node is less likely to generate a bit error than that from an unsatisfied one. The proposed mixed scheduling is formally described in Algorithm 6.

3.4 Complexity Analysis and Simulation Results

3.4.1 Complexity Analysis

This subsection analyzes the computational complexity of the proposed iRBP and syndrome-based mixed schedules as well as the conventional static schedules (the flooding the LBP) and two IDS strategies (the RBP and the NS). Let E be the number of edges in the bipartite graph of an LDPC code. Then, the flooding schedule updates E VTC and E CTV messages in every iteration as described in Section 2.3.1. In the LBP schedule, an iteration consists of M check node updates, each of which updates d_c CTV and d_c VTC messages. Therefore, the LBP schedule also updates $d_c \times M = E$ VTC and E CTV messages in every iteration.

In IDS schedules, an iteration is defined as E CTV message updates [22, 23]. Therefore, the stopping rule is checked when the number of CTV message updates reaches the number of edges in the graph, E. In the RBP decoding, a CTV message and $d_v - 1$ VTC messages are generated and propagated in each *message update*, where $(d_v - 1)(d_c - 1)$ residuals are also computed. Therefore, the RBP decoding updates $E(d_v - 1)$ VTC and E CTV messages and computes $E(d_v - 1)(d_c - 1)$ residuals in every iteration.

The NS decoding generates and propagates d_c CTV and $d_c(d_v - 1)$ VTC messages and computes $d_c(d_v - 1)(d_c - 1)$ residuals in each message update. Thus, an iteration consists of $E/d_c = M$ message updates in the NS decoding. Consequently, the NS

decoding updates $M \cdot d_c(d_v - 1) = E(d_v - 1)$ VTC and $M \cdot d_c = E$ CTV messages and computes $M \cdot d_c(d_v - 1)(d_c - 1) = E(d_v - 1)(d_c - 1)$ residuals, which is the same to that of the RBP decoding.

In the RBP and NS schedules, the residuals of the propagated CTV messages from an SC are set to zeros. Hence, both schedules require E set-to-zero operations, i.e., $r(\cdot) = 0$ operation. When compared to the RBP decoding, the proposed iRBP needs d_c set-to-zero operations in each message update, thus resulting in $E \cdot d_c$ set-to-zero operations per iteration. Note that the number of message updates and that of residual computations are the same to those of the RBP decoding.

Because the proposed mixed scheduling is based on the syndrome of an SC, the number of set-to-zero operations as well as that of the message updates depends on the ratio of satisfied, or unsatisfied, check nodes. Denoting the ratio of satisfied check nodes in each iteration as ρ_l, the mixed scheduling performs ρ_l iRBP and $1 - \rho_l$ NS decoding operations. Hence, the number of residual computations as well as that of message updates is the same to that of the iRBP or NS decoding, while the number of set-to-zero operations is computed as $(\rho_l(d_c - 1) + 1)E$, which is the sum of $\rho_l \cdot E \cdot d_c$, for the iRBP, and $(1 - \rho_l)E$, for the NS. Note that the number of set-to-zero operations for the mixed scheduling is smaller than or equal to that for the iRBP because $\rho_l \leq 1$.

The complexities of the above schedules are summarized in Table 3.1.

3.4.2 Simulation Results

In this subsection, we present the performance of the proposed iRBP and the syndrome-based mixed scheduling over the AWGN channel. The floating-point BP algorithm was employed, and the frame error rate (FER) performance was measured until at

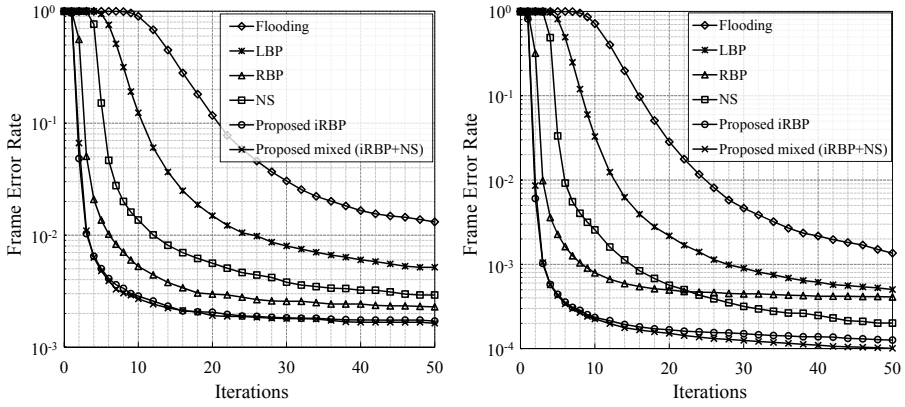
Table 3.1: Computational complexity of schedules

Schedules	**VTC updates**	**CTV updates**	**Residual computations**	$r(\cdot)=0$ **operations**
Flooding	E	E	-	-
LBP	E	E	-	-
RBP	$E(d_v-1)$	E	$E(d_v-1)(d_c-1)$	E
NS	$E(d_v-1)$	E	$E(d_v-1)(d_c-1)$	E
iRBP	$E(d_v-1)$	E	$E(d_v-1)(d_c-1)$	$E\cdot d_c$
Mixed	$E(d_v-1)$	E	$E(d_v-1)(d_c-1)$	$(\rho_l(d_c-1)+1)E$

least 100 frame errors were observed. The decoder checks the stopping rule when the number of message updates reaches the number of edges in the bipartite graph of an LDPC code [22, 23].

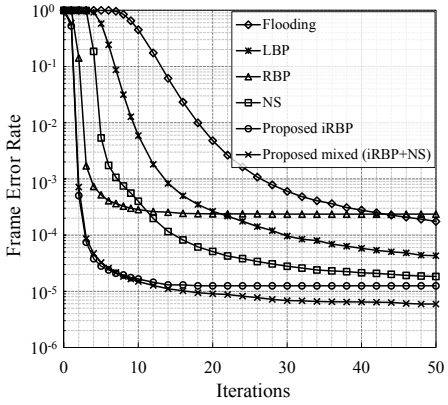
Figures 3.2, 3.3, and 3.4 show the FER performance of the rate-1/3 1920-bit [87], the rate-1/2 1944-bit [15], and the rate-3/4 1944-bit LDPC codes [15], respectively, with the flooding, the LBP, the RBP, the NS, the two-staged IDS [24], the adaptive mixed scheduling [25], the proposed iRBP, and the syndrome-based mixed schedule. The FER performance of the adaptive IDS [26], the quota-based RBP (QRBP) [80], and the silent-variable-node-free RBP (SVNF-RBP) [80] are also plotted in Fig. 3.3. The signal to noise ratios (SNRs) were set to $E_b/N_0 = 1.75\,\mathrm{dB}$ for Fig. 3.3 and $E_b/N_0 = 3.10\,\mathrm{dB}$ for Fig. 3.4. In order to validate the proposed syndrome-based mixed strategy, we also consider the mixed scheduling of the RBP and the NS as well as that of the iRBP and the NS in Fig. 3.3 and Fig. 3.4.

The proposed iRBP exhibits the fastest convergence speed among all of the scheduling schemes and converges about two times faster than the RBP decoding. Moreover, the proposed iRBP performs consistently better than the RBP decoding in terms of FER. This is because avoiding the continuous update not only improves the convergence speed at the early stage of decoding but also removes trapping sets and thereby



(a) For a E_b/N_0 of 1.50 dB

(b) For a E_b/N_0 of 1.75 dB



(c) For a E_b/N_0 of 2.00 dB

Figure 3.2: FER performance of the rate-1/3 1920-bit LDPC code

improves the error performance.

Furthermore, the proposed syndrome-based mixed scheduling of the iRBP and the NS outperforms all other scheduling schemes because the proposed method exploits not only the iRBP for fast convergence but also the iRBP and the NS for removing trapping sets. The performance gap between the proposed syndrome-based mixed scheduling and other schedules is more pronounced at high SNR as shown in

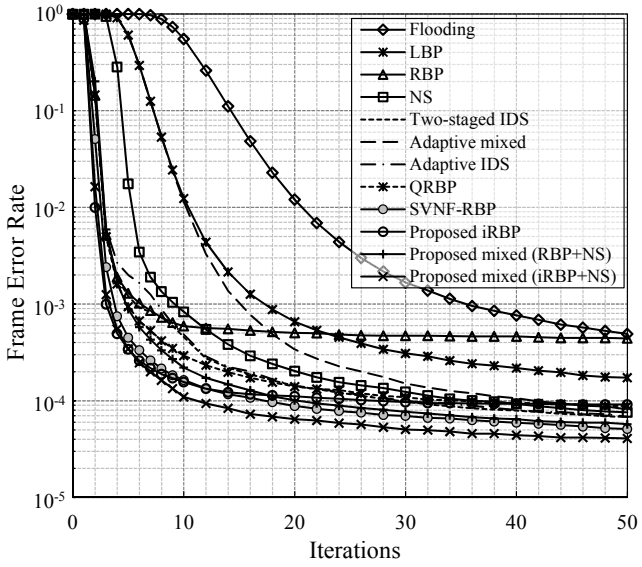


Figure 3.3: FER performance of the rate-1/2 1944-bit LDPC code for a E_b/N_0 of 1.75 dB

Fig. 3.2. We note that the proposed syndrome-based mixed scheduling also performs well when combined with the RBP, especially when compared with the two-staged IDS studied in [24], which is also a mixed strategy of the RBP and the NS.

Figures 3.5, 3.6, and 3.7 show the FER performance of the rate-1/3 1920-bit, the rate-1/2 1944-bit, and the rate-3/4 1944-bit LDPC codes, respectively, where the maximum iterations are set to 50 for Fig. 3.5, 10 for Fig. 3.6, and 15 for Fig. 3.7. The rate-1/2 1944-bit LDPC code was also simulated with the maximum iteration of 50 as shown in Fig. 3.8. From these figures we can find that the proposed syndrome-based mixed scheduling of the iRBP and the NS shows the best error performance when compared with the other scheduling schemes. In particular, the proposed mixed strategy suffers less from the error floor behavior.

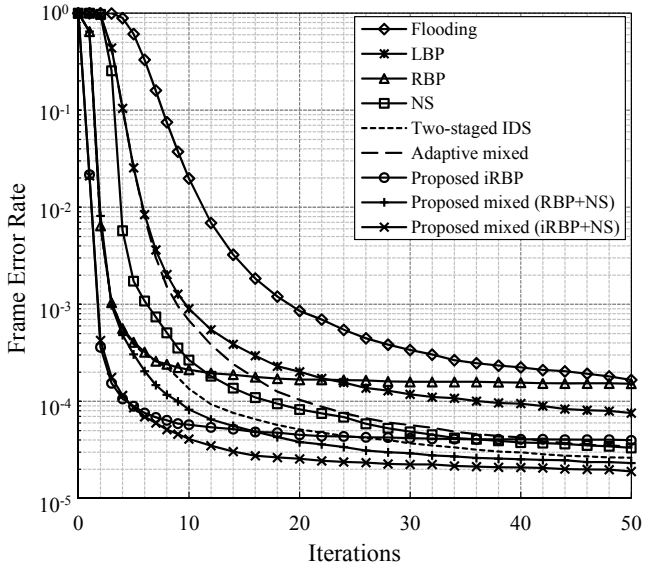


Figure 3.4: FER performance of the rate-3/4 1944-bit LDPC code, where E_b/N_0 is fixed to 3.10 dB

3.5 Concluding Remarks

In this chapter, we propose an improved residual belief propagation (iRBP) and a syndrome-based mixed scheduling scheme combining the iRBP and the node-wise scheduling (NS). The proposed iRBP forces the residuals of a *selected check* (SC) to zeros, which prevents updating reliable variable nodes and thus improves the convergence speed of the RBP by approximately two times. The proposed mixed scheduling performs either the iRBP or the NS based on the syndrome of the SC. Simulation results show that the proposed mixed-scheduling yields significant performance improvement when compared to all other scheduling schemes.

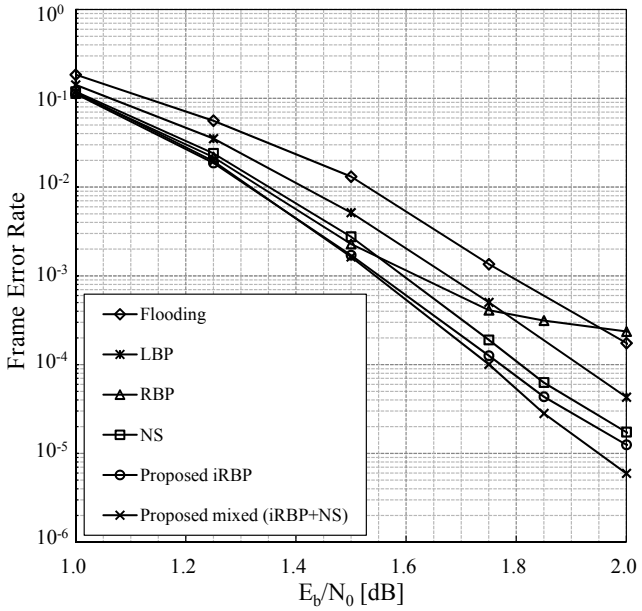


Figure 3.5: FER performance of the rate-1/3 1920-bit LDPC code with the maximum iteration number of 50

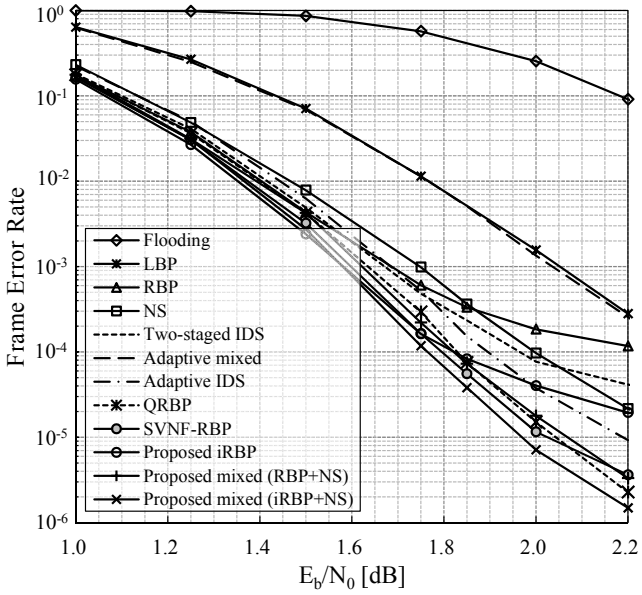


Figure 3.6: FER performance of the rate-1/2 1944-bit LDPC code for the maximum iteration number of 10

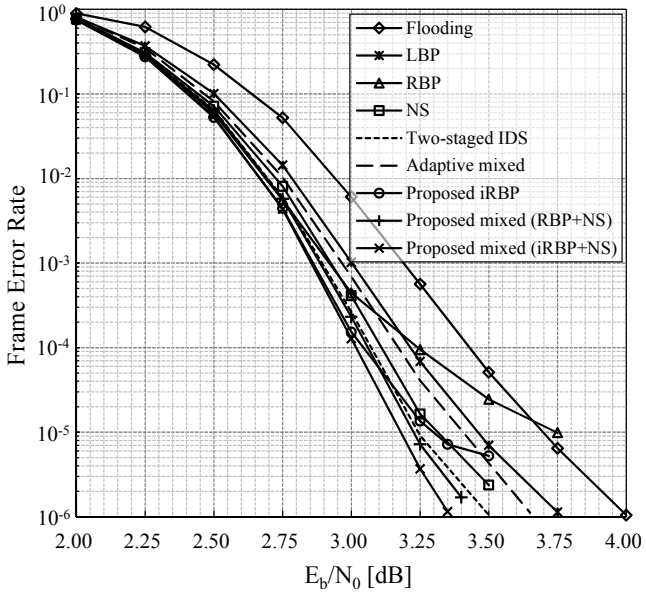


Figure 3.7: FER performance of the rate-3/4 1944-bit LDPC code over AWGN channel, where the maximum iteration number is set to 15

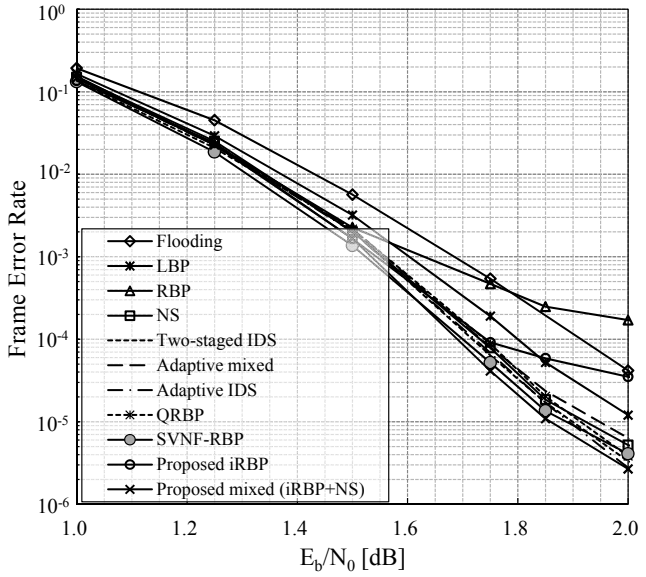


Figure 3.8: FER performance of the rate-1/2 1944-bit LDPC code for the maximum iteration number of 50

Chapter 4

A Pipelined Parallel Architecture for Decoding of Finite-Geometry LDPC Codes

4.1 Introduction

High-throughput LDPC decoders for NAND flash memory controllers need to be implemented in hardware. As hardware implementation of LDPC decoders is directly affected by decoding and scheduling algorithms as well as hardware architectures, low-complexity algorithms with fast convergence speed and their hardware implementation have been extensively studied in the literature [27, 29, 30, 31, 35, 88]. The early studies for the hardware implementation featured fully parallel LDPC decoders with the belief propagation (BP) algorithm [27, 28]. However, since fully parallel decoders demand very complex interconnection networks, most high-throughput LDPC decoders usually employ partially parallel architectures with the min-sum (MS) al-

gorithm or its variants and a serial scheduling algorithm [29, 30, 31, 88] to lower the interconnection complexity and increase the decoding throughput. Although the performance of LDPC codes converges to the thresholds predicted by the density evolution (DE) [43] as the code length increases, only little work has been devoted to the implementation of LDPC decoders with a large code length [37, 38, 39].

In this chapter, a pipelined parallel architecture with the serial-C schedule is proposed for finite geometry (FG) LDPC codes. The decoding hardware employs the normalized *a posteriori* probability (APP) based algorithm that not only shows good error-correcting performance for FG-LDPC codes but also simplifies both the variable and check node operations. A conditional variable node update scheme is proposed to make the normalized APP-based algorithm resilient to decoding failure as well as to reduce circuit switching activities in the node processing units. In order to increase the decoding throughput while minimizing the chip area, the decoder adopts pipelined parallel architecture and employs three memory size reduction techniques, which are optimizing the word-length of extrinsic information, compressing the extrinsic information, and approximating the second minimum magnitudes. The implementation results are given for a (69615, 66897) Euclidean geometry (EG) LDPC code.

The remainder of this chapter is organized as follows. Section 4.2 introduces the property of FG-LDPC codes and the proposed serial schedule of normalized APP-based algorithm with conditional node update. In Section 4.3, the decoder architecture and optimization strategies are presented. The implementation results are provided in Section 4.4. Section 4.5 concludes this chapter.

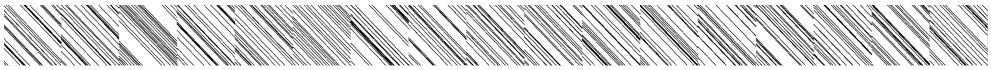
4.2 Finite-Geometry LDPC Codes and Conditional Variable Node Update Algorithm

4.2.1 Finite-Geometry LDPC codes

FG-LDPC codes are constructed based on Euclidean and projective geometries over finite fields. FG-LDPC codes have large minimum distances and perform well with iterative decoding algorithms [86]. They show fast convergence speed [86] and have no harmful trapping sets with the size smaller than their minimum weights, thus resulting in good error-floor performance [89]. It is reported that (1024, 781) EG-LDPC code has the error-floor below the bit error rate (BER) of 10^{-23} [90]. Moreover, the parity-check matrices of FG-LDPC codes contain redundant rows that give additional improvement in error performance [86, 72]. FG-LDPC codes have either cyclic or quasi-cyclic (QC) structure for their parity-check matrices, which allows efficient encoder implementation. The encoder can be implemented with linear feedback shift registers. However, since the row and column weights, d_c and d_v, respectively, are quite large when compared to other classes of LDPC codes, it is very challenging to implement a high-throughput FG-LDPC decoder.

Figure 4.1 shows the parity-check matrices of a rate-0.77 (1057, 813) projective geometry (PG) LDPC code and a rate-0.96 (69615, 66897) EG-LDPC code. The (1057, 813) PG-LDPC code has the cyclic parity-check matrix with $d_v = d_c = 33$. The parity-check matrix of the (69615, 66897) EG-LDPC code consists of 17 sub-matrices, and each sub-matrix is a cyclically right-shifted $4{,}095 \times 4{,}095$ matrix with the row and column weights of 16. As a result, the parity-check matrix has a $4{,}095 \times 69{,}615$ structure with the row weight d_c of 272 and the column weight d_v of

(a) A rate-0.77 (1057, 813) PG-code



(b) A rate-0.96 (69615, 66897) EG-LDPC code

Figure 4.1: Parity-check matrices

16.

A shortened EG-LDPC code is considered for the application to NAND flash memory as follows. Since the number of information bits contained in the (69615, 66897) EG-LDPC code is different from the page size of NAND flash memory devices, which is typically 8 kbytes, the (68254, 65536) shortened EG-LDPC code is constructed by removing 1,361 (= 66,897 - 65,536) data bits from the original EG-LDPC code as shown in Fig. 4.2, where the parameter γ represents the number of parity bits of the EG-LDPC code. In this work, we assume that a flash memory page consists of 8 kbytes of user data and 3,450 bits (5 %) of spare data. Among the spare bits, 2,718 bits are used for error correction and the remaining 732 bits are reserved for other purposes such as cell-to-cell interference (CCI) cancellation and the operation of flash translation layer. Implementing this shortening process is straightforward both for encoding and decoding. At the encoding process, 1,361 zeros are inserted in the information part of the (69615, 66897) EG-LDPC code. At the decoding process, this zero-filled bit positions are regarded to have zero received values with a large

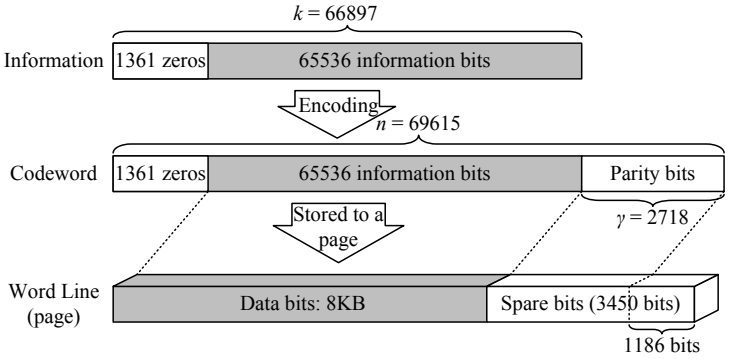


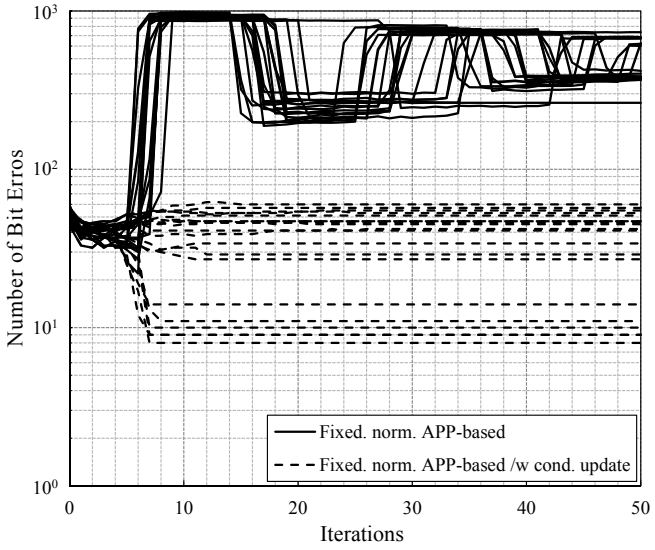
Figure 4.2: Codeword structure of the (68254, 65536) shortened EG-LDPC code and its mapping method

reliability.

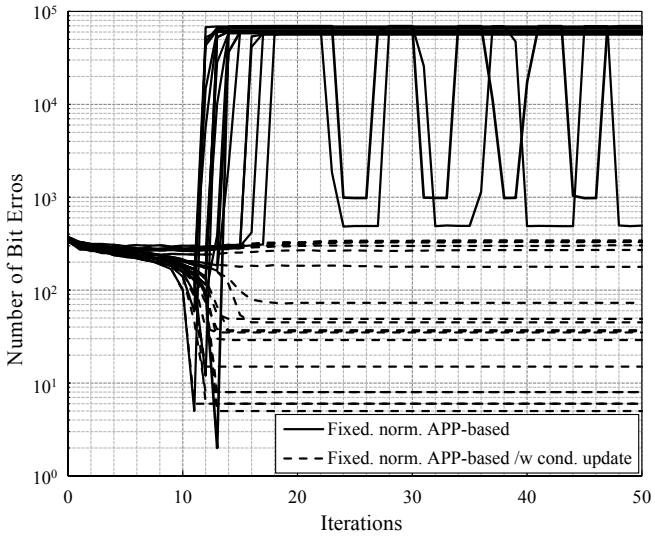
4.2.2 Conditional Variable Node Update Algorithm for Fixed-Point Normalized APP-Based Algorithm

Hardware-based implementation of FG-LDPC codes is considered very difficult because of their large row and column weights that demand a complex interconnection network. Also, when the code length is very large as shown in Fig. 4.1(b), a large memory size is required for the implementation. Thus, it is very needed to lower the implementation complexity of the codes, especially for high-throughput decoding with parallel architecture. For a reduced complexity implementation, the normalized APP-based algorithm [61], which performs well with FG-LDPC codes [72], is employed. This algorithm simplifies the variable node operations by substituting the extrinsic messages from a variable node with the *a posteriori* log-likelihood ratio (LLR) of the corresponding variable node [61].

The normalized APP-based algorithm, however, causes a large number of bit errors for undecodable blocks. Figure 4.3 shows the number of bit errors of the (1057,



(a) The rate-0.77 (1057, 813) PG-LDPC code



(b) The rate-0.96 (68254, 65536) EG-LDPC code

Figure 4.3: Number of bit errors of the two LDPC codes with the normalized APP-based algorithm for 20 undecodable blocks

813) PG-LDPC code and the (68254, 65536) EG-LDPC code for 20 undecodable blocks, where E_b/N_0 is set to 3.5 dB and 5.5 dB, respectively. As shown in Fig. 4.3, the APP-based decoding reduces the number of bit errors at the early stage but, rather, increases it as the iteration proceeds. This is caused by the correlation among propagated messages, which continues to increase the reliabilities (magnitudes) of uncorrected variable nodes and saturates some of them. The unstopped reliability increase of the uncorrected variable nodes also affects the correct ones and eventually leads to a large number of bit errors after several iterations. In order to solve this problem, we develop *a conditional variable node update algorithm* for the fixed-point normalized APP-based decoding. The proposed conditional node update algorithm finishes updating a variable node as soon as the reliability of the corresponding *a posteriori* LLR reaches the maximum fixed-point value. As a result, the proposed algorithm can prevent the reliability decrease of correct variable nodes that is caused by unstopped reliability increase of uncorrected variable nodes. Figure 4.3 shows the number of bit errors as the iteration proceeds for several decoding failed blocks, and it illustrates the effect of the conditional update.

Along with the reduced complexity decoding algorithm, the serial-C schedule that can halve the number of decoding iterations when compared to the conventional flooding schedule is applied to achieve a high throughput [17]. As discussed in Section 2.3, the serial schedule converges almost twice faster than the conventional flooding schedule.

The algorithm operates as follows. Consider a regular (N, K) LDPC code defined by an $M \times N$ parity-check matrix $\boldsymbol{H}$ with the row weight d_c and the column weight d_v. Let α be a normalization factor. Then, the serial schedule of the normalized APP-based algorithm with conditional node update is described in Algorithm 7.

Algorithm 7 Serial schedule of normalized APP-based algorithm with conditional node update

1: Initialize $k = 0$
2: Initialize all $L_{mn}^{(-1)} = 0$
3: Initialize all $Z_n = I_n$
4: **for** $m = 1$ to M **do**
5: **for** every $n \in \mathcal{N}(m)$ **do**
6:

$$L_{mn}^{(k)} = \prod_{n' \in \mathcal{N}(m) \backslash n} \operatorname{sign}(Z_{n'}) \cdot \min_{n' \in \mathcal{N}(m) \backslash n} |Z_{n'}| \cdot \alpha \tag{4.1}$$

7: **end for**
8: **for** every $n \in \mathcal{N}(m)$ **do**
9:

$$Z_n = \begin{cases} Z_n, & \text{if } |Z_n| = 2^{q-1} - 1 \\ Z_n + L_{mn}^{(k)} - L_{mn}^{(k-1)}, & \text{otherwise} \end{cases} \tag{4.2}$$

10: **end for**
11: **end for**
12: Decide a hard-decision vector $\hat{\boldsymbol{w}} = \{\hat{w}_1, \ldots, \hat{w}_N\}$ based on

$$\hat{w}_n = \begin{cases} 0, & \text{if } Z_n^{(k)} \geq 0 \\ 1, & \text{otherwise} \end{cases}$$

13: **if** $\boldsymbol{H}\hat{\boldsymbol{w}}^T = 0$ or the maximum iteration number is reached **then**
14: Output the hard-decision $\hat{\boldsymbol{w}}$
15: **else**
16: $k = k + 1$
17: Go to line 4;
18: **end if**

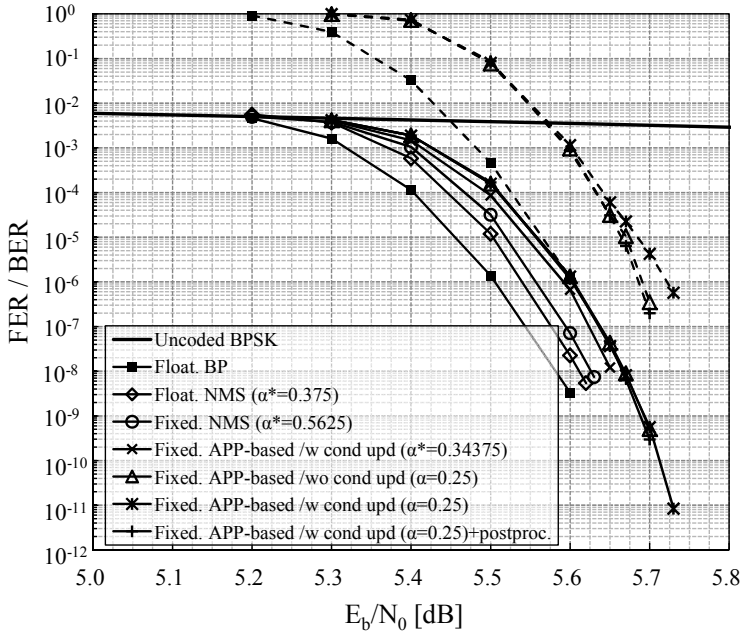


Figure 4.4: (Dashed line) Frame- and (solid line) bit-error performance of the (68254, 65536) shortened EG-LDPC code with the serial-C schedule

Figure 4.4 shows the error performance of the (68254, 65536) shortened EG-LDPC code with various decoding algorithms over the additive white Gaussian noise (AWGN) channel, which include floating-point BP, floating-point normalized MS (NMS), fixed-point NMS, and fixed-point normalized APP-based algorithm with and without conditional node update. The serial-C schedule is employed, and the maximum iteration number is set to eight. The normalization factors of the floating-point NMS, fixed-point NMS, and fixed-point APP-based algorithms are set to 0.375, 0.5625, and 0.34375, respectively, which yield the best error performance. The performances of the fixed-point normalized APP-based algorithms with the normalization factor of 0.25, which leads to simple hardware, are also shown. All of the fixed-point simulations employ the word-length, q, of seven, which includes one bit for

the sign, four bits for the integer, and two bits for the fractional part [91]. The BP algorithm shows the best error-correcting performance, and the gap between the BP and the floating-point NMS algorithms is about 0.033 dB at the BER of 10^{-7}. However, because of the quantization noise, the fixed-point NMS decoding shows slight performance degradation of 0.052 dB compared to the BP algorithm. The fixed-point normalized APP-based algorithms with and without conditional node update show good error performance that is close to the NMS decoding, when the normalization factors of 0.34375 or 0.25 are used. Note that the error performance degradation due to the conditional node update in the normalized APP algorithm is less than 0.01 dB at the frame error rate (FER) of 10^{-4}. This is because the variable node cannot be corrected under the conditional node update scheme when an incorrect variable node is saturated to the maximum fixed-point value, $\pm 2^{q-1} - 1$. However, in high signal-to-noise ratio (SNR) region, only a small number of variable nodes remain unchanged and the FER increase is very minor. When the SNR is 5.7 dB, 92 % of erroneous frames contain only one uncorrected bit error in each frame. Thus, if needed, we can remove these remaining bit errors by applying the LDPC decoding without the conditional node update again on the decoded data output (consult the "+" marked curves in Fig. 4.4, where one decoding iteration without the conditional node update is applied after finishing seven decoding iterations with the conditional update).

The proposed conditional variable node update algorithm can be compared with the reduced computational complexity algorithms proposed in [92] and [93]. Both papers have proposed similar algorithms that stop updating reliable variable nodes using a pre-defined [92] or a dynamic thresholds [93]. Although their complexity reduction was reported to be up to 60 % and 35 % in [92] and [93], respectively, some error performance degradation was observed due to erroneously identified variable

nodes. Furthermore, these algorithms require extensive simulations to determine the optimum threshold, and their error performance and computational complexity reduction are very sensitive to the threshold. In contrast to those methods, the proposed conditional node update does not require a precise threshold, and the error performance degradation is almost negligible.

4.3 Decoder Architecture

In this section, we discuss the architecture optimization of the decoder for efficient VLSI implementation. As the serial schedule increases the path delay when compared to the conventional flooding schedule, it is essential to employ the pipelining technique. In addition, since the application demands a very high throughput, it is also needed to adopt the parallel architecture. We first present the baseline architecture that sequentially executes the developed algorithm, and then propose a pipelined-parallel architecture for increasing the throughput. The memory requirement is also greatly reduced by optimizing the word-length and minimizing the number of internal variables. Although the architecture is given for the (69615, 66897) EG-LDPC code, it is applicable to FG-LDPC codes.

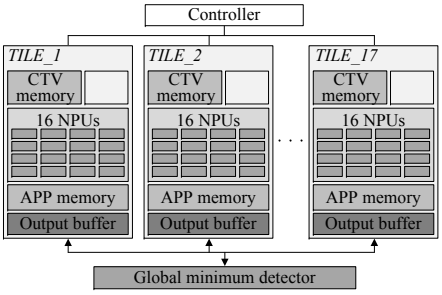
4.3.1 Baseline Sequential Architecture

Figure 4.5(a) shows the overall baseline architecture that contains 17 tiles, a global minimum detector, and control logic. Each tile, shown in Fig. 4.5(c), conducts the operations assigned to each sub-matrix of the parity-check matrix. The exclusive OR (XOR) gate tree that computes the overall sign is omitted for clarity. Each tile contains 16 node processing units (NPUs), an APP memory block, a check-to-variable

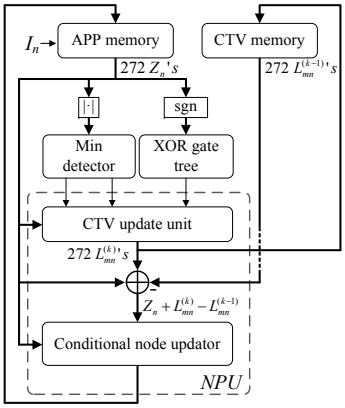
(CTV) memory block, an output buffer, and a local minimum detector. The APP memory consists of 4,095 $\times$ q-bit shift registers for storing *a posteriori* LLRs. The CTV memory, which consists of a 4 k $\times$ 16 $\times$ q-bit dual-port SRAM block, keeps 4,095 $\times$ 16 CTV messages. We use 7 bits for the word-length, q, as explained in Section 4.2.2. The capacity of the CTV memory is 7.44 Mbits, which results from 4 k checks $\times$ 272 CTVs/check $\times$ 7 bits/CTV.

The overall dataflow of the decoder is shown in Fig. 4.5(b). At the initialization phase, the channel LLRs obtained by reading the flash memory with multiple sensing reference voltage are transferred to the APP memory, while the CTV memory is initialized to zero. The initialization phase takes 4,095 clock cycles, which is determined by the number of rows in the parity-check matrix, M. Because we consider the (68254, 65536) shortened EG-LDPC code and seven bits for the channel LLR, the channel LLR of the shortened bit position is initialized to 63 according to Eq. (2.5), where $x_n = 2c_n - 1$ is used, which corresponds to a zero received value with a large reliability as described in Section 4.2.1. After the initialization, the local minimum detectors find the two smallest magnitudes among 16 *a posteriori* LLRs in each tile, whereas the global minimum detector selects the two smallest values among the output of the local detectors and provides the result to all the tiles. Then, each NPU updates the *a posteriori* LLR. Finally, the newly updated *a posteriori* LLRs and CTV messages are written back to the APP and CTV memories, respectively. Since a check node and its neighboring variable nodes are updated at each clock cycle and M is 4,095, it takes 4,095 clock cycles to complete one iteration. At the end of each iteration, the sign bits of the current *a posteriori* LLRs are stored in the output buffer.

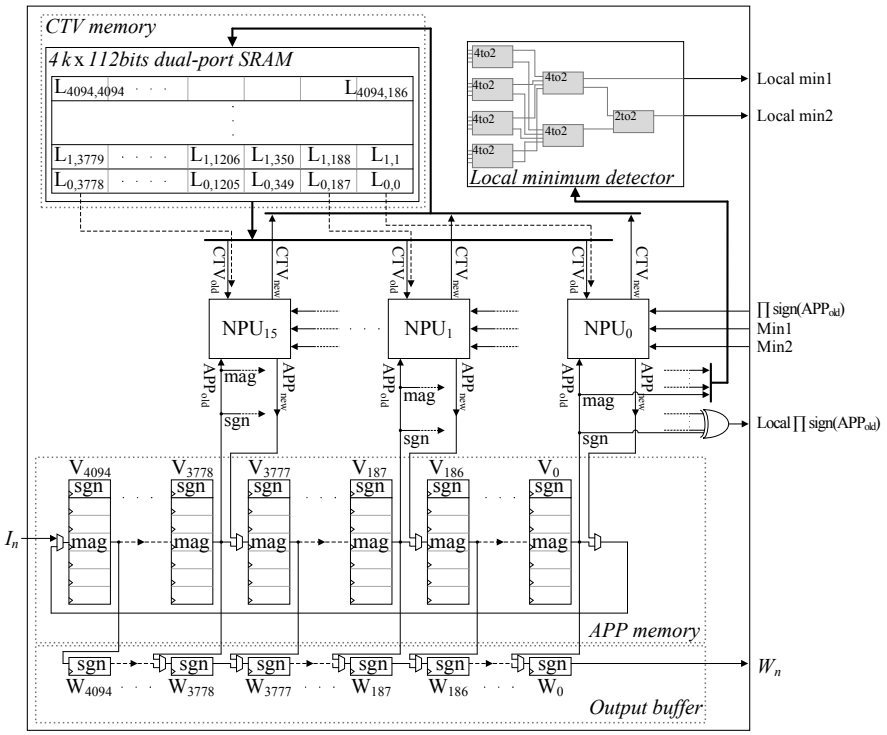
Because the parity-check matrix has the QC property and the row weight of each sub-matrix is quite large, the APP memory is implemented using shift registers in-



(a) Overall architecture



(b) Overall dataflow of the proposed decoder



(c) Structure of a tile

Figure 4.5: Baseline architecture of the proposed LDPC decoder

stead of SRAM blocks. As a result, the interconnection lines between the shift registers and NPUs are fixed, thus the decoder does not need a permutation network. In other words, the interconnection in the i-th tile is determined by the first row of the i-th sub-matrix. As SRAM-based storage demands smaller chip area than that with shift registers, many partially parallel LDPC decoders for QC-LDPC codes whose sub-matrix is a permuted identity matrix or a zero matrix employ SRAM-based architecture for the APP memory [34, 35], where only one message is read from a memory block at each cycle. However, the developed (69615, 66897) EG-LDPC code has the sub-matrix row weight of 16, which requires a large number of small SRAM blocks to increase the memory bandwidth and renders placement and routing for VLSI design very difficult. Furthermore, even if we implement the APP memory with d_c SRAM segments such as in [29], complex switching networks are required to resolve memory access conflicts.

The structure of an NPU is shown in Fig. 4.6. The *min selector* chooses the magnitude of a CTV message between the minimum and the second minimum magnitudes, and the result is right-shifted by two bits for normalization with $\alpha\,(=0.25)$. The subtractor and the following adder compute the *a posteriori* LLR, and the quantizer saturates the output to prevent overflows. Finally, the *conditional updater* selects the *a posteriori* LLR according to Eq. (4.2), and then the newly updated *a posteriori* LLR is formatted to the 7-bit sign-magnitude representation to be stored to the APP memory.

The data in the APP memory employs the sign-magnitude format to save the power consumption. As the APP memory consists of shift registers, there are many switching activities when the input signal varies much. Because EG-LDPC codes converge fast, most *a posteriori* LLRs are quickly saturated within a few iterations,

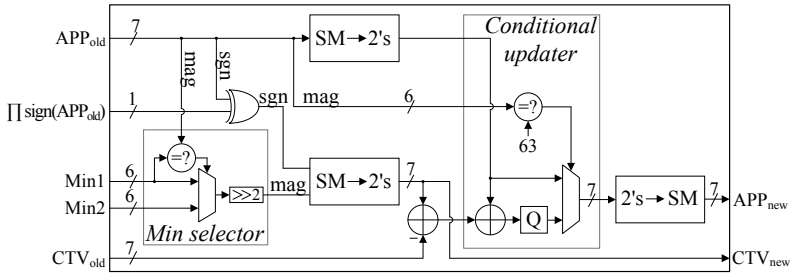


Figure 4.6: Structure of a node processing unit

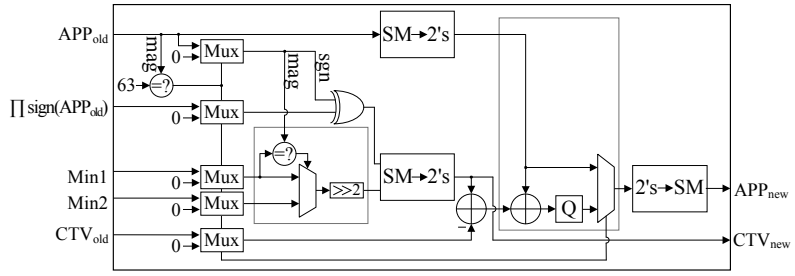
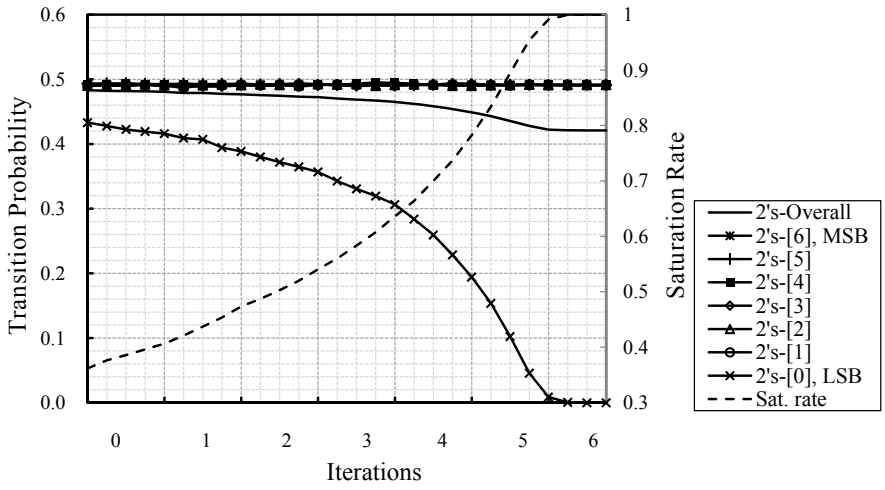
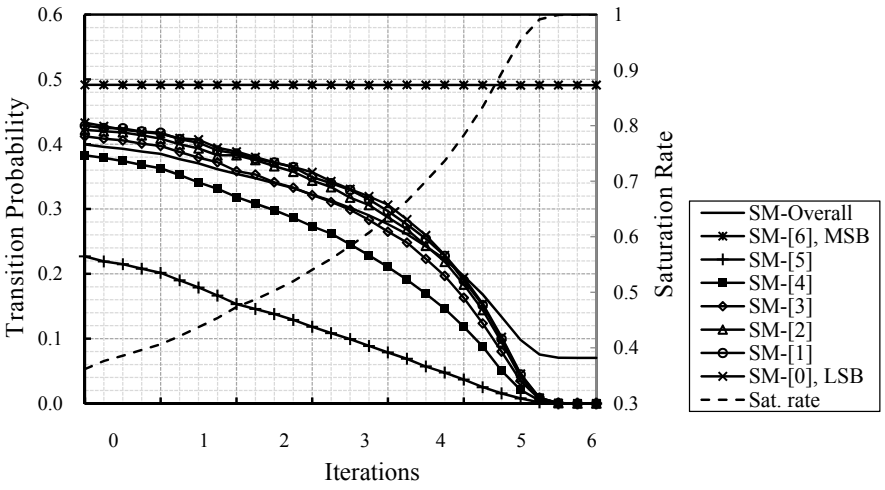


Figure 4.7: Structure of a modified node processing unit

which increases the probability that the input to a shift register has the saturated value. In addition, once the *a posteriori* LLR is saturated to either $2^{q-1}-1$ or $-2^{q-1}+1$ ($+63$ and -63, respectively, for $q=7$), the proposed conditional node update scheme keeps its magnitude unchanged. Figure 4.7 shows the modified node processing unit that employs the conditional node update. Five multiplexors are added to force all the input to have zero values when the corresponding *a posteriori* LLR is saturated. If the LLRs for all the nodes are saturated, the magnitude of them is always $2^{q-1}-1$ and only the sign bit changes with the sign-magnitude format when the input for successive nodes are applied to the APP memory. For example, when the current and the next input values are $+63$ and -63, respectively, the number of bit transitions for the two's complement format is six, while only the sign bit changes for the



(a) With two's complement format



(b) With sign-magnitude format

Figure 4.8: Saturation rate of *a posteriori* LLRs and bit transition probabilities at the SNR of 5.5 dB

sign-magnitude format. Figure 4.8 shows the saturation rate of the *a posteriori* LLRs (dotted line) as well as the transition probabilities for the two's complement and sign-magnitude formats. The lines with markers represent the transition probabilities of bit

positions, whereas the solid lines show the overall transition probability at the shift registers. Test vector 1 in the Table 4.1 was used for this simulation. The transition probabilities of bits with the two's complement format remain approximately 50 %, while those with the sign-magnitude format decrease as the decoding proceeds and are inversely proportional to the saturation rate except the sign bit. As the LLR saturates, the circuit operation with the sign-magnitude format mostly changes only the sign and consumes much less dynamic power.

We compare the power consumption of the proposed decoder with that of our prior work [45] in Table 4.1. In our earlier report, the two's complement format was used for storing the *a posteriori* LLRs, and the conditional node updater and the multiplexors were not employed in the node processor. For this comparison, we also implemented the decoder of our earlier work [45] and estimated the power consumption from the post-layout simulation result. To improve the accuracy of power estimation, randomly generated information bits were encoded and Gaussian noise was added for preparing the test vectors. From Table 4.1, it is clear that the sign-magnitude format reduces the internal and switching power of the APP memory by up to 57 and 50 %, respectively. The power consumed at the combinational logic is also reduced by up to 54 % due to the modified node processor. Finally, although the clock network consumes the largest portion (around 50 %) of the total power consumption in the chip, the combination of the sign-magnitude format and the conditional node update scheme efficiently reduces the total power consumption by more than 1 W or 24 % for all cases.

In the conventional decoders employing the serial schedule, such as [45], [31], and [88], CTV messages are updated using variable-to-check (VTC) messages. Hence,

Table 4.1: Comparison of power consumption (in W)

		Register				Clock network	Combinational logic	etc	Total
		Internal	Switching	Leakage	Sub total				
Test vector 1	Proposed (SM)	0.3118	0.0979	0.0065	0.4161	1.9321	0.5856	0.3247	3.2585
	[45] (2's compl.)	0.5794	0.1561	0.0072	0.7427	2.0923	1.0494	0.4068	4.2912
Test vector 2	Proposed (SM)	0.3144	0.0988	0.0065	0.4197	1.9511	0.6122	0.3553	3.3383
	[45] (2's compl.)	0.6097	0.1641	0.0072	0.7809	2.1194	1.1088	0.4224	4.4315
Test vector 3	Proposed (SM)	0.3016	0.0951	0.0065	0.4032	1.9660	0.5869	0.3304	3.2865
	[45] (2's compl.)	0.6339	0.1705	0.0072	0.8116	2.1400	1.1706	0.4344	4.5566
Test vector 4	Proposed (SM)	0.3254	0.1021	0.0065	0.4340	2.0840	0.6503	0.3822	3.5505
	[45] (2's compl.)	0.7548	0.2033	0.0072	0.9653	2.2944	1.4160	0.4978	5.1735
Test vector 5	Proposed (SM)	0.2660	0.0846	0.0064	0.3570	1.9067	0.5020	0.2880	3.0537
	[45] (2's compl.)	0.5303	0.1424	0.0072	0.6799	2.0555	0.9369	0.3854	4.0577

the front-end of the NPUs needs to contain a circuit that recovers VTC messages from the *a posteriori* LLRs and CTV messages as well as a logic that converts the data format of VTC messages from the two's complement to the sign-magnitude format. On the other hand, the proposed APP-based algorithm does not need these circuits at the front-end of the NPU because the *a posteriori* LLRs are used to compute CTV messages and the sign-magnitude data format is employed. This simplification reduces the critical path delay. When compared to our earlier work in [45], the critical path delay is reduced from 22.7 to 16.3 ns, and hence the number of pipeline stages can be reduced from six to five.

4.3.2 Pipelined-Parallel Architecture

In the serial schedule, a check node is updated first, and then the variable nodes connected to the renewed check node are changed using the newly modified CTV messages, which results in a long critical path. As the critical path delay of the baseline architecture is 16.3 ns, the maximum clock frequency is limited to around 59 MHz unless pipelining technique is employed. In this case, the minimum decoding throughput is 106 Mb/s. In order to reduce the critical path delay, four pipeline registers are inserted in the node processing unit and the minimum detectors as shown in Fig. 4.9. Figure 4.10 compares the cell area, the critical path delay, and the minimum throughput of the four decoders: 1) the baseline, 2) the pipelined, 3) the pipelined-parallel, and 4) the proposed decoders (i.e., the pipelined-parallel architecture with the three memory reduction techniques), where L_p and N_p stand for the level of parallelism and the number of pipeline stages, respectively. The pipelined architecture reduces the critical path delay from 16.3 to 4.4 ns with only 0.8 % area overhead.

The pipelining incurs hazards when two 1's appear contiguously in the parity-

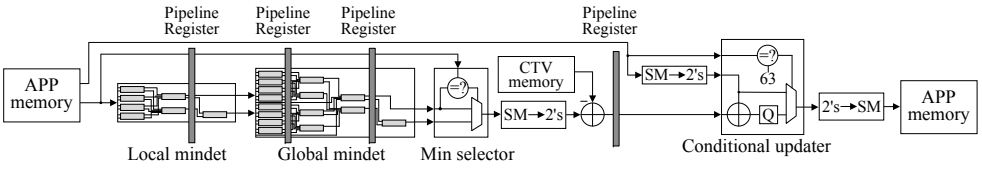


Figure 4.9: Critical path splitting through pipelining

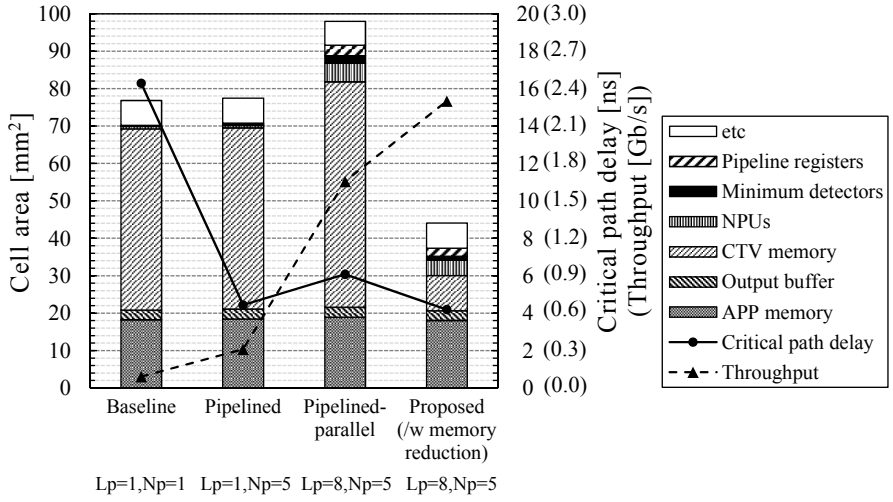


Figure 4.10: Comparison of the cell area, the critical path delay, and the minimum decoding throughput of the four decoders that were synthesized in 0.13-μm CMOS technology

check matrix. Due to the pipelining delay in the update of variable nodes, we can only read old variable node values (*a posteriori* LLRs) in this situation. However, in the proposed five-stage pipelined eight-way parallel decoder, only 1.5 % of variable nodes are affected by the delayed update. We observe no error performance degradation due to this pipelining hazards.

The pipelined architecture is then parallelized to increase the decoding throughput. An L_p-parallel decoder processes L_p check nodes in parallel. Hence, there are L_p global minimum detectors, and every tile has L_p times more NPUs and local min-

imum detectors when compared to the pipelined architecture, which increases the corresponding cell areas by L_p times as shown in Fig. 4.10. In order to deliver L_p times more *a posteriori* LLRs to NPUs, the shift registers in the APP memory are grouped into L_p segments, each of which has $\lceil M/L_p \rceil \times q$ bit registers [94]. Note that the parallel architecture does not demand more capacity for the CTV memory but needs increased bandwidth when compared to the baseline architecture. For this purpose, the CTV memory is also divided into L_p blocks, and the size of each block is reduced from 4 k $(=M)$ to $\lceil M/L_p \rceil$. In this work, the level of parallelism is set to eight, thus each tile contains eight 512×112-bit dual-port SRAM blocks. Although the capacity of the CTV memory remains unchanged, dividing the memory increases the corresponding cell area by 25 % compared to the pipelined decoder as shown in Fig. 4.10. Note also that the critical path delay was increased from 4.4 to 6.1 ns because the parallelization increases the capacitance of the decoding circuit.

In the parallel decoder architecture, the same *a posteriori* LLR can be used to update different check nodes simultaneously. In this case, the *a posteriori* LLR should be partially updated using all the participating check nodes. Thus, Eq. (4.2) is modified as follows:

$$Z_n = \begin{cases} Z_n, & \text{if } |Z_n| = 2^{q-1} - 1 \\ Z_n + \sum_{m' \in \mathcal{M}_c(n)} \Delta L^{(k)}_{m'n}, & \text{otherwise,} \end{cases} \tag{4.3}$$

where $\Delta L^{(k)}_{m'n} = L^{(k)}_{m'n} - L^{(k-1)}_{m'n}$ and $\mathcal{M}_c(n)$ is defined as the set of check nodes that participate in the n-th *a posteriori* LLR *in the current sub-iteration*. Note that Z_n is quantized after the summation is conducted. Therefore, the word-length for the inter-

mediate value of Z_n should be increased appropriately in order to prevent overflows.

As it takes M clock cycles for the baseline architecture to conduct one iteration for a code block, the throughput, T, of the pipelined LDPC decoder can be represented as

$$T = \frac{N \cdot f_{clk}}{\lceil M/L_p \rceil \cdot (I_t + 2)}, \tag{4.4}$$

where N, M, f_{clk}, L_p, and I_t are the code length, the number of rows in the parity-check matrix, the clock frequency, the level of parallelism, and the maximum iteration number, respectively. For the baseline architecture, the level of parallelism L_p is set to one. Note that $I_t + 2$ is used instead of I_t due to the initialization and the parity-check phases. With the five-stage pipelined eight-way parallel architecture and the maximum iteration limit of 8, we can achieve the minimum throughput of 1.6 Gb/s (200 MB/s) which corresponds to the speed of the ONFI (Open NAND Flash Interface) 2.1 [95]. Here, we use the clock frequency of 125 MHz considering the post-layout delay.

4.3.3 Memory Capacity Reduction

In order to reduce the capacity of the CTV memory, three memory reduction techniques are applied: word-length optimization of CTV messages, compression of CTV messages, and approximation of the second minimum magnitudes.

The word-length of CTV messages is reduced by changing the order of computation. In the proposed algorithm, the magnitude computation of a CTV is to find the two minimum magnitudes among d_c *a posteriori* LLRs, and then to multiply the normalization factor α to the minimum values. However, this process is identical to the

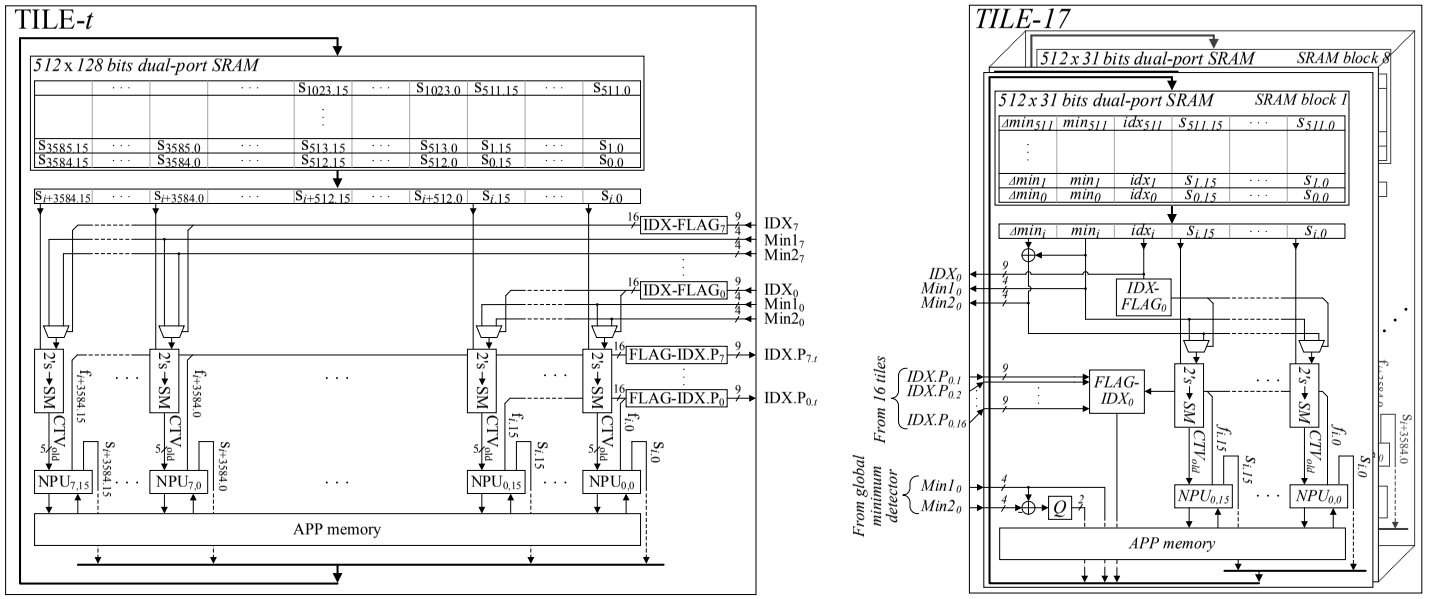
operation that first multiplies the normalization factor to d_c *a posteriori* LLRs, and then finds the two minimum values:

$$L_{mn}^{(k)} = \prod_{n' \in \mathcal{N}(m) \setminus n} \operatorname{sign}(Z_{n'}) \cdot \min_{n' \in \mathcal{N}(m) \setminus n} |Z_{n'} \cdot \alpha| . \tag{4.5}$$

With $\alpha = 0.25$, the number of bits required for CTV messages can be reduced by two bits (from 7 to 5 bits). This efficiently reduces the capacity of the CTV memory as well as the interconnection complexity. In particular, the proposed word-length optimization reduces the critical path delay from 6.1 to 4.2 ns as shown in Fig. 4.10, which further increases the decoding throughput by 39 %. Note that the optimal normalization factor α for MS decoding is usually around 0.75. However, unlike conventional LDPC codes, FG-LDPC codes such as EG-LDPC codes need smaller normalization factors to achieve the best error-correcting performance as stated in [61]. For instance, the normalization factors of 0.5 and 0.25 were used for a (273, 191) and a (1057, 813) PG-LDPC codes, respectively.

In order to further reduce the capacity of the CTV memory, we compress CTV messages [30] and approximate the second minimum magnitude [36]. CTV messages are stored in the compressed form with four components: $\{signs, index, min, \Delta min\}$, where *signs*, *index*, *min*, and Δmin are the set of signs of d_c CTV messages, the index of the minimum magnitude, the minimum magnitude, and the q_d-bit quantized value of the difference between the two smallest magnitudes. As $d_c = 272$, the word-lengths of *signs*, *index*, and min_1 are 272, $\lceil \log_2(272) \rceil = 9$, and four bits, respectively. For this decoder, q_d is set to two bits, which incurs negligible performance loss.

Due to the memory reduction techniques, the capacity of the CTV memory is



(a) For the first to 16th tiles

(b) For the 17th tile

Figure 4.11: Organization of CTV memory and its connection to NPUs

reduced from 7.44 to 1.12 Mbits, which results from 4 k checks $\times$ (272 bits/signs +9 bits/index + 4 bits/min + 2 bits/Δmin). When compared to the pipelined-parallel architecture, these techniques decrease the cell area of the CTV memory by 84 %, and thereby reducing the total cell area by 55 % as shown in Fig. 4.10.

Figure 4.11 illustrates the organization of CTV memory and its connection to relevant NPUs, where $s_{m.l}$ denotes the sign bit of a CTV message that is used for l-th NPU when processing the check node m. Each of the first to 16th tiles contains a 512×128-bit dual-port SRAM for storing the sign bits, while the 17th tile has eight 512×31-bit dual-port SRAM blocks for keeping the sign bits as well as the *index*, *min*, and Δmin. Note that the number of SRAM blocks in the 17th tile is proportional to the parallel factor. Because eight check nodes $\{C_{i+j\cdot 512} : 0 \leq j \leq 7\}$ are processed simultaneously at the i-th clock cycle in the proposed 8-way parallel architecture, a set of signs $\{s_{(i+j\cdot 512).l}\}$ is fetched from the CTV memory in each tile, while $\{index_{i+j\cdot 512}\}$, $\{min_{i+j\cdot 512}\}$, and $\{\Delta min_{i+j\cdot 512}\}$ are read from the CTV memory in the 17th tile to recover CTV messages, where $0 \leq j \leq 7$, and $0 \leq l \leq 15$. The first and second minimum magnitudes (min_1 and min_2, respectively) are recovered in the 17th tile before transmitting to tiles. In order to lower the interconnection complexity, the flags that are needed to select the magnitude of CTV between min_1 and min_2 are generated from the *IDX-FLAG* module in each tile, rather than in the 17th tile.

4.4 Implementation Results

The proposed pipelined-parallel LDPC decoder was synthesized, placed, and routed in a 0.13-μm CMOS technology using Synopsys tools. Table 4.2 shows the

Table 4.2: Implementation results

	This work			DVB [a] [37]		DVB [a] [38]	DVB [a] [39]
	Baseline	Proposed decoder					
	Synthesis	Synthesis	Post-layout				
Code length	68254			16200 / 64800		16200 / 64800	64800
Code rate	0.96			0.25~0.9		0.25~0.9	0.25~0.9
Decoding algorithm	Normalized APP-based with conditional node update			MS with a correction factor			Modified λ-min
Word-length	7 bits	7 for a posteriori LLR, 5 for CTV		6 bits		6 bits	6 bits
Technology	130 nm 6 LM 1.2 V			130 nm 8 LM 1.2 V	90 nm 7 LM 1.0 V	65 nm 7 LM 1.2 V	90 nm
Iteration limit	8			~1,023 [b]	~1,023 [b]	~1,023 [b]	40
Level of parallelism	1	8		12~90 [c]	12~90 [c]	6~45 [c]	3~23 [c]
Clock frequency	59 MHz	185 MHz	131 MHz	200 MHz	300 MHz	174 MHz	270 MHz
Memory area (RAM & ROM)	48.42 mm^2	9.45 mm^2	9.45 mm^2	16.8 mm^2	5.2 mm^2	3 mm^2	8.78 mm^2
Area /wo memory	28.40 mm^2	34.66 mm^2	53.63 mm^2	26.68 mm^2 [d]	10.6 mm^2	3.07 mm^2	3.19 mm^2 [d]
Area	76.82 mm^2	44.11 mm^2	63.08 mm^2	43.48 mm^2 [d]	15.8 mm^2	6.07 mm^2	11.97 mm^2 [d]
Area (scaled to 65nm)	19.21 mm^2	11.03 mm^2	15.77 mm^2	10.87 mm^2 [d]	8.24 mm^2	6.07 mm^2	6.24 mm^2 [d]
Throughput	0.09 Gb/s	2.30 Gb/s	1.63 Gb/s	90 Mb/s [e]	135 Mb/s [e]	135 Mb/s [e]	180 Mb/s
Power consumption	-	-	2.09~3.81 W	1.54 W	700 mW	130~476 mW	477 mW
Energy efficiency [f]	-	-	89~813 pJ/bit	6084 pJ/bit	3835 pJ/bit	685~2507 pJ/bit	1,361 pJ/bit
TAR [f]	-	-	106 $Mbs^{-1}mm^{-2}$	8 $Mbs^{-1}mm^{-2}$	16 $Mbs^{-1}mm^{-2}$	22 $Mbs^{-1}mm^{-2}$	29 $Mbs^{-1}mm^{-2}$

a DVB decoders include both BCH and LDPC codecs, but the BCH codec and the LDPC encoder occupies small chip area (less than 10 %).
b The maximum iteration is programmable up to 1023.
c The LDPC decoders presented here serially update CTV messages, thus a node processing operation takes d_c clock cycles. For a fair comparison with the proposed decoder that computes d_c CTV messages simultaneously, the level of parallelism is defined as the number of node processing units divided by the degree of a check node.
d These do not include the area of the BCH codec and the LDPC encoder.
e The throughput is measured with 41 to 77 iterations depending on code rate.
f The energy efficiency and TAR are estimated in 65-nm technology with the operating voltage of 1.0 V.

implementation results of the baseline and the pipelined-parallel decoders along with three recently published VLSI circuits for LDPC decoding. When comparing the baseline and proposed decoders, we can find that the area of the CTV memory is reduced by 80 % and the throughput is increased by approximately 17 times.

Figure 4.12 shows the layout of the pipelined-parallel decoder that occupies the core area of 63.08 mm^2 with 70 % logic utilization. Shift registers for storing *a posteriori* LLRs are spread across the floorplan. Sixteen 64-kbit dual-port synchronous SRAMs keeping the signs of CTV messages are placed in the upper and lower sides of the chip because each SRAM block is accessed locally by a specific tile. On the other hand, eight 16-kbit dual-port synchronous SRAMs for storing the two minimums, indices, and signs are located at the center of the chip to allow convenient access from all of the tiles.

Under the worst case condition, the critical path delay from the synthesis result was estimated to be 4.2 ns. However, the post-layout delay was increased to 7.62 ns due to the wire delay, hence the maximum operating clock frequency after the layout became 131 MHz. As the iteration limit is set to eight, the minimum decoding throughput is 1.63 Gb/s at the maximum clock frequency of 131 MHz.

The maximum and minimum power consumption were estimated to be 3.81 W and 2.09 W at the minimum and maximum throughput, respectively. When the channel SNR varies from 5.2 to 8.0 dB, the average power consumption and the throughput of the decoder operating at 131 MHz with 1.2 V supply are shown in Fig. 4.13. In the region below 5.3 dB, where most of the received frames cannot be decoded, the decoder consumes over 3.68 W and shows the lowest throughput. However, as the channel SNR grows, the decoding throughput quickly increases because of the fast convergence characteristics of EG-LDPC codes and the serial scheduling scheme.

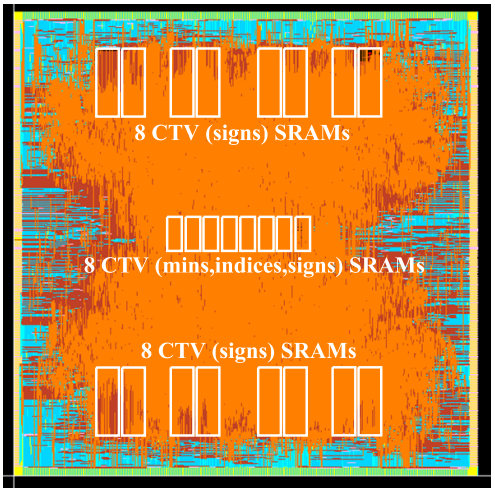


Figure 4.12: Layout of the proposed LDPC decoder

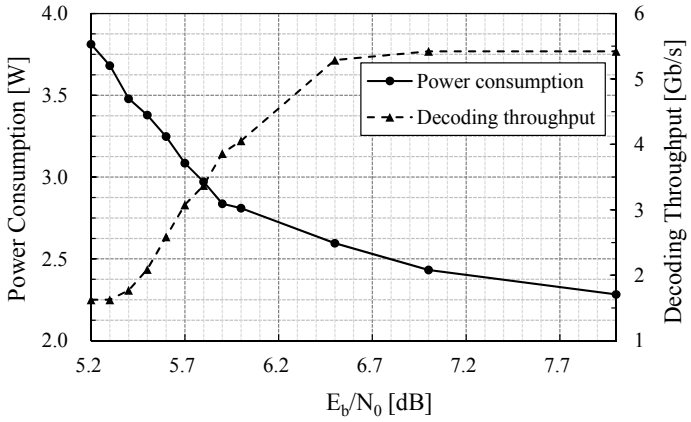


Figure 4.13: Average power consumption and throughput of the decoder

When compared with other implementations in [37, 38, 39], whose code length is comparable to the developed decoder, the proposed one seems to consume a larger chip area because of the higher node degree and shift register-based APP memory. However, this decoder achieves fairly high decoding throughput considering the degree of parallelism mainly thanks to the serial schedule and fast convergence property

of EG-LDPC codes.

For fair comparison, the energy efficiency as well as the throughput-to-area ratio (TAR) [96] are computed for all decoders as follows:

$$\text{Energy efficiency (in pJ/bit)} = \frac{\text{Power consumption (in mW)}}{\text{Throughput (in Gb/s)}}, \tag{4.6}$$

$$\text{TAR (in Mbs}^{-1}\text{mm}^{-2}) = \frac{\text{Throughput (in Mb/s)}}{\text{Area (in mm}^2)}. \tag{4.7}$$

Note that when computing the energy efficiency and TAR, the power consumptions and chip areas are scaled down to 65-nm technology with the operating voltage of 1.0 V according to [97]. As can be observed from Table 4.2, the proposed decoder achieves the highest energy efficiency as well as the area efficiency (TAR). The energy efficiency is a few times better and the area efficiency is at least 4.5 times higher than those of others.

4.5 Concluding Remarks

In this chapter, a decoder architecture for FG-LDPC codes is proposed. The architecture employs the normalized *a posteriori* probability (APP) based algorithm and the serial schedule to reduce the complexity of the node processors and the number of decoding iterations, respectively. A conditional variable node update method is also employed for the normalized APP-based algorithm to reduce the number of bit errors in undecodable blocks and lower the number of switching activities in the decoder. To increase the decoding throughput and minimize the memory requirements, the decoder adopts five-stage pipelined eight-way parallel architecture and a

few chip area reduction techniques including word-length optimization, compression of check-to-variable messages, and approximation of the second minimum. The developed LDPC decoder achieves the maximum throughput of 8.13 Gb/s with the chip area of 63.08 mm^2 in 0.13-μm CMOS process technology.

Chapter 5

Low-Energy Error Correction of NAND Flash Memory through Soft-Decision Decoding

5.1 Introduction

In this chapter, we analyze the energy consumption of a NAND flash memory error correction system that adopts soft-decision LDPC decoding. The energy consumed in NAND flash memory as well as that in the LDPC decoder is all considered. A VLSI circuit-based decoder implementing a rate-0.96 (68254, 65536) LDPC code is used for error performance and energy estimation. Especially, the effect of energy consumption when increasing the output precision of NAND flash memory is analyzed. The LDPC decoder tends to consume more energy when the precision of NAND flash memory output is very low. However, increasing the precision also demands more energy in NAND flash memory for memory sensing and data transfer. As a result, the

optimum precision is closely related to the signal quality of NAND flash memory. We analyze this relation quantitatively, and also propose a method that can find the optimum precision using the iteration count of an LDPC decoder.

The rest of this chapter is organized as follows. Section 5.2 explains the read operation of NAND flash memory and its energy consumption. In Section 5.3, the error performance of LDPC decoding with soft-decision flash memory output is presented. Section 5.4 describes the energy consumption of a rate-0.96 (68254, 65536) LDPC decoder implemented with a 65-nm technology and compares the hardware performance of the LDPC decoder with that of two Bose-Chaudhuri-Hocquenghem (BCH) decoding circuits. In Section 5.5, we analyze the total energy consumption of a NAND flash memory system employing LDPC code-based soft-decision decoding and also propose an LDPC decoder-assisted precision selection method. Finally, Section 5.6 concludes this chapter.

5.2 Energy Consumption of Read Operations in NAND Flash Memory

5.2.1 Voltage Sensing Scheme for Soft-Decision Data Output

In 2-bit multi-level cell (MLC) NAND flash memory, each memory cell has one of four different threshold voltages that have Gaussian-like distributions as illustrated in Fig. 5.1, where the left-most peak corresponds to the erased state (symbol *11*), and the remaining ones are three different programmed states (symbol *01*, *00*, and *10*, respectively). The read operation of NAND flash memory can be considered a quantization process. In conventional flash memory with hard-decision data output,

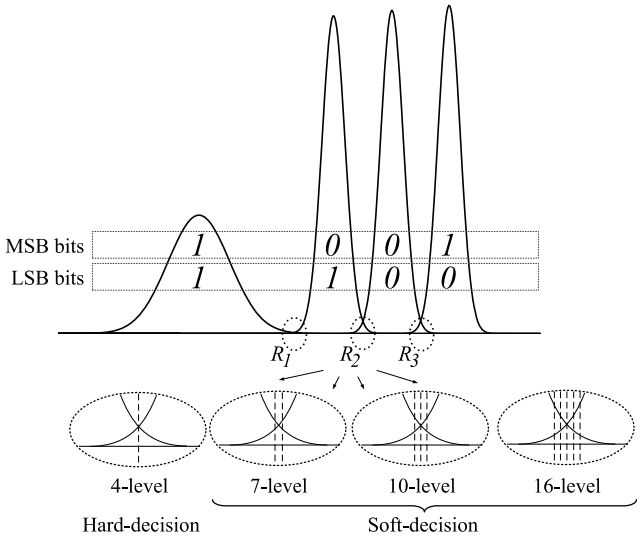
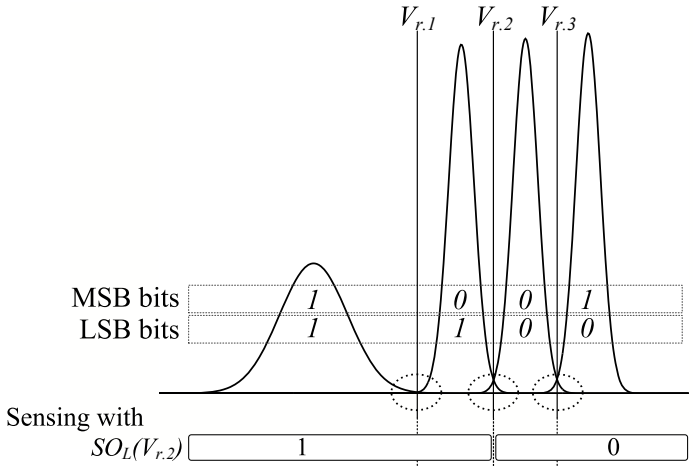


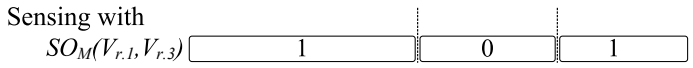
Figure 5.1: Threshold voltage distributions and voltage sensing schemes of 2-bit MLC NAND flash memory

three sensing reference voltages (SRVs), namely, $V_{r.1}$, $V_{r.2}$, and $V_{r.3}$, are needed to fully resolve the four threshold voltage distributions, which corresponds to 4-level signal quantization in Fig. 5.1, where the dashed line at each overlapping region R_i represents an SRV. Note that $V_{r.1}$ resolves the boundary between the symbols *11* and *01*, while $V_{r.2}$ is for the boundary of the symbols *01* and *00*, and $V_{r.3}$ is for the symbols *00* and *10*. Since a pair of LSB and MSB pages is mapped into a word-line and the bits are gray coded, $V_{r.1}$ and $V_{r.3}$ are required to read MSB pages, while only $V_{r.2}$ is needed for LSB pages as illustrated in Fig. 5.2. The LSB sensing operation (SO) with $V_{r.2}$ is referred to $SO_L(V_{r.2})$, and the MSB sensing operation with $V_{r.1}$ and $V_{r.3}$ is represented by $SO_M(V_{r.1}, V_{r.3})$.

For soft-decision error correction, each memory cell should be sensed with an increased number of SRVs. Especially, it is needed to increase the resolution in the overlapping regions, where most of bit errors are occurred, as shown in Fig. 5.1. The



(a) For LSB pages



(b) For MSB pages

Figure 5.2: Voltage sensing scheme of 4-level signal quantization

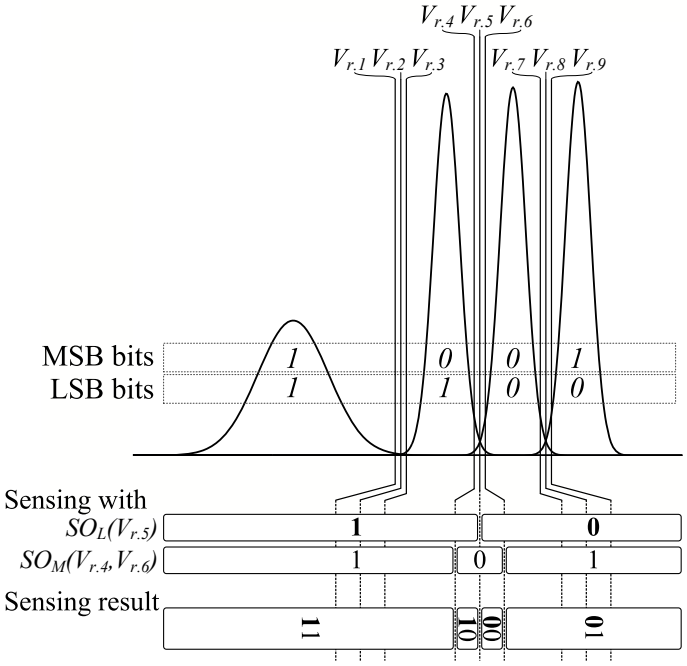
simplest form of soft-decision memory sensing is to provide an erasure region at each symbol boundary. In this case, we need six SRVs and can obtain seven different quantized values. The lowest voltage region can be considered a strong *11* symbol, and the next lowest region is a value between *11* and *01*. Figure 5.1 shows four different sensing schemes, including the conventional sensing for hard-decision data output. Increasing the number of sensing operations at each symbol boundary can provide more accurate reliability information, which, however, increases the latency and energy consumption in NAND flash memory.

Since conventional NAND flash memory devices do not naturally provide soft-decision memory sensing, obtaining the soft-decision data from conventional memory requires multiple hard-decision sensing and data output operations. Note that con-

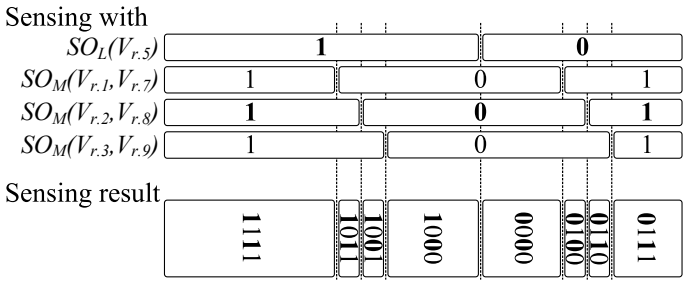
Table 5.1: The number of sensing and data output (DO) operations for hard- and soft-decision sensing with conventional NAND flash memory

Precision	LSB pages			MSB pages		
	SO_L	SO_M	DO	SO_L	SO_M	DO
4-level	1	0	1	0	1	1
7-level	0	1	1	1	2	3
10-level	1	1	2	1	3	4
16-level	1	2	3	1	5	6

ventional NAND flash memory devices provide command sequences that can change the SRVs. Figure 5.3 illustrates the voltage sensing scheme for 10-level soft-decision data output with conventional hard-decision memory sensing operations, where $V_{r.i}$'s are SRVs for $1 \leq i \leq 9$. With a hard-decision LSB sensing operation $SO_L(V_{r.5})$ and an MSB sensing operation $SO_M(V_{r.4}, V_{r.6})$ around the overlapping region R_2, an LSB bit is read with four levels as shown in Fig. 5.3(a). In this case, two data output operations are performed. Meanwhile, because an MSB bit has two overlapping regions, R_1 and R_3, three MSB sensing operations, $SO_M(V_{r.1}, V_{r.7})$, $SO_M(V_{r.2}, V_{r.8})$, and $SO_M(V_{r.3}, V_{r.9})$ are needed. In addition, one LSB sensing operation $SO_L(V_{r.5})$ is also performed to distinguish the region below $V_{r.1}$ and that above $V_{r.9}$ as illustrated in Fig. 5.3(b). As a result, in order to read an MSB bit with eight levels, one SO_L and three SO_M are demanded, which results in four times many data output operations when compared to the conventional hard-decision mode. Table 5.1 summarizes the number of sensing operations for the 4-level hard-decision and the 7-, 10-, and 16-level soft-decision memory sensing. Note that the sensing results are mapped to log-likelihood ratio (LLR) values by using a look-up table in the flash memory controller.



(a) For LSB pages



(b) For MSB pages

Figure 5.3: Voltage sensing scheme of 10-level signal quantization

5.2.2 LSB and MSB Concurrent Access Scheme for Low-Energy Soft-Decision Data Output

As explained in the previous subsection, the soft-decision scheme with conventional memory demands multiple hard-decision sensing and data transfer operations to increase the resolution in the overlapping region. Moreover, an additional LSB sensing operation is needed to access an MSB page as shown in Fig. 5.3(b). This scheme incurs a large amount of data output operations when high precision data are needed. In order to reduce the energy consumption of soft-decision data output, we consider a method that senses the LSB and MSB bits simultaneously with multiple SRVs.

In this scheme, an $(N_s + 1)$-level read operation is performed with N_s SRVs for a row of transistors, i.e., a word-line, in the NAND flash array, and all the sensing results are stored at the data register in N_b bits, where $N_b = \lceil \log_2(N_s + 1) \rceil$. Assuming that up to 16-level quantization is used, $N_b = 4$ bits are needed to represent all kinds of soft-decision sensing results. Of course, this scheme needs increased hardware of $4 \times N_{pagebits}$ data registers to store the soft-decision sensing results as shown in Fig. 5.4, while the conventional NAND flash memory has only $N_{pagebits}$ data registers, where $N_{pagebits}$ is the number of bits in each page.

When compared to the soft-decision sensing using conventional NAND flash memory described in the previous subsection, this concurrent access scheme greatly reduces the number of data transfer operations, only N_b bits for both LSB and MSB data, because the data are composed within a memory device. Thus, this method reduces not only the data output latency but also the energy consumption for off-chip data transfer. Therefore, we only consider the LSB and MSB concurrent access scheme in this dissertation.

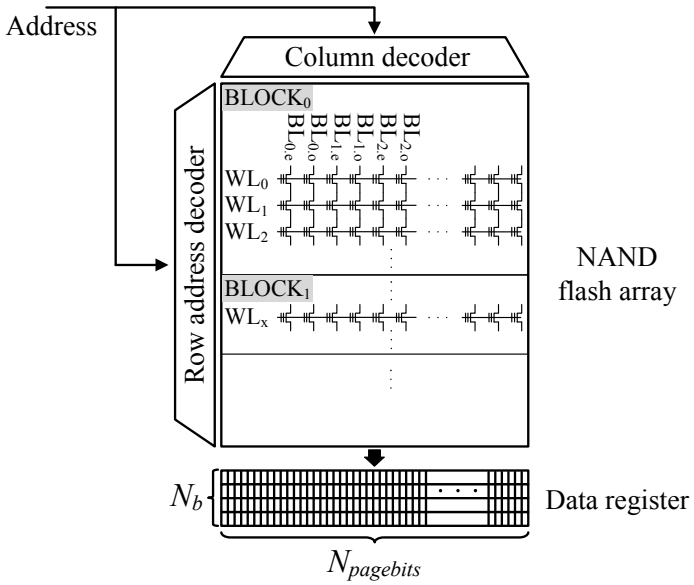


Figure 5.4: NAND flash memory with internal soft-decision data composition

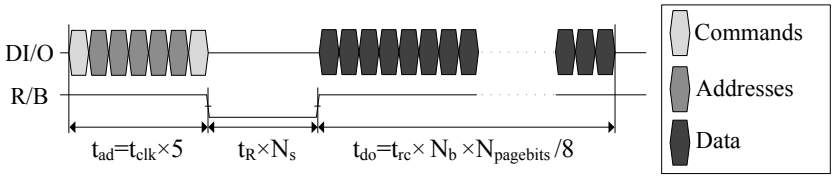


Figure 5.5: Timing diagram of read page mode

5.2.3 Energy Consumption of Read Operations in NAND Flash Memory

The read operation of NAND flash memory involves *address decoding*, *NAND flash array access*, and *data output*. Conventional NAND flash memory supports various types of read operations such as *read page* and *read page cache*. The *read page* mode accesses only one page, whereas the *read page cache* mode reads the next sequential pages in a block consecutively, while concurrently outputting data from the

data register to increase the throughput. The timing diagram of the read page mode is illustrated in Fig. 5.5, where t_{clk}, t_R, and t_{rc} denote the clock period, NAND flash array access time for each voltage sensing operation, and read cycle time, respectively. The array access time, t_R, includes the threshold voltage sensing operation time as well as the data transfer time from the NAND flash array to either the data or cache register.

In this section, we analyze the energy consumption of reading 2-bit MLC NAND flash memory. We estimate the energy consumption consulting the electrical specifications listed in the data book from Micron technology [6]. The energy consumption of reading NAND flash memory is modeled as the sum of the energy for array access (E_{ac}) and that for data output (E_{do}), where

$$E_{ac} = V_{cc} I_{cc} t_R N_s, \tag{5.1}$$

$$E_{do} = V_{ccq} I_{io} t_{do}. \tag{5.2}$$

Note that we only concern the active energy and ignore the idle energy. V_{cc} and V_{ccq} are the core and the I/O supply voltages, while I_{cc} and I_{io} represent the core and the I/O supply currents, respectively. Finally, the data output time is represented by t_{do}, which is determined by the number of bytes to output and the period of data output clock, as a result $t_{do} = t_{rc} \times N_b \times N_{pagebits}/8$.

Since the read operation is performed simultaneously for both LSB and MSB data, the energy consumption of LSB and MSB pages is considered as follows. Let $E_{r.LSB}$ and $E_{r.MSB}$ be the read energy for an LSB page and an MSB page, respectively. In 2-bit MLC, reading an MSB page uses two times many SRVs than that of an LSB page access, hence the energy consumption of the array access operations for an LSB

Table 5.2: The voltage, current, and timing parameters of 2-bit MLC NAND flash memory

	Asynchronous	**Synchronous**	**Unit**
t_{clk}	20, 25, 30, 35, 50, 100	10, 12, 15, 20, 30, 50	ns
V_{cc}	3.3	3.3	V
V_{ccq}	1.8, 3.3	1.8, 3.3	V
I_{cc}	25	25	mA
I_{io}	8	20	mA
t_{ad}	150–450	168–288	ns
t_R	12.5	12.5	μs/sensing
t_{rc}	t_{clk}	$0.5 \times t_{clk}$	ns

page and an MSB page can be modeled as $E_{ac}/3$ and $E_{ac} \times 2/3$, respectively. Because two pages of data are delivered simultaneously in the LSB and MSB concurrent access scheme, the data output energy of each page is modeled as $E_{do}/2$. Therefore, the energy consumption of each page can be represented as follows:

$$E_{r.LSB} = \frac{1}{3}E_{ac} + \frac{1}{2}E_{do}, \tag{5.3}$$

$$E_{r.MSB} = \frac{2}{3}E_{ac} + \frac{1}{2}E_{do}. \tag{5.4}$$

Table 5.2 shows the voltage, current, and timing parameters noted in the 34-nm 2-bit MLC NAND flash data book from Micron technology [6]. Table 5.3 shows the estimated energy consumption and the latency of read operation for different memory signal quantization cases. Since the data output operation takes a long time due to the limited number of I/O ports, the operating condition that needs the smallest t_{rc} in the synchronous mode shows the minimum energy consumption. In this simulation, NAND flash memory that operates at 100 MHz and V_{ccq} of 1.8 V in the synchronous mode consumes the minimum read energy. Since the energy consumption of the *read page* mode is almost similar to that of the *read page cache* mode, we only consider

Table 5.3: The energy consumption of a read operation for LSB and MSB pages

	E_{ac} (nJ/byte)	E_{do} (nJ/byte)	E_r (nJ/byte)
LSB pages			
4-level	0.12	0.18	0.30
7-level	0.24	0.27	0.51
10-level	0.36	0.36	0.72
16-level	0.60	0.36	0.96
MSB pages			
4-level	0.24	0.18	0.42
7-level	0.48	0.27	0.72
10-level	0.72	0.36	1.08
16-level	1.19	0.36	1.55

the *read page* mode of the above operating condition ($t_{clk} = 10\,\text{ns}$, $V_{ccq} = 1.8\,\text{V}$, and synchronous mode).

As summarized in Table 5.3, the 7-, 10-, 16-level signal quantization of an LSB page consume 1.7, 2.4, and 3.2 times more energy, respectively, when compared to the 4-level quantization that yields hard-decision data output. MSB pages consume approximately 1.5 times more energy than LSB pages.

5.3 The Performance of Soft-Decision Error Correction over a NAND Flash Memory Channel

This section estimates the error performance of an LDPC code using a 2-bit MLC NAND flash memory simulation model that includes random telegraph noise, incremental step pulse programming, cell-to-cell interference (CCI), and non-uniform quantization [8, 8, 98, 99, 100, 101] as illustrated in Fig. 5.6. In order to support soft-decision LDPC decoding, we evaluate the effects of memory signal quantization using the LLR computation method proposed in [44], where the four threshold volt-

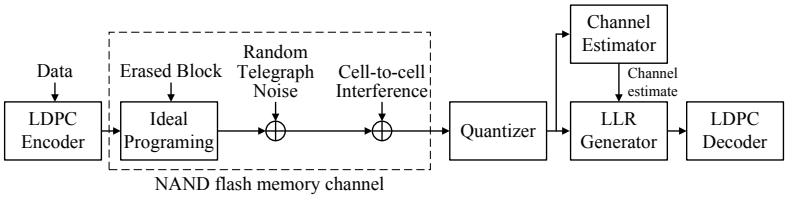


Figure 5.6: MLC NAND flash memory channel model

age distributions are assumed as Gaussian distributions and the partial cumulative distribution functions of the Gaussian distributions are used to compute quantized LLRs. Thus, the LLR computation method only requires the means and the variances of the distributions obtained by channel estimation [42]. Note that the LLR computation can be implemented using a look-up table.

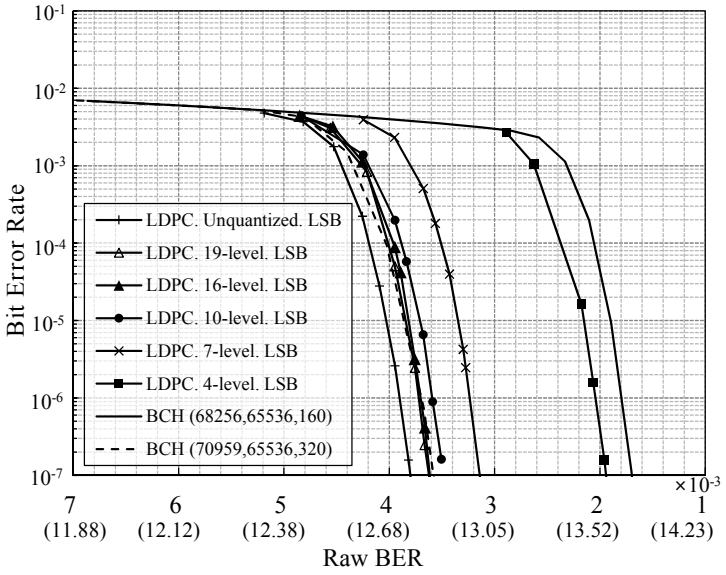
For the error correction in NAND flash memory, we employ a rate-0.96 (68254, 65536) shortened Euclidean geometry (EG) LDPC code that can accommodate one page of the 128-Gbit 2-bit MLC NAND flash memory as described in Section 4.2.1. Note that finite geometry (FG) LDPC codes are good candidates for error correction of NAND flash memory because FG-LDPC codes have the characteristics of low error-floor performance, fast convergence speed, good error-correcting performance, and cyclic or quasi-cyclic (QC) properties.

We assume that the erased state (symbol *11*) has a Gaussian distribution whose mean and standard deviation are 1.0 V and 0.32 V, respectively, and the target programming voltages for the symbol *01*, *00*, and *10* are 2.6 V, 3.2 V, and 3.8 V, respectively. In order to generate the NAND flash memory channel with different bit error rates (BERs), we change the cell-to-cell coupling coefficient factor (CCF) [40, 44]. The CCF primarily affects the variances of the threshold voltage distributions. Hence, increasing the CCF results in high raw BER (RBER) because of the increased vari-

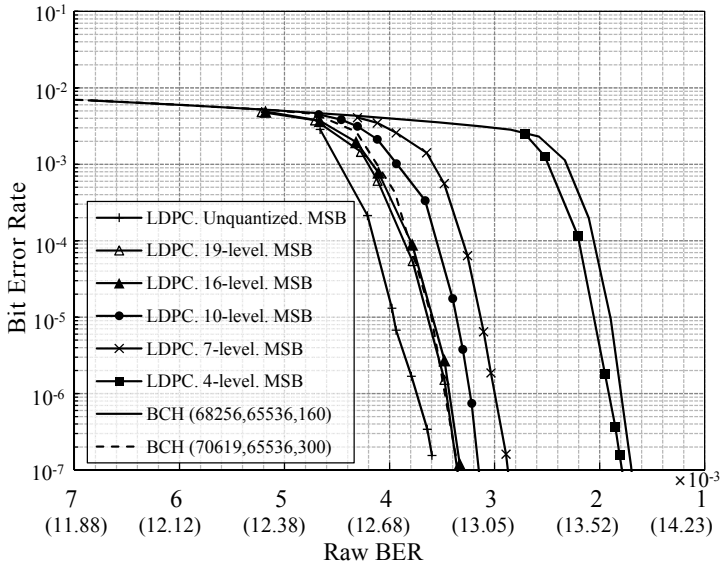
ance. The probability ratios that determine quantization boundaries [40] are set to three and eight for the 7-level and the other levels, respectively.

The error performances of a rate-0.96 (68254, 65536) EG-LDPC code and two BCH codes over the NAND flash memory channel are plotted in Fig. 5.7 for LSB and MSB pages, where the serial-C schedule and the fixed-point normalized APP-based algorithm with conditional variable node update are used for low-complexity LDPC decoding. The word-length and the maximum iteration number is set to seven bits and eight, respectively, as described in Section 4.2.2. The error performance of the decoding with unquantized channel output is also shown for comparison. The x-axis represents RBER, and the numbers in parentheses are the corresponding signal-to-noise ratio (SNR) values, which are computed assuming a 4-pulse amplitude modulation channel with additive white Gaussian noise (AWGN).

As shown in Fig. 5.7, increasing the precision, i.e., the number of quantization levels, improves the error-correcting performance, and the unquantized channel output yields the best result. However, the improvement is not much noticeable when the precision is larger than 16-level. In order to compare the soft-decision LDPC decoding with hard-decision decoding, we employed BCH codes. The (68256, 65536, 160) BCH code, which has the same code rate of 0.96, shows a much worse performance than the soft-decision LDPC decoding for LSB and MSB pages. To have the comparable performance to the LDPC decoding with 16-level quantization, the error-correcting capability t of the BCH code needs to be almost doubled, $t = 320$ for LSB pages and $t = 300$ for MSB pages. This translates that the code rate of the BCH codes needs to be lowered to 0.92, which demands twice the amount of parity data. The comparison of soft-decision LDPC and hard-decision BCH codes clearly shows the advantage of the soft-decision decoding.



(a) For LSB pages



(b) For MSB pages

Figure 5.7: Error-performance of the (68254, 65536) EG-LDPC code over the NAND flash memory channel

Table 5.4: The operating regions according to memory signal quantization

	RBER ($\times 10^{-3}$)		**Memory signal quantization**
	LSB pages	**MSB pages**	**needed for 10^{-7}BER**
Region I	~ 1.95	~ 1.79	4-, 7- 10-, and 16-level
Region II	1.95–3.15	1.79–2.90	7- 10-, and 16-level
Region III	3.15–3.50	2.90–3.15	10- and 16-level
Region IV	3.50–3.62	3.15–3.33	16-level
Region V	3.62+	3.33+	–

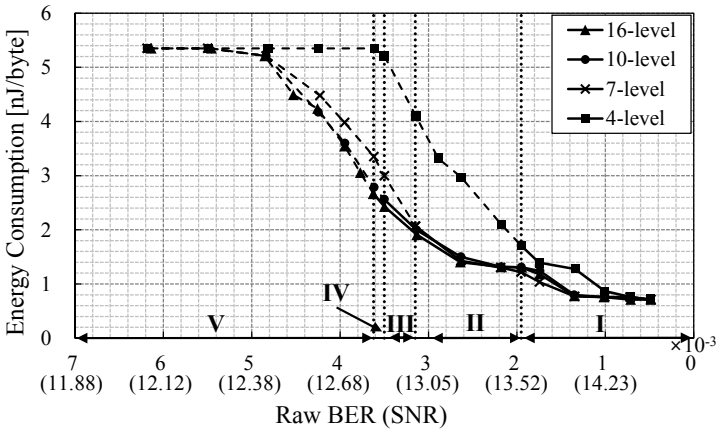
In Fig. 5.7(a), we can find that even 4-level hard-decision decoding works when the RBER is lower than 1.95×10^{-3}. However, when the RBER is between 1.95×10^{-3} and 3.15×10^{-3}, the 4-level hard-decision decoding does not work and only soft-decision decoding can remove most of the errors. When the RBER is greater than 3.62×10^{-3}, even 16-level soft-decision decoding cannot correct the data properly. From this observation, we can divide the RBER values into five regions as shown in Table 5.4. Although a NAND flash memory system requires error-free decoding with BER less than 10^{-15}, here we set the target BER to 10^{-7} because the simulation of the LDPC decoding takes much time to observe the minimum requirement. Note again that EG-LDPC codes show very low error-floor performance and have fast convergence speed. Table 5.4 summarizes the results for LSB and MSB pages. Here, we can find that the 7-level quantization enhances the error-correcting performance very much when compared to 4-level hard-decision decoding. However, further increasing the precision brings diminishing returns. As a result, the region II is quite wider than region III or IV.

5.4 Hardware Performance of the (68254, 65536) LDPC Decoder

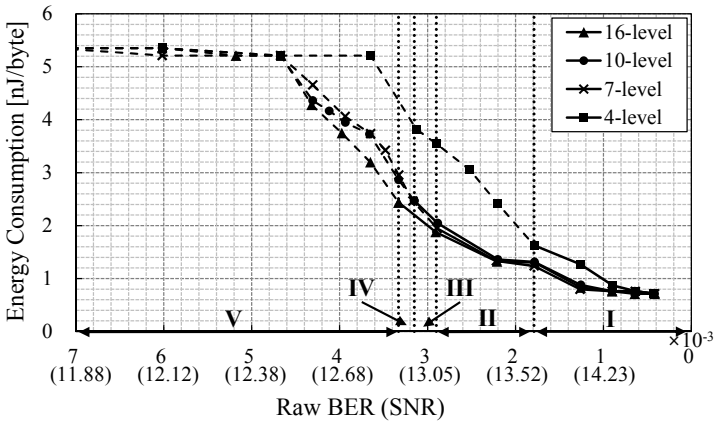
5.4.1 Energy Consumption of the LDPC Decoder

In Chapter 4, we have implemented the (68254, 65536) EG-LDPC decoder employing the normalized APP-based algorithm, the serial-C schedule, and the conditional variable node update technique. This subsection assesses the energy consumption of the LDPC decoder for read operation of NAND flash memory and compares the developed decoder with two BCH decoding circuits showing the comparable performance in terms of the parity ratio, chip area, decoding throughput, and energy consumption.

The LDPC decoder was synthesized, placed, and routed in 0.13-μm CMOS technology using Synopsys tools, then parasitic resistances and capacitances were extracted to estimate the energy consumption accurately. Randomly generated information bits were encoded and Gaussian noise was added to make test vectors. Then, the power consumption, iteration count, and decoding latency were estimated by using Synopsys PrimeTime. From the simulation results, we obtained the average energy consumption as a first-order function of the iteration count. Finally, the energy consumption of the LDPC decoder was computed using the average iteration counts found by simulations for each precision and RBER. In order to consider the implementation with a recent process technology, the decoding energy of the LDPC decoder is scaled down to a 65-nm technology. The core supply voltages of 130 nm and 65 nm nodes are 1.2 V and 1.0 V, respectively. In addition, the maximum clock frequencies are assumed to be the same, 131 MHz, for both processes. Considering



(a) For LSB pages



(b) For MSB pages

Figure 5.8: The energy consumption of the (68254, 65536) LDPC decoder (65-nm VLSI) over NAND flash memory channel

the process technologies and the supply voltages, the energy consumption is scaled down by a factor of 2.88 ($= \left[(65/130\,\text{nm}) \times (1.0/1.2\,\text{V})^2\right]^{-1}$) for the 65-nm technology node according to [97].

The energy consumption of the (68254, 65536) LDPC VLSI with the 65-nm technology for hard- and soft-decision memory sensing is shown in Fig. 5.8, where the

clock frequency and the maximum iteration limit were set to 131 MHz and eight, respectively. Since the implemented LDPC decoder shows very fast convergence speed, the decoding energy consumption decreases rapidly at low RBER (high SNR). For the low RBER region below 10^{-3}, decoding with all of the precision considered demands mostly one decoding iterations, thus resulting in the minimum energy consumption of 0.7 nJ/byte. For the region exceeding the RBER of 10^{-3}, decoding with soft-decision data consumes less energy than that with the 4-level hard-decision data because of the decreased number of iterations. In addition, in the region below the RBER of 3×10^{-3}, all soft-decision decoding shows similar energy consumption.

At the high RBER region where only 16-level soft-decision decoding is allowed to use, i.e., region IV, we can find that the average energy consumption of the LDPC decoder is 1.6 to 8.4 times higher than that of the read operation in MLC NAND flash memory. However, in the low RBER (high SNR) region in which all kinds of precision can be used, i.e., region I, the LDPC decoder consumes only 0.5 to 2.3 times of the energy needed for the read operation in MLC NAND flash memory. Therefore, we can consider that the total energy consumption is significantly affected by the LPDC decoder in the high RBER region but is more influenced by the read operation of NAND flash memory in the low RBER region.

5.4.2 Performance Comparison of the LDPC Decoder and Two BCH Decoders

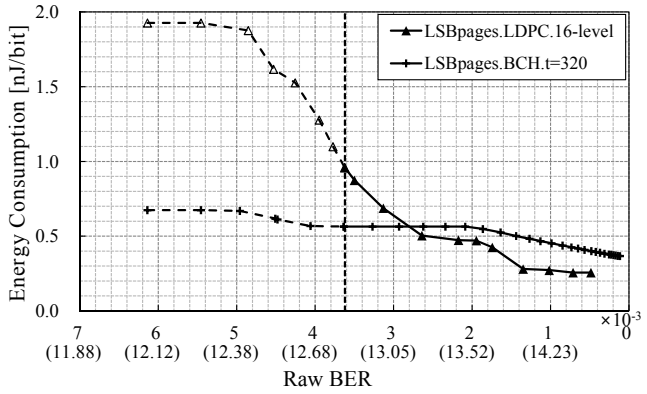
We also compare the performance of the developed LDPC decoder with that of BCH code-based hard-decision decoding. The (70959, 65536, 320) and the (70619, 65536, 300) BCH codes show comparable error performance to the developed (68254, 65536) EG-LDPC code with 16-level quantization for LSB and MSB pages, respectively.

Table 5.5: The parallel factors of the two BCH decoders

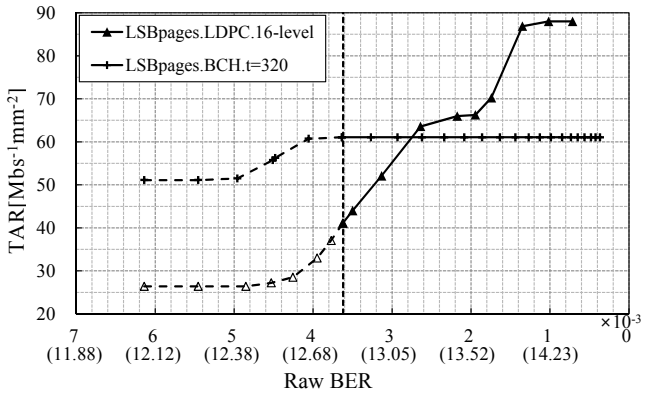
	Parallel factors			Time-multiplexing factor
	Encoder	Syndrome generator	Chien seach	
(70959, 65536, 320) BCH decoder	16	10	16	1
(70619, 65536, 300) BCH decoder	16	9	16	1

These BCH decoders were implemented using the same 0.13-μm CMOS technology. The BCH decoders employ the architecture proposed in [2], where the BCH encoder-assisted syndrome generator, simplified inversion-less Berlekamp-Massey algorithm (SiBMA), and resource sharing technique between the syndrome generator and the Chien search module are proposed. To achieve almost the same minimum decoding throughput of 1.63 Gb/s, the parallel factors for the encoder, the syndrome generator, and the Chien search module were chosen as shown in Table 5.5. The time-multiplexing factor of the SiBMA module was set to one to support the needed decoding throughput.

The developed (70959, 65536, 320) and (70619, 65536, 300) BCH decoders occupy the core areas of 32.74 mm^2 and 29.47 mm^2 with 70 % logic utilization and consume 1.10 W and 1.07 W at the minimum decoding throughput, respectively. The energy efficiency and throughput-to-area ratio (TAR) of both decoders are computed as shown in Fig. 5.9 and Fig. 5.10 for LSB and MSB pages, respectively. Assuming that the target BER of a NAND flash memory system is 10^{-7}, both the soft-decision LDPC decoder with 16-level quantization and the hard-decision BCH decoder work up to the RBER of 3.6×10^{-3} and 3.3×10^{-3} for LSB and MSB pages, respectively, as shown in Fig. 5.7. In other words, the right side of the vertical dotted line in Fig. 5.9 and Fig. 5.10 is the region in which the decoder corrects the data properly at the target BER of 10^{-7}.



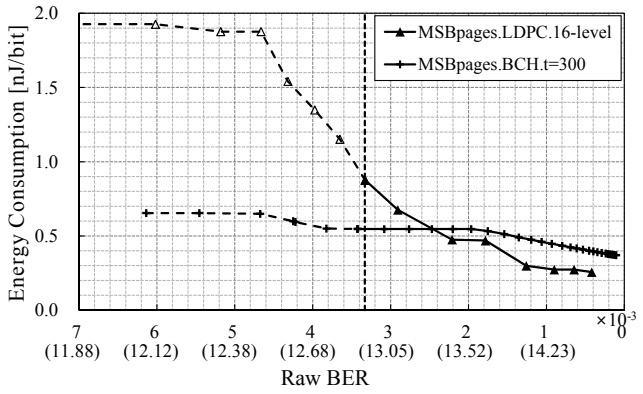
(a) Energy efficiency



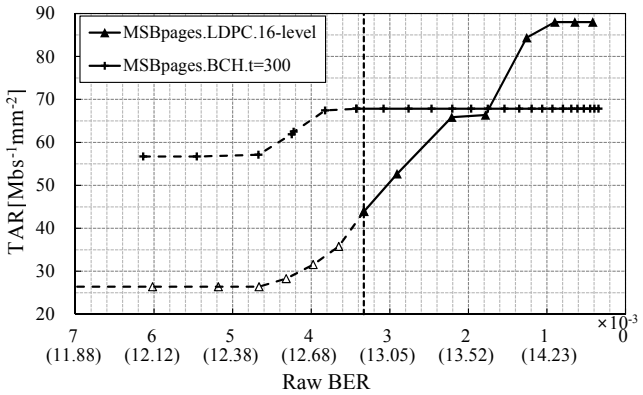
(b) TAR

Figure 5.9: Energy efficiency and TAR of the (68245, 65536) LDPC and (70959, 65536, 320) BCH decoders for LSB pages

From Fig. 5.9 and Fig. 5.10 we can find that the LDPC decoder outperforms the BCH decoders for a broad range of the RBER because of the advantage of early termination, while the BCH decoders yield better results only when the RBER is near the undecodable region. Most of all, the soft-decision LDPC decoding demands only about a half of the parity ratio when compared to the hard-decision BCH decoding.



(a) Energy efficiency



(b) TAR

Figure 5.10: Energy efficiency and TAR of the (68245, 65536) LDPC and (70619, 65536, 300) BCH decoders for MSB pages

5.5 Low-Energy Error Correction Scheme for NAND Flash Memory

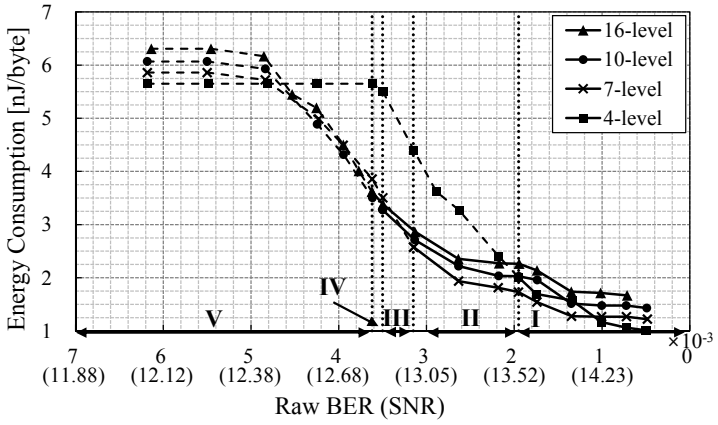
5.5.1 Optimum Precision for Low-Energy Decoding

The total energy consumption of NAND flash memory access can be obtained by adding the energy consumption for memory access and that for error correction. We observe that high precision increases the energy for memory access, while it can reduce the LDPC decoding energy.

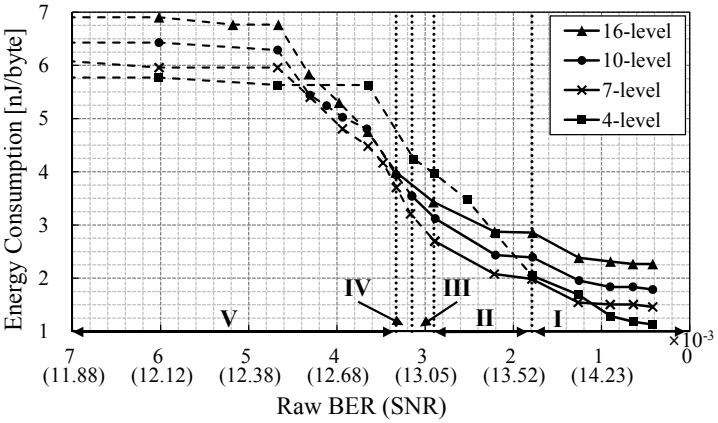
Figure 5.11 shows the total energy consumption of NAND flash memory with the LDPC decoder for LSB and MSB pages, where NAND flash memory operates at 100 MHz and V_{ccq} of 1.8 V in the synchronous data output mode. The vertical dotted lines divide the operating regions according to Table 5.4.

In the region I, where all hard- and soft-decision decoding operate, the decoding with the 4-level quantization shows the smallest energy consumption when the RBER is very low, while the 7-level soft-decision decoding consumes less energy than the hard-decision decoding as RBER increases. In the region II, the decoding with the 7-level read operation results in the lowest energy consumption, while in the region III, the 10-level quantization leads to the lowest consumption. Finally, in the region IV, there is no other choice except the 16-level soft-decision decoding.

In summary, for each operating region, decoding with the lowest precision allowed consumes the least energy among possible decoding schemes, especially for decoding MSB pages. Although the 16-level soft-decision decoding shows the best error-correcting performance over all RBER regions, it consumes up to two times more energy than the 4-level hard-decision decoding at the low RBER (high SNR)



(a) For LSB pages



(b) For MSB pages

Figure 5.11: The total energy consumption

region because of the additional memory sensing operations. Therefore, depending on the channel condition, an appropriate precision should be chosen to minimize the total energy consumption.

We also studied the trend of total energy consumption when considering both program-and-erase (PE) cycling and data retention. The NAND flash memory channel estimation proposed in [42] was used to decide the SRVs and the lowest precision

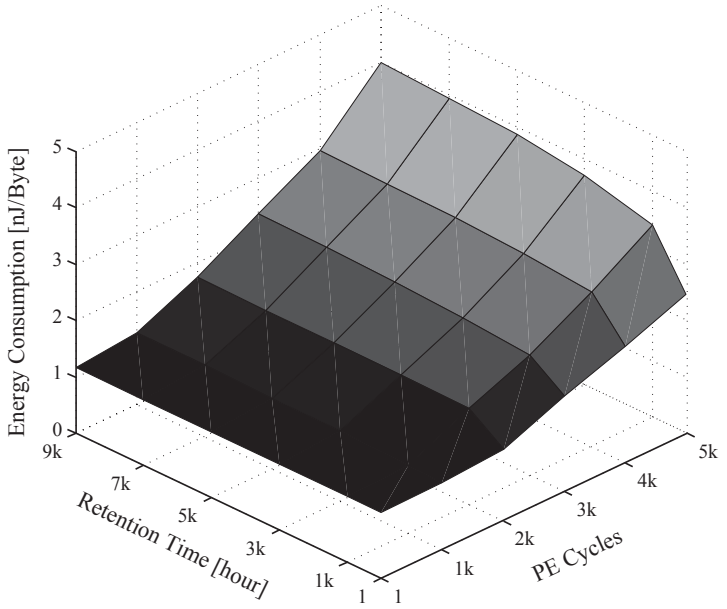


Figure 5.12: The total energy consumption for MSB pages with the number of PE cycles and retention time

was chosen among the possible decoding schemes. Figure 5.12 shows the total energy consumption for MSB pages. The number of PE cycles and retention time vary from 1 to 5 k times and from 1 to 9 k hours, respectively. The coupling coefficients of the x and $x-y$ directions are set to 0.1034 and 0.006721, respectively, in order to consider 20-nm flash memory technology [9, 102]. We can find that the total energy consumption is very strongly affected by the PE cycling. When the number of PE cycles is less than or equal to 1K, the total energy consumption shows the least amount, which is around 1 nJ/byte regardless of the retention time. However, the total energy consumption also increases with the retention time when the number of PE cycles is larger than 1 k.

5.5.2 Iteration Count-Based Precision Selection

The presented experimental results show that optimum precision selection is very important for low-energy soft-decision decoding of NAND flash memory. One straightforward idea is to conduct *failure-based* precision selection. In this method, the precision is increased when the decoding is failed. For example, the decoding begins with 4-level quantization, and if it fails, the decoding is retried with an increased precision. Although this method is very simple and there is no need of storing the precision information, this can consume a large amount of energy when the decoding fails because LDPC decoders iterate many cycles. Of course, the failure-based scheme also incurs additional time-delay for retrying the decoding with an increased precision.

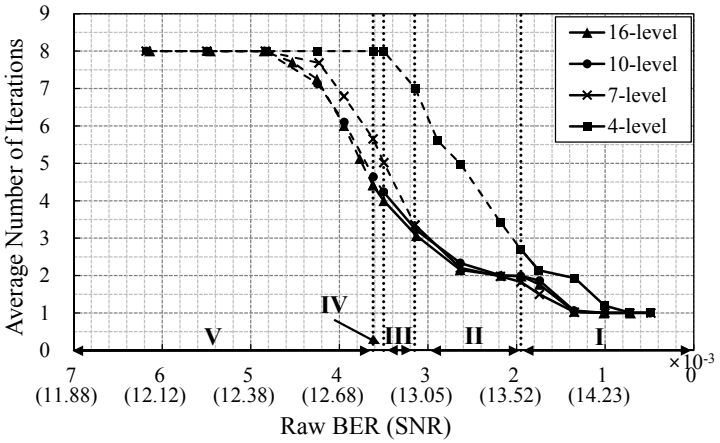
Another approach is to estimate the signal quality of NAND flash memory periodically with channel estimation algorithms [42]. By sensing the signal with multiple threshold voltages, we can estimate the mean and the variance of each symbol. This method, however, demands extra time and energy for signal quality estimation. Considering that the signal quality deteriorates when the number of PE cycles and the retention time increase, the overhead of periodic estimation can be quite high, especially for a large capacity solid-state drives (SSDs).

We propose a precision selection method that utilizes the iteration count of the LDPC decoder. In this explanation, we use the precision of 4-, 7-, and 16-level because the optimum operating range of the 10-level read is quite narrow as shown in Fig. 5.13(b). When the RBER is very low, such as less than 1.0×10^{-3}, the average iteration count is around one even with the 4-level quantization. Thus, employing the 4-level read is the best for low energy decoding in this region. However, as the

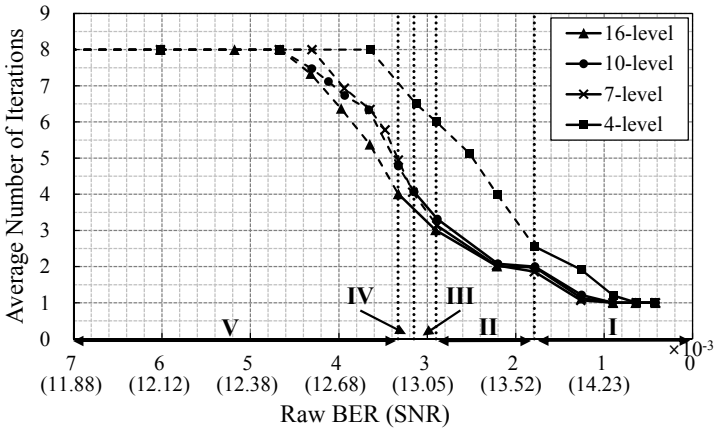
RBER grows and when it is approximately between 1.0×10^{-3} and 1.79×10^{-3}, the decoding with the 4-level quantization demands an increased number of iterations. Thus, we need to increase the precision to 7-level for lowering the energy when the iteration count with the 4-level read is repeatedly two or greater. Of course, the opposite path is also needed. If the iteration count is repeatedly only one with the 7-level quantization, then it is needed to lower the precision into 4-level. A similar scenario happens when the RBER is close to 3.0×10^{-3}. At this region, the decoding with 7-level demands the iteration count of three or more. This means that it is the time to increase the precision to 16-level. Of course, when the iteration count with 16-level quantization is repeatedly equal to or less than two, we need to decrease the precision to 7-level. Since the precision is adjusted before the decoding failure, we can avoid the energy loss and delay. Finally, The channel estimation is performed only when the iteration count of the decoding with 16-level quantization is repeatedly four or greater.

5.6 Concluding Remarks

We studied the optimum memory signal quantization of NAND flash memory for low-energy soft-decision error correction. The energy consumed at NAND flash memory as well as the LDPC decoder is considered. This study shows that the optimum precision of flash memory data for soft-decision LDPC decoding depends on the signal quality, which implies that knowing the signal-to-noise ratio (SNR) of NAND flash memory is quite important for low-energy error correction. When the SNR is relatively high, the conventional 4-level (hard-decision) decoding for 2-bit multi-level cell (MLC) leads to the lowest energy consumption because of minimum sensing and



(a) For LSB pages



(b) For MSB pages

Figure 5.13: Average number of decoding iterations of the (68254, 65536) LDPC decoder

output energy consumed at NAND flash memory. However, as the SNR decreases, the optimum precision for low energy needs to be increased. We find that the precision of 7-level for signal quantization, which represents providing an erasure region at each signal boundary, leads to minimum energy decoding at a broad range of signal quality. We also propose an adaptive, feedback-based, precision selection scheme

that needs virtually no overhead.

Chapter 6

Conclusion

In this dissertation, we have studied low-energy and high-performance error correction for NAND flash memory systems. For this purpose, a low-density parity-check (LDPC) code-based decoding algorithm, scheduling methods, and low-power hardware architecture is investigated.

Chapter 3 of this dissertation is devoted to the study of informed dynamic scheduling algorithms. Since message updates are concentrated on a specific check node in the residual belief propagation (RBP), we propose an improved RBP (iRBP) algorithm that avoids such concentration and increases the convergence speed. We have also developed a syndrome-based mixed scheduling that dynamically conduct either the iRBP or the node-wise scheduling and obtained the result that outperforms all other schedules tested in this work.

In Chapter 4, we have proposed a conditional variable node update scheme for the normalized *a posteriori* probability (APP)-based algorithm to develop hardware efficient and low-power LDPC decoding systems. Simulation results show that the

proposed algorithm is robust to decoding failure and reduces switching activities of the decoding circuits. In addition, we have proposed a decoder hardware architecture for FG-LDPC codes that perform well with the normalized APP-based decoding. The architecture employs the normalized APP-based decoding, the serial scheduling algorithm, and the proposed conditional variable node update technique, which lead to simple functional units, halved decoding iterations, and low-power consumption, respectively. The decoder also adopts five-stage pipelined eight-way parallel architecture for high throughput and a few memory-reduction techniques. The developed decoder can achieve the minimum and maximum decoding throughput of 1.6 and 8.1 Gb/s, respectively, with the chip area of 63.08mm^2 in 0.13-μm CMOS process technology.

In Chapter 5 of the dissertation, we have studied the low-energy error correction of NAND flash memory through soft-decision error correction. The error performance of an LDPC code improves as the precision of data fed to the decoder increases, which demands an increased number of memory sensing operations in NAND flash memory. Although high precision data lowers the energy consumption in the LDPC decoder, multiple memory sensing operations obviously increase the energy consumption of NAND flash memory. We have analyzed the total energy consumption of the read operation for a NAND flash memory system equipping an LDPC decoder and proposed an LDPC decoder-assisted precision selection method that minimizes the total energy consumption.

The research works in this dissertation can contribute to the design of high-performance NAND flash memory systems that support strong error correction, high bandwidth, and low-power energy consumption.

Bibliography

[1] S. Gregori, A. Cabrini, O. Khouri, and G. Torelli, "On-chip error correcting techniques for new-generation flash memories," *Proc. IEEE*, vol. 91, no. 4, pp. 602–616, Apr. 2003.

[2] W. Liu, J. Rho, and W. Sung, "Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS'06)*, Oct. 2006, pp. 303–308.

[3] R. Micheloni *et al.*, "A 4Gb 2b/cell NAND flash memory with embedded 5b BCH ECC for 36MB/s system read throughput," in *ISSCC Dig. Tech. Papers*, Feb. 2006, pp. 497–506.

[4] F. Sun, S. Devarajan, K. Rose, and T. Zhang, "Design of on-chip error correction systems for multilevel NOR and NAND flash memories," *IET Circuits, Devices and Syst.*, vol. 1, no. 3, pp. 241–249, June 2007.

[5] T.-H. Chen, Y.-Y. Hsiao, Y.-T. Hsing, and C.-W. Wu, "An adaptive-rate error correction scheme for NAND flash memory," in *Proc. IEEE VLSI Test Symp. (VTS'09)*, May 2009, pp. 53–58.

[6] *64Gb, 128Gb, 256Gb, 512Gb Asynchronous/Synchronous NAND Features*, Micron Technology Inc., Boise, ID, 2009. [Online]. Available: http://www.micron.com/products/nand-flash/mlc-nand

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

[8] 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, Reg. Papers*, vol. 57, no. 10, pp. 2718–2728, Oct. 2010.

[9] K. Prall, "Scaling non-volatile memory below 30nm," in *NVSMW Tech. Dig.*, Aug. 2007, pp. 5–10.

[10] L. M. Grupp *et al.*, "Characterizing flash memory: Anomalies, observations, and applications," in *Proc. 42nd Annual IEEE/ACM Int. Symp. Microarch.*, Dec. 2009, pp. 24–33.

[11] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[12] ——, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[13] *Digital Video Broadcasting (DVB): Second Generation System for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications*, ETSI Std. 302 307, 2005.

[14] *IEEE Standard for Information Technology-Telecommunications and Information Exchange between Systems-Local and Metropolitan Area Networks-Specific Requirements Part 3: Carrier Sense Multiple Access with Collision*

Detection (CSMA/CD) Access Method and Physical Layer Specifications, IEEE Std. 802.3an, 2006.

[15] *IEEE Standard for Information technology–Local and metropolitan area networks–Specific requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11n, 2009.

[16] *IEEE Standard for Local and Metropolitan Area Networks–Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, IEEE Std. 802.16e, 2005.

[17] E. Sharon, S. Litsyn, and J. Goldberger, "An efficient message-passing schedule for LDPC decoding," in *Proc. 23rd IEEE Convention Elect. Electron. Eng. Israel*, Sept. 2004, pp. 223–226.

[18] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS'04)*, Oct. 2004, pp. 107–112.

[19] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

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

[21] D. Levin, E. Sharon, and S. Litsyn, "Lazy scheduling for LDPC decoding," *IEEE Commun. Lett.*, vol. 11, no. 1, pp. 70–72, Jan. 2007.

[22] A. Vila Casado, M. Griot, and R. Wesel, "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *Proc. IEEE Int. Conf. Commun. (ICC'07)*, June 2007, pp. 932–937.

[23] ——, "LDPC decoders with informed dynamic scheduling," *IEEE Trans. Commun.*, vol. 58, no. 12, pp. 3470–3479, Dec. 2010.

[24] S. Kim, K. Ko, J. Heo, and J.-H. Kim, "Two-staged informed dynamic scheduling for sequential belief propagation decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 13, no. 3, pp. 193–195, Mar. 2009.

[25] S. Kim, S. Lee, J. Heo, and C. Rim, "Adaptive mixed scheduling for correcting errors in trapping sets of LDPC codes," *IEEE Commun. Lett.*, vol. 14, no. 7, pp. 664–666, July 2010.

[26] S. Kim, "Trapping set error correction through adaptive informed dynamic scheduling decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 16, no. 7, pp. 1103–1105, July 2012.

[27] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[28] P. Ciao, G. Colavolpe, and L. Fanucci, "A parallel VLSI architecture for 1-Gb/s, 2048-b, rate-1/2 turbo Gallager code decoder," in *Proc. Euromicro Symp. Digital System Design (DSD'04)*, Aug. 2004, pp. 174–181.

[29] Z. Wang and Z. Cui, "Low-complexity high-speed decoder design for quasi-cyclic LDPC codes," *IEEE Trans. VLSI Syst.*, vol. 15, no. 1, pp. 104–114, Jan. 2007.

[30] ——, "A memory efficient partially parallel decoder architecture for quasi-cyclic LDPC codes," *IEEE Trans. VLSI Syst.*, vol. 15, no. 4, pp. 483–488, Apr. 2007.

[31] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 985–994, Aug. 2009.

[32] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Linköping University, S-581 83 Linköping, Sweden, 1996.

[33] M. P. C. Fossorier, M. Mihaljević, 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.

[34] M. M. Mansour and N. R. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, Mar. 2006.

[35] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29 mm^2 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 μm CMOS process," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 672–683, Mar. 2008.

[36] B. Xiang, R. Shen, A. Pan, D. Bao, and X. Zeng, "An area-efficient and low-power multirate decoder for quasi-cyclic low-density parity-check codes," *IEEE Trans. VLSI Syst.*, vol. 18, no. 10, pp. 1447–1460, Oct. 2010.

[37] P. Urard *et al.*, "A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes," in *ISSCC Dig. Tech. Papers*, vol. 1, Feb. 2005, pp. 446–609.

[38] P. Urard, L. Paumier, V. Heinrich, N. Raina, and N. Chawla, "A 360mW 105Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes enabling satellite-transmission portable devices," in *ISSCC Dig. Tech. Papers*, Feb. 2008, pp. 310–311.

[39] S. Muller, M. Schreger, M. Kabutz, M. Alles, F. Kienle, and N. Wehn, "A novel LDPC decoder for DVB-S2 IP," in *Proc. Design, Automation, Test in Eur. Conf. Exhibition (DATE'09)*, Apr. 2009, pp. 1308–1313.

[40] G. Dong, N. Xie, and T. Zhang, "On the use of soft-decision error-correction codes in NAND flash memory," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 2, pp. 429–439, Feb. 2011.

[41] J. Wang, T. Courtade, H. Shankar, and R. Wesel, "Soft information for LDPC decoding in flash: Mutual-information optimized quantization," in *Proc. IEEE Globecom*, Dec. 2011, pp. 1–6.

[42] D. Lee and W. Sung, "Estimation of NAND flash memory threshold voltage distribution for optimum soft-decision error correction," *IEEE Trans. Signal Process.*, vol. 61, no. 2, pp. 440–449, Jan. 2013.

[43] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[44] J. Kim, D. Lee, and W. Sung, "Performance of rate 0.96 (68254, 65536) EG-LDPC code for NAND Flash memory error correction," in *Proc. IEEE Int. Conf. Commun. (ICC'12), Workshop Emerging Data Storage Technol.*, June 2012.

[45] J. Kim, J. Cho, and W. Sung, "A high-speed layered min-sum LDPC decoder for error correction of NAND Flash memories," in *Proc. IEEE Int. Midwest Symp. Circuits Syst. (MWSCAS'11)*, Aug. 2011.

[46] J. Kim and W. Sung, "Low-energy error correction of NAND Flash memory through soft-decision decoding," *EURASIP Journal on Advances in Signal Processing*, vol. 2012, no. 1, p. 195, Sept. 2012.

[47] ——, "A rate-0.96 LDPC decoding VLSI for soft-decision error correction of NAND flash memory," *IEEE Trans. VLSI Syst.*, 2013, accepted.

[48] ——, "Improved residual belief propagation algorithm and syndrome-based mixed scheduling for decoding of low-density parity-check codes," *IEEE Commun. Lett.*, Submitted.

[49] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sept. 1981.

[50] 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.

[51] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1988.

[52] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996, reprinted in *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[53] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, "Turbo decoding as an instance of pearl's 'belief propagation' algorithm," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 140–152, Feb. 1998.

[54] X. Wei and A. N. Akansu, "Density evolution for low-density parity-check codes under Max-Log-MAP decoding," *Electron. Lett.*, vol. 37, no. 18, pp. 1125–1126, Aug. 2001.

[55] S.-Y. Chung, "On the construction of some capacity-approaching coding schemes," Ph.D. dissertation, M.I.T., Sept. 2000.

[56] E. Eleftheriou, T. Mittelholzer, and A. Dholakia, "Reduced-complexity decoding algorithm for low-density parity-check codes," *Electron. Lett.*, vol. 37, no. 2, pp. 102–104, Jan. 2001.

[57] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *Proc. IEEE Globecom*, vol. 2, Dec. 2001, pp. 1036–1036.

[58] J. Chen and M. P. C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Trans. Commun.*, vol. 50, no. 3, pp. 406–414, Mar. 2002.

[59] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proc. IEEE Globecom*, vol. 2, Dec. 2001, pp. 1021–1025.

[60] 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.

[61] ——, "Decoding low-density parity check codes with normalized APP-based algorithm," in *Proc. IEEE Globecom*, vol. 2, Nov. 2001, pp. 1026–1030.

[62] F. Guilloud, E. Boutillon, and J.-L. Danger, "λ-min decoding algorithm of regular and irregular LDPC codes," in *Proc. 3rd Int. Symp. Turbo Codes, Related Topics*, Sept. 2003, pp. 451–454.

[63] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding low-density parity-check LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 4, pp. 549–554, Apr. 2005.

[64] S. L. Howard, C. Schlegel, and V. C. Gaudet, "A degree-matched check node approximation for LDPC decoding," in *Proc. Int. Symp. Inf. Theory (ISIT'05)*, Sept. 2005, pp. 1131–1135.

[65] M. Jiang, C. Zhao, L. Zhang, and E. Xu, "Adaptive offset min-sum algorithm for low-density parity check codes," *IEEE Commun. Lett.*, vol. 10, no. 6, pp. 483–485, June 2006.

[66] J. Zhang, M. P. C. Fossorier, D. Gu, and J. Zhang, "Two-dimensional correction for min-sum decoding of irregular LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 3, pp. 180–182, Mar. 2006.

[67] H. Zhong, W. Xu, N. Xie, and T. Zhang, "Area-efficient min-sum decoder design for high-rate quasi-cyclic low-density parity-check codes in magnetic recording," *IEEE Trans. Magn.*, vol. 43, no. 12, pp. 4117–4122, Dec. 2007.

[68] V. Savin, "Self-corrected min-sum decoding of LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT'08)*, July 2008, pp. 146–150.

[69] D. Oh and K. K. Parhi, "Min-sum decoder architectures with reduced word length for LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 1, pp. 105–115, Jan. 2010.

[70] J. Chen and M. P. C. Fossorier, "Density evolution for BP-based decoding algorithms of LDPC codes and their quantized versions," in *Proc. IEEE Globecom*, vol. 2, Nov. 2002, pp. 1378–1382.

[71] J. Heo, "Analysis of scaling soft information on low density parity check code," *Electron. Lett.*, vol. 39, no. 2, pp. 219–221, Jan. 2003.

[72] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[73] D. Oh and K. K. Parhi, "Performance of quantized min-sum decoding algorithms for irregular LDPC codes," in *Proc. IEEE Symp. Circuits Syst. (ISCAS'07)*, May 2007, pp. 2758–2761.

[74] S. Kim, G. Sobelman, and H. Lee, "Adaptive quantization in min-sum based irregular LDPC decoder," in *Proc. IEEE Symp. Circuits Syst. (ISCAS'08)*, May 2008, pp. 536–539.

[75] C.-H. Chung, Y.-L. Ueng, M.-C. Lu, and M.-C. Lin, "Adaptive quantization for low-density-parity-check decoders," in *Proc. Int. Symp. Inf. Theory, Its Appl. (ISITA'10)*, Oct. 2010, pp. 13–18.

[76] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[77] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," in *Proc. 22nd Conf. Uncertainty Artificial Intelligence*, July 2006.

[78] J.-H. Kim, M.-Y. Nam, and H.-Y. Song, "Variable-to-check residual belief propagation for LDPC codes," *Electronics Letters*, vol. 45, no. 2, pp. 117–118, Jan. 2009.

[79] Y. Gong, X. Liu, W. Yecai, and G. Han, "Effective informed dynamic scheduling for belief propagation decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 59, no. 10, pp. 2683–2691, Oct. 2011.

[80] H.-C. Lee, Y.-L. Ueng, S.-M. Yeh, and W.-Y. Weng, "Two informed dynamic scheduling strategies for iterative LDPC decoders," *IEEE Trans. Commun.*, vol. 61, no. 3, pp. 886–896, Mar. 2013.

[81] Y. Mao and A. Banihashemi, "Decoding low-density parity-check codes with probabilistic scheduling," *IEEE Commun. Lett.*, vol. 5, no. 10, pp. 414–416, Oct. 2001.

[82] H. Kfir and I. Kanter, "Parallel versus sequential updating for belief propagation decoding," *Physica A*, vol. 330, no. 1-2, pp. 259–270, Dec. 2003.

[83] P. Radosavljevic, A. de Baynast, and J. Cavallaro, "Optimized message passing schedules for LDPC decoding," in *Proc. 39th Asilomar Conf. Signals, Syst. Computers*, Oct. 2005, pp. 591–595.

[84] V. Savin, "Iterative LDPC decoding using neighborhood reliabilities," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT'07)*, June 2007, pp. 221–225.

[85] O. Golov and O. Amrani, "Edge-based scheduled BP in LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT'07)*, June 2007, pp. 2376–2380.

[86] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[87] D. J. C. MacKay. Encyclopedia of sparse graph codes. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[88] Z. Cui, Z. Wang, and Y. Liu, "High-throughput layered LDPC decoding architecture," *IEEE Trans. VLSI Syst.*, vol. 17, no. 4, pp. 582–587, Apr. 2009.

[89] Q. Huang, Q. Diao, S. Lin, and K. Abdel-Ghaffar, "Cyclic and quasi-cyclic LDPC codes on constrained parity-check matrices and their trapping sets," *IEEE Trans. Inf. Theory*, vol. 58, no. 5, pp. 2648–2671, May 2012.

[90] H. Tang, J. Xu, S. Lin, and K. A. S. Abdel-Ghaffar, "Codes on finite geometries," *IEEE Trans. Inf. Theory*, vol. 51, no. 2, pp. 572–596, Feb. 2005.

[91] S. Kim, K.-I. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 11, pp. 1455–1464, Nov. 1998.

[92] E. Zimmermann, G. Fettweis, P. Pattisapu, and P. K. Bora, "Reduced complexity LDPC decoding using forced convergence," in *Proc. Int. Symp. Wireless Personal Multimedia Commun. (WPMC'04)*, Sept. 2004, pp. 12–15.

[93] A. Blad, O. Gustafsson, and L. Wanhammar, "An early decision decoding algorithm for LDPC codes using dynamic thresholds," in *Proc. Eur. Conf. Circuit Theory Design*, Aug.-Sep. 2005, pp. III/285–III/288.

[94] J. Cho, J. Kim, and W. Sung, "VLSI implementation of a high-throughput soft-bit-flipping decoder for geometric LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 5, pp. 1083–1094, May 2010.

[95] *Open NAND Flash Interface Specification*, ONFI Std. 2.1, 2009.

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

[97] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Jul.-Aug. 1999.

[98] G. Dong, Y. Pan, N. Xie, C. Varanasi, and T. Zhang, "Estimating information-theoretical NAND flash memory storage capacity and its implication to mem-

ory system design space exploration," *IEEE Trans. VLSI Syst.*, vol. 20, no. 9, pp. 1705–1714, Sept. 2012.

[99] C. M. Compagnoni, M. Ghidotti, A. L. Lacaita, A. S. Spinelli, and A. Visconti, "Random telegraph noise effect on the programmed threshold-voltage distribution of flash memories," *IEEE Electron Device Lett.*, vol. 30, no. 9, pp. 984–986, Sept. 2009.

[100] K.-D. Suh *et al.*, "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1149–1156, Nov. 1995.

[101] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron Device Lett.*, vol. 23, no. 5, pp. 264–266, May 2002.

[102] P. Poliakov, P. Blomme, M. M. Corbalan, J. V. Houdt, and W. Dehaene, "Cross-cell interference variability aware model of fully planar NAND Flash memory including line edge roughness," *Microelectron. Reliab.*, vol. 51, no. 5, pp. 919–924, May 2011.

국문 초록

반도체 공정의 미세화에 따라 비트 에러율이 증가하는 낸드 플래시 메모리에서 고성능 에러 정정 방법은 필수적이다. Low-density parity-check (LDPC) 부호와 같은 연판정 에러 정정 부호는 뛰어난 에러 정정 성능을 보이지만, 높은 구현 복잡도로 인해 플래시 메모리 시스템에 적용되기 힘든 단점이 있다. 본 논문에서는 LDPC 부호의 효율적인 복호를 위해 고성능 메시지 전파 스케줄링 방법과 저 복잡도 복호 알고리즘을 제안한다. 특히 finite geometry (FG) LDPC 부호에 대한 효율적인 디코더 아키텍쳐를 제안하며, 구현된 디코더를 이용하여 낸드 플래시 메모리에 대해 연판정 복호시의 에너지 소모량에 대해 연구한다.

본 논문의 첫 번째 부분에서는 동적 스케줄링 (informed dynamic scheduling, IDS) 알고리즘의 성능향상 방법에 대해 연구한다. 이를 위해 우선 기존의 가장 빠른 수렴 속도를 보이는 IDS 알고리즘인 레지듀얼 신뢰 전파 (residual belief propagation, RBP) 알고리즘의 동작 특성을 분석하고, 이를 바탕으로 특정 노드에 메시지 갱신이 집중되는 것을 방지하여 RBP 알고리즘의 수렴속도를 증가시킨 improved RBP (iRBP) 알고리즘을 제안한다. 또한 iRBP의 뛰어난 수렴속도와 기존의 NS 알고리즘의 우수한 에러 정정 능력을 모두 갖춘 신드롬 기반의 혼합 스케줄링 (mixed scheduling) 방법을 제안한다. 끝으로 다양한 부호율의 LDPC 부호에 대한 모의실험을 통해 제안된 신드롬 기반의 혼합 스케줄링 방법이 본 논문에서 시험된 다른 모든 스케줄링 알고리즘의 성능을 능가함을 확인하였다.

논문의 두 번째 부분에서는 복호 실패시 많은 비트 에러를 발생시키는 *a posteriori* probability (APP) 알고리즘의 개선 방안에 방안을 제안한다. 또한 빠른

수렴속도와 우수한 에러 마루 (error-floor) 성능으로 데이터 저장장치에 적합한 FG-LDPC 부호에 대해 제안된 알고리즘이 적용된 하드웨어 아키텍처를 제안하였다. 제안된 아키텍처는 높은 노드 가중치를 가지는 FG-LDPC 부호에 적합하도록 쉬프트 레지스터 (shift registers)와 SRAM 기반의 혼합 구조를 채용하며, 높은 처리량을 얻기 위해 파이프라인된 병렬 아키텍처를 사용한다. 또한 메모리 사용량을 줄이기 위해 세 가지의 메모리 용량 감소 기법을 적용하며, 전력 소비를 줄이기 위해 두 가지의 저전력 기법을 제안한다. 본 제안된 아키텍처는 부호율 0.96의 (68254, 65536) Euclidean geometry LDPC 부호에 대해 0.13-μm CMOS 공정에서 구현하였다.

마지막으로 본 논문에서는 연판정 복호가 적용된 낸드 플래시 메모리 시스템의 에너지 소모를 낮추는 방법에 대해 제안한다. 연판정 기반의 에러 정정 알고리즘은 높은 성능을 보이지만, 이는 플래시 메모리의 센싱 수와 에너지 소모를 증가 시키는 단점이 있다. 본 연구에서는 앞서 구현된 LDPC 디코더가 채용된 낸드 플래시 메모리 시스템의 에너지 소모를 분석하고, LDPC 디코더와 BCH 디코더 간의 칩 사이즈와 에너지 소모량을 비교하였다. 이와 더불어 본 논문에서는 LDPC 디코더를 이용한 센싱 정밀도 결정 방법을 제안한다. 본 연구를 통해 제안된 복호 및 스케줄링 알고리즘, VLSI 아키텍쳐, 그리고 읽기 정밀도 결정 방법을 통해 낸드 플래시 메모리 시스템의 에러 정정 성능을 극대화 하고 에너지 소모를 최소화 할 수 있다.

주요어 : 낸드 플래시 메모리, 동적 스케줄링, LDPC 부호, 연판정 센싱 방법, 연판정 오류 정정

학번 : 2009-30185

감사의 글

이제 지난 4년 반의 박사과정이 모두 끝났습니다. 학위 논문 심사와 논문 수정으로 바빴던 지난 몇 달 동안 정말 수없이 많은 밤을 지새우며 정신 없이 보냈는데, 어느덧 제본소로 논문을 보낼 시간이 가까워 오니 곧 졸업한다는 점이 비로소 실감이 납니다. 이 한 권의 논문이 완성되기까지 정말 많은 분들의 도움이 있었습니다. 그 분들께 이 지면을 빌어 감사의 마음을 전하고자 합니다.

우선 저의 지도 교수님이신 성원용 교수님께 깊은 감사를 드립니다. 석사과정 2 년을 포함하여 총 6년 반 동안 교수님께서는 열과 성을 다하여 저를 지도해 주셨습니다. 특히, 교수님의 하드웨어 아키텍쳐와 고정 소수점에 대한 강의는 이 논문이 완성되는 데에 정말 큰 도움이 되었습니다. 또한, 학위 과정 동안 제 논문들을 정성껏 수정해주신 점에 대해 진심으로 감사 드립니다. 그리고 교수님께서 말씀해주신 수많은 명언들을 잊지 않겠습니다. 정말 감사 드립니다. 학위 논문 심사에 귀중한 시간을 할애해 주신 노종선 교수님, 채수익 교수님, 이정우 교수님, 그리고 조준호 박사님께 깊은 감사를 드립니다. 심사 위원님의 값진 조언으로 인해 이 논문이 더욱 완성도 있게 마무리될 수 있었습니다.

연구실의 선·후배님들에게도 감사의 마음을 전하고 싶습니다. 박사 후 연구원으로 계시며 연구에 대한 조언뿐만 아니라 제 진로에 대한 고민을 상담해 주신 안재우 박사님, 연구실의 든든한 버팀목이셨던 호석이 형과 원철이 형, 하드웨어 구현에 대해 알려준 효진이 형, 연구실의 음성인식과제를 성공적으로 이끌며 타의 모범이 된 기선이 형, 하드웨어 합성과 PNR하는데 큰 도움이 되었던 상식선배, 연구실의 신입생이었던 제게 여러 가지 도움을 준 경환이 형,

늘 긍정적인 자세로 연구에 매진하며 자칫 삭막할 수 있는 연구실 생활에 활력소가 되어준 영준이 형과 영규, 그리고 지아쿤, 박사 과정 초반 연구에 대한 고민을 함께 나누었던 백상이 형 모두에게 감사 드립니다. 또한 오트곤과 현우, 수진이, 민수, 디마에게도 앞으로 하는 일에 큰 성공이 있기를 기원합니다. 2010년 초반 음성인식 MPW를 하며 함께 여러 날을 지새웠던 정욱이와 정석이에게도 고마운 마음을 전하며, 유학기간 동안 좋은 논문을 많이 쓰길 기원합니다. 그리고 제가 LDPC 부호에 대한 연구를 시작했을 때부터 지금까지 연구에 대한 깊은 통찰력으로 제게 너무나 큰 도움을 주고 계신 준호 형과, 낸드 플래시 메모리 채널을 구현하여 저의 연구에 많은 도움을 준 동환이에게 정말 고맙다는 말을 전하고 싶습니다. 특히, 저의 수많은 고민을 함께 나눈 연구실 동기인 지은이와 성건이에게 진심으로 고맙다는 말을 전하며 앞으로 좋은 일만 가득하기를 기원합니다. 그리고, 현재 재학 중인 연구실 후배 사지드, 준희, 규연이, 승열이, 민재, 동윤이, 창헌이, 성호 모두 열심히 연구하여 뜻한 바를 이루고 졸업하길 바랍니다.

또한, 늘 저의 의견을 존중해 주시고 오랜 학위기간 동안 흔들리지 않고 학업에 전념할 수 있도록 도와주신 사랑하는 아버지와 어머니, 정말 고맙습니다. 앞으로도 부모님의 자랑스러운 아들이 될 수 있도록 더욱 노력하겠습니다. 그리고 힘들고 바쁘다는 핑계로 많은 얘기를 나누지 못했던 사랑하는 동생에게도 계획한 일이 순조롭게 진행되어 크게 성공하길 바랍니다. 그리고, 항상 저를 위해 기도해주셨던 사랑하는 조부모님과 외조부모님께도 진심으로 감사의 마음을 전합니다. 끝으로, 오랜 시간 옆에서 묵묵히 지켜보며 제게 큰 힘이 되어준 사랑하는 민영이에게 고마운 마음을 전하고 싶습니다.

2013년 8월 2일

김 종 홍