The impact of faulty memory bit cells on the decoding of spatially-coupled LDPC codes


Published in:
2015 49th Asilomar Conference on Signals, Systems and Computers

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
© 2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.
The Impact of Faulty Memory Bit Cells on the Decoding of Spatially-Coupled LDPC Codes


Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.
The Impact of Faulty Memory Bit Cells on the Decoding of Spatially-Coupled LDPC Codes

Jiandong Mu†, Aida Vosoughi†, Joao Andrade*, Alexios Balatsoukas-Stimming‡, Georgios Karakonstantis§, Andreas Burg‡, Gabriel Falcao*, Vitor Silva*, Joseph R. Cavallaro†
† Department of ECE, Rice Univ., Houston, TX, USA; * Instituto de Telecomunicações, Univ. of Coimbra, Portugal
§ Queen’s University, Belfast, UK ‡ Telecommunications Circuits Lab, EPFL, Lausanne, Switzerland;

Abstract—In this paper, we investigate the decoding performance of spatially-coupled LDPC codes in the case of faulty memory bit-cells within the storage modules of the decoder. Our study characterizes error resilience, by measuring the BER degradation from such errors and we focus on the application of error mitigation techniques that further aid the inherent error resilience. In particular, we propose mitigation strategies based on the use of methods that consider the algorithmic significance of each bit-cell fault, such as MSB protection and self-correction of messages.

I. INTRODUCTION

Low-density parity-check (LDPC) codes are error-correcting codes well-known for supporting high data rates and having only linear growth in decoding complexity under message passing decoding algorithms, such as belief propagation. Spatially-coupled LDPC (SC-LDPC) codes [1] are new capacity-approaching codes that provide superior error correcting performance with respect to block LDPC codes by combining the advantages of turbo codes and block LDPC codes. Therefore, SC-LDPC codes support high data rates as well as flexible code lengths and they also enable decoding output with fine granularity. Furthermore, the use of sliding-window decoding algorithms solves the typical problems associated with the high decoding latency of long LDPC block codes [2], [3], making them highly appealing to the realization of powerful and yet very low latency forward error correction (FEC) systems.

While semiconductor technology scaling continues to enhance the performance and efficiency of error correction encoder and decoder circuits for modern wireless communication systems, it also leads to considerable process variations that result in various hardware failures [4]. Embedded memories are particularly sensitive to technology variations in sub-45 nm technology nodes, meaning that they often fail to reliably retain the stored data. In addition, voltage scaling, which is commonly employed for low-power operation, worsens the memory bit-cell failure problem. Traditional memory fault mitigation techniques help in limiting the number of errors by using larger memory bit-cells or applying circuit-level error-correcting codes (ECCs), but unfortunately they incur high design overheads and are not energy-efficient. Therefore, as technology keeps scaling down, alternative cost-effective paradigms must be explored to address the inevitable memory unreliabilities.

Approximate computing is such an alternative design paradigm, where 100% reliable operation requirements are relaxed in order to limit the overhead of classical fault mitigation techniques [5]. Under this new paradigm, it is possible to reduce design overheads by allowing the use of low cost unreliable memories for FEC systems [6] which exploit the system level redundancy of the communication system itself [7], as well as some additional low-overhead error mitigation techniques.

Studies on the impact of arithmetic errors and faulty memories on FEC decoders have been explored for decoders of LDPC block codes [8]–[13], for turbo codes [14], [15], as well as for polar codes [16]. The study of the error resilience and decoding behavior of the promising class of SC-LDPC codes under unreliable memory storage, however, remains an open question. Due to the intricate decoding algorithms utilized for the decoding of SC-LDPC codes [3], the error resilience of the decoder is influenced by several factors: 1) the memories in the decoder’s architecture that are affected by bit-cell failures, 2) the error mitigation techniques that are used to address these errors, and, most importantly, 3) the decoding window and partial syndrome-check window dimensions and the overlap between consecutive decoding windows, among others. The last item, in particular, makes SC-LDPC codes distinct when compared to other ECCs [1], which have fewer degrees of freedom that can influence the error resilience to faults.

In this paper, we study the error-resilience of SC-LDPC decoders considering some of the above-mentioned factors. Moreover, we aim to propose energy-efficient fault mitigation solutions to enable the implementation of high-performance SC-LDPC decoding under unavoidable memory unreliabilities.

II. SPATIALLY-COUPL ED LDPC CODES

In this work, we consider a (3, 6, 72) SC-LDPC code with quasi-cyclic (QC) structure which is described by the parity check matrix described in Fig. 1. In the used protograph, there are 72 copies of sliding rectangles coupled together so that the variable node (VN) degree $d_v$ is 3 and the check node (CN) degree $d_c$ is 6 (excluding the first and last two VNs/CNs). Then the parity-check matrix entries are expanded by a factor $Z$,
whereupon each entry in the matrix is replaced by an identity-matrix of size $Z \times Z$ permuted by the entry value—similarly to how quasi-cyclic block LDPC codes are constructed [1], [3]. Thus, in Fig. 1 each entry denotes a $Z \times Z$ cyclically shifted identity-matrix in the SC-LDPC code.

It is possible to decode SC-LDPC codes like block LDPC codes, but this will leave the advantages purported by SC-LDPC codes untapped and the resource usage and latency of the resulting decoder would be impractical. Alternatively, in sliding-window decoding methods the entire matrix is not decoded at once, but bits are decoded within a smaller window, called a decoding window of size $W$, by applying a standard message-passing algorithm. Once the bits inside the partial syndrome-check window, which is a sub-window of size $W$, have converged, the decoding window is shifted to the right by a step $s$ so that its next position partially overlaps with the one that has just converged. This way, reliable log-likelihood ratios (LLRs) are fed to the next decoding window, thus accelerating the convergence speed.

III. UNRELIABLE MEMORY MODEL

The following methodology is applied to determine how the number of bit-cell failures translates to loss in bit error rate (BER). Information bits are first encoded and modulated into BPSK symbols. Then, the symbols are transmitted over an additive white Gaussian noise (AWGN) channel and, at the receiver side, symbols are demodulated and the corresponding LLRs are passed to the SC-LDPC decoder. To model the unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stuck-at fault model that matches the standard failure model of embedded unreliable memory storage, we consider the stick-at fault model that matches the standard failure model of embedded memories in nanometer CMOS affected by process variations [17]. In this model, each bit-cell fails independently with failure probability $P_s$. The bit-cell failure locations are assumed to be distributed uniformly across the entire memory. A memory block $m$ has capacity for $W_m$ words, with word width $B_m$ bits, meaning that the total number of bits is $S_m = W_m \times B_m$. The average number of stuck-at bit-cells is given by $E_m = \lfloor S_m \times P_s \rfloor$, which is always considered for the simulations carried out. For each Monte Carlo simulation run, a random block of information bits is generated and for each memory block $m$, $E_m$ cells are marked as stuck-at based on a uniform distribution. The considered SC-LDPC decoder architecture is shown in Fig. 2.

IV. PROPOSED BER DEGRADATION MITIGATION STRATEGIES

In this section we introduce two strategies to mitigate the effect of memory errors on the BER of SC-LDPC decoding, namely MSB protection and a self-correction technique.

A. MSB protection

In the LLR-based decoding domain the sign of the LLRs is used directly to perform bit-decisions. Thus, faults introduced by bit-cell failures on the most significant bit (MSB) (where the sign is stored) intuitively have a higher impact on the BER than faults in the remaining bits. This behavior has also been verified experimentally, e.g. [12]. Therefore, we use a simple unequal error mitigation strategy where only the MSB bits are protected, by using, e.g., larger and more reliable cells. We assume that the protected cells are no longer subject to bit-cell failures, i.e., $P_s = 0$.

B. MSA with Self-correction

Alternatively, instead of protecting the MSB by applying a bit-cell level technique, an algorithmic approach can be taken, relying on a self-correction mechanism. Qualitatively, the main idea behind the self-correction mechanism is that, after a certain number of iterations has been reached, the VN output messages should no longer change signs very frequently from one iteration to the next. Thus, it can be assumed with relative safety that any sign flips come from faulty bit-cells, and not from the algorithm itself. This idea was first applied to correcting the overestimation of certain LLR messages in the iterative LDPC decoding algorithm [18] and has shown its potential to mitigate the BER degradation resulting from noisy hardware in the min-sum algorithm (MSA) iterative LDPC decoder [9]. The min-sum algorithm that uses self-correction is denoted by self-corrected min-sum algorithm (SCMSA). An example of the self-correction procedure is shown in Fig. 3, under 2’s-complement.
Figure 3. VN update under MSA and SCMSA depicting the LLR exchange in 2’s-C representation. LLRs greater or equal than 0x80 are negative, otherwise are positive. Whenever the sign-bit is due for a change between the \((i-1)\)-th and \(i\)-th iteration, the SCMSA introduces an erasure, except if an erasure had already been introduced at the \((i-1)\)-th iteration.

Table I

<table>
<thead>
<tr>
<th>Experiments</th>
<th>Unreliable Memories</th>
<th>MSB Protection</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>None</td>
<td>N/A</td>
<td>Fig. 4, Fig. 6</td>
</tr>
<tr>
<td>II</td>
<td>Channel</td>
<td>No</td>
<td>Fig. 4a), Fig. 6a</td>
</tr>
<tr>
<td>III</td>
<td>Channel</td>
<td>Yes</td>
<td>Fig. 4a), Fig. 6a</td>
</tr>
<tr>
<td>IV</td>
<td>All</td>
<td>No</td>
<td>Fig. 4b), Fig. 5, Fig. 6b</td>
</tr>
<tr>
<td>V</td>
<td>All</td>
<td>Yes</td>
<td>Fig. 4b), Fig. 5, Fig. 6b</td>
</tr>
</tbody>
</table>

Decoding parameters: \(W=15\), \(W_s=8\) and \(s = 2 \times Z\)

V. EVALUATION METHODOLOGY

A. Monte Carlo simulations

We study the sensitivity of the windowed SC-LDPC decoder to faults introduced by bit-cell failures in the different memories via Monte Carlo simulations. Moreover, we also evaluate the fault mitigation techniques that exploit algorithm or bit-cell level schemes to reduce the impact of these faults. An equal mixture of both stuck-at 0 and stuck-at 1 faults are injected at a \(P_s\) probability in the different memories. The experiments carried out are summarized in Table I. First, we establish the BER performance of the SC-LDPC code when the memories operate reliably (experiment I). Then, we introduce errors in the channel memory only and on all of the memories (experiments II and IV, respectively). This study is performed for a range of bit-cell failure probabilities \(P_s\).

B. Density evolution

Density evolution (DE) is a method for analyzing the performance of ensembles of LDPC codes in the limit of infinite blocklength without the need for lengthy Monte Carlo simulations [19]. The main idea behind DE is that, under certain conditions, the probability density functions of the messages that are exchanged within the decoder over the course of the decoding iterations can be tracked analytically. The bit error rate can then be easily derived from the aforementioned message densities. While DE is not entirely accurate for the analysis of a particular, finite-length, LDPC code (like the one given in Fig. 1), it still allows us to get an idea of how the performance of SC-LDPC codes is affected by the presence of memory faults.

Two crucial properties that make DE computationally tractable are the all-zero codeword assumption and message independence over the iterations. The stuck-at memory fault model we consider in this paper does not preserve message symmetry [13], meaning that the DE analysis cannot be restricted to the all-zero codeword. However, stuck-at faults can easily be converted to random bit-flips (which do preserve message symmetry) by assuming the existence of a simple xor-based randomization mechanism. Moreover, the presence of early termination introduces dependence between the messages of consecutive iterations. To avoid this issue, the DE results presented in the following section are obtained without early termination. For the same non-independence issue, it is unfortunately computationally infeasible to perform DE for the case where self-correction is used.

Due to space limitations, we do not describe DE for

![Figure 4](image-url)
SC-LDPC codes with faulty memories in detail and we focus more on the specific results for the $(3, 6, 72)$ SC-LDPC code ensemble to which the code in Fig. 1 belongs. We note that the DE procedure for faulty SC-LDPC code decoding can be derived relatively easily by combining the DE for SC-LDPC codes given in [20] with the faulty DE equations given in [13].

VI. SIMULATION RESULTS

In our simulations we follow the unreliable memory error model and simulation methodology introduced in Section III and Section V. For all Monte Carlo simulations and DE results, the LLRs are represented in 8-bit fixed-point, using a Q6.2 representation, i.e., 6 bits for sign and magnitude and 2 bits for decimal representation. A uniform distributed error with a probability of $P_s \in \{10^{-4}, 10^{-3}\}$ is injected into the memory.

A. Monte Carlo simulations

![Graph](image1)

Figure 5. SC-LDPC BER under unreliable memory for fault injection rates of $P_s \in \{10^{-4}, 10^{-3}\}$ and for Experiments IV and V running the SCMSA.

1) MSB protection: When bit-cell failures occur only in the channel memory, as seen in Fig. 4a), the memory faults can significantly deteriorate the BER if the fault $P_s$ is high enough. For example, at a BER of $10^{-4}$, the SNR loss when the memory fault rate is $P_s=10^{-4}$ is only 0.15 dB. For a fault rate of $P_s=10^{-3}$, however, the BER is affected severely. Fortunately, MSB protection is very successful in mitigating the BER degradation due to memory faults for both $P_s=10^{-3}$ and $P_s=10^{-4}$, with the signal-to-noise ratio (SNR) loss being in the order of 0.1 dB.

When all memories suffer from bit-cell failures, as expected, the BER performance is further degraded as shown in Fig 4b). In this case, at a BER of $10^{-4}$, the SNR loss when the memory fault rate is $P_s=10^{-4}$ is about 0.2 dB, while for a memory fault rate of $P_s=10^{-3}$ the SNR loss is significant and higher than what we observe in our simulations. As in the previous case, MSB protection is very successful in mitigating the impact of memory faults on the BER performance. More specifically, for both memory fault rates, we observe that the SNR loss at a BER of $10^{-4}$ can again be limited to 0.1 dB.

2) Self-corrected MSA: The self-correction outperforms the plain MSA algorithm, as observed in Fig. 5. Thus, two interpretations to the obtained results are possible. First, with regards to the SCMSA BER performance under reliable memories, the MSB protection is shown to be also an effective technique to improve the BER performance of the SCMSA under unreliable memory to close to the case where the memory is reliable. On the other hand, the self-correction is able to improve on the performance of the MSA by itself, even if the sign-bit, on which it depends to introduce erasures, is not protected, but only for low fault-injection rates ($P_{s0}=10^{-4}$). For higher injection rates ($P_{s0}=10^{-3}$), the self-correction technique is highly dependent of the MSB protection to achieve the original SCMSA performance. Unlike the case where soft-errors are introduced by faulty arithmetic units [8], [9], the BER performance achieved when stuck-at memory bit-cells are present can only be improved to its optimal level when MSB protection is also provided.
B. Density evolution analysis

The DE results for the (3, 6, 72) SC-LDPC code ensemble (to which the code of Fig. 1 belongs) are presented in Fig. 6a) and Fig. 6b) for experiments I, II and III, and I, IV, and V, respectively. We observe that the results generally agree with the Monte Carlo simulations in the sense that the presence of errors in all memories has a much more significant effect on the BER than the presence of errors only in the channel memory and that MSB protection provides significant improvements in the BER. Note that, as DE is idealized, there is some expected mismatch between the BER vs. SNR curves. Nevertheless, DE seems to be a useful low-complexity tool for an initial evaluation, with more accurate and specific results still requiring Monte Carlo simulations.

VII. CONCLUSION

In this paper, we characterize the BER performance of the SC-LDPC decoder under unreliable memory storage with bit-cell stuck-at-faults. More specifically, the BER degradation due to memory errors is quantified and evaluated through both Monte Carlo simulations and density evolution analysis. In addition we evaluate the performance improvements obtained by using two low-overhead mechanisms to mitigate memory unreliabilities: 1) the protection of most significant bit of each word in the memory modules, 2) a message self-correction technique. We observed that both MSB protection and self-correction are very effective in mitigating memory faults in SC-LDPC code decoding, and that a combination of the two methods can reduce the BER degradation to essentially zero with respect to the fault-free case for memory fault rates as high as $10^{-3}$.

REFERENCES