High Speed Low Error Floor Hardware Implementation and Fast and Accurate Error Floor Estimation of LDPC Decoders

by

Sina Tolouei, M.Sc.

A dissertation submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario

September, 2014

©Copyright

Sina Tolouei, 2014

The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the dissertation

**High Speed Low Error Floor Hardware Implementation and
Fast and Accurate Error Floor Estimation of LDPC
Decoders**

submitted by **Sina Tolouei, M.Sc.**

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Professor Amir H. Banihashemi, Thesis Supervisor

Professor Vincent Gaudet, External Examiner

Professor Roshdy Hafez, Chair,
Department of Systems and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

September, 2014

Abstract

This thesis explores hardware implementation of low error floor and high speed low-density parity-check (LDPC) decoders. At first, a novel partially-parallel implementation of a protograph-based quasi-cyclic (QC) LDPC decoder, using a new column-layered message-passing schedule is presented. A new design for serial-input check nodes and switch networks, results in lower hardware complexity while having a higher throughput and comparable latency in comparison with all existing partially-parallel architectures.

Thereafter, we introduce a multi-step scheme for the input quantization of message-passing decoders for LDPC codes. The scheme is based on successive re-quantization and re-decoding of the input blocks that cause the decoder to be trapped in a trapping set, until the decoding is successful or a maximum number of re-quantization/re-decodings is reached. The proposed scheme, which is applicable to both regular and irregular codes, lowers the error floor significantly at the cost of small increase in complexity, memory and latency.

Furthermore, to facilitate the process of code design and performance evaluation in the error floor region, we design a fast and accurate technique to estimate the error floor of variable-regular LDPC codes under quantized iterative decoding algorithms. This technique, which is based on enumerating the dominant elementary trapping sets of the code, provides significant improvement over existing methods in terms of speed and accuracy.

Finally, we examine the harmfulness of trapping sets of variable-regular LDPC codes for soft-decision iterative decoding algorithms. We show that, other than the size of a trapping set, the number of unsatisfied check nodes and the trapping set's topological properties, the position of its subgraph in the Tanner graph of the code can also have an effect on its harmfulness. We also examine dominant trapping sets

of the quantized min-sum (MS) decoding algorithm and two of its variants (offset-MS and MS with successive relaxation (SR)), for both regular and irregular LDPC codes. We show that the distribution of dominant trapping sets of these algorithms is similar for regular codes and different for irregular codes. Based on these results, we evaluate the error floor performance of a decoder, consisting of MS, offset-MS and SR-MS decoding algorithms working in parallel.

To My Wife

Acknowledgments

First, I would like to thank my supervisor, Professor Amir H. Banihashemi for leading and encouraging me during my research. When I started as his M.A.Sc. student nine years ago, my dream was to acquire a set of skills. Amir helped me to achieve them with invaluable advice and excellent guidance.

I would also like to thank M. Samy Hosny, the CEO of SiloconPro Inc. for supporting my research and providing me with his professional experience.

I can not express my appreciation to my wonderful wife, Nona who devoted her never ending love to me and was my partner towards success, and to my adorable son Diyaco, who was patient with Daddy being always busy. I also appreciate the love and support of my family, specially my parents who are always the source inspiration and encouragement for me.

And finally, many thanks to my thesis committee members for their helpful suggestions and discussions.

Table of Contents

Abstract	iii
Acknowledgments	vi
Table of Contents	vii
List of Tables	x
List of Figures	xii
Nomenclature	xvi
1 Introduction	1
1.1 Motivation	1
1.2 Related work	4
1.2.1 High-Throughput Decoder Architectures	4
1.2.2 Lowering the Error Floor of LDPC Codes	7
1.2.3 Error Floor Estimation of Quantized Iterative Decoders for Variable-Regular LDPC codes	8
1.2.4 Harmfulness of Trapping Sets LDPC Codes	9
1.3 Thesis Contributions	11
1.4 Thesis Organization	13

2	Background and Definitions	14
2.1	Protograph-Based QC-LDPC Codes and Multilevel Liftings	14
2.2	Channel Model	16
2.3	Decoding algorithm	17
2.3.1	MS Decoding Algorithm	19
2.3.2	Offset-MS Decoding Algorithm	20
2.3.3	SR-MS	21
2.4	Quantization Scheme	21
3	A High-Throughput Architecture for Protograph-Based Quasi-Cyclic LDPC Decoders	22
3.1	Introduction	22
3.2	Decoder Architectures for Protograph-Based QC-LDPC Codes and throughput analysis	26
3.2.1	BPP and PPP Architectures	26
3.2.2	High-Throughput PPP Architecture Based on Two-Level Liftings	28
3.3	PPP Architecture with Serial Check Nodes	29
3.3.1	SCN Structure	31
3.3.2	Serial Check Node Message-passing (SCMP)	34
3.3.3	Multiplexer Networks	36
3.3.4	Pipeline	39
3.3.5	Early termination	40
3.4	FPGA Implementations	41
3.5	ASIC Implementations	47
3.5.1	Medium-Length Decoder Implementations	49
3.5.2	Large-Length Decoder Implementations	52

3.6	Conclusion	53
4	Lowering the Error Floor of LDPC Codes Using Multi-Step Quantization	55
4.1	Introduction	55
4.2	The Proposed Scheme	56
4.3	Simulation Results	60
4.4	Conclusion	66
5	Estimating the Error Floor of variable-regular LDPC codes	67
5.1	Introduction	67
5.2	The Proposed Scheme	68
5.3	Experimental results	71
5.4	Conclusion	77
6	Harmfulness of trapping sets of LDPC codes	78
6.1	Introduction	78
6.2	Harmfulness of trapping sets of variable-regular LDPC codes	80
6.2.1	ETS layered growth algorithm	80
6.3	Dominant trapping sets of MS and two of its variants for regular and irregular LDPC codes	88
6.4	Conclusion	96
7	Conclusions and future work	97
7.1	Conclusions	97
7.2	Future work	99
	List of References	100

List of Tables

3.1	Resource utilization of different processing blocks of BPP, PPP and PPP-SCN architectures based on 5-bit quantization	43
3.2	Comparison of Resource Utilizations for a 24×24 Multiplexer Network Using Architectures of Fig. 3.6(a) and (b)	43
3.3	Comparison of resource utilization parameters of different architectures	44
3.4	Comparison of timing parameters of different architectures Considering a maximum number of iterations equal to 10	45
3.5	Comparison of medium-length LDPC decoders	50
3.6	Comparison of large-length LDPC decoders	54
4.1	Multiplicity of different classes of trapping sets after successive application of the re-quantization/re-decoding operations for the optimal scheme with $m = 6$ for the (1008,504) code at SNR=4.6 dB using MS	62
4.2	Latency Information of the Proposed Scheme for Trapped Cases . . .	65
5.1	$(a, b_{min}^{(a)})$ ETS classes with $a \leq 10$ and their multiplicity for $\mathcal{C}_1 - \mathcal{C}_3$ . .	72
6.1	Multiplicity and failure rate of each of the dominant trapping sets of the (1008, 504) code.	82
6.2	Multiplicity of the dominant trapping sets of the (775, 312) code. . .	87
6.3	Optimum waterfall clipping threshold and the error floor region SNR for all codes and algorithms, respectively.	89

6.4	Dominant trapping sets of each code and decoding algorithm and the percentage of appearance of dominant trapping sets at the output of the decoder.	91

List of Figures

2.1	An example of protograph lifting (a) Original base graph (protograph) with a parity-check matrix of size (3,2) (b) The application of a degree-4 cyclic lifting to the base graph of part (a) which makes a (12,4) protograph-based QC-LDPC code	16
2.2	An example of a two-level lifting: (a) The original base protograph (b) The application of two degree-2 liftings to the protograph of Part (a)	17
3.1	The (a) block-based partially-parallel (BPP) and (b) protograph-based partially-parallel (PPP) architectures for the decoder of the code with the parity-check matrix given in Fig. 2.1(b). Variable nodes, check nodes, memory units, barrel shifters for variable and check nodes are denoted by “V”, “C”, “M”, “BSV”, “BSC”, respectively.	26
3.2	The parity-check matrix of a (30,12) QC-LDPC code which is SCN compatible	31
3.3	The block diagram of the PPP-SCN architecture for the decoder of the code with the parity-check matrix of Fig. 3.2	32
3.4	The structure of serial check node (SCN)	33

3.5	SCMP for the QC-LDPC code of Fig. 2.1(b): (a)-(d) The first stage of the first iteration where variable nodes pass messages to SCNs. (e) The first clock cycle of the second stage of the first iteration where check nodes pass messages to variable nodes in the first copy of the protograph.(f) The second clock cycle of the second stage of one iteration where check nodes pass messages to variable nodes in the second copy of the protograph and variable nodes pass messages to check nodes in the first copy of the protograph (this corresponds to the first clock cycle of the first stage of the next iteration).	37
3.6	A 5×6 <i>VtoC</i> Multiplexer (a) Fast architecture, (b) Compact architecture	39
3.7	Pipeline diagram of SCMP, showing the edge of clock cycles corresponding to different stages.	40
3.8	The performance of QC-LDPC codes with a parity-check matrix of size (9216,7680) using different architectures	48
3.9	Bit error rate curves of different message-passing schedules. (...) The (2304, 1920) QC-LDPC code from [1] with TPMP schedule, MS algorithm, 8 iterations ($\circ$) and 16 iterations ($\square$). (-.-) The (2304, 1920) QC-LDPC code from [1] with row-layered schedule, MS algorithm, 8 iterations ($\circ$) and 16 iterations ($\square$). (- -) The (2304, 1920) QC-LDPC code from [1] with column-layered, MS algorithm, 8 iterations ($\circ$) and 16 iterations ($\square$). (—) The (2304, 1920) two-level lifted code, with SCMP schedule, MS algorithm, 8 iterations ($\circ$) and 16 iterations ($\square$).	53

4.1	Block error rate curves of the (1008,504) code of [2]. (a) MS ($\circ$) and offset-MS (∇) with (-.-) and without (—) optimal multi-step quantizer ($m = 5$). (b) Optimal ($\triangle$), greedy (+) and equal-increment ($\square$) schemes applied to offset-MS in the error floor region: the scheme is optimized at each SNR (—); and the scheme optimized at 4.6 dB is applied to every SNR (-.-)	61
4.2	Block error rate curves of the (775,312) code of [3]. (a) MS ($\circ$) and offset-MS (∇) with (-.-) and without (—) optimal multi-step quantizer ($m = 5$). (b) Optimal ($\triangle$), greedy (+) and equal-increment ($\square$) schemes applied to offset-MS in the error floor region: the scheme is optimized at each SNR (—); and the scheme optimized at 4 dB is applied to every SNR (-.-)	61
5.1	Normal graphs of three non-LSS ETS structures in a Tanner graph with variable degree 4 and girth 6.	69
5.2	Simulation and estimation results of $\mathcal{C}_1$ decoded by 3-bit MS ($C_{th} = 1.8$ and $I_{max} = 50$)	75
5.3	Simulation and estimation results of $\mathcal{C}_2$ decoded by 3-bit MS ($C_{th} = 1.2$ and $I_{max} = 30$)	76
5.4	Simulation and estimation results of $\mathcal{C}_3$ decoded by 3-bit MS ($C_{th} = 1.2$ and $I_{max} = 30$)	77
6.1	(a),(b),(c): Configurations $\{1\}$ and $\{2\}$ and $\{3\}$ of the (6,2) ETS class with an LSS number of 8 in a Tanner graph with variable degree 3 and girth 6, respectively.	81
6.2	Configuration $\{4\}$ of the (6,2) ETS class with an LSS number of 10 in a Tanner graph with variable degree 3 and girth 6, respectively. . . .	81

6.3	(a) A (3,3) ETS in a graph with a girth of $g = 6$ and variable and check node degrees of $d_v = 3$ and $d_c = 6$, respectively.(b) The result of one level growth.	83
6.4	(a) The $\{26,193,383,597,744,746\}$ (6,2) ETS. (b) The (8,4) trapping set resulted after the growth.	84
6.5	Configuration $\{2\}$ of the (6,2) ETS and the (7,3) trapping set as a result of its growth in the Tanner graph of the (1008, 504) code. . . .	85
6.6	Configuration $\{4\}$ of the (6,2) ETS and the (7,3) trapping set as a result of its growth in the Tanner graph of the (1008, 504) code . . .	85
6.7	Configuration $\{1\}$ of the (8,2) ETS and the (9,3) trapping sets as a result of its growth in the Tanner graph of the (1008, 504) code. . . .	86
6.8	Block error rate curves of the regular codes with different decoding algorithms. (a) Code $\mathcal{C}_1$ with MS ($\circ$), offset-MS (∇), SR-MS ($\times$) and parallel decoder ($\square$). (b) Code $\mathcal{C}_2$ with MS ($\circ$), offset-MS (∇), SR-MS ($\times$) and parallel decoder ($\square$).	92
6.9	(a) A (7,1) trapping set of $\mathcal{C}_3$. (b) A (8,1) trapping set of $\mathcal{C}_3$. (c) A (9,1) trapping set of $\mathcal{C}_3$	93
6.10	The (13,0) trapping set of $\mathcal{C}_3$	93
6.11	(a) A (3,2) trapping set of $\mathcal{C}_4$. (b) A (4,2) trapping set of $\mathcal{C}_4$. (c) A (5,2) trapping set of $\mathcal{C}_4$	94
6.12	Block error rate curves of the irregular codes with different decoding algorithms. (a) Code $\mathcal{C}_3$ with MS ($\circ$), offset-MS (∇), SR-MS ($\times$) and parallel decoder ($\square$). (b) Code $\mathcal{C}_4$ with MS ($\circ$), offset-MS (∇), SR-MS ($\times$) and parallel decoder ($\square$).	95

List of Abbreviations

Acronyms	Definition
LDPC	Low-Density Parity-Check
QC	Quasi-Cyclic
RS	Reed-Solomon
BP	Belief Propagation
MS	Min-Sum
SR	Successive Relaxation
SNR	Signal-to-Noise Ratio
BER	Bit Error Rate
FER	Frame Error Rate
LLR	Log-Likelihood Ratio
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase Shift Keying
SP	Sum-Product

FPGA	Field Programmable Gate Array
ASIC	Application-Specific Integrated Circuit
LUT	Look Up Table
BPP	Block-Based Partially-Parallel
PPP	Protograph-Based Partially-Parallel
TPMP	Two-Phase Message-Passing
TDMP	Turbo-Decoding Message-Passing
ETS	Elementary Trapping Set
SCN	Serial-input Check Node
SCMP	Serial Check node Message-Passing
TAR	Throughput-To-Area
LSS	Layered Superset

Chapter 1

Introduction

1.1 Motivation

After their rediscovery by Mackay and Neal [4], low-density parity-check (LDPC) codes [5] have attracted much attention in the research community and have been adopted in a number of standards such as WiMax [1], DVB-S2 [6], ITU-T G.hn [7] and IEEE 802.3an [8]. The main reason for the popularity of LDPC codes is their near Shannon limit performance with relatively low decoding complexity that increases only linearly with the block length. However, the hardware implementation of high throughput and low error floor LDPC decoders remains to be a challenge, and is considered the main road block in wide spread application of LDPC codes. This is particularly the case in applications such as optical communications and storage devices where high throughput and low error rate is required, or in real-time applications where low latency is essential. Many such applications also demand the error correction codes to have high rate. This further complicates the hardware design since high-rate LDPC codes are often represented by high-degree check nodes that are more complex to implement and introduce more delay in the critical path of the decoder. Thus, the design of high throughput LDPC decoders is still an active area of research.

On the other hand, error floor, generally defined as an abrupt change in the slope of the error rate curves versus signal-to-noise ratio (SNR), hampers the application of LDPC codes in low error rate applications. In practice, iterative message-passing algorithms are implemented in fixed-point. This motivates the research on methods to estimate and lower the error floors of quantized decoding algorithms. Error floor, is known to be caused by certain graphical structures of the code's Tanner graph [9], called *trapping sets* [10]. For a given LDPC code and iterative decoding algorithm, the knowledge of trapping sets harmful to the error correction performance is important. Although a great deal of research has been devoted to the harmfulness of trappings sets in many contexts, their effect on the failures of iterative decoding algorithms is still not well understood. More research is thus required to fully understand the trapping sets and their effect on the error floor.

In this thesis, we address the above challenges as described in the following. In the first part of the thesis, we present a new partially-parallel architecture for a multilevel lifted protograph-based quasi-cyclic (QC) LDPC decoder. The new architecture also induces a new column-layered message-passing schedule which outperforms the standard parallel (flooding) message-passing schedule in terms of error rate performance.

In the rest of the thesis, we focus the error floor of LDPC codes. First, we present a simple method to lower the error floor of quantized LDPC decoders. As mentioned earlier, in hardware, iterative message-passing algorithms are implemented in fixed-point. Quantization is thus required to map the real values at the output of the channel to a finite set of values at the input to the decoder. Zhao *et al.* [11] demonstrated that clipping as part of the quantization process creates an early error floor and that for a given LDPC code, there exists an optimal clipping threshold which results in the best performance in the waterfall region. We introduce a new multi-step quantization scheme to lower the error floor of LDPC codes. This scheme, which can

be applied to quantized decoding algorithms for regular and irregular LDPC codes, has minimum effect on complexity, memory and latency of the decoder.

Then, we introduce a fast and accurate method to estimate the error floor. Hardware platforms allow a substantial acceleration in the emulation of LDPC codes. However, simulating the performance of LDPC codes in hardware requires implementing and synthesizing the emulation engine, which may not be easily possible for example for randomly constructed codes of large length, or when many codes with different structures are being studied. Thus, fast and accurate estimation of the error floor of low-density parity-check (LDPC) codes under quantized iterative message-passing decoding algorithms [10, 12–15] is of great interest. In this thesis, we improve the method of [15] to estimate the error floor of LDPC codes faster and more accurately.

And finally, we study the harmfulness of trapping sets of LDPC codes. We show that the relative harmfulness of a trapping set depends on more factors than its size and the topological configuration of its subgraph. All possible topological configurations of trapping sets of variable-regular LDPC codes are well defined and categorized in [16]. We show that, in terms of harmfulness, the position of the induced subgraph of a trapping sets in the Tanner graph of the code also plays an important role. Furthermore, we show that for irregular codes, even the decoding algorithm has an effect on the harmfulness of trapping sets.

1.2 Related work

1.2.1 High-Throughput Decoder Architectures

Field programmable logic gate arrays (FPGA) have often been used to implement LDPC decoders [17–23]. This is either as an initial step to application-specific integrated circuit (ASIC) implementation, or as a substitute for the much slower software simulations particularly for applications, such as optical communications and storage devices, where low error rates need to be reached. ASIC implementations result in faster and larger decoders, and there are also many high-throughput ASIC designs in the literature [24–32]. Structured LDPC codes such as Protograph-based LDPC codes [33] and Reed-Solomon-based LDPC codes (RS-LDPC) [34] are suitable for hardware implementation and have been widely used in almost all high-throughput LDPC decoders.

Protograph-based LDPC codes [33] have been shown to have excellent performance both in waterfall and error floor regions [3, 35, 36], while having a simple encoder and decoder structure [17, 20, 23, 28, 31, 32, 35, 37–40]. In particular, if the permutation matrices that produce the parity-check matrix of the lifted code from that of the base code are cyclic, the resulted (lifted) code is quasi-cyclic (QC). Multilevel lifted or Hierarchical QC-LDPC codes [41] are also shown to have high-girth and good performance.

In the implementation of LDPC decoders, the architecture with the highest throughput is the fully-parallel architecture where the processing units corresponding to the complete Tanner graph of the code are laid out in circuitry [18, 19, 21, 26, 29, 30, 32, 42]. This however translates to the abundant use of resources and a large circuit, which even for short codes may not fit into the state-of-the-art FPGA's and occupies a large area in ASIC implementation.

Modified fully-parallel architectures [26, 29, 30] make it suitable for hardware implementation, at the cost of lowering the error correction performance or increasing the latency. For example, in [26], check node operations are partitioned into localized operations to reduce the routing congestion and increase the throughput. However, there is a 0.3 dB to 0.7 dB loss in performance compared to the non-partitioned architecture. Stochastic decoding is used in [29] to reduce the hardware complexity, which increases the throughput, but also causes an increase in the latency and a performance loss of 0.4 dB compared to a non-stochastic decoder. The architecture in [30] uses bit-serial message-passing to reduce the number of wires, which results in a high throughput, but the decoder still occupies a large area.

As a viable alternative for implementation, partially-parallel decoder architectures for LDPC codes with some structure, such as QC-LDPC decoders [17, 20, 22, 23, 28, 32, 35, 37, 38, 43–45] or Reed-Solomon based LDPC (RS-LDPC) decoders [24–27, 29, 30] have been an active area of research. QC-LDPC codes are receiving more attention since they have a simpler structure compared to RS-LDPC codes. In the context of choosing a good decoder architecture, the goal is often to strike a good balance between the decoder throughput/latency and its complexity. For structured LDPC codes, there are two general categories of partially-parallel decoder architecture referred to as *block-based partially-parallel (BPP) architecture* and *protograph-based partially-parallel (PPP) architecture*. In BPP, which can be used for both QC-LDPC and RS-LDPC codes, and has been used for example in [22, 24, 25, 27, 28, 43–45], a block of rows and/or columns of the parity-check matrix is processed in parallel, and within one iteration, the decoder processes different blocks sequentially. In PPP, which can be only used for QC-LDPC codes, and has been adopted for example in [17, 20, 23, 35, 37, 38], all variable nodes and check nodes in one copy of the protograph are updated simultaneously, and the decoder processes different copies of the protograph

serially within each iteration. Depending on the size of the protograph and the lifting degree, BPP and PPP offer different tradeoffs between the throughput and the decoder complexity. For a small protograph, say with about 10 to 20 nodes, and a large lifting degree, say about 100 to 1000, BPP would result in a higher throughput and a more complex decoder compared to PPP. The trend will however change if the size of the protograph is increased and/or the lifting degree is decreased.

In high-rate LDPC codes, check nodes often have very high degrees. Check nodes with a high degree occupy a large area and create a large delay in the critical path, which results in a low throughput for the decoder. To alleviate these problems, serial-input check nodes within the BPP architecture are used in [24, 27, 28]. Serial-input check nodes need less resources and introduce only a small delay.

In current decoder designs, different message-passing schedules are used to perform LDPC decoding. In Two-Phase Message-Passing (TPMP) schedule [5, 24–26, 29, 30, 32], each iteration consists of two subsequent phases of check nodes and variable nodes operations. In the first phase of TPMP, all check nodes process messages received from variable nodes and prepare messages for variable nodes, while in the second phase, all variable nodes process messages from check nodes and prepare new messages for check nodes.

Layered message-passing schedules [43, 46] provide an increase in convergence speed, defined as the average number of iterations in a specific SNR, by combining the two phases of the decoding algorithm. This comes at the cost of increases in memory, the number of clock cycles per iteration and latency.

In row-layered message-passing schedule, also known as layered message-passing or Turbo-Decoding Message-Passing (TDMP) [28, 31, 43], check nodes (rows of the parity check matrix) are processed in layers. Before the next layer of check nodes starts processing, variable nodes are updated with messages from current group of

check nodes, and subsequently they create a new set of messages for the next group of check nodes. This increases the convergence speed. The cost of this gain is the increase in the size of memory to store messages of check nodes in different layers, and the increase in the number of clock cycles needed to complete one iteration. This consequently results in a higher latency, and by increasing the time needed for each iteration, reduces the throughput.

In column-layered message-passing schedule, also known as shuffled decoding [27, 46, 47], variable nodes (columns of the parity check matrix) are processed in groups. After each group is processed, messages are sent to check nodes. Check nodes keep all their inputs stored, and after receiving a new input, reprocess all inputs and update the messages for variable nodes, before the next group of variable nodes start processing. This also results in an increase in the convergence speed and decoding performance [27, 46]. The main problem with this message-passing schedule is the increase in memory, since check nodes have to store all inputs in order to update their output after receiving a new input from variable nodes. Also, the complexity of check nodes is still high because they have to process all messages every time they receive new inputs.

1.2.2 Lowering the Error Floor of LDPC Codes

One approach to improve the error floor of LDPC codes is to modify the quantization scheme [48], [49], [50], or the decoding algorithm [51], [52] with the purpose of helping the decoder to avoid getting trapped in most problematic trapping sets. Post-processing techniques such as those proposed in [51] and [52], not only increase the complexity of decoding, but also require a detailed knowledge of dominant (most harmful) trapping sets, which may not be easy to obtain.

In [48], a dual quantization scheme was proposed to lower the error floor of belief

propagation (sum-product) algorithm. For a fixed number of quantization bits, this was performed by using different quantization step sizes to represent messages at the output of variable nodes and check nodes. In [49], [50], Zhang and Siegel studied the error floor performance of message-passing algorithms with uniform quantization in comparison with floating-point decoders and concluded that the decoder message saturation plays a key role in trapping the decoder in small trapping sets and in creating early error floors. To lower the error floor, they thus proposed a quasi-uniform quantizer (with a fixed small quantization step for small messages and exponentially larger step sizes for larger messages) that significantly increased the dynamic range of the quantizer.

1.2.3 Error Floor Estimation of Quantized Iterative Decoders for Variable-Regular LDPC codes

In [10], a semi-analytical technique was used to estimate the performance of a given LDPC code in the error floor region over the additive white Gaussian noise (AWGN) channel. In [12–14], importance sampling was used to estimate the frame error rate (FER) of LDPC decoders in the error floor region. In [15], an error floor estimation method for quantized symmetric iterative decoders of LDPC codes over the AWGN channel was proposed. It was assumed that the all-zero codeword was transmitted and the method was devised based on the enumeration and the examination of certain input vectors with small Euclidean distance to the most reliable vector $\vec{R}$ (the vector with the largest positive quantized value $Q = 2^{q-1} - 1$ in all coordinates). The input vectors were generated using a technique called *modified cycle enumeration*, where the *error locations* (coordinates with values less than Q) belonged to the set T_k consisting of the variable sets of all the cycles of length $g, \dots, g + 2k$ in the Tanner graph of the

code, with g being the *girth* of the Tanner graph. In modified cycle enumeration, the *error patterns* (coordinate values of error locations) were selected in the increasing order of their distance to $\vec{R}$. The generated input vectors were then applied to the iterative decoder under consideration and the error floor was estimated based on those input vectors that failed the decoder. Such input vectors were categorized in the sets $\tilde{E}_d$, $d = \tilde{d}_m, \dots$, where $\tilde{E}_d$ and $\tilde{d}_m$ were estimates for the set of input vectors at distance d from $\vec{R}$ that failed the decoder and the minimum distance of the failed input vectors to $\vec{R}$, respectively. The estimate for the error floor was obtained based on sets $\tilde{E}_d$.

Although the method of [15] was demonstrated to be very effective for short LDPC codes with variable degree 3, its application to block lengths larger than about 1000 and variable degrees larger than 3 is hampered by large computational complexity (see Section IV of [15]). In particular, to render an accurate estimate for the error floor of LDPC codes with larger variable degrees, larger values of k for the set T_k need to be selected. This in turn increases the size $|T_k|$ of T_k and consequently the complexity of the method. For example, for a randomly constructed regular code with $g = 6$ and variable- and check-node degrees of 5 and 10, respectively, increasing k from 1 to 2, will increase $|T_k|$ from about 218,000 by a factor of about 29 to about 6.3×10^6 [53]. For $q = 3$, this corresponds to increasing the number of test input vectors by a factor of roughly 200 from about 5×10^8 to about 10^{11} . This is while, based on our results, to obtain an accurate estimate, one may still need to increase k even further.

1.2.4 Harmfulness of Trapping Sets LDPC Codes

Trapping sets are normally classified by a pair (a, b) , in which a is the number of variable nodes whose induced subgraph has b check nodes of odd degree [10, 16, 48,

49, 54–59]. Trapping sets with smaller values of a and b are generally believed to be more harmful to iterative decoding.

The harmfulness of a trapping set is studied in many contexts, such as decoding algorithm [48], quantization scheme [49, 58] and graphical structure [10, 16, 56, 57]. Richardson defined elementary trapping sets (ETS) in [10] as those having only check nodes of degree one or two in their induced subgraphs. In his work, ETSs are shown to be dominant trapping sets. Karimi *et al.* [16] showed that ETSs that are *layered supersets* (LSS) of short cycles, i.e. the ones that are the result of incremental growth of short cycles in the Tanner graph of the code, are the most harmful ones.

In [54], Vasic *et al.* build a database of all possible topological configurations for trapping sets of size up to 8 in variable-regular LDPC codes with variable node degree 3. In their search algorithm, they considered transmission over binary symmetric channel (BSC) and Gallager A/B hard-decision algorithms. Using this database in [55], they discuss the relative harmfulness of trapping sets for the sum-product algorithm (SPA). Focusing on ETSs, they show that smaller trapping sets (with smallest a) are not necessarily the dominant trapping sets of the code. They also show that for trapping sets having the same number of variable nodes a , the ones having the smallest number of unsatisfied check nodes b are not always the most harmful. Furthermore, they show that for some trapping sets having the same a and b , some with specific topological configurations can be more harmful than others. They characterize the relative harmfulness of trapping sets based on their parent-child relationship, which they extract from the database generated in [54]. Based on this analysis, they select ETSs with specific configurations to avoid while constructing LDPC codes.

In [48], the authors show that the multiplicity of different trapping sets can have a relationship with the decoding algorithm. For two regular LDPC codes, they implement quantized sum-product and approximate sum-product decoding algorithms

such as min-sum (MS) and MS with unconditional correction [11] (also known as offset-MS). Using hardware simulation, they show that in implementation of quantized sum-product and approximate sum-product decoders for regular LDPC codes, the latter have lower error floor.

1.3 Thesis Contributions

The main contributions in this thesis are as follows:

- We develop a new serial-input check node design for the implementation of high-degree check nodes in a MS decoder of a protograph-based QC-LDPC code. We demonstrate that for high-rate codes, using the proposed architecture within the protograph-based partially-parallel (PPP) framework results in a competitive throughput/latency compared to the existing layered, partially-parallel and fully-parallel architectures with comparable circuitry. In addition to delivering competitive throughput/area/latency trade-off, the proposed architecture induces a new column-layered message-passing schedule which outperforms the standard parallel (flooding) message-passing schedule in terms of error rate performance. Synthesis results in a 65-nm CMOS technology are presented for medium-length (2304, 1920) and large-length (32768, 26624) QC-LDPC codes. These decoders have a silicon area of 1.02 mm² and 14.19 mm² and attain average throughput of 17.28 Gbps and 99.9 Gbps, respectively.
- We propose a simple scheme to lower the error floor of LDPC codes decoded by fixed-point message-passing decoders, without sacrificing the waterfall performance. Our proposed scheme is universal in the sense that it can be applied to both regular and irregular codes and to any message-passing decoder, and

results in an improvement of about 2 to 3 orders of magnitude in the error floor region for different codes and decoding algorithms.

- We present a fast and accurate technique for the error floor estimation of variable-regular LDPC codes. The results of this technique extend the reach of the estimation method of [15] to larger block lengths and larger variable degrees by significantly improving the speed and the accuracy in comparison with the modified cycle enumeration of [15].
- We study the relative harmfulness of trapping sets based on the position of their induced subgraph in the Tanner graph of the code. We show that, not all trapping sets with the same class (a, b) and the same topological configuration are as harmful. We demonstrate that the location of the induced subgraph of a trapping sets in the Tanner graph, has a significant effect on its harmfulness. While our findings can help to relax the constraint of removing a complete class or topological configuration of trapping sets in code design, like what was done in [55], justifying these observations will be an interesting topic for future research.
- We also examine dominant trapping sets of the quantized min-sum (MS) decoding algorithm and two of its variants (offset-MS and SR-MS), for both regular and irregular LDPC codes. We show that the distribution of dominant trapping sets of these algorithms is similar for regular codes and different for irregular codes. Based on these results, we evaluate the error floor performance of a decoder, consisting of MS, offset-MS and SR-MS decoding algorithms working in parallel and demonstrate that using such a decoder the error floor, particularly for irregular codes, can be significantly improved.

1.4 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, basic definitions and backgrounds are provided. In Chapter 3, we provide the proposed high throughput architecture and the new column-layered message-passing schedule. We also present FPGA and ASIC implementation results. The multi-step quantization scheme for lowering the error floor of LDPC codes is presented in Chapter 4. Chapter 5 includes the technique for estimating the error floor of variable-regular LDPC codes under quantized iterative decoding algorithms. In Chapter 6, we first study the harmfulness of trapping sets of variable-regular LDPC codes based on the position of their subgraph in the Tanner graph. We then study dominant trapping sets of MS decoding algorithm and two of its variants (offset-MS and SR-MS), for both regular and irregular LDPC codes. We conclude this thesis in Chapter 7 and highlight directions for future research.

Chapter 2

Background and Definitions

2.1 Protograph-Based QC-LDPC Codes and Multilevel Liftings

Consider a binary (n, k) LDPC code $\mathcal{C}$ of length n and dimension k with a parity-check matrix H with $m = n - k$ rows and n columns. Corresponding to H , code $\mathcal{C}$ can be represented by a Tanner graph $G = (V_b \cup V_c, E)$, where V_b and V_c are the sets of n variable nodes and m check nodes corresponding to the columns and the rows of H , respectively, and E is the set of edges in the graph, corresponding to the nonzero elements (ones) of H .

Consider the group C_N of the N circulant permutations defined by $\pi_d(i) = i + d \bmod N$, $d \in Z_N$, where $Z_N \triangleq \{0, \dots, N - 1\}$. Now, consider the following construction of a Tanner graph $\tilde{G} = (\tilde{V}_b \cup \tilde{V}_c, \tilde{E})$ from a Tanner graph $G = (V_b \cup V_c, E)$: We first make N copies of G such that for each node $v \in V = V_b \cup V_c$, we have N copies $\tilde{v} \triangleq \{v^0, \dots, v^{N-1}\}$ in $\tilde{V}$. For each edge $e = \{u, v\} \in E$ with $u \in V_b$ and $v \in V_c$, we apply a permutation $\pi^e \in C_N$ to the N copies of e in $\tilde{E}$ such that an edge $\{u^i, v^j\}$ belongs to $\tilde{E}$ if and only if $\pi^e(j) = i$. The set of these edges is denoted by $\tilde{e}$. The graph $\tilde{G}$ is called a *cyclic N -lifting* of G , and G is referred to as the *base graph*,

or *protograph* corresponding to $\tilde{G}$. To the lifted graph $\tilde{G}$, we associate an LDPC code $\tilde{\mathcal{C}}$, referred to as the *lifted code*, such that the $mN \times nN$ parity-check matrix $\tilde{H}$ of $\tilde{\mathcal{C}}$ is equal to the biadjacency matrix of $\tilde{G}$. Due to the circular nature of the permutations, the lifted code $\tilde{\mathcal{C}}$ is quasi-cyclic (QC) and thus easy to implement in hardware. The parity-check matrix $\tilde{H} = [\tilde{H}_{ij}]$ of the protograph-based QC-LDPC code $\tilde{\mathcal{C}}$ can be presented as an $m \times n$ array of $N \times N$ circulants, where each circulant $\tilde{H}_{ij}$ corresponds to all the parallel edges of the base graph between the i th check node and the j th variable node. If each such edge e has a circulant permutation $\pi^e = \pi_d$, we have $\tilde{H}_{ij} = \sum_e I^{(d)}$, where $I^{(d)}$ is a *circular permutation matrix* whose rows are obtained by cyclically shifting all the rows of the $N \times N$ identity matrix I_N by d to the right. For a base graph with no parallel edges, each circulant $\tilde{H}_{ij}$ has only one ‘1’ in each column and in each row. In this case, which is of particular interest here, the $m \times n$ matrix D consisting of the permutation shifts d is called *permutation shift matrix*. If $\tilde{H}_{ij} = 0$ for some values of i and j , the corresponding element of D is arbitrarily set to be ∞ .

It is easy to see that the degree distribution of the base code remains intact during the lifting process.

Example 1 *A toy example of the parity-check matrix of a $(12, 4)$ protograph-based QC-LDPC code is shown in Fig. 2.1(b). The empty locations in the matrix correspond to zero values. To obtain the code, a cyclic lifting of degree 4 is applied to the base graph of Fig. 2.1(a). In Fig. 2.1(a), variable nodes and check nodes are represented by circles and squares, respectively.*

The lifting process, as explained above, can be applied to a protograph multiple times, hence the term *multilevel lifting*. Multilevel liftings can be used for a variety of reasons. For example, the first level of lifting can be simply used to remove the parallel

Figure 2.1: An example of protograph lifting (a) Original base graph (protograph) with a parity-check matrix of size (3,2) (b) The application of a degree-4 cyclic lifting to the base graph of part (a) which makes a (12,4) protograph-based QC-LDPC code

edges of the base graph and produce a new (larger) base graph with no parallel edges [35]. This new base graph can then be lifted to the desired block length. Multilevel liftings can also be used as a technique for trading off resources versus throughput in hardware implementation of iterative decoders [60]. In such scenarios, a two-level lifting with degree $N = N_1 \times N_2$ can be designed, where N_1 and N_2 are the lifting degrees of the first and the second lifting, respectively. In a PPP architecture, the goal of the design can be to maximize N_1 and thus to maximize the number of processing units working in parallel (within the constraint of the available hardware resources) so that the throughput is consequently maximized.

Example 2 A two-level lifting of degree 4 has been applied to the base graph of Fig. 2.1(a) in Fig. 2.2, where each of the two levels of lifting has degree 2.

2.2 Channel Model

In an $(n, n-m)$ LDPC code C , any n -tuple v is a codeword in the code C if and only if $v \cdot H^t = 0$. The code rate is defined as $R = (n - m)/n$. The code is used for

Figure 2.2: An example of a two-level lifting: (a) The original base protograph (b) The application of two degree-2 liftings to the protograph of Part (a)

communication over a binary-input additive white Gaussian (AWGN) channel, where the input signal x corresponding to the code bit c is given by $x = 1 - 2c$. The output of the channel y is equal to $y = x + N$, where N is Gaussian with zero-mean and variance $\sigma^2 = \frac{1}{(2RE_b/N_0)}$, in which E_b is the average energy per information bit and N_0 is the one-sided power spectral density of the AWGN.

2.3 Decoding algorithm

LDPC codes are decoded using an iterative algorithm. The belief propagation (BP) decoding algorithm [5] is optimal for cycle-free graphs but is complex. In iterative decoding algorithms, messages are exchanged between variable nodes and check nodes.

Variable nodes are initially filled with a function of the channel output. Check nodes receive the initial beliefs from the neighboring variable nodes along the edges of the Tanner graph, and in return, they calculate the parity message for each connected variable node based on other connected variable nodes and send this extrinsic information back to variable nodes. In the next step, each variable node uses the received new extrinsic information and refines its initial decision. The sequence of bits in all variable nodes (vector v) should form a codeword to be accepted as a corrected sequence. Hence, $v.H^t=0$ is checked. If this condition is satisfied, we have a codeword. If the condition is not satisfied, variable nodes will create new messages for check nodes. Each variable node, for each connected check node, uses the information from the channel and messages from all other neighboring check nodes to create a new message for that check node. Then variable nodes send the messages back to check nodes. The iterative algorithm continues until we reach the maximum number of iterations or $v.H^t=0$ is satisfied and we have a codeword. The belief propagation decoding algorithm is usually implemented in log-likelihood ratio (LLR) domain due to better robustness (assuming finite precision) in this domain. All the multiplications are converted to additions in the logarithm domain. So in this domain, we have better numerical stability and low cost operation in variable nodes. We use the LLR domain to present messages. In the LLR domain, we use the notation $L(q_{ij})$ for the message passed from the variable node i to check node j , and $L(r_{ji})$ for the message from check node j to variable node i . The min-sum (MS) decoding algorithm is an approximation of BP in the LLR domain [61], which is much less complex. In the min-sum algorithm, there is no change in the iterative message passing and variable node calculations. Calculations in check nodes are however simplified to finding the minimum of extrinsic incoming messages and they are less complex compared to the BP algorithm. Different modifications are applied to the MS algorithm to improve

its performance. In [62] and [11], a correction method is introduced. In this method, an additive correction factor is applied to the magnitude of the output messages from the check nodes. In [62], this method is called offset BP-based algorithm. In [11], it is called MS with unconditional correction. In this thesis, we call this decoding algorithm “offset-MS”. Offset-MS is popular because it is simpler than BP but performs as well as BP or even in some cases better for finite-length LDPC codes [11]. Inspired by the dynamics of analog decoders, a new framework for iterative message-passing called “successive relaxation (SR)”, has been introduced in [63]. In this thesis, we apply SR to MS decoding algorithm in LLR domain and call it “SR-MS”. It is shown in [64] that with 7-bit quantization, SR-MS outperforms floating-point BP, particularly at high SNR values. MS, offset-MS and SR-MS decoding algorithms are used throughout this thesis.

2.3.1 MS Decoding Algorithm

The MS decoding algorithm is described by the following steps in each iteration:

Step 0) The initial messages at variable nodes are set to:

$$L(q_{ij}) = L(c_i) = y_i \quad (2.1)$$

Step 1) Check node update:

$$\begin{aligned} L(r_{ji}) &= \left(\prod_{i' \in V_j \setminus i} \alpha_{i'j} \right) \cdot \min_{i' \in V_j \setminus i} (\beta_{i'j}) \\ \alpha_{i'j} &= \text{sign}(L(q_{i'j})) \\ \beta_{i'j} &= |L(q_{i'j})| \end{aligned} \quad (2.2)$$

where $V_j \setminus i$ is the set of variable nodes connected to check node j excluding variable

node i .

Step 2) Decision at variable nodes:

$$L(Q_i) = L(c_i) + \sum_{j \in C_i} L(r_{ji})$$

$$\hat{c} = \begin{cases} 1 & \text{if } L(Q_i) < 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.3)$$

where C_i is the set of check nodes connected to variable node i and $\hat{c}$ is the estimate of the code bit i . The algorithm stops if $(\hat{c}_1, \dots, \hat{c}_n) \cdot H^t = 0$, or if the maximum number of iterations is reached.

Step 3) Variable node update:

$$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \setminus j} L(r_{j'i}) \quad (2.4)$$

These messages are sent to check nodes and step 1 is repeated till one of the exit conditions are met.

2.3.2 Offset-MS Decoding Algorithm

Offset-MS has the same variable node operation as MS does. The only difference is in the magnitude of the check node outgoing messages:

$$L(r_{ji}) = (\prod_{i' \in V_j \setminus i} \alpha_{i'j}) \cdot \max\left(\left| \min_{i' \in V_j \setminus i} (\beta_{i'j}) \right| - y_c, 0\right)$$

$$\alpha_{i'j} = \text{sign}(L(q_{i'j})) \quad (2.5)$$

$$\beta_{i'j} = |L(q_{i'j})|$$

where $V_j \setminus i$ is the set of variable nodes connected to check node j excluding variable node i and y_c is called the *additive correction factor*.

2.3.3 SR-MS

Compared to standard MS, SR-MS has the same initial messages for variable nodes, and the same check node operations. The variable node messages however, are calculated by:

$$L^{(\lambda+1)}(q_{ij}) = L^{(\lambda)}(q_{ij}) + \beta \left(\left[L(c_i) + \sum_{j \in C_i} L^{(\lambda)}(r_{ji}) \right] - L^{(\lambda)}(q_{ij}) \right) \quad (2.6)$$

where λ is the iteration number and β is called the relaxation factor.

2.4 Quantization Scheme

We consider a quantization scheme (q, c_{th}) , in which the received values are clipped symmetrically at a threshold $\pm c_{th}$, and then uniformly quantized in the range $[-c_{th}, c_{th}]$. There are $2^q - 1$ quantization intervals, symmetric with respect to the origin, and each represented by q quantization bits. Integer numbers $-(2^{q-1} - 1), \dots, (2^{q-1} - 1)$ are assigned to the quantization intervals.

Chapter 3

A High-Throughput Architecture for Protograph-Based Quasi-Cyclic LDPC Decoders

3.1 Introduction

In this chapter, we propose a new PPP-based decoder architecture for protograph-based multilevel lifted QC-LDPC codes. As the decoding algorithm, we use the widely popular min-sum algorithm [61]. The proposed architecture is based on a novel implementation of the column-layered decoding in which each check node processing unit has only one serial input and one serial output, and processes messages in an interleaved fashion along with the variable node processing units. At the end of each iteration, check nodes store some information consisting of their two inputs with the smallest and the second smallest magnitude along with the index of the former in registers. In the next iteration, as new messages are received from variable nodes, the same information is also stored and updated continuously in another set of registers. The output messages of check nodes to variable nodes are then formed using the smallest of the two sets according to the index of the variable node and following

the MS updating rule. This new architecture induces a new column-layered message-passing schedule, referred to as *serial check node message-passing (SCMP)*.

Our work fits into the general category of decoders with layered message-passing schedules (layered decoders). To evaluate the proposed decoder, first we compare it to two recent layered decoders of [28] and [27] which use column-layered and row-layered message-passing, respectively. Both these decoders use serial-input check nodes within BPP architecture. In the following, we use minimum throughput-to-area (TAR_min), and when available, average throughput-to-area (TAR_avg) as well as latency as comparison metrics. TAR_min and TAR_avg are calculated based on maximum and average number of iterations, respectively. To have a fair comparison, when calculating TAR_min, we consider a maximum number of iterations of 8 for all medium-length ASIC decoders.

The closest work in the literature to the proposed architecture is [27] which is a column-layered decoder with serial-input check nodes. The first difference between our work and that of [27], is in the way that the check node updates are performed. In [27], to perform check node updates, an ordered list of input messages of previous and current iterations is stored and updated (resorted) each time a new input arrives. The sorting blocks increase the complexity of check nodes. In SCMP, on the other hand, inside each serial-input check node (SCN), we store two sets of information, as described above, one for previous iteration, which is only updated at the end of each iteration, and one for the current iteration, which needs to be updated within the current iteration each time a new input arrives. The output message is then formed using two comparators and a multiplexer, which is much simpler than the sorting block designed in [27]. We will later show that SCMP, while having a lower complexity, has the same performance as the column-layered message-passing schedule in [27].

The second main difference with the work of [27] is that in our work, we use

the PPP architecture, and exploit a two-level lifting construction to maximize the throughput with a constraint on the available resources. Using FPGA implementation, we particularly show that as a result of combining SCMP with PPP, which we call PPP-SCN, and two-level lifting, we can develop a high-throughput PPP-based decoder with competitive performance, while consuming less resources. For example, we consider a (9216, 7680) QC-LDPC code. First, we construct this code using one-level lifting and implement the decoder using BPP and PPP architectures, and show that BPP results in a faster decoder with a larger resource utilization than PPP. Then we apply PPP-SCN to a two-level lifted code of the same size and rate, and show that we can achieve a better throughput and latency compared to BPP and at the same time consume less resources on the FPGA. FPGA implementation of the PPP-SCN decoder, on an Altera Stratix IV EP4SGX530 FPGA achieves a throughput and latency of 8.5 Gbps and 2.26 μ sec, respectively, with a total resource utilization of 64%. To compare with the decoder implemented in [27], we synthesize the decoder of a medium-length (2304, 1920) QC-LDPC code, which has a similar code length and the same rate as the (2048, 1723) RS-LDPC code implemented in [27]. We synthesize the decoder in a 65-nm CMOS technology. This decoder has a minimum throughput of 7.71 Gbps, and occupies a core area of 1.02 mm². The minimum throughput-to-area (TAR_{min}) of the decoder is thus 7.56 Gbps/mm² which is about 70% higher than the TAR_{min} 4.45 Gbps/mm² of the column-layered decoder in [27]. The latency of the proposed decoder, however, is 300 nsec, which is about 40% higher than that of the decoder in [27] with a latency of 211 nsec.

As the second layered decoder, we compare our work to the decoder in [28], which is a row-layered decoder with serial-input check nodes. The main disadvantage of the row-layered decoder in [28] is that, each layer has to finish processing all columns before the next layer can start processing. This causes an increase in the number

of clock cycles needed to complete one iteration. The advantage, however, is the increase in convergence speed gained by using layered-decoding. On the other hand, the column-layered decoders in [27] and our work, achieve essentially the same increase in convergence speed without any significant increase in the number of clock cycles per iteration. ASIC implementation results in [28] are reported for a (2304, 1920) QC-LDPC code from [1], which has the same length, rate and base matrix as the code we construct in our design. The decoder in [28] has a TAR_{min} of 1.64 Gbps/mm², which is significantly lower than that of the proposed decoder (7.56 Gbps/mm²), and a latency of 2430 nsec, which is more than 8 times higher than the latency of the proposed decoder.

We also compare the proposed architecture with other medium-length high-rate partially-parallel and fully-parallel designs in the literature. We show that the TAR_{min} of the proposed decoder is higher than two non-layered partially-parallel architectures in [24] and [25], while having comparable latency. Furthermore, we show that the TAR_{min} and latency of the proposed decoder in most cases is better or comparable with many fully-parallel architectures, while having a better decoding performance.

To demonstrate the advantage of this architecture in decoders for large-length codes, we implement a decoder for a regular (32768, 26624) QC-LDPC code and show that compared to the decoder with the same code length and rate implemented in [29], the proposed decoder has a better latency and TAR_{min}.

3.2 Decoder Architectures for Protograph-Based QC-LDPC Codes and throughput analysis

3.2.1 BPP and PPP Architectures

Protograph-based QC-LDPC codes are attractive due to their simple hardware implementations [35, 39]. It has also been shown that they can be designed to have very good performance in both waterfall and error floor regions [3, 36]. There are two general partially-parallel hardware architectures for implementing protograph-based QC-LDPC decoders: block-based partially-parallel (BPP) architecture and protograph-based partially-parallel (PPP) architecture. These two architectures for the decoder of the code with the parity-check matrix of Fig. 2.1(b) are shown in Figs. 3.1(a) and (b), respectively.

Figure 3.1: The (a) block-based partially-parallel (BPP) and (b) protograph-based partially-parallel (PPP) architectures for the decoder of the code with the parity-check matrix given in Fig. 2.1(b). Variable nodes, check nodes, memory units, barrel shifters for variable and check nodes are denoted by “V”, “C”, “M”, “BSV”, “BSC”, respectively.

In the BPP architecture, a block of rows and/or columns of $\tilde{H}$, corresponding to the rows and/or columns of one of $\tilde{H}_{ij}$ sub-matrices, are processed in parallel. These processing units are denoted by “V” and “C”, in Fig. 3.1, respectively. This requires N variable node and N check node processing units to operate in parallel. The processing of rows and columns corresponding to $\tilde{H}_{ij}$ sub-matrices for different i and j values are then performed sequentially through each decoding iteration. Barrel shifters (BS) are used to rearrange memory contents based on cyclic permutations of the QC-LDPC code and to store them back into memory. In Fig. 3.1(a), barrel shifters for variable and check nodes are represented by “BSV” and “BSC”, respectively, and memory units are denoted by “M”.

In the PPP architecture, shown in Fig. 3.1(b), on the other hand, each copy of the protograph is processed simultaneously, requiring n variable node and m check node processing units to operate in parallel. There is no need for barrel shifters as the memory address generator is a simple counter.

It is easy to see that the throughput (in Mbps) of the BPP and PPP architectures for a one-level lifting can be approximated by the following equations:

$$\eta_{BPP} \approx \frac{nNf}{(n+m)I_{max}} \quad , \quad \eta_{PPP} \approx \frac{nNf}{2NI_{max}} = \frac{nf}{2I_{max}} \quad , \quad (3.1)$$

where f and I_{max} are the clock frequency (in MHz) and the maximum number of iterations, respectively. In deriving (3.1), it is assumed that the decoder is already filled with channel data, and the extra cycles for read and write from memory and for pipelining are ignored. Corresponding to (3.1), the two architectures have the following decoding delays (in μsec):

$$\tau_{BPP} \approx \frac{(n+m)I_{max}}{f} \quad , \quad \tau_{PPP} \approx \frac{2NI_{max}}{f} \quad . \quad (3.2)$$

Using (3.1) and (3.2), it can be seen that if $N > (m+n)/2$, the BPP architecture has a larger throughput and a smaller latency in comparison with the PPP architecture.

3.2.2 High-Throughput PPP Architecture Based on Two-Level Liftings

The PPP architecture is commonly used to implement decoders for protograph-based QC-LDPC codes [17, 20, 23, 37, 38]. This architecture, for a one-level lifting, has a simple structure and usually requires less resources compared to BPP. It however suffers from having a relatively low throughput. To overcome this problem, a two-level lifting can be used within the framework of a resource constrained scenario, to maximize the throughput of the PPP architecture. In such a scenario, given a base graph with m check nodes and n variable nodes, a two-level lifting of a given degree $N = N_1 \times N_2$ is designed such that a desired block length of $n \times N$ is attained, while N_1 is maximized within the constraint of hardware resources. It is important to note that in this context, since the degree distribution, the block length, and the rate of the lifted code are all independent of the choice of N_1 , so is the error rate performance of the code. The throughput and the delay of the PPP architecture based on a two-level lifting are given by

$$\eta_{PPP}^{(2)} \approx \frac{nN_1f}{2I_{max}} \quad , \quad \tau_{PPP}^{(2)} \approx \frac{2N_2I_{max}}{f} \quad , \quad (3.3)$$

where the superscript (2) is for the “two-level” lifting.

In many high-speed applications such as optical networks, the LDPC codes have high rate and large check node degrees, see, e.g., [65–68]. As an example, the code of [68] has check nodes of degree 24 compared to variable nodes of degree only 4. In such scenarios, the check nodes are the bottleneck for increasing the throughput

and decreasing the delay. In the following, we propose a novel architecture with serial input and serial output for the check node processing unit that reduces the number of clock cycles required to complete one iteration of the decoding process. The new architecture is implemented within the framework of PPP for a two-level lifted QC-LDPC code (with no parallel edges after the first lifting), and is based on the min-sum decoding algorithm. For high-rate codes, the new architecture is shown to outperform even the BPP architecture in terms of the throughput while using comparable resources. In addition, as a result of the new message-passing schedule imposed by the new architecture, the error rate performance of the decoder is improved.

3.3 PPP Architecture with Serial Check Nodes

We recall that in the standard PPP architecture, for a one-level lifting, there are n variable nodes and m check nodes processing messages in parallel. In the proposed architecture, we still have n variable nodes, corresponding to block-columns of $\tilde{H}$, but the number of check nodes is increased to $m \times N$, where each check node represents one row of the parity-check matrix $\tilde{H}$. We refer to the new check nodes as *serial check nodes (SCN)* since they process messages serially. The message-passing is designed such that at each clock cycle, each check node receives messages from at most one variable node. This requires all the non-infinity elements in each row of the permutation matrix D to be unequal. This, for a fully-connected base graph (a base graph in which each variable node is connected to all the check nodes and vice versa), implies that $N \geq n$. We refer to a protograph-based QC-LDPC code (corresponding to a parity-check matrix $\tilde{H}$) with this property *SCN compatible*.

Example 3 Consider the (30, 12) QC-LDPC code with the parity-check matrix $\tilde{H}$

given in Fig. 3.2. This code is obtained by the application of a cyclic lifting of degree 6 to a 3×5 fully-connected base graph. The permutation shift matrix for this code is the following:

$$D = \begin{bmatrix} 1 & 4 & 2 & 0 & 3 \\ 5 & 4 & 1 & 3 & 2 \\ 2 & 0 & 4 & 1 & 5 \end{bmatrix},$$

and is SCN compatible. In the new architecture, to implement the decoder of this code, we need 5 variable nodes and 18 SCNs. The fact that the code is SCN compatible implies that each row of $\tilde{H}$ has at most one '1' in the five k th columns ($1 \leq k \leq 6$) of five block-columns corresponding to the 5 variable nodes. The five columns have labels $6l + k$, $0 \leq l \leq 4$. For $k = 1$, these columns are shaded in Fig. 3.2. As we will see later, in the new architecture, the messages from variable nodes are passed in parallel and in the same clock cycle to all the check nodes that have value '1' in their rows and in the columns labeled by $6l + k$, $0 \leq l \leq 4$, for different values of k .

In the following, we describe the PPP architecture with SCN, PPP-SCN for brief, based on the example of Fig. 3.2 described in Example 3.

In PPP-SCN, instead of having m high-degree check nodes working in parallel, as in the standard PPP architecture, we have $m \times N$ serial-input check nodes receiving data in parallel. The SCN compatibility of the code ensures that in each clock cycle, only a maximum of one variable node is passing messages to each check node. Check nodes can thus be designed as serial-input serial-output units. Fig. 3.3 shows the block diagram of the PPP-SCN architecture for the code of Fig. 3.2. As it can be seen in Fig. 3.3, instead of 3 check nodes of degree 5, we have 18 serial-input serial-output check nodes (SCN). There are also 5 variable nodes of degree 3. Variable nodes

C1	1			1			1			1							1
C2		1				1			1		1						1
C3			1		1				1			1					1
C4				1		1				1			1		1		
C5					1		1		1					1		1	
C6	1					1		1					1			1	
C7				1			1		1				1				1
C8	1						1			1				1			1
C9		1			1					1				1			1
C10			1			1				1	1						1
C11				1			1			1		1		1			
C12					1			1				1			1		
C13			1			1				1		1					1
C14				1			1				1		1		1		
C15					1			1				1				1	
C16						1			1				1			1	
C17	1						1		1					1			1
C18		1						1		1							1
		V1		V2		V3		V4		V5							

Figure 3.2: The parity-check matrix of a (30,12) QC-LDPC code which is SCN compatible

process messages in parallel, the same as in the standard PPP architecture. The first and second minimums along with the index of the first minimum for both current and previous iterations are stored in registers inside SCN blocks. *VtoC* and *CtoV* multiplexer networks transfer messages between variable nodes and check nodes. *Concat* blocks concatenate outputs of multiple check nodes to be sent to the corresponding variable nodes.

3.3.1 SCN Structure

The structure of the SCN is shown in Fig. 3.4. The serial input is first decomposed into the sign *Input_sign* and the magnitude *Input_magnitude*. The *Input_magnitude*

Figure 3.3: The block diagram of the PPP-SCN architecture for the decoder of the code with the parity-check matrix of Fig. 3.2

is compared with the current values of first minimum *First_min* and second minimum *Second_min* registers. Based on the result of the comparison, the block *Data_select* will update *First_min* and *Second_min*. The counter *Counter* keeps

Figure 3.4: The structure of serial check node (SCN)

track of the index of the current input, and updates the index of the first minimum, *First_min_index*, based on the comparison results from *Data_select*. The value of *Counter* is also used to store the sign of inputs in a buffer. Also, in the memory *Total_sum*, the XOR of signs of all inputs is stored. The *Total_sum* value is used to check whether the check node is satisfied. It is also used to determine the sign of the output messages. If *Total_sum* is zero, the sign of output messages remains unchanged, and if *Total_sum* is one, the sign of output messages is inverted. At the end of each iteration, values of *First_min*, *Second_min*, and *First_min_index* are stored in *Previous First_min*, *Previous Second_min* and *Previous First_min_index*, respectively. The block *Output_select* is responsible for passing the appropriate message to the output based on *Output_index*, *First_min*, *Second_min*, *First_min_index*, *Input_signs*, *Total_sum*, *Previous First_min*, *Previous Second_min* and *Previous First_min_index*. Passing messages to check nodes in serial will result in a new column-layered message-passing schedule called *serial check node message-passing (SCMP)*. This is explained in the following subsection.

3.3.2 Serial Check Node Message-passing (SCMP)

In the serial check node message-passing (SCMP), variable nodes read messages from memory, process them and instead of storing them in memory, as they would do in the standard PPP architecture with parallel message-passing, send them directly to check nodes. Check nodes then receive and process these messages serially. We explain this new message-passing schedule using the Tanner graph of the lifted code of Fig. 2.1(b).

In the PPP-SCN architecture, the decoder of this code has two variable nodes of degree 2 and one variable node of degree 1. The decoder also has 8 serial check nodes, four of which accept 2 input messages in serial and the other four accept 3 serial input messages to complete the check node update process.

Fig. 3.5 explains SCMP for this code. We assume that variable nodes are already filled with the initial messages from the channel. For PPP-SCN, we need three variable nodes $V1 - V3$ and eight SCN processing units $C1 - C8$. Figures 3.5(a)-(d) show the first 4 clock cycles (first stage) of the first iteration of the message-passing algorithm. In this stage, at each clock cycle, variable nodes of one copy of the protograph send messages to SCNs. Based on the SCN compatibility constraint, at each clock cycle, each check node receives a maximum of one message. After the 4th clock cycle, check nodes have received all their inputs and are ready to pass messages to variable nodes. Fig. 3.5(e) shows the fifth clock cycle. This cycle is the first cycle of the second stage of the first iteration in SCMP. In this clock cycle, check nodes send back messages to the variable nodes in the first copy of the protograph. For example, in Fig. 3.5(e), check nodes $C1$ and $C5$ send messages to $V1$, check nodes $C4$ and $C7$ send messages to $V2$ and check node $C6$ sends a message to variable node $V3$. Thus, variable nodes $V1 - V3$ have all messages they need to start the next iteration by processing the first copy of the protograph. The processing that takes place during the 6th clock

cycle is shown in Fig. 3.5(f). In this cycle, variable nodes start processing the first copy of the protograph for the next iteration. The messages passed by variable nodes to serial check nodes is similar to those of the first clock cycle. At the same time, check nodes pass messages to variable nodes in the second copy of the protograph. Variable nodes need these messages to process the second copy of the protograph in the next clock cycle. In other words, the first stage of the next iteration is performed with one clock cycle delay from the second stage of the current iteration. In the steady-state, each iteration is completed in N clock cycles due to the full overlap between the message-passings in the two directions. In fact, starting from the second iteration, the message-passings described in Figures 7(d) and 7(e) also overlap. The throughput and the latency of the PPP-SCN architecture are thus given by

$$\eta_{PPP-SCN} \approx \frac{nf}{I_{max}} \quad , \quad \tau_{PPP-SCN} \approx \frac{NI_{max}}{f} . \quad (3.4)$$

Note that for this code, a complete iteration in the standard PPP architecture takes 8 clock cycles to be completed. From the 8 cycles, four is for the three variable nodes to perform the processing required for the variable nodes in all four copies of the protograph and send messages to check nodes, and 4 clock cycles for the check nodes to process all copies of the protograph and send messages back to variable nodes. In general, each iteration in the standard PPP architecture is completed in $2N$ clock cycles rather than N required for the PPP-SCN architecture. This number for the BPP architecture is $m + n$, which is still larger than that of PPP-SCN if $m + n > N$.

The overlap between the second stage of the current iteration and the first stage of the next iteration in SCMP implies column-layered decoding, in which the updated check node messages propagate through the graph faster than they would in parallel (flooding) message-passing. At each clock cycle, check nodes that are updating a

certain copy of the protograph for the next iteration are themselves being updated at the same time by the variable nodes of the previous copy of the protograph in the current iteration. For example in Fig. 3.5(f), check nodes $C5$ and $C7$ are being updated by variable nodes $V1$ and $V2$ from the next iteration. The correspondingly updated messages of these check nodes will then propagate in the future copies of the protograph in the current iteration. This, for example, can be seen in Fig. 3.5(g), in which $C5$ will update $V2$ and $C7$ will update $V1$ based on the updated message received from $V1$ and $V2$ in the previous copy of the protograph. This expedited propagation of updated messages results in faster convergence speed and better decoding performance.

3.3.3 Multiplexer Networks

The SCMP schedule, described above, needs two multiplexer networks: one network to transfer messages from variable nodes to check nodes, ($VtoC$), and another one to transfer messages from check nodes to memory, ($CtoV$).

Focusing on the decoder of Fig. 3.3, there are 3 $VtoC$ blocks in this decoder. Each block operates as a cyclic multiplexer based on a control sequence as explained in the following. In Fig. 3.2, for the first clock cycle, check node $C1$ receives a message from variable node $V4$, check node $C2$ does not receive any messages from any variable nodes and check nodes $C3 - C6$ receive messages from $V2, V5, V3, V1$, respectively. Corresponding to this message-passing, we define vector $\mathbf{V}_{in} \triangleq (V4, -, V2, V5, V3, V1)$ as the input message vector for check nodes $C1 - C6$ in the first clock cycle. In the second clock cycle, the input message vector is $(V1, V4, -, V2, V5, V3)$, which is a cyclically shifted version of $\mathbf{V}_{in}$ by one to the right. In general, at clock cycle $t, 1 \leq t \leq 6$, the input message vector is a cyclic shift

Figure 3.5: SCMP for the QC-LDPC code of Fig. 2.1(b): (a)-(d) The first stage of the first iteration where variable nodes pass messages to SCNs. (e) The first clock cycle of the second stage of the first iteration where check nodes pass messages to variable nodes in the first copy of the protograph.(f) The second clock cycle of the second stage of one iteration where check nodes pass messages to variable nodes in the second copy of the protograph and variable nodes pass messages to check nodes in the first copy of the protograph (this corresponds to the first clock cycle of the first stage of the next iteration).

of $\mathbf{V}_{in}$ by $t - 1$ to the right. The $VtoC$ blocks are thus implemented based on the control sequences $\mathbf{V}_{in}$.

The $CtoV$ blocks have a similar structure to $VtoC$ blocks. Referring to Fig. 3.3, the modulo 6 counters, which generate the memory addresses, are initialized based on the number of shifts for each permutation matrix. For Example, the first set of 5 counters in the top $CtoV$ block in Fig. 3.3 is initialized with the values in the first row of D , and so on. At each clock cycle, a cyclically shifted control sequence sends the index of the message that needs to be generated to check nodes, and in the next clock cycle, stores their output in memory.

We propose two architectures for multiplexer networks, as shown in Fig. 3.6. Both architectures can be used for either $VtoC$ or $CtoV$ blocks. Fig. 3.6, however, shows the architectures for the $VtoC$ block.

In the first architecture (Fig. 3.6 (a)), for each variable node, a demultiplexer assigns the input to one of the check nodes. The selection signal of this demultiplexer is generated by a counter which is initialized based on the amount of shift in the permutation shift matrix D . There will be only one input for each check node at each clock cycle. Therefore, each check node simply XORs outputs of all connected demultiplexers.

In the second architecture (Fig. 3.6 (b)), each check node has a multiplexer connected to all inputs, a counter shifts the cyclic control sequence, and sends this to the bottom multiplexer. Each multiplexer then shifts the control sequence coming from the multiplexer below it to the right by one and passes the result to the multiplexer above it. This is how the cyclic sequence corresponding to $\mathbf{V}_{in}$ is realized. The architecture of Fig. 3.6(b) is more compact but slower than that of Fig. 3.6(a).

Figure 3.6: A 5×6 *VtoC* Multiplexer (a) Fast architecture, (b) Compact architecture

3.3.4 Pipeline

A four stage pipeline, as shown in Fig. 3.7, is designed for PPP-SCN. The 4 stages are allocated for variable nodes, *VtoC*, *SCN* and *CtoV* blocks to process their inputs and update their output registers. A 4 cycle pipeline stall is also added to make sure that the last *CtoV* has messages ready for variable nodes, before the next iteration starts. In PPP-SCN, extra caution has to be exercised while increasing the stages of pipeline. As explained in Section 3.3.2, the gain in convergence speed comes from the fact that within each iteration, the updated messages of check nodes can propagate in the future copies of the protograph, and will be used by variable nodes when they

process those copies. Increasing the number of pipeline stages results in an increase in the clock frequency, but it will also introduce delays in the propagation of messages within the same iteration. For example, in the four stage pipeline in Fig. 3.7, at the edge of clock 0, a variable node processes the first copy of the protograph using messages from previous iteration. At the edge of clocks 1, 2 and 3, that variable node processes the next 3 copies of the protograph, respectively, using messages from the previous iteration. In these 3 clock cycles, messages created by the variable node at the edge of clock 1, are processed by *VtoC*, *SCN* and *CtoV* blocks, respectively. Thus, it is only at the edge of clock 4, that the variable node can use the propagated messages from the first copy of the protograph. We found that, for high-rate codes having check nodes of degree 24 and 32, using a 4-stage pipeline results in timing closure with negligible effect on the convergence speed.

Figure 3.7: Pipeline diagram of SCMP, showing the edge of clock cycles corresponding to different stages.

3.3.5 Early termination

The *Total_sum* in each check node determines if the check node is satisfied. In early termination, instead of continuing the iterative decoding process until the maximum number of iterations is reached, the satisfaction of all check nodes is monitored, and if achieved, the decoder stops and proceeds to the next block. In high-throughput

decoders, which normally operate in the low error rate region, early termination is used to increase the throughput. To help timing closure, we use two tree-like *AND* networks, each *AND*ing the result of half of the check nodes in one clock cycle, and subsequently *AND* the result of the two trees in another clock cycle, to detect if all check nodes are satisfied.

3.4 FPGA Implementations

The results reported below on resource utilization of different architectures and different hardware blocks are for implementations on an Altera Stratix IV EP4SGX530 FPGA device [69]. We use Altera’s Quartus II as the synthesis tool [70].¹

To compare different decoder architectures, we consider a regular QC-LDPC code of rate 5/6 with a parity-check matrix of size 1536×9216 . This code can be used, for example, in OTN4 [65]. The base graph is a fully-connected graph of size $m = 4$ and $n = 24$. Similar codes have been considered in [66–68]. In a construction with one-level lifting, the lifting degree will be equal to 384. As it is shown in Fig. 3.1(a), in BPP there are 4 processing units: variable nodes, check nodes, “BSV” and “BSC”. Thus, for BPP, we consider a 4-stage pipeline, each stage allocated for a processing unit, and a 4 cycle pipeline stall. For PPP, shown in Fig. 3.1(b), variable nodes and check nodes are the main processing units, thus we choose a 2-stage pipeline and a 2 cycle pipeline stall. For PPP-SCN, as we shown in section 3.3.4, we choose a 4 stage pipeline and 4 cycle pipeline stall. The BPP architecture in this case requires $m + n + 4 = 32$ clock cycles to complete one iteration. The PPP architecture, on the

---

¹To the best of our knowledge, all publications on the FPGA implementation of partially-parallel LDPC decoders, including the works in [17, 20, 22, 23], report throughput values less than or close to 1 Gbps. On the other hand, the throughputs that we obtain on our FPGA platform based on PPP-SCN architecture are in the order of a few Gbps, which handily outperform the similar existing results.

other hand, will need $2N + 2 = 770$ clock cycles for one iteration. This will reduce to $N + 4 = 388$ clock cycles for PPP-SCN, which is still much larger than the 32 cycles of BPP architecture. Both PPP and PPP-SCN however, require much less resources compared to BPP. To increase the throughput of PPP architectures, we consider a two-level lifting and maximize the degree of the first lifting with the constraint of the available resources on the FPGA device. This results in $N_1 = 16$ and $N_2 = 24$. In this case, if we apply the BPP architecture to the code resulted from the first level of lifting as the base code, then each iteration will take $N_1 \times (m + n) + 4 = 452$ clock cycles. This is while the PPP and PPP-SCN architectures will require only $2N_2 + 2 = 50$ and $N_2 + 4 = 28$ clock cycles, respectively.

To compare the resource utilizations of different architectures, we consider the number of quantization bits for messages to be 5. Table 3.1 shows the resource utilization of different processing blocks for the BPP, PPP and PPP-SCN architectures, in terms of look-up tables (LUTs) and registers.

In Table 3.1, we have used the architecture of Fig. 3.6(b) for the *VtoC* and *CtoV* multiplexers. For a comparison between the two architectures of Fig. 3.6, we have provided the resource utilization and the delay of the two architectures for different number of quantization bits for a 24×24 multiplexer network in Table 3.2. As it is shown in Table 3.2, the architecture of Fig 3.6(a) is faster but generally requires more resources, particularly for smaller number of quantization bits. (In general, the delay of the architecture of Fig 3.6(b) is about 50% higher than that of the architecture of Fig 3.6(a).) There is an exception for the 8-bit realization, for which the architecture of Fig 3.6(b) requires more LUTs than that of Fig 3.6(a). This is due to the fact that in Altera Startix IV FPGAs, each LUT can handle a function with maximum of 6 inputs. For functions with more than 6 inputs, more than 1 LUT are used and then

Table 3.1: Resource utilization of different processing blocks of BPP, PPP and PPP-SCN architectures based on 5-bit quantization

	Variable Node (V)	Check Node (C)	Barrel Shifter	SCN	$VtoC\ 24 \times 24$	$CtoV\ 24 \times 24$
LUTs	114	573	1,536	72	639	767
Registers	21	31	384	63	64	83

the design is optimized by the synthesizer to reduce the number of LUTs.

Table 3.2: Comparison of Resource Utilizations for a 24×24 Multiplexer Network Using Architectures of Fig. 3.6(a) and (b)

Architecture	Quantization		4-bit	5-bits	8-bits
Fig. 3.6(a)	Resources	LUTs	598	658	838
		Registers	1249	1549	2449
	Delay(nsec)		2.7	2.6	3.1
Fig. 3.6(b)	Resources	LUTs	540	639	938
		Registers	51	64	99
	Delay(nsec)		3.8	3.83	4.6

The comparison of the resource utilization of different architectures for one-level and two-level liftings are summarized in Table 3.3 for 5-bit messages. Timing and throughput parameters such as the number of clock cycles required to complete one iteration, the maximum operating frequency reported by the synthesizer (F_{max}), the average number of iterations reported for SNR=4.3 dB using early termination, and the calculated maximum latency, average and minimum throughputs are presented in Table 3.4. For the FPGA implementation, the maximum number of iterations is considered to be 10.

In Tables 3.3 and 3.4, the first row shows the results for the BPP architecture

Table 3.3: Comparison of resource utilization parameters of different architectures

Architecture	Variable Node (V)		Check Node (C)		BS		SCN		V2C		C2V		Block RAMs	Total	Final Utilization	
	# of units	LUTs Registers	# of units	LUTs Registers	# of units	LUTs Registers	# of units	LUTs Registers	# of units	LUTs Registers						
BPP One-Level lifting	384	43,776	384	220,032	28	43,008	0	0	0	0	0	96	306,816	30,720	79%	
		8,064		11,904		10,752		0		0						
PPP One-Level lifting	24	2,736	4	2,292	0	0	0	0	0	0	0	96	5,028	628	2%	
		504		124		0		0								
PPP Two-Level lifting 16×24	384	43,776	64	36,672	0	0	0	0	0	0	0	1,536	80,448	10,048	29%	
		8,064		1,984		0		0								
PPP-SCN Two-Level lifting 16×24	384	43,776	0	0	0	0	1,536	110,592	64	40,896	64	49,075	0	248,947	117,325	64%
		8,064		0		0		96,768		4,096		5,325				

Table 3.4: Comparison of timing parameters of different architectures Considering a maximum number of iterations equal to 10

Architecture	Clock cycles per Iteration	Fmax (MHz)	Latency (μ sec)	Average Iterations at 4.3 dB	Average Throughput(Gbps)	Min Throughput (Gbps)
BPP One-Level lifting	32	112	2.86	6.2	5.20	3.23
PPP One-Level lifting	770	213	36.15	6.2	0.41	0.25
PPP Two-Level lifting 16×24	50	150	3.33	6.2	4.46	2.76
PPP-SCN Two-Level lifting 16×24	28	124	2.26	4.8	8.50	4.08

applied to the one-level lifting. The decoder uses 306,816 LUTs, 30,720 registers, 96 block RAMs, has a total utilization of 79% and a maximum frequency of 112 MHz. At SNR=4.3 dB, the average number of iterations for this decoder is 6.2. It requires 32 clock cycles to complete one iteration, and has a latency of 2.86 μ sec and average and minimum throughputs of 5.20 Gbps and 3.23 Gbps, respectively. The second row shows the results for the standard PPP architecture applied to the one-level lifting. The number of LUTs and registers are 5,028 and 628, respectively, which is much less than that of BPP, and the total utilization is only 2%. At SNR=4.3 dB, the average number of iterations for this decoder is also 6.2. Each iteration takes 770 clock cycles, and while operating at a higher frequency of 213 MHz, it has a latency of 36.15 μ sec and average and minimum throughputs of only 410 Mbps and 250 Mbps, respectively. In the third row, we have the PPP architecture for the two-level lifting. In this case, the resource utilization (LUTs, registers and block RAMs) is increased by a factor of $N_1 = 16$ compared to the PPP for the one-level lifting. In return, the number of clock cycles to complete one iteration is reduced to 50 from 770. At SNR=4.3 dB, the average number of iterations for this decoder also remains at 6.2. The maximum operating frequency is reduced to 150 MHz from 213 MHz for the PPP architecture

with one-level lifting, resulting in a latency of $3.33 \mu\text{sec}$ and average and minimum throughputs of 4.46 Gbps and 2.76 Gbps, respectively. In the last row of Tables 3.3 and 3.4, we show the results for the PPP-SCN architecture applied to the two-level lifting. In this architecture, the number of LUTs is less than that of BPP by about 20% but almost 3 times that of PPP for the two-level lifting. The number of registers has increased significantly to about four times that of BPP and almost twelve times that of PPP for the two-level lifting. This however is not a major issue, since FPGAs normally have an equal number of LUTs and registers. In other words, each cell in an FPGA contains an equal number of LUTs and registers, and thus the final cell usage on an FPGA is determined by the maximum number of LUTs and registers. As it can be seen in Table 3.3, the final resource utilization is 64%, which is lower than the resources needed by BPP. Also, in the PPP-SCN architecture, there is no need for block RAMs, making it very suitable for ASIC implementations. Finally, PPP-SCN requires the least number of clock cycles (28) among all the architectures to complete one iteration. Also, the average number of iterations for this decoder is reduced to 4.8, which with the maximum operating frequency of 124 MHz, translates to a latency of $2.26 \mu\text{sec}$, and average and minimum throughputs of 8.50 Gbps and 4.08 Gbps, respectively. These parameters are considerably superior to those of the other architectures. The second highest throughput is attained by the BPP architecture for one-level lifting with average and minimum throughputs of 5.20 Gbps and 3.23 Gbps, respectively.

To study the performance of SCMP, we consider three QC-LDPC codes: (i) A one-level lifting of degree 384 of the fully-connected base graph with $n = 24$ and $m = 4$. For this code, all the cyclic permutations are selected randomly; (ii) A two-level lifting of the same base graph with lifting degrees $N_1 = 16$ and $N_2 = 24$. For this code also, the cyclic permutation shifts for both liftings are selected randomly; (iii)

Similar construction to (ii) with the only difference being that the cyclic permutations for the second lifting are selected randomly with the constraint that they are SCN compatible. The three codes are decoded using a 5-bit MS decoder with maximum number of iterations 10 implemented on the FPGA device using the BPP, PPP and PPP-SCN architectures, respectively. The FER and BER of the three codes/decoders are shown in Fig. 3.8. We have also included the error rate curves for sum-product (belief propagation (BP)) algorithm with 5-bit quantization and flooding schedule and with the same maximum number of iterations 10 applied to LDPC code (iii) in Fig. 3.8, for reference. As expected, both the FER and BER curves for the first two codes/decoders coincide (as they both correspond to a parallel message-passing applied to two codes with similar degree distributions and block lengths). On the other hand, the performance of the third code/decoder is superior to that of the other two code/decoders. This can be attributed to the better performance of SCMP compared to parallel (flooding) message-passing schedule. In comparison with MS decoders, the BP decoder demonstrates a very early error floor. The average number of iterations for this decoder at $\text{SNR} = 4$ dB, which is at the start of the error floor region, is 6.9. This is significantly larger than the average number of iterations for MS with the PPP-SCN architecture.

3.5 ASIC Implementations

In this section, we use the proposed PPP-SCN architecture to implement decoders of two high-rate QC-LDPC codes, with medium and large code lengths, respectively. We use a TSMC 65-nm CMOS process with a core voltage of 1.2 V. The synthesis results are generated using Synopsys Design Compiler. Placement and Routing is

Figure 3.8: The performance of QC-LDPC codes with a parity-check matrix of size (9216,7680) using different architectures

performed using Synopsys IC Compiler. To calculate the power, parasitics are extracted from IC Compiler and back-annotated in the post-layout RTL simulations for power calculation using power analysis tool Synopsys PrimeTime PX. We also compare the proposed implementation with similar existing results in the literature. As one comparison metric, we consider minimum throughput-to-area (TAR_{min}), and when available, average throughput-to-area (TAR_{avg}), which are the minimum and average throughputs divided by the normalized core area, respectively. In addition, we measure the latency, power consumption and energy efficiency of the decoder. The energy efficiency is calculated as the ratio of total power dissipation over throughput and normalized with respect to the number of iterations. To have a fair comparison, when calculating TAR_{min}, we consider a fixed maximum number of iterations of 8 for all medium-length decoders. For decoders whose area measurement is extracted

from the layout report, the core area is normalized based on the reported utilization. Also, for decoders that do not use a 65-nm process or use a different supply voltage, the area, clock frequency, latency and power consumption are scaled to those of the 65-nm with 1.2 V supply voltage using the following equations [71], [72]:

$$area' = area(\frac{1}{S^2}) \quad , \quad freq' = freq \times S \quad , \quad power' = power(V^2), \quad (3.5)$$

where S is the ratio of the feature size of the technology to 65nm, and V is the ratio of 1.2 V to the supply voltage used, and $area'$, $freq'$ and $power'$ are the area, frequency and power scaled to 65nm and 1.2 V, respectively.

3.5.1 Medium-Length Decoder Implementations

We first consider a (2304, 1920) QC-LDPC code. The (2304, 1920) QC-LDPC code in [1] is generated using an irregular base graph of size $m = 4$ and $n = 24$, and a lifting degree of 96. We use the same base graph, and lift it using a two-level lifting with degrees 4×24 . The number of quantization bits is chosen to be 5. This decoder completes each iteration in 28 clock cycles. Using SCMP, at SNR=4.5 dB, the average number of iterations is 3.5, and the maximum achievable frequency is 750 MHz. Thus, the latency of this decoder is 300 nsec and average and minimum throughputs are 17.63 Gbps and 7.71 Gbps, respectively. The synthesis result shows that this decoder occupies 1.02 mm², resulting in TAR_avg and TAR_min of 17.28 Gbps/mm² and 7.56 Gbps/mm², respectively.

Table 3.5 compares this decoder with the state-of-the-art designs for high-rate medium-length LDPC decoders.

First, we compare the proposed decoder with two layered decoders in [27] and [28]. The former has a TAR_min of 4.19 Gbps/mm², and the latter has a TAR_min of 2.93

Table 3.5: Comparison of medium-length LDPC decoders

	[27]	[28]	[25]	[73]	[32]	This Work
Report	Layout	Layout	Layout	Layout	Layout	Layout
Code Block Length	2048	2304	2048	672	672	2304
Code Rate	5/6	5/6	5/6	1/2	13/16	5/6
Code Structure	RS-LDPC	QC-LDPC	RS-LDPC	QC-LDPC	QC-LDPC	QC-LDPC
Base Matrix	32×6	24×4	32×6	16×8	16×3	24×4
Lifting Degree	64	96	64	42	42	4×24
Architecture	Serial Check Node	Serial Check Node	Bit Parallel	Parallel Check node	Iterations Unrolled	Serial Check Node
Framework	BPP	BPP	BPP	BPP	Fully Parallel	PPP-SCN
Scheduling	Column-Layered	Row-Layered	TPMP	TPMP	TPMP	SCMP
Decoding Algorithm	Offset MS	Scaled MS	Offset MS	Offset MS	MS	MS
Quantization Bits	4	6	4	5	4	5
Clocks per Iteration	8	52	12	12	1	28
Average no. Iterations	-	-	1.5 (@ 5.5 dB)	-	-	3.5 (@ 4.4 dB)
Max no. Iterations	8	10	8	10	9	8
Technology	90 nm	0.13 μ m	65 nm	65 nm	65 nm	65 nm
Voltage (V)	1.00	1.20	1.20	1.15	1.20	1.20
Core area normalized to 65 nm	3.20	0.76	5.35	1.60	12.09	1.02
Normalized Clock Freq (MHz)	420	428	700	540	257	750
Normalized Latency (nsec)	153	972	137	178	31	299
Normalized Avg. Throughput (Gbps)	-	-	47.79		-	17.63
Normalized Min Throughput (Gbps)	13.43	2.22	13.27	11.34	160.80	7.71
Normalized TAR _{avg} (Gbps/mm ²)			8.93	-		17.29
Normalized TAR _{min} (Gbps/mm ²)	4.19	2.93	2.48	7.09	13.30	7.56
Power (mW)	1392	397	2800	783	5360	980
Normalized Energy Efficiency (pJ/bit/iteration)	13.48	11.17	26.37	9.40	4.17	15.88
Normalized Power	2004	397	2800	852	5360	980
Normalized Energy Efficiency (pJ/bit/iteration)	18.66	22.34	26.37	9.40	4.17	15.88
Coding Gain Relative to Flooding	+0.1 dB	+0.1 dB	-	-	-	+0.1 dB

- : Not Reported

Gbps/mm². Compared to these two layered decoders, the proposed decoder has a higher TAR_{min} of 7.56 Gbps/mm². Regarding the latency, the proposed decoder with a latency of 299 nsec, has a much lower latency than the row-layered decoder of [28] with a latency of 972 nsec. The latency of the proposed decoder, however, is about 2 times that of [27]. In terms of power consumption and energy efficiency, while the power consumption of the proposed decoder (980 mW) is smaller than the decoder of [27] (2004 mW) and larger than that of [28] (397 mW), the proposed decoder is more energy efficient than both with an energy efficiency of 15.88 pJ/bit/iteration compared to 18.66 and 22.34 pJ/bit/iteration for the decoders of [27] and [28], respectively.

We also compare the proposed decoder with two BPP based decoders which use TPMP schedule in [25] and [73]. These decoders, have TAR_{min} of 2.48 Gbps/mm² and 7.09 Gbps/mm², respectively, which are both lower than the TAR_{min} of the proposed decoder (7.56 Gbps/mm²). In [25], the TAR_{avg} is also reported to be 8.93 Gbps/mm², which is lower than TAR_{avg} of the proposed decoder (17.28 Gbps/mm²). The latency of these decoders are 137 nsec and 178 nsec, respectively, which are both lower than the latency of the proposed decoder which is 300 nsec. In terms of power consumption and energy efficiency, the proposed decoder consumes less power than the decoder of [25] (980 mW vs. 2800 mW) and is more energy efficient (15.88 pJ/bit/iteration vs. 26.37 pJ/bit/iteration), and is less power efficient than the decoder of [73], with a power efficiency of 9.4 pJ/bit/iteration. No information on power consumption is reported in [24].

Finally, we compare the proposed decoder with the fully-parallel decoder [32]. Compared to the decoder of [32], the proposed decoder has lower TAR_{min} and higher latency, and is less energy efficient. However, the the decoder of [32], has an area of 12.09 mm², which is 12 times larger than that of the proposed decoder, while implementing a code with much smaller block length (672 vs. 2304). This makes the

decoder of [32] unsuitable for medium and large-length code implementation.

To end this section, we have reported the BER performance of the (2304, 1920) two-level lifted code decoded by 5-bit MS with SCMP schedule in Fig. 3.9. The BER curves for maximum number of iterations 8 and 16 are provided. In the same figure, we have also given the the BER curves of the (2304, 1920) code used in WiMax [1] decoded by 5-bit MS with TPMP, row-layered and column-layered message-passing schedules. As can be seen in Fig. 3.9, SCMP, row layered and column-layered message-passing schedules perform more or less the same, all outperforming TPMP by about 0.1 dB and 0.2 dB, when the maximum number of iterations are 8 and 16, respectively.

3.5.2 Large-Length Decoder Implementations

In [29], the synthesis results of a stochastic decoder for a (32768, 26624) LDPC code is presented. Stochastic decoders have the advantage of high throughput and low area, but suffer from a high latency, due to the increased number of clock cycles needed to obtain a good performance. For comparison, we construct a regular (32768, 26624) QC-LDPC code using a fully-connected graph of size $m = 6$ and $n = 32$, with a two-level lifting of degrees 32×32 . With a maximum number of iterations set to 10, the decoder has an average number of iterations of 4.1 at SNR=4.8 dB, resulting in average and minimum throughputs of 99.9 Gbps and 40.96 Gbps, respectively. The proposed decoder occupies an area of 14.19 mm², and can operate with a maximum clock frequency of 450 MHz. The TAR_avg is 7.04 Gbps/mm², which is lower than the TAR_avg of 32.72 Gbps/mm² reported in [29] for the same SNR. However, the latency is 800 nsec which is 33% lower than the latency of 1200 nsec reported in [29]. In addition, the TAR_min is 2.89 Gbps/mm², which is higher than the TAR_min of 1.87 Gbps/mm² in [29] for the same SNR.

Figure 3.9: Bit error rate curves of different message-passing schedules. (...) The (2304, 1920) QC-LDPC code from [1] with TPMP schedule, MS algorithm, 8 iterations ($\circ$) and 16 iterations ($\square$). (-.-) The (2304, 1920) QC-LDPC code from [1] with row-layered schedule, MS algorithm, 8 iterations ($\circ$) and 16 iterations ($\square$). (- -) The (2304, 1920) QC-LDPC code from [1] with column-layered, MS algorithm, 8 iterations ($\circ$) and 16 iterations ($\square$). (—) The (2304, 1920) two-level lifted code, with SCMP schedule, MS algorithm, 8 iterations ($\circ$) and 16 iterations ($\square$).

3.6 Conclusion

In this chapter, we proposed a serial-input design for the implementation of high-degree check nodes in a min-sum decoder of a protograph-based multi-level lifted QC-LDPC code. A new design for serial-input check nodes and switch networks,

Table 3.6: Comparison of large-length LDPC decoders

	[29]	This Work
Technology	STM 90 nm	TSMC 65 nm
Code	(32768, 26624)	(32768, 26624)
Normalized Core 65 nm (mm ²)	14.6	14.19
Clock Freq (MHz)	500	450
Average no. of Iterations	34.3	4.1
SNR (dB)	4.8	4.8
Max no. Iterations	600	10
Avg. Throughput (Gbps)	477.67	99.90
Minimum Throughput (Gbps)	27.31	40.96
Latency (nsec)	1200	800.00
TAR _{avg} (Gbps/mm ²)	32.72	7.04
TAR _{min} (Gbps/mm ²)	1.87	2.89

resulted in lower hardware complexity while having a higher throughput and comparable latency in comparison with all existing partially-parallel architectures. In addition to superior throughput/area/latency trade-off, the SCN architecture induces a new column-layered message-passing schedule which outperforms the standard parallel (flooding) message-passing schedule in terms of error rate performance. Synthesis results in a 65-nm CMOS technology are presented for medium-length (2304, 1920) and large-length (32768, 26624) QC-LDPC decoders. These decoders have a silicon area of 1.02 mm² and 14.19 mm² and attain average throughput of 17.28 Gbps and 99.9 Gbps, respectively.

Chapter 4

Lowering the Error Floor of LDPC Codes Using Multi-Step Quantization

4.1 Introduction

In this chapter, we propose a very simple scheme to lower the error floor of LDPC codes decoded by fixed-point message-passing decoders. The scheme is based on successive re-quantization and re-decoding of the input blocks that cause the decoder to be trapped in a trapping set, until the decoding is successful or a maximum number of re-quantization/re-decodings is reached. At each step, the re-quantization is simply performed by increasing the clipping threshold (dynamic range of the input quantizer) of a uniform quantizer while keeping the same number of quantization bits. Our experiments with a variety of codes and decoding algorithms show that with a rather small number of re-quantization/re-decodings, a large majority of the input blocks that originally failed the decoder will be successfully decoded. Our proposed scheme is universal in the sense that it can be applied to both regular and irregular codes and to any message-passing decoder. Unlike the schemes of [48], [49] and [50], the modification to the quantization in the re-quantization process only affects the input

to the decoder without affecting the internal modules of the decoder. It is important to note that while the idea of increasing the dynamic range of the quantizer to lower the error floor is already known [11], [48], [49], [50], the simple implementation of this idea without sacrificing the waterfall performance, as presented here, is novel. In [49], [50], the good waterfall performance is maintained by the embedded uniform quantizer within the quasi-uniform quantizer, and the improvement in error floor is achieved by the non-uniform component of quantization which allows for very large dynamic ranges. In this thesis, however, the quantization is uniform and the dynamic ranges are not nearly as large as those used in [49], [50]. In addition, the results presented in [49], [50] are limited to variable-regular LDPC codes. The scheme proposed here however, can be applied to any regular or irregular code.

4.2 The Proposed Scheme

In the waterfall region, for each decoding algorithm, the proposed scheme quantizes the input using the optimal clipping threshold c_{opt} for the best waterfall performance [11]. There is no re-quantization/re-decoding needed for the waterfall region.¹ In the error floor region, the proposed scheme starts by quantizing the input using the clipping threshold c_{opt} . If the decoding is successful, in that, it converges to a codeword, then the decoder moves on to the next input block. If unsuccessful, the quantization/decoding is repeated, this time using c_2 from the set $\mathcal{C} = \{c_1, c_2, \dots, c_m\}$, where $c_1 = c_{opt}$ and $c_i > c_{opt}$ for $i = 2, \dots, m$. This process of re-quantization/re-decoding

---

¹To distinguish between waterfall and error floor regions, the decoder can rely on the SNR value if that information is available at the receiver. In the absence of such information, the decoder can use the number of unsatisfied check equations to identify the two regions. In the error floor region, when the decoder is trapped, the number of unsatisfied check equations is often rather small. In the waterfall region however, if the decoder fails to converge to a codeword, the number of unsatisfied check equations is often larger. More details on the choice of the threshold τ on the number of unsatisfied check equations to distinguish between the two regions are given at the end of this section and in Section 4.3.

will continue until the decoding is successful or the last (m th) re-quantization/re-decoding fails. In the latter case, a failure is declared. Parameter m controls the tradeoff between the performance and the latency/complexity of the scheme. In general, larger m corresponds to better performance at the cost of increased latency and complexity. The motivation behind the increase of the clipping threshold in the error floor region is that at high SNR values, to help the decoder to recover from small trapping sets, one should allow for the larger input values to be properly represented without undue saturation (clipping) [48], [50]. Increasing the dynamic range of the quantizer while keeping the number of quantization bits fixed, will also reduce the magnitude of the integer values corresponding to the few bits that are highly affected by noise and have small LLR values with the wrong sign. This in turn helps with the correction of such bits in the decoding process.

In this thesis, we use Monte Carlo simulations to evaluate the performance of the proposed scheme.² Without loss of generality, we assume $c_{opt} < c_2 < \dots < c_m$. For a fixed m , finding the optimal set of clipping thresholds in the set $\mathcal{C}$ is a hard problem. The solution would depend on the code, the decoding algorithm and the SNR, and has to be found among an infinitely large set of possibilities if clipping thresholds are assumed to be real numbers. If, however, we constrain the clipping thresholds to belong to a finite set $\mathcal{A}$, and assume that $c_1 = c_{opt}$, then our search will be limited to $\binom{|\mathcal{A}|}{m-1}$ choices which would be manageable for small values of $|\mathcal{A}|$ and m . In the following, without loss of generality, we assume $\mathcal{A}$ to be the set of equi-distanced values $c_{opt} + i\Delta, i = 1, \dots, |\mathcal{A}|$. To find the optimal set $\mathcal{C} \setminus \{c_{opt}\}$ for a given m at a given SNR, γ , we first find a sufficiently large set $\mathcal{S}$ of input blocks that trap the decoder by using Monte Carlo simulations with the clipping

---

²If the information about the dominant trapping sets of code/decoding algorithm is available, an alternate approach to evaluate (estimate) the performance of the proposed scheme in the error floor region is to use that information together with importance sampling techniques.

threshold c_{opt} at γ . The set $\mathcal{S}$ is then re-quantized/re-decoded by different clipping thresholds, starting from $c_{opt} + \Delta$ all the way up to the maximum value $c_{opt} + |\mathcal{A}|\Delta$ by increments of Δ . Suppose the subset of $\mathcal{S}$ that is resolved by using the threshold $c_{opt} + i\Delta, i = 1, \dots, |\mathcal{A}|$, is denoted by s_i . The optimal set $\mathcal{C} \setminus \{c_{opt}\}$ is then obtained by finding a collection of $m - 1$ sets from $\{s_1, \dots, s_{|\mathcal{A}|}\}$ whose union has the largest size. Suppose this collection is $s_{i_1}, \dots, s_{i_{m-1}}$. Then the optimal $\mathcal{C} \setminus \{c_{opt}\}$ is the set $\{c_{opt} + i_j\Delta : j = 1, \dots, m - 1\}$. In the proposed scheme, parameter Δ represents the resolution of the clipping thresholds, and $|\mathcal{A}|$ determines the maximum dynamic range of the quantization scheme. In general, higher resolution and larger maximum dynamic range (smaller values of Δ and larger values of $|\mathcal{A}|$) would improve the performance at the expense of more complex search for finding the optimal clipping thresholds.

The search process can be simplified by selecting clipping thresholds of the set $\mathcal{C} \setminus \{c_{opt}\}$ in a greedy fashion, i.e., the first element is selected to be the clipping threshold with maximum $|s_i|$. Let $j = \arg \max_i (|s_i|)$. Then the second element of $\mathcal{C} \setminus \{c_{opt}\}$ is selected to be the threshold that can resolve the maximum number of blocks in $\mathcal{S} \setminus s_j$. The process continues until all the blocks in $\mathcal{S}$ are resolved or until $m - 1$ clipping thresholds (in addition to c_{opt}) are selected.

Our experiments show that both the optimal and the greedy schemes, described above, are rather sensitive to the change in SNR. In the absence of perfect knowledge of the SNR at the receiver, this would degrade the performance of the proposed scheme. To mitigate this problem, we have devised an even simpler scheme that assigns the clipping thresholds using $c_i = c_{opt} + (i - 1)\delta, i = 2, \dots, m$, where $\delta > 0$ is selected to maximize the number of resolved inputs from the set $\mathcal{S}$. This scheme, referred to as multi-step quantizer with *equal increments* performs slightly inferior to the greedy scheme but is generally more robust to the changes of SNR in the error floor region.

The parameter δ depends mainly on the code and the decoding algorithm.

It is important to note that the improvement in the error floor performance by the proposed scheme, in any of its incarnations, comes at the cost of a rather small increase in complexity, memory requirement and the latency of the decoder. The added memory is to store the non-quantized outputs of the coherent detector (to be used in possible subsequent attempts for re-quantization). An input buffer that accommodates a maximum of 16 bits per detector output would be sufficient to represent the outputs with the precision required for the implementation of the proposed scheme. The increased in latency is a result of re-quantization/re-decodings of trapped blocks. Since this operation is invoked rarely, the effect on average latency is negligible. In the worst-case scenario, the latency can increase by a factor of about m . Using techniques such as early trap detection [74], the worst-case latency, however, can be reduced significantly. The added complexity is for the extra operations required for re-quantization/re-decodings. This would include the complexity of the early trap detection module.

To implement the early trap detection, we use the algorithm described in Section III.A of [74]. This algorithm is called within each iteration of the decoding algorithm after the number of unsatisfied check nodes $|C_{cur}|$ is counted. One should note that the early trap detection module can also function as the unit responsible for distinction between the waterfall and error floor regions (see Footnote 1). For this, the value $|C_{cur}|$ is compared with a predetermined threshold τ . If $|C_{cur}| \leq \tau$, then the early trap detection algorithm is evoked. Parameter τ is selected as the maximum number of unsatisfied check nodes in steady-state decoding iterations of the 1000 blocks in $\mathcal{S}$.

4.3 Simulation Results

To analyze the performance of the proposed multi-step quantizers in the error floor region, we use a field programmable gate array (FPGA) to emulate the MS and offset-MS decoders. In our simulations, we set the maximum number of iterations to 100. We start our experiments with two regular LDPC codes: a (1008, 504) code with variable and check degrees 3 and 6, respectively [2]; and a quasi-cyclic (775, 312) code constructed in [3]. The latter code is particularly designed to have a good error floor performance. For offset-MS, we select the additive correction factor to be 1. For the (1008, 504) code, the optimal clipping threshold in the waterfall region c_{opt} is found to be 5.5 and 4.0 for MS and offset-MS, respectively. These values for the (775, 312) code are found to be 1.5 and 3.5, respectively. For both codes and decoders, we use $q = 5$, and at each SNR, we continue the simulations until 1000 block errors occur ($|\mathcal{S}| = 1000$). The performance of the two codes under MS and offset-MS with c_{opt} are reported in Figures 4.1(a) and 4.2(a), respectively. The figures show that for both codes and decoding algorithms, the error floor starts at SNR of about 3.25 dB.

To devise the proposed schemes, we select $\Delta = 0.1$ and $|\mathcal{A}| = 200$. For the (1008, 504) code, the sets that trap MS with $c_{opt} = 5.5$ at SNR = 4.6 dB are categorized in Table 4.1. In the first column of this table, a pair (a, b) denotes a class of trapping sets with a variable nodes whose induced subgraph in the code's Tanner graph has b check nodes of odd degree. In the second column, we have the multiplicity of different classes of trapping sets. The total number of trapping sets is $|\mathcal{S}| = 1000$. In 19 out of 1000 cases, the decoder is trapped in oscillatory patterns. Using the sets $s_i, i = 1, \dots, 200$, we verify that $\cup_{i=1}^{200} s_i = \mathcal{S}$, which means that the sequential application of the 200 clipping thresholds will resolve all the 1000 block errors. We then

Figure 4.1: Block error rate curves of the (1008,504) code of [2]. (a) MS ($\circ$) and offset-MS (∇) with (---) and without (—) optimal multi-step quantizer ($m=5$). (b) Optimal ($\triangle$), greedy (+) and equal-increment ($\square$) schemes applied to offset-MS in the error floor region: the scheme is optimized at each SNR (—); and the scheme optimized at 4.6 dB is applied to every SNR (---)

Figure 4.2: Block error rate curves of the (775,312) code of [3]. (a) MS ($\circ$) and offset-MS (∇) with (---) and without (—) optimal multi-step quantizer ($m=5$). (b) Optimal ($\triangle$), greedy (+) and equal-increment ($\square$) schemes applied to offset-MS in the error floor region: the scheme is optimized at each SNR (—); and the scheme optimized at 4 dB is applied to every SNR (---)

search for the optimal set $\mathcal{C} \setminus c_{opt}$ for different values of m starting from $m=2$. Our search indicates that $m=6$ is the minimum value of m for which all the 1000 blocks

in $\mathcal{S}$ can be resolved with the proposed multi-step quantizer, i.e., there is no need to apply 200 re-quantization/re-decodings, only five suffices. The corresponding thresholds are 5.6, 8.3, 11.3, 19.3, 21.9. The size of the subset of $\mathcal{S}$ that can be resolved by the optimal scheme for values of $m = 2, 3, 4$ and 5 are respectively 947, 992, 998 and 999. This indicates that by using only one re-quantization/re-decoding ($m = 2, c_2 = 18.2$), one can still improve the error floor significantly. In columns three to seven of Table 4.1, we have shown how the multiplicity of different classes of trapping sets changes by successive application of the re-quantization/re-decoding operations for the optimal scheme with $m = 6$. The table demonstrates that an increasingly larger portion of small trapping sets are eliminated by the proposed scheme until they are all resolved.

Table 4.1: Multiplicity of different classes of trapping sets after successive application of the re-quantization/re-decoding operations for the optimal scheme with $m = 6$ for the (1008,504) code at SNR=4.6 dB using MS

Trapping Set Class	$c_{opt} = 5.5$	$c_2 = 5.6$	$c_3 = 8.3$	$c_4 = 11.3$	$c_5 = 19.3$	$c_6 = 21.9$
(6,2)	673	542	98	24	0	0
(8,2)	240	208	49	19	4	0
(9,3)	1	1	0	0	0	0
(10,2)	52	45	9	0	0	0
(12,2)	11	10	6	3	0	0
(13,3)	2	0	0	0	0	0
(14,2)	1	1	0	0	0	0
(15,3)	1	0	0	0	0	0
Oscillation	19	11	0	0	0	0

The trends for offset-MS are similar to those of MS. For the (1008,504) code at SNR = 4.6 dB, the maximum number of blocks in $\mathcal{S}$ that can be resolved with the proposed scheme with $\Delta = 0.1$ is 998. The smallest m that can achieve this

performance is 8, and the sequence of thresholds are 4.0, 4.5, 5.7, 6.6, 7.1, 7.2, 7.4, 7.5. With only one re-quantization/re-decoding ($m = 2, c_2 = 6.7$), 947 out of 1000 blocks are resolved.

For the (775, 312) code and MS, at SNR = 4 dB, all the blocks in $\mathcal{S}$ can be resolved with the proposed scheme using thresholds 1.5, 1.6, 8.3, 10.5 and 12.5 ($m = 5$). With only one re-quantization/re-decoding with $c_2 = 12.3$, 983 out of 1000 blocks are resolved. For offset-MS at 4 dB, 996 of the blocks in $\mathcal{S}$ are resolved with $m = 5$ using thresholds 3.5, 5.6, 6.3, 6.7 and 7.1. Increasing m beyond 5 does not provide further improvement. For $m = 2$ with $c_2 = 6.7$, 942 blocks are resolved.

In Figures 4.1(a) and 4.2(a), we have plotted the performance of the optimal multi-step quantizer with $m = 5$ for MS and offset-MS for the two codes, respectively. The point with a downward arrow shows that the error is lower than the estimated value on the figure. This happens when all the blocks in $\mathcal{S}$ are resolved by the proposed scheme, and the error is over-estimated by assuming one block error (rather than zero). The figures show significant improvement in the error floor region for both codes and both decoding algorithms.³

To compare the optimal, greedy and equal-increment multi-step quantizers, we consider the application of MS to the (1008, 504) code at 4.6 dB. We fix $m = 5$ and the resolution of quantization to $\Delta = 0.1$. For this scenario, the optimal scheme resolves 999 out of 1000 blocks. The greedy quantizer has thresholds 5.5, 9.9, 11.3, 18.2, 19.6 and resolves 996 of the trapped blocks. The equal-increment quantizer with $m = 5$ performs the best with $\delta = 5$ and resolves 994 of the blocks. In general, the optimal quantizer has the best performance, followed by the greedy and the equal-increment quantizers. To examine the sensitivity of the proposed quantizers to SNR, we consider the application of MS to the (1008, 504) code but at SNR = 3.8 dB. At this SNR,

---

³It should be noted that due to the significantly smaller number of block errors for the dot-dashed curves compared to solid curves, the former curves may be less accurate than the latter.

we find the best performing quantizer in each category (optimal, greedy and equal-increment) for $m = 5$. These quantizers resolve 994, 992 and 977 out of 1000 trapped blocks, respectively. Now, if we apply the best-performing quantizers devised for 4.6 dB at 3.8 dB, the number of resolved blocks will decrease to 957, 980 and 973, respectively. This corresponds to a change of about 3.7%, 1.2% and 0.4%, respectively, indicating that the optimal quantizer is the most sensitive, followed by the greedy quantizer, and the least sensitive is the equal-increment quantizer. In Figures 4.1(b) and 4.1(b), we have reported the performance of each of the three schemes with offset-MS for two scenarios: the scheme is optimized at each SNR (full line), and the scheme optimized at the highest SNR is applied to every SNR (dash-dotted line). Similar trends as described above can be seen in these figures.

To investigate how the proposed scheme affects the latency of the decoder, in Table 4.2, we have reported the average number of iterations for all re-quantization/re-decodings of the input blocks in $\mathcal{S}$ under the optimal multi-step quantizer with $m = 5$ for both codes and both decoding algorithms at the highest SNR reported in Figures 4.1 and 4.2. As a reference, for each case, we have also reported the average number of iterations it takes to detect that the decoder with c_{opt} is trapped in a trapping set for the blocks in $\mathcal{S}$. For both the initial trap detection and the trap detection during the re-decodings, we use the early trap detection Algorithm of [74]. Parameters (N_{rep}^{max}, τ) for the four cases of Table II are selected to be (3,11), (4,10), (3,18) and (4,11), respectively. In Table 4.2, we have also reported the largest number of iterations for blocks in $\mathcal{S}$ as they are processed through the sequence of thresholds in $\mathcal{C}$. This corresponds to the worst-case latency. These results indicate that, on average, the proposed scheme increases the latency of the trapped cases by a factor of about two. Since the decoder is rarely trapped, this has negligible effect on the overall average latency of the decoder. In the worst-case scenario, the latency is increased by

a factor of about 4. (The latency results of Table II do not appear to be very sensitive to the change of SNR for the tested cases. For example, the results of MS for the (1008, 504) code at 3.8 dB are 10.20, 12.5 and 46, respectively.) The size of the input buffer required to store the detector outputs for the (1008, 504) and (775, 312) codes is about 16 Kbits and 12 Kbits, respectively.

Table 4.2: Latency Information of the Proposed Scheme for Trapped Cases

Code	SNR (dB)	Decoding algorithm	Corrected Blocks	latency (number of iterations)		
				re-decodings	initial decoding	worst case
(1008,504)	4.6	MS	999	9.43	10.4	42
(1008,504)	4.6	offset-MS	994	10.75	11.3	45
(775,312)	4	MS	1000	12.80	11.2	38
(775,312)	4	offset-MS	996	9.61	11.4	43

As the final example, we consider the irregular (2304, 1152) code in IEEE 802.16e WiMAX standard. For the decoding, we choose offset-MS with $q = 6$, $c_{opt} = 4.0$ and additive correction factor 1. To implement this decoder in FPGA, we have used layered architecture [43] with the maximum number of iterations 20. An error floor starts at SNR of about 2.2 dB. We devise an optimal multi-step quantizer with $m = 6$ for this combination of code/decoder at 2.4 dB. This quantizer has clipping thresholds 4.0, 5.0, 5.7, 6.2, 7.3, 7.5, and is capable of resolving 991 out of 1000 blocks in $\mathcal{S}$. Using this quantizer, the error floor at 2.4 dB, which was about 3×10^{-6} , is thus lowered to about 2.7×10^{-8} . In terms of the added latency for the trapped blocks, using the early trap detection, the average number of iterations for re-decodings is 11.02 compared to 12.3 for the average number of iterations of the initial decoding. The largest number of iterations for trapped blocks is 39.

4.4 Conclusion

In this chapter, we proposed a simple method to lower the error floor of LDPC codes decoded by fixed-point message-passing decoders, without sacrificing the waterfall performance. In this method, we used a multi-step scheme for the input quantization of message-passing decoders for LDPC codes. The proposed scheme resulted in an improvement of about 2 to 3 orders of magnitude in the error floor region for different codes and decoding algorithms.

Chapter 5

Estimating the Error Floor of variable-regular LDPC codes

5.1 Introduction

In this chapter, we propose a fast and accurate technique to estimate the error floor of variable-regular LDPC codes under quantized iterative decoding algorithms. As mentioned earlier, although the method of [15] was demonstrated to be very effective for short LDPC codes with variable degree 3, its application to block lengths larger than about 1000 and variable degrees larger than 3 is hampered by large computational complexity. In this chapter, we address this issue by modifying the way that the error locations are selected.¹ Instead of using the set T_k , we use a different set denoted by $T_{A,B}$ which consists of all the (a, b) *elementary trapping sets* (ETS) [10,16] of size a less than or equal to A , where for each a , we use the ETSs with the value b in the range of $b_{min}^{(a)} \leq b \leq B$ with $b_{min}^{(a)}$ being the smallest b for (a, b) ETSs with a given a . We demonstrate that using the new set with the proper choice of A and B , the

---

¹It is important to note that our work in this work is focused only on the way that error locations are selected. We thus make no change in the approach of [15] for generating the error patterns or the technique used in [15] for the estimation of error floor.

total number of test input vectors that are required to provide an accurate estimate of the error floor reduces substantially compared to the modified cycle enumeration technique of [15]. We also demonstrate that using only the *absorbing sets* [14] (rather than all ETSs) does not provide an accurate estimate of the error floor.

5.2 The Proposed Scheme

For the error floor estimation, we follow the general approach of Algorithm 1 in [15], known as “modified cycle enumeration.” The main change that we make is the replacement of the set T_k with the set $T_{A,B}$, described in the previous section. Correspondingly, we refer to the new algorithm, which is described in Algorithm 1, as “ETS enumeration.” The list $PL_{A,B}$ at the input of the algorithm contains error patterns corresponding to the elements in the set $T_{A,B}$ in increasing order of the distance to $\vec{R}$, and $D'_{A,B}$ is the list of distances corresponding to the error patterns in $PL_{A,B}$. Algorithm 1 continues with testing input vectors up until the distance d of the tested error patterns is large enough to satisfy $d^2 > d_m^2 + \beta$, where d_m is the minimum distance of error patterns that failed the decoder, and β is a constant selected to provide an accurate estimate, as will be discussed in Section III. To generate the error rate estimate in Step 5 of Algorithm 1, we only use the first term in Equation (6) of [15]. As discussed in [15], the second term, $P(S_{d>d_{th}})$, which denotes the probability of input vectors whose Euclidean distance to $\vec{R}$ is larger than a certain threshold d_{th} , mainly contributes to the accuracy of the estimate in the waterfall region, and has negligible effect in the error floor region.

Algorithm 1: ETS Enumeration

- 1: Create $T_{A,B}$, $PL_{A,B}$ and $D'_{A,B}$.
- 2: $d = \min\{x : x \in D'_{A,B}\}$, $d_m = \infty$.
- 3: Decode all patterns in $PL_{A,B}$ of distance d to $\vec{R}$ and corresponding to locations in $T_{A,B}$.
- 4: If there are patterns that cannot be correctly decoded, $d_m = \min\{d, dm\}$ and update set $\tilde{E}_d$.
- 5: If $d_m = \infty$ or $d^2 < d_m^2 + \beta$, then $d =$ the next larger distance in $D'_{A,B}$, and go to step 3;
else, stop and generate the error rate estimate.

To generate the set $T_{A,B}$ for a given code, we mainly rely on [16] which characterizes the (a, b) ETSs of variable-regular LDPC codes with relatively small values of a and b . Based on the results presented in [16], we are able to efficiently find *all* the ETSs in the ranges of a and b that are of interest in this work.

Figure 5.1: Normal graphs of three non-LSS ETS structures in a Tanner graph with variable degree 4 and girth 6.

In [16], it is proved that the vast majority of (a, b) ETS structures with small values of a and b are LSS of short cycles, in the sense that for each such structure there is a nested sequence of ETSs that starts from a short cycle and grows, one variable node at a time, to generate the structure. Associated with the LSS property, there is an efficient search algorithm (Algorithm 1 of [16]) that starts from the short cycles of the graph and can find *all* ETSs with LSS property in a guaranteed fashion.

For ETSs with LSS property, we thus use Algorithm 1 of [16] to find them. For the few ETS structures that are not LSS of any cycles, we carefully examine their structures. Our study shows that all such ETS structures are composed of two ETSs with LSS structure sharing a variable-node, or connected by either one check-node or a chain consisting of two check-nodes and one variable-node. An example of each structure for a code with variable degree 4 and girth 6 is presented in Fig 5.1. The structures are shown by their normal graph in Fig. 5.1.² Fig. 5.1(a) is the normal graph of a (7, 8) ETS which consists of two LSS ETSs (3, 6) and (5, 6) sharing a variable-node. Fig. 5.1(b) is the normal graph of an (8, 6) ETS which consists of two LSS ETSs (3, 6) and (5, 2) connected by a check-node, and Fig. 5.1(c) is the normal graph of an (8, 10) ETS which consists of two LSS ETSs (3, 6) and (4, 4) connected by two check-nodes and one variable-node. Knowing all the structures of non-LSS (a, b) ETSs in the range of a and b values of interest, we then identify their constituent LSS structures and search the graph for those constituent LSS structures that are so connected to create the non-LSS structures (through a shared variable-node, a single check-node or two check-nodes and a variable-node).

To compare the complexity of our technique with that of modified cycle enumeration, we mainly focus on the number of input vectors that need to be tested to obtain the error estimates in each case. This is the same as the number of iterative decoding instances that are required for the derivation of each estimate, and accounts for the majority of computational complexity.

---

²The normal graph of an ETS in a variable-regular Tanner graph is obtained from the induced subgraph of the ETS in the Tanner graph by removing all the check nodes of the subgraph: for degree-one check nodes, the node and its adjacent edge are removed; for degree-two check nodes, the node and its two adjacent edges are replaced with an edge [16].

5.3 Experimental results

We have applied our method of selecting the error locations along with the general estimation method of [15] to a large number of variable-regular LDPC codes. In this section, however, we only report our results for three randomly constructed codes. We focus on random codes here because they have a large variety of TSs compared to structured codes and are thus more difficult to analyze. The block length n and dimension k of the codes are: $(n, k) = (1008, 504), (8000, 4000), (816, 408)$. We refer to these codes as $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$, respectively. The codes $\mathcal{C}_1$ and $\mathcal{C}_2$ have variable-node and check-node degrees of $d_v = 4$ and $d_c = 8$, respectively, and are obtained from [15] and [2], respectively. Code $\mathcal{C}_3$ is taken from [2] and has $d_v = 5$ and $d_c = 10$. All three codes have a girth of $g = 6$. For $\mathcal{C}_1$, following [15], we use $C_{th} = 1.8$ and $I_{max} = 50$, where I_{max} is the maximum number of iterations for the MS algorithm. For $\mathcal{C}_2$ and $\mathcal{C}_3$, we use $C_{th} = 1.2$ and $I_{max} = 30$. In general, clipping thresholds C_{th} are selected to minimize the frame error rate (FER) in the waterfall region [11]. For all simulations, the number of quantization bits q is chosen to be 3. For larger number of quantization bits, the results for smaller values of q (say $q = 3$), obtained by the technique presented here, can be used as the input to Algorithm 2 of [15] to efficiently and accurately estimate the error floor.

Our study indicates that, for all three codes, $A = 10$ is large enough to cover the range of all ETS classes with non-negligible effect on the error floor. Thus, we first find all the (a, b) ETSs with $a \leq 10$ and $b \leq 10$. We then investigate the effect of different classes of ETSs on the error floor. Our results in general indicate that for each value of a , one only needs to include the class of $(a, b_{min}^{(a)})$ in the error floor estimation to obtain a good estimate. We denote the set of all (a, b) ETSs with $a \leq A$ and $b = b_{min}^{(a)}$ for each a in this range by $T_{A, b_{min}}$. The list of such ETS classes and

their multiplicity for $A = 10$ are provided in Table 5.1 for the three codes. The last row of the table shows the total number of $(a, b_{min}^{(a)})$ ETSs with $a \leq 10$, which is the same as $|T_{10, b_{min}}|$, for each code.

Table 5.1: $(a, b_{min}^{(a)})$ ETS classes with $a \leq 10$ and their multiplicity for $\mathcal{C}_1 - \mathcal{C}_3$

$(1008, 504)_{(4,8)}$		$(8000, 4000)_{(4,8)}$		$(816, 408)_{(5,10)}$	
ETS Class	Multiplicity	ETS Class	Multiplicity	ETS Class	Multiplicity
(3,6)	1560	(3,6)	1620	(3,9)	7879
(4,6)	347	(4,6)	55	(4,8)	30
(5,6)	105	(5,6)	2	(5,9)	98
(6,4)	1	(6,8)	177	(6,8)	1
(7,6)	13	(7,8)	10	(7,9)	5
(8,6)	5	(8,10)	941	(8,8)	1
(9,6)	5	(9,10)	90	(9,9)	3
(10,6)	3	(10,10)	12	(10,10)	4
$ T_{10, b_{min}} $	2039	$ T_{10, b_{min}} $	2907	$ T_{10, b_{min}} $	8021

From Table 5.1, one can see the random nature of the existing ETSs and their multiplicity for the three codes. Comparing the results for $\mathcal{C}_1$ and $\mathcal{C}_2$, we can see that increasing the block length in general increases $b_{min}^{(a)}$. This would in turn translate to an improved error floor. The comparison of the results for $\mathcal{C}_1$ and $\mathcal{C}_3$ demonstrates a similar trend when the variable- and check-node degrees are increased.

We note that while within each ETS class of each code, there may exist a number of different non-isomorphic structures,³ the study of such structures and their relative

³Two graphical structures G_1 and G_2 with undirected edges and node sets V_1 and V_2 , respectively, are isomorphic if there is a bijection $f : V_1 \rightarrow V_2$ such that nodes $v_1, v_2 \in V_1$ are joined by an edge if and only if $f(v_1)$ and $f(v_2)$ are joined by an edge.

harmfulness is beyond the scope of this work. Instead, we would like to focus on the fact that despite the variety of non-isomorphic structures within each class, the total number of $(a, b_{min}^{(a)})$ ETSs with $a \leq 10$ for each code (the last row of the table) is relatively small. In addition, a more careful inspection of the structures in Table 5.1 reveals that, for each code, many of the smaller ETSs are subsets of larger ones, and thus can be removed from the set $T_{10, b_{min}}$ since their contributions will be accounted for through their supersets. Taking this into account, the size of $T_{10, b_{min}}$ can be further reduced to 1130, 2520 and 7286, for the three codes, respectively.

In Figs. 5.2–5.4, we have provided the error floor estimates obtained using our technique and based on the ETSs listed in Table 5.1. Through the estimation process, if the minimum Euclidean distance from $\vec{R}$ of error patterns that fail the decoder are found to be $\tilde{d}_m$, to obtain the estimates, we generate and test error patterns with distances up to $\sqrt{\tilde{d}_m^2 + 10}$ from $\vec{R}$. Increasing the range of distances beyond this value will have negligible effect on improving the estimates. In Figs. 5.2–5.4, for comparison, we have also included the Monte Carlo simulation results. To demonstrate the convergence behavior of the estimates as larger ETSs are included, for each code, we have plotted the estimates based on $T_{A, b_{min}}$ for $A = 5, \dots, 10$. As can be seen, the estimates improve and demonstrate a convergence behavior as larger ETSs are included. For all codes, the estimate based on $T_{10, b_{min}}$ closely matches the Monte Carlo simulation.

It is important to note that for the modified cycle enumeration technique to provide an estimate with similar accuracy as those presented here, a much larger number of input vectors need to be tested. For $\mathcal{C}_1$, to obtain an estimate as accurate as ours using modified cycle enumeration, one needs to use cycles of length 6, 8 and 10. The number of such cycles in $\mathcal{C}_1$, which is the same as $|T_2|$, is 434,317. This corresponds to about 7×10^9 input test vectors. For our method, on the other hand, the number

of test vectors based on the ETSs of Table 5.1 is only about 10^9 which is less than what is required for modified cycle enumeration by a factor of 7. For comparison, the estimate obtained by the modified cycle enumeration using T_2 is also given in Fig. 5.2. As can be seen, this estimate is very close to ours. Both techniques find the same estimate $\tilde{d}_m = \sqrt{91}$.

For $\mathcal{C}_2$, using modified cycle enumeration with T_2 results in a very poor estimate of the error floor. Even by enlargening the set of error locations to T_3 , still the estimate is rather far from the Monte Carlo simulations. This can be seen in Fig. 5.3. Note that $|T_3|$ for $\mathcal{C}_2$ is about 7.6×10^6 and the corresponding number of input test vectors is about 8.4×10^{11} . In comparison, the number of input test vectors for our scheme is about 1.2×10^{10} which is less than that of modified cycle enumeration by a factor of about 70. For this code, $\tilde{d}_m$ for our scheme and modified cycle enumeration with T_3 is $\sqrt{179}$ and $\sqrt{205}$, respectively. So the modified cycle enumeration misses some of the most dominant error patterns, and thus results in a poor estimate of the error floor. (Note that for $\mathcal{C}_2$, using T_4 in modified cycle enumeration is practically infeasible since the number of cycles of length 14 in the Tanner graph of $\mathcal{C}_2$ is more than 10^9 .)

Similar to $\mathcal{C}_2$, the error floor of $\mathcal{C}_3$ is also out of the reach of modified cycle enumeration. The estimate obtained using T_2 (consisting of cycles of length 6, 8 and 10) is reported in Fig. 5.4. As can be seen, this estimate is off by a factor of about 50. The estimated value $\tilde{d}_m$ in this case is $\sqrt{145}$ versus $\sqrt{130}$ obtained by our technique. Note that $|T_2|$ for this code is about 6.3×10^6 , and for the implementation of modified cycle enumeration about 10^{11} input vectors need to be tested. For our scheme, on the other hand, only about 1.2×10^9 input test vectors are required, more than a factor of 83 less than what is required by modified cycle enumeration. (Note that for $\mathcal{C}_3$, using T_3 in modified cycle enumeration is practically infeasible as the number of

Figure 5.2: Simulation and estimation results of $\mathcal{C}_1$ decoded by 3-bit MS ($C_{th} = 1.8$ and $I_{max} = 50$)

cycles of length 12 in the Tanner graph of $\mathcal{C}_3$ is more than 10^9 .)

It is worth noting that using a desktop computer with an Intel Xeon 2.27GHz CPU and 2GB memory, the ETS enumeration (Algorithm 1) implemented in C language took about 7, 112 and 13 hours to obtain the estimates presented in Figs. 5.2- 5.4, respectively.⁴

As another experiment, we obtained all the ETSs of the three codes that are also

⁴The running times are based on testing only the error patterns that have at least one negative coordinate. This change, which was also implemented in [15], has negligible effect on the accuracy of estimates.

Figure 5.3: Simulation and estimation results of $\mathcal{C}_2$ decoded by 3-bit MS ($C_{th} = 1.2$ and $I_{max} = 30$)

absorbing sets. For $\mathcal{C}_1$, among all the ETSs listed in Table 5.1, only two are absorbing sets: the only $(6, 4)$ ETS and one of the five $(9, 6)$ ETSs. For $\mathcal{C}_2$, none of the ETSs in Table 5.1 is an absorbing set. Finally, for $\mathcal{C}_3$, out of 8021 ETSs of Table 5.1, only 69 are absorbing sets: $30 \times (4, 8)$, $35 \times (5, 9)$, $1 \times (6, 8)$, $1 \times (7, 9)$, $1 \times (8, 8)$, $1 \times (10, 10)$. The estimation results based on the absorbing sets for $\mathcal{C}_1$ and $\mathcal{C}_3$ are presented in Figs. 5.2, and 5.4, respectively. As can be seen, none of the estimates is very accurate. For $\mathcal{C}_2$, no estimation can be made as none of the ETSs in Table 5.1 is an absorbing set.

Figure 5.4: Simulation and estimation results of $\mathcal{C}_3$ decoded by 3-bit MS ($C_{th} = 1.2$ and $I_{max} = 30$)

5.4 Conclusion

In this chapter, we presented a fast and accurate technique for the error floor estimation of variable-regular LDPC codes. This technique, which is based on enumerating the dominant elementary trapping sets of the code, provides significant improvement over existing methods in terms of speed and accuracy.

Chapter 6

Harmfulness of trapping sets of LDPC codes

6.1 Introduction

In this chapter, we study the relative harmfulness of trapping sets with the same class (a, b) and the same topological configuration in variable-regular LDPC codes. We show that, other than the size of a trapping set, the number of unsatisfied check nodes and its topological properties, the position of its induced subgraph in the Tanner graph of the code can also affect its harmfulness. We focus on ETSs that are LSS of short cycles, and shown to be the most harmful ones [16]. We look into two cases by simulating the performance of two selected regular LDPC codes using quantized min-sum (MS) algorithm in the log-likelihood ratio (LLR) domain over the AWGN channel.

Then, we examine dominant trapping sets of the quantized min-sum (MS) decoding algorithm and two of its variants (offset-MS and SR-MS), for both regular and irregular LDPC codes. We present our results based on 2 regular and 2 irregular codes. For each set of regular/irregular codes, we examine one randomly constructed

and one quasi-cyclic code. We show that for regular codes, the relative harmfulness of these ETSs is similar for MS and its two variants. In irregular codes, however, we show that dominant trapping sets change by changing the decoding algorithm. As explained in [75], most irregular LDPC codes have a substantial proportion of degree 2 variable nodes to have a good performance. However, these degree-2 variable nodes, forming 2-chains, play an important role in formation of dominant trapping sets in these codes [75]. We show that the harmfulness of these dominant trapping sets formed by degree-2 variable nodes, change depending on the decoding algorithm. For offset-MS, 2-chains involving small number of variable nodes are the dominant trapping sets. For MS, 2-chains with larger number of variable nodes or trapping sets composed of 2-chains are the dominant trapping sets. And for SR-MS, larger trapping sets involving cycles and 2-chains are the most dominant trapping sets. Based on these results, we evaluate the dominant trapping sets of a parallel decoder, consisting of MS, offset-MS and SR-MS decoding. For the parallel decoder, the dominant trapping sets would be the intersection of all trapping sets. In regular codes, the parallel decoder slightly improves the error floor performance while in irregular codes, it improves the error floor by one order of magnitude.

Finally, using the error floor performance results of the parallel decoder, we present a simple scheme to lower the error floor of irregular LDPC codes. We show that this scheme, with a cost of small increase in latency and memory of the decoder, lowers the error floor of irregular LDPC codes by one order of magnitude.

6.2 Harmfulness of trapping sets of variable-regular LDPC codes

6.2.1 ETS layered growth algorithm

In [16], for a given variable node degree d_v and girth g , all the non-isomorphic structures of rather small ETSs are categorized to be either LSS of a short cycle or not. As mentioned earlier, Algorithm 1 in [16], provides a method to find all ETSs of a variable-regular LDPC code with LSS property, which are known to be the most harmful trapping sets.

In this section, we look at the harmfulness of trapping sets having the same class and topological configuration. We start with the following case study. We consider the variable-regular (1008, 504) code obtained from [2]. This code has a girth of $g = 6$ and variable and check node degrees of $d_v = 3$ and $d_c = 6$, respectively. We use an FPGA and emulate the error rate of this code over the AWGN channel. In our simulations, the decoding algorithm is quantized MS with the number of quantization bits equal to 5 and the maximum number of iterations is set to 100. With these simulation parameters, the error floor for this code starts at an SNR equal to 3.4 dB. Using the hardware emulator, we continue the simulation well into the error floor region, and at 4.6 dB, continue the simulation until 3000 block errors occur. Simulation results show that the dominant ETSs are the (6,2) and (8,2) ETS classes. Using the graph isomorphism tool *nuaty* [76] and the method explained in [16], we generate all non-isomorphic topological configurations of the (6,2) and (8,2) ETS classes. We also find all (6,2) and (8,2) ETSs in the code using Algorithm 1 in [16]. We define the LSS number of a trapping set as the length of the shortest cycle in the Tanner graph of the code for which the trapping set satisfies the LSS property. In a left-regular

graph with $d_v = 3$ and $d_c = 6$, there are 3 (6,2) ETS structures with an LSS number of 8 and one (6,2) ETS with an LSS number of 10. Figures 6.1 and 6.2 show the (6,2) configurations with LSS numbers of 8 and 10, respectively. In these figures, and the rest of this thesis, variable nodes are represented by circles, and satisfied and unsatisfied check nodes are shown by empty and full squares, respectively.

Figure 6.1: (a),(b),(c): Configurations {1} and {2} and {3} of the (6,2) ETS class with an LSS number of 8 in a Tanner graph with variable degree 3 and girth 6, respectively.

Figure 6.2: Configuration {4} of the (6,2) ETS class with an LSS number of 10 in a Tanner graph with variable degree 3 and girth 6, respectively.

Also, for the (8,2) ETS class, there are 9, 7 and 1 non-isomorphic topological

configurations, with LSS numbers of 10, 12 and 14, respectively.

Table 6.1 shows all (6,2) and (8,2) trapping sets of this code. The first and second columns show the ETS class and the index of variable nodes in the ETS, respectively. The third and forth columns show the topological configuration and LSS number of the trapping set, respectively. And the fifth and sixth columns show the multiplicity and the fail rate of the ETS, respectively.

Table 6.1: Multiplicity and failure rate of each of the dominant trapping sets of the (1008, 504) code.

ETS Class	Variable nodes in the ETS	Configuration	LSS Number	Multiplicity	Fail Rate
(6,2)	26 193 383 597 744 746	{1}	8	26	3.97E-10
	99 330 419 595 697 885			962	1.47E-08
	107 247 388 566 797 900	{2}	8	29	4.43E-10
	71 287 431 517 854 938	{3}	8	1000	1.53E-08
	146 223 263 295 465 898	{4}	10	0	0.00E+00
(8,2)	12 17 154 632 669 745 771 968	{1}	10	36	5.50E-10
	19 130 193 208 290 535 744 771	{2}	10	295	4.50E-09
	62 200 221 322 436 511 905 972	{3}	10	356	5.44E-09

As can be seen in Table 6.1, the first two (6,2) ETSs have the same topological configuration {1} and thus the same LSS number of 8. However, the multiplicity and resulted failure rate of these two trapping sets are substantially different. The (6,2) ETSs in the third and forth rows which have the same LSS number of 8 and different configurations of {2} and {3}, respectively, also have different multiplicity and failure rates. The (6,2) ETSs in the fifth row has an LSS number of 10, and multiplicity and failure rate of zero. The three (8,2) ETSs, all have the same LSS number of 10 and different configurations. The multiplicity of the first (8,2) ETS with configuration

$\{1\}$ is lower than the other two (8,2) ETSs with configurations $\{2\}$ and $\{3\}$.

Now, we investigate the relationship between the position of the induced subgraph of the (6,2) and (8,2) ETSs in the Tanner graph of the (1008, 504) code and their harmfulness. We grow the subgraph of each ETS within the Tanner graph of the code by one level. To do this, for all the nodes in the subgraph, including variable nodes, satisfied and unsatisfied check nodes, we add all the neighboring nodes. Fig. 6.3 shows this process for a (3,3) ETS in a graph with a girth of $g = 6$ and variable and check node degrees of $d_v = 3$ and $d_c = 6$, respectively. The original ETS is shown in Fig. 6.3(a), and the expanded graph is shown in Fig. 6.3(b). We find that in some cases, the new graph forms new cycles and larger trapping sets, i.e., there is an intersection between the set of the new added nodes, and the set of the original nodes in the subgraph.

Figure 6.3: (a) A (3,3) ETS in a graph with a girth of $g = 6$ and variable and check node degrees of $d_v = 3$ and $d_c = 6$, respectively. (b) The result of one level growth.

We grow all the ETSs in Table 6.1 with the method explained above. For (6,2) ETSs with configuration $\{1\}$, growing the $\{26,193,383,597,744,746\}$ ETS, unlike the $\{99,330,419,595,697,885\}$ ETS, results in a new trapping set. Fig. 6.4 shows the $\{26,193,383,597,744,746\}$ ETS and the result of its growth. As it is shown in Fig. 6.4(b), growing this (6,2) ETS will result in a (8,4) trapping set. Variable node

130, which is connected to variable node 193 through check node 437, is also connected to variable nodes 744 and 597 through check node 313. Also, variable node 985, the is neighbor of two satisfied check nodes 464 and 313. In the grown trapping sets, check node 437 becomes satisfied check node and check node 464 becomes unsatisfied. Also, two new unsatisfied check node are added to the graph, resulting in a new (8,4) TS, which is not an ETS.

The (6,2) ETS with configuration $\{2\}$, as shown in Fig. 6.5, also grows into a (7,3) trapping set. The (6,2) ETS with configuration $\{3\}$ does not grow into a new trapping sets. And as shown in Fig. 6.4, the (6,2) ETS with an LSS number of 10 and configuration $\{4\}$, also grows into a (7,3) trapping set.

Figure 6.4: (a) The $\{26,193,383,597,744,746\}$ (6,2) ETS. (b) The (8,4) trapping set resulted after the growth.

Figure 6.5: Configuration $\{2\}$ of the (6,2) ETS and the (7,3) trapping set as a result of its growth in the Tanner graph of the (1008, 504) code.

Figure 6.6: Configuration $\{4\}$ of the (6,2) ETS and the (7,3) trapping set as a result of its growth in the Tanner graph of the (1008, 504) code

The same trend happens in the (8,2) class, in which the two ETSs with configurations $\{2\}$ and $\{3\}$, which are more harmful, do not grow into larger trapping sets, and

the ETS with configuration $\{1\}$, which is less harmful than others, grows into a larger trapping set. The latter ETS and the results of its growth are shown in Fig. 6.7.

Figure 6.7: Configuration $\{1\}$ of the (8,2) ETS and the (9,3) trapping sets as a result of its growth in the Tanner graph of the (1008, 504) code.

We continue with another example. We use a quasi-cyclic (775,312) code constructed in [3]. This code has a girth of $g = 10$ and variable and check node degrees of $d_v = 3$ and $d_c = 5$, respectively. This code is a quasi-cyclic code with cyclic liftings of degree 5, and all trapping sets appear in multiples of 5. For each group of 5 ETSs, we only consider the first one as their representative, and count the group of 5 as 1 and add their multiplicities in the results below.

Hardware simulations are performed for this code using a 5-bit MS decoder and at SNR=4.0 dB, which is in the error floor region, and 3000 block errors are collected. Simulations show that in this code, the (12,4) class is the most harmful. Table 6.2 shows all (12,4) ETSs in this code and their multiplicity out of the collected 3000 block errors. As can be seen in Table 6.2, other than one of the two ETSs with configuration $\{4\}$, all (12,4) ETSs with an LSS number of 16 have similar multiplicities between

110-137. And (12,4) ETSs with an LSS number of 18 and configuration $\{12\}$ have the lowest multiplicities between 19-32. The result of the growth of all (12,4) ETSs shows that the ETS with variable nodes $\{41,69,107,122,218,254,352,487,537,552,622,646\}$, is the only ETS which grows in a larger trapping set and also has the lowest multiplicity among all ETSs with the same LSS number. Also, (12,4) ETSs with an LSS number of 18 have smaller multiplicity than the ones with an LSS number of 16.

Table 6.2: Multiplicity of the dominant trapping sets of the (775, 312) code.

ETS Class	Variable nodes in the trapping set	Configuration	LSS Number	Multiplicity
(12,4)	16 205 321 395 436 486 602 641 687 726 755 757	{1}	16	110
	6 35 133 163 282 290 291 406 435 618 706 715	{2}		115
	6 154 199 243 303 387 399 406 458 484 535 749	{3}		120
	26 105 120 135 330 470 550 565 650 690 735 740	{4}		137
	41 69 107 122 218 254 352 487 537 552 622 646			70
	81 92 211 285 301 326 380 388 430 482 616 697	{5}		125
	51 66 126 131 326 496 561 571 646 671 686 731	{6}		111
	71 96 222 285 321 465 539 566 600 641 651 676			123
	36 160 222 270 328 375 414 466 509 706 746 773	{7}		127
	36 222 270 328 345 375 414 466 509 618 746 773	{8}		113
	41 76 97 206 226 322 421 487 557 617 642 727	{9}		125
	61 145 148 271 278 371 439 491 510 559 666 693	{10}	120	
	41 66 131 150 231 280 336 444 496 515 561 671	{11}	117	
	101 126 131 326 466 561 571 646 656 686 731 736	{12}	18	19
	101 126 131 326 466 561 571 646 671 686 731 736			29
	101 126 131 326 466 561 571 646 656 671 686 731			24
	26 40 55 70 120 485 500 550 560 660 675 740			32

The above experiments suggest that there is a strong relationship between the LSS

number and the position of the subgraph of an ETS in the Tanner graph of the code, and its harmfulness. These findings can help to relax the constraint of removing a complete class or topological configuration of trapping sets in code design, like what was done in [55]. However, we have not been able to find a general convincing explanation for this phenomenon.

6.3 Dominant trapping sets of MS and two of its variants for regular and irregular LDPC codes

In this section, we study dominant trapping sets of MS, offset-MS and SR-MS, and evaluate the error floor performance of a decoder, consisting of all three decoding algorithms working in parallel. We perform our experiments with four LDPC codes: a randomly constructed $(1008, 504)$ regular code with variable and check degrees 3 and 6, respectively [2]; a quasi-cyclic $(775, 312)$ regular code constructed in [3] with variable and check degrees 3 and 5, respectively; a randomly constructed $(1008, 504)$ irregular code from [2], having a variable node and check node degree distributions of $\lambda(x) = 0.4772x + 0.2808x^2 + 0.0347x^3 + 0.0972x^4 + 0.0089x^6 + 0.0010x^{13} + 0.1002x^{14}$ ¹ and $\rho(x) = 0.0099x^6 + 0.9782x^7 + 0.0119x^8$, respectively; and a quasi-cyclic $(1152, 576)$ irregular code from [1], having a variable node and check node degree distributions of $\lambda(x) = 0.4583x + 0.3333x^3 + 0.2083x^4$ and $\rho(x) = 0.6667x^5 + 0.3333x^6$, respectively. We refer to these codes as $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$, respectively.

In our simulations, we set the maximum number of iterations to 100 and use $q = 5$. For offset-MS, we select the additive correction factor to be 1, which gives the best results for $q = 5$. And for SR-MS, we consider the relaxation factor to be 0.5, which results in good performance and is preferred for hardware implementation. Using

---

¹In this representation, the coefficients are the percentage of edges of each degree

FPGA, we emulate the error rate of codes with MS, offset-MS, SR-MS, and parallel decoding algorithm over the AWGN channel. The motivation for studying a parallel decoder consisting of MS offset-MS, SR-MS is that it can be easily implemented in hardware using minimal changes to the same decoder. For example, a decoder can be implemented in such way that first, it tries to decode the input block using offset-MS, which has the best performance in the waterfall region. In case of a failure, the decoder repeats the decoding process by changing the additive correction factor inside check nodes to zero, which will change the decoding algorithm to MS, and in case of a second failure, shift and add blocks inside variable nodes are activated to apply the SR-MS update and decoder repeats decoding the input block. Also, similar to the technique in Chapter 4, early trap detection [74] can be used to reduce the latency.

For each LDPC code and decoding algorithm, we first find the optimal clipping threshold which results in the best performance in the waterfall region (c_{opt}), then continue the simulation until we reach a SNR which is well into the error floor region. At this point, we collect 3000 block errors. Also, to evaluate the error floor performance of the parallel decoder, we pass the exact same inputs through all the decoders in parallel and continue the simulation until we collect 100 block errors.

Table 6.3 shows the optimum clipping threshold and the tested SNR for each code and algorithm, respectively.

Table 6.3: Optimum waterfall clipping threshold and the error floor region SNR for all codes and algorithms, respectively.

Code	$\mathcal{C}_1$			$\mathcal{C}_2$			$\mathcal{C}_3$			$\mathcal{C}_4$		
Decoding Algorithm	MS	Offset-MS	SR-MS	MS	Offset-MS	SR-MS	MS	Offset-MS	SR-MS	MS	Offset-MS	SR-MS
c_{opt}	5.5	4	6.5	1.5	3.5	4	2.5	4	6	2	3	3.5
SNR (dB)	4.6			4			4.2			3.3		

Considering an (a, b) trapping set, in the error floor region, there are three types of error events [51]: 1) unstable error events, which dynamically change from iteration to iteration and for which a and b are typically large, 2) stable error events, for which a and b are small and are the main cause of the error floor and 3) oscillating trapping sets, which periodically vary with the number of decoder iterations and which are some times subsets of stable trapping sets.

Our simulations show that for tested codes and the three tested decoding algorithms, majority of error events are either stable or oscillatory trapping sets. Table 6.4 shows the percentage of appearance of dominant trapping sets of each code and decoding algorithm, at the output of the decoder. In this table, only trapping sets with a multiplicity of more than 1% of the total number of block errors are shown. For trapping sets with multiplicity of less than 1%, which also includes unstable cases, the total multiplicity is shown as “Other”.

First, we look at the harmfulness of trapping sets of the regular codes $\mathcal{C}_1$, $\mathcal{C}_2$. In $\mathcal{C}_1$, the (6,2), (8,2) and (10,2) trapping set classes are dominant trapping sets for all three decoding algorithms. All these trapping sets are ETSs with LSS property. Similarly, for $\mathcal{C}_2$, the (8,4), (10,4) and (12,4) trapping set classes are dominant for all three decoding algorithms, and they are also ETSs with LSS property. Since for both regular codes, the dominant trapping set classes are similar using all three decoders, a decoder consisting of all of them working in parallel also has the same dominant trapping sets. However, the parallel decoder does not have any oscillation cases.

Fig. 6.8 shows the performance of the regular codes with different algorithms. This figure shows that offset-MS performs better than MS and SR-MS, and the parallel decoder has similar (but slightly lower) error floor performance compared to offset-MS.

Table 6.4: Dominant trapping sets of each code and decoding algorithm and the percentage of appearance of dominant trapping sets at the output of the decoder.

Code	Trapping Set Class	Decoding Algorithm			
		MS	offset-MS	SR-MS	Parallel
$\mathcal{C}_1$	(6,2)	67.3	75.3	60.9	73
	(8,2)	23	16.5	28	22
	(10,2)	5.2	3.3	7.7	5
	Oscillation	1.9	3.2	0.3	0
	Other	2.6	1.7	3.1	0
	Total	100	100	100	100
$\mathcal{C}_2$	(8,4)	6.6	10	14.3	13
	(10,4)	38.1	18.1	19.2	19
	(12,4)	53.9	54.5	49	53
	Oscillation	1.2	3.8	1.6	0
	Other	0.2	13.6	15.9	18
	Total	100	100	100	100
$\mathcal{C}_3$	(1,2)	0	52.6	0	0
	(2,2)	0	6.6	0	0
	(7,1)	11.5	0	0	0
	(8,1)	10.3	0	0	0
	(9,1)	10.4	0	7.3	0
	(10,1)	0	0	11	0
	(13,0)	0	0	50.5	36
	Oscillation	44.3	3.9	6.6	23
	Other	23.5	36.9	24.6	42
	Total	100	100	100	100
$\mathcal{C}_4$	(1,2)	0	37.6	0	0
	(2,2)	0	6.4	0	0
	(3,2)	18.5	8.2	0	0
	(4,2)	27.3	0	0	0
	(5,2)	13.6	0	0	0
	(17,2)	0	0	7.4	0
	(18,2)	0	0	5.6	0
	(19,0)	0	0	16.5	10
	Oscillation	5.9	7.1	10.5	13
	Other	34.7	40.7	60	79
	Total	100	100	100	100

Now, we look at the harmfulness of trapping sets of the irregular codes $\mathcal{C}_3, \mathcal{C}_4$. As it is shown in Table 6.4, for irregular codes, dominant trapping sets change with the decoding algorithm.

Figure 6.8: Block error rate curves of the regular codes with different decoding algorithms. (a) Code $\mathcal{C}_1$ with MS ($\circ$), offset-MS (∇), SR-MS ($\times$) and parallel decoder ($\square$). (b) Code $\mathcal{C}_2$ with MS ($\circ$), offset-MS (∇), SR-MS ($\times$) and parallel decoder ($\square$).

Focusing on $\mathcal{C}_3$, using offset-MS, the dominant trapping set classes are: (1,2) and (2,2). The (1,2) and (2,2) classes are 2-chains consisted of one or two variable nodes, respectively. For MS, dominant trapping set classes are (7,1), (8,1) and (9,1). Further examination shows that these trapping sets are composed of 2-chains. Examples of (7,1), (8,1) and (9,1) trapping sets of $\mathcal{C}_3$, all consisting of 2-chains and variable nodes of different degrees, are shown in Figures 6.9(a), 6.9(b) and 6.9(c), respectively. For SR-MS, the dominant trapping sets are (9,1), (10,1) and (13,0). All these trapping sets are composed of 2-chains of 2, 3 and 4 variable nodes. For the parallel decoder, the dominant trapping set is (13,0), which is a codeword. This trapping set is shown in Fig. 6.10. Unlike regular codes, the number of oscillation cases are very high. And the number of cases recorded as “Other” is also high. These cases include a wide range of trapping sets of different sizes each appear in less than 1% of the total block errors collected, as well as some unstable cases. This means that, for this irregular code,

since stable dominant trapping sets of the three decoding algorithms are disjoint, trapping sets of large sizes become dominant trapping sets of the parallel decoder.

Figure 6.9: (a) A (7,1) trapping set of $\mathcal{C}_3$. (b) A (8,1) trapping set of $\mathcal{C}_3$. (c) A (9,1) trapping set of $\mathcal{C}_3$.

Figure 6.10: The (13,0) trapping set of $\mathcal{C}_3$.

Focusing on $\mathcal{C}_4$, the dominant trapping sets using offset-MS are (1,2), (2,2) and (3,2), which are 2-chains consisted of 1, 2 and 3 variable nodes, respectively. For MS, dominant trapping sets are (3,2), (4,2) and (5,2). These trapping sets are shown in Fig. 6.11(a)-(c), respectively. These trapping sets are 2-chains consisted of 3, 4 and 5 variable nodes, respectively. And for SR, the dominant trapping sets are (17,2), (18,2) and (19,0). These trapping sets contain 2-chains of 3 and 4 variable nodes, respectively. For the parallel decoder, the dominant trapping set is (19,0). In $\mathcal{C}_4$, similar to $\mathcal{C}_3$, the number of oscillation cases are very high, which we can relate to the

disjoint of dominant trapping sets of different algorithms working in parallel. Also, similar to $\mathcal{C}_3$, the number of cases recorded as “Other” is also high. These are also the addition of multiplicities of a wide range of trapping sets, each with a multiplicity of less than 1%, together with a few unstable cases.

Figure 6.11: (a) A (3,2) trapping set of $\mathcal{C}_4$. (b) A (4,2) trapping set of $\mathcal{C}_4$. (c) A (5,2) trapping set of $\mathcal{C}_4$.

Fig. 6.12 shows the performance of the irregular codes with different algorithms. In both irregular codes, SR-MS performs better than MS and offset-MS, and the parallel decoder improves the error floor of SR-MS by about one order of magnitude.

Using these results, we come up with a simple and practical scheme to improve the error floor of irregular LDPC codes. In this scheme, we start with offset-MS which has the best performance in the waterfall region. In the case of an unsuccessful decoding, we re-decode the input block using MS, for which we only need to set the additive correction factor to zero. And in the case of a second decoding failure, we activate a shift and addition block inside variable nodes to repeat the decoding process using SR-MS. The cost of this method is a slight increase in decoder complexity, increased input buffer and added latency.

As shown in Table 6.3, each decoding algorithm has a different optimum clipping threshold, so when we switch decoders, the input has to be re-quantized based on the optimum clipping threshold of the new decoder. Thus, an input buffer is required to keep the quantized input blocks for the second and third decoding algorithms. For

Figure 6.12: Block error rate curves of the irregular codes with different decoding algorithms. (a) Code $\mathcal{C}_3$ with MS ($\circ$), offset-MS (∇), SR-MS ($\times$) and parallel decoder ($\square$). (b) Code $\mathcal{C}_4$ with MS ($\circ$), offset-MS (∇), SR-MS ($\times$) and parallel decoder ($\square$).

example, for $\mathcal{C}_4$, assuming a 5-bit quantization, the size of the input buffer required to store quantized inputs is about 23Kbits.

As for the added latency, our simulations show that, the majority of input blocks which fail the first decoding algorithm (offset-MS), are decoded successfully using MS. For example, for $\mathcal{C}_3$, out of 3000 collected block errors at SNR=4.2 dB, 2460 blocks are resolved using MS and 41 blocks are resolved by further decoding using SR-MS. This process lowers the error floor of offset-MS by almost two orders of magnitude. And for $\mathcal{C}_4$, out of 3000 collected block errors at SNR=3.3 dB and offset-MS, 1713 blocks are resolved using MS, and 945 blocks are resolved by further decoding using SR-MS. For this code, the proposed scheme lowers the error floor of offset-MS by one order of magnitude. Moreover, by using technique such as early trap detection [74], we can reduce the added latency significantly.

6.4 Conclusion

In this chapter, first we studied the relative harmfulness of trapping sets with the same class and the same topological configuration in variable-regular LDPC codes. We showed that, the position of the induced subgraph of a trapping set in the Tanner graph of the code, can affect its harmfulness.

Then, we examined dominant trapping sets of the quantized min-sum (MS) decoding algorithm and two of its variants (offset-MS and SR-MS), for both regular and irregular LDPC codes. Based on these results, we evaluated the dominant trapping sets of a parallel decoder, consisting of MS, offset-MS and SR-MS decoding. Finally, we presented a simple scheme to lower the error floor of irregular LDPC codes. This scheme, with a cost of small increase in latency and memory of the decoder, lowers the error floor of irregular LDPC codes by one order of magnitude.

Chapter 7

Conclusions and future work

7.1 Conclusions

We proposed a serial-input design for the implementation of high-degree check nodes (SCN architecture) in a min-sum decoder of a protograph-based QC-LDPC code. We demonstrated that for high-rate codes, using the proposed architecture within the protograph-based partially-parallel (PPP) framework results in a superior throughput/latency compared to the existing layered, partially-parallel and fully-parallel architectures with comparable circuitry. In addition to superior throughput/area/latency trade-off, the SCN architecture induces a new column-layered message-passing schedule which outperforms the standard parallel (flooding) message-passing schedule in terms of error rate performance. While the results presented in this thesis were based on the combination of SCN and PPP architectures for a min-sum decoder, SCN architecture can also be used to implement modified min-sum algorithms.

We also proposed a method to lower the error floor of LDPC codes. In this method, we used a multi-step scheme for the input quantization of message-passing decoders for LDPC codes. The proposed scheme resulted in an improvement of about

2 to 3 orders of magnitude in the error floor region for different codes and decoding algorithms.

We also presented a fast and accurate technique for the error floor estimation of variable-regular LDPC codes. Our technique addresses the main limitation of the error floor method of [15]. Although, our results here are limited to variable-regular codes, they do extend the reach of the estimation method of [15] to larger block lengths and larger variable degrees by significantly improving the speed and the accuracy in comparison with the modified cycle enumeration of [15].

In addition, based on the results of this work, we established that ETSs are the relevant graphical structures in the error floor region of LDPC codes (at least variable-regular ones). On the other hand, our examples demonstrated that, in general, absorbing sets alone do not provide an accurate estimate of the error floor of randomly constructed LDPC codes.

We also discussed the relative harmfulness of trapping sets of variable-regular LDPC codes, based on the position of their induced subgraph in the Tanner graph of the code. Using two case studies, we showed that for ETSs in the same class, and even with the same configuration, harmfulness can be different. We related the relative harmfulness to the position of the subgraph in the Tanner graph of the code. Our findings can help to relax the constraint of removing a complete class or topological configuration of trapping sets in code design, like what was done in [55].

Furthermore, we studied the dominant trapping sets of MS decoding algorithm and two of its variants (offset-MS and SR-MS). We showed that in regular codes, the relative harmfulness of trapping sets is similar for MS and its two variants. For irregular codes, we showed that the harmfulness of trapping sets changes by depending on the decoding algorithm. Thus for irregular codes, a parallel decoder consisting of MS, offset-MS and SR-MS decoding algorithms, improves the error floor by one order

of magnitude. We also presented a simple and practical scheme to lower the error floor of irregular LDPC codes.

7.2 Future work

There are some interesting open problems for future research on high throughput and low error floor LDPC codes, which we briefly describe below:

- In our proposed architecture, we presented a constraint in code design to implement PPP-SCN (*SCN compatible*). As we showed in this thesis, this constraint does not affect the performance of the LDPC code. Removing this constraint can generalize this architecture to implement any protograph-based LDPC code. The design of serial-input check nodes and switch networks can be further enhanced to remove this constraint.
- We studied the harmfulness of dominant trapping sets of LDPC codes. We showed that not all trapping sets with the same size and configuration have the same harmfulness. The relationship between the harmfulness of trapping sets and the position of their subgraph in the Tanner graph, has to be further analyzed. This knowledge can be used in the design of LDPC codes with low error floor.
- The research on harmfulness of trapping sets is mostly performed on regular codes. As we showed in this thesis, for irregular codes, more factors such as decoding algorithm are involved in the harmfulness of trapping sets. Extending the research on harmfulness of trapping sets to irregular codes, is a very important topic for future research.

List of References

- [1] *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1*, Std., 2006.
- [2] [Online]. Available: <http://www.inference.phy.cam.ac.uk/mackay/codes/>
- [3] R. Asvadi, A. H. Banihashemi, and M. Ahmadian-Attari, “Lowering the error floor of LDPC codes using cyclic liftings,” *IEEE Trans. on Info. Theory*, vol. 57, no. 4, pp. 2213 – 2224, Apr. 2011.
- [4] D. MacKay and R. Neal, “Near shannon limit performance of low density parity check codes,” *Electronics Letters*, vol. 32, no. 18, p. 1645, Aug. 1996.
- [5] R. Gallager, “Low-density parity-check codes,” *IEEE Trans. on Info. Theory*, vol. 8, pp. 21 – 28, 1962.
- [6] *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*, ETSI Std. EN 302 307, Aug. 2009.
- [7] *ITU-TG.hn Standard for Wired Home Networking*, Std. [Online]. Available: <http://www.homegridforum.org/home/>
- [8] *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, Sep. 2006.

- [9] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Info. Theory*, vol. 27, no. 5, pp. 533 – 547, Sep. 1981.
- [10] T. Richardson, "Error floors of LDPC codes," in *Proc. 41th Annual Allerton Conf. on Commun., Control, and Computing*, Oct. 2003.
- [11] 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. on Commun.*, vol. 53, no. 4, pp. 549 – 554, Apr. 2005.
- [12] C. A. Cole, S. G. Wilson, E. K. Hall, and T. R. Giallorenzi. A general method for finding low error rates of LDPC codes. [Online]. Available: <http://arxiv.org/abs/cs/0605051>
- [13] E. Cavus, C. Haymes, and B. Daneshrad, "An IS simulation technique for very low BER performance evaluation of LDPC codes," in *Proc. 2006 IEEE Inter. Conf. Commun.*, vol. 3, 2006, pp. 1095–1100.
- [14] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright, "Predicting error floors of structured LDPC codes: deterministic bounds and estimates," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 908–917, 2009.
- [15] H. Xiao, A. H. Banihashemi, and M. Karimi, "Error rate estimation of low-density parity-check codes decoded by quantized soft-decision iterative algorithms," *IEEE Trans. on Commun.*, vol. 61, no. 2, pp. 474–484, 2013.
- [16] M. Karimi and A. H. Banihashemi. On characterization of elementary trapping sets of variable-regular LDPC codes. [Online]. Available: <http://arxiv.org/abs/1308.1259>
- [17] T. Zhang and K. Parhi, "A 54 mbps (3,6)-regular FPGA LDPC decoder," in *Proc. IEEE Workshop on Signal Process. Syst.*, Oct. 2002, pp. 127 – 132.
- [18] R. Zarubica, S. Wilson, and E. Hall, "Multi-gbps FPGA-based low density parity check (LDPC) decoder design," in *Proc. IEEE Global Telecommun. Conf.*, Nov. 2007, pp. 548 – 552.
- [19] S. Tolouei and A. H. Banihashemi, "FPGA implementation of variants of min-sum algorithm," in *Proc. 24th Biennial Symp. on Commun.*, June 2008, pp. 80 – 83.

- [20] X. Chen, Q. Huang, S. Lin, and V. Akella, "FPGA-based low-complexity high-throughput tri-mode decoder for quasi-cyclic LDPC codes," in *Proc. 47th Annual Allerton Conf. on Commun., Control, and Computing*, Oct 2009, pp. 600–606.
- [21] V. Chandrasetty and S. Aziz, "FPGA implementation of high performance LDPC decoder using modified 2-bit min-sum algorithm," in *Proc. 2010 Second Int. Conf. on Computer Research and Development*, May. 2010, pp. 881–885.
- [22] S. Kim, G. E. Sobelman, and H. Lee, "A reduced-complexity architecture for LDPC layered decoding schemes," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, vol. PP, no. 99, pp. 1–5, Jun 2011.
- [23] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory system optimization for FPGA-based implementation of quasi-cyclic LDPC codes decoders," *IEEE Trans. Circuits and Syst. I: Reg. Papers*, vol. 58, no. 1, pp. 98–111, Jan. 2010.
- [24] J. Sha, J. Lin, Z. Wang, L. Li, and M. Gao, "Decoder design for RS-Based LDPC codes," *IEEE Trans. on Circuits and Syst. II: Express Briefs*, vol. 56, pp. 724–728, Sep 2009.
- [25] Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," *IEEE J. of Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, Apr 2010.
- [26] T. Mohsenin, D. Truong, and B. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders," *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 57, no. 5, pp. 1048–1061, Apr 2010.
- [27] Y.-L. Ueng, C.-J. Yang, K.-C. Wang, and C.-J. Chen, "A multimode shuffled iterative decoder architecture for high-rate RS-LDPC codes," *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 57, no. 10, pp. 2790–2803, Oct 2010.
- [28] B. Xiang, D. Bao, S. Huang, and X. Zeng, "An 847-955 mb/s 342-397 mw dual-path fully-overlapped QC-LDPC decoder for WiMAX system in 0.13 um CMOS," *IEEE J. of Solid-State Circuits*, vol. 46, no. 6, pp. 1416–1432, Jun 2011.
- [29] A. Naderi, S. Mannor, M. Sawan, and W. Gross, "Delayed stochastic decoding of LDPC codes," *IEEE Trans. on Signal Process.*, vol. 59, no. 11, pp. 5617–5626, Nov 2011.

- [30] P. Marshall, V. Gaudet, and D. Elliott, “Deeply pipelined digit-serial LDPC decoding,” *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 59, no. 12, pp. 2934–2944, Dec 2012.
- [31] P. Schläfer, C. Weis, N. Wehn, and M. Alles, “Design space of flexible multigigabit LDPC decoders,” *VLSI Des.*, vol. 2012, pp. 1–10, Jan. 2012.
- [32] P. Schläfer, N. Wehn, M. Alles, and T. Lehnigk-Emden, “A new dimension of parallelism in ultra high throughput LDPC decoding,” in *Proc. IEEE Workshop on Signal Process. Syst.*, Oct 2013, pp. 153–158.
- [33] J. Thorpe, “Low-density parity-check (LDPC) codes constructed from protograph,” *IPN Progress Report 42-154*, Aug. 2003.
- [34] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, “A class of low-density parity-check codes constructed based on reed-solomon codes with two information symbols,” *Communications Letters*, vol. 7, no. 7, pp. 317–319, Jul 2003.
- [35] D. Divsalar, S. Dolinar, C. Jones, and K. Andrews, “Capacity-approaching protograph codes,” *IEEE J. on Selected Areas in Comm.*, vol. 27, no. 6, pp. 876 – 888, Aug. 2009.
- [36] R. Asvadi, A. H. Banihashemi, and M. Ahmadian-Attari, “Design of finite-length irregular protograph codes with low error floors over the binary-input AWGN channel using cyclic liftings,” *IEEE Trans. on Commun.*, vol. 60, no. 4, pp. 902–907, Apr. 2012.
- [37] Y. Dai and Z. Yan, “Optimal overlapped message passing decoding for quasi-cyclic low-density parity-check codes,” in *Proc. IEEE Global Telecommun. Conf.*, Dec. 2005, pp. 5 pp. – 2399.
- [38] Y. Chen and K. Parhi, “Overlapped message passing for quasi-cyclic low-density parity check codes,” *IEEE Trans. on Circuits and Syst. I: Reg. Papers*, vol. 51, no. 6, pp. 1106 – 1113, June. 2004.
- [39] A. Abbasfar, D. Divsalar, and K. Yao, “Accumulate-repeat-accumulate codes,” *IEEE Trans. on Commun.*, vol. 55, no. 4, pp. 692 –702, Apr. 2007.
- [40] Y.-L. Wang, Y.-L. Ueng, C.-L. Peng, and C.-J. Yang, “Processing-task arrangement for a low-complexity full-mode WiMAX LDPC codec,” *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 58, no. 2, pp. 415–428, 2011.

- [41] Y. Wang, S. Draper, and J. Yedidia, "Hierarchical and high-girth QC LDPC codes," *IEEE Trans. on Info. Theory*, 2013.
- [42] A. Blanksby and C. Howland, "A 690-mw 1-gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. of Solid-State Circuits*, vol. 37, no. 3, pp. 404 – 412, Mar. 2002.
- [43] M. Mansour, "A turbo-decoding message-passing algorithm for sparse parity-check matrix codes," *IEEE Trans. on Signal Process.*, vol. 54, no. 11, pp. 4376 – 4392, Nov. 2006.
- [44] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Trans. on Circuits and Syst. II: Express Briefs*, vol. 54, no. 6, pp. 542 – 546, June. 2007.
- [45] D. Oh and K. Parhi, "Low-complexity switch network for reconfigurable LDPC decoders," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 1, pp. 85 – 94, Jan. 2010.
- [46] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Trans. on Commun.*, vol. 53, no. 2, pp. 209–213, Feb 2005.
- [47] Y.-L. Ueng, B.-J. Yang, C.-J. Yang, H.-C. Lee, and J.-D. Yang, "An efficient multi-standard LDPC decoder design using hardware-friendly shuffled decoding," *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 60, no. 3, pp. 743–756, 2013.
- [48] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Trans. on Commun.*, vol. 57, no. 11, pp. 3258 – 3268, Nov. 2009.
- [49] X. Zhang and P. Siegel, "Quantized min-sum decoders with low error floor for LDPC codes," in *Proc. IEEE Inter. Symp. on Info. Theory (ISIT)*, 2012, pp. 2871–2875.
- [50] X. Zhang and P. H. Siegel. Quantized iterative message passing decoders with low error floor for LDPC codes. [Online]. Available: <http://arxiv.org/abs/1212.6465>
- [51] Y. Han and W. Ryan, "Low-floor decoders for LDPC codes," *IEEE Trans. on Commun.*, vol. 57, no. 6, pp. 1663 – 1673, Jun 2009.

- [52] G. B. Kyung and C.-C. Wang, "Finding the exhaustive list of small fully absorbing sets and designing the corresponding low error-floor decoder," *IEEE Trans. on Commun.*, vol. 60, no. 6, pp. 1487–1498, 2012.
- [53] M. Karimi and A. H. Banihashemi, "Message-passing algorithms for counting short cycles in a graph," *IEEE Trans. on Commun.*, vol. 61, no. 2, pp. 485–495, 2013.
- [54] B. Vasic, S. Chilappagari, D. Nguyen, and S. Planjery, "Trapping set ontology," in *Proc. 47th Annual Allerton Conf. on Commun., Control, and Computing*, 2009, pp. 1–7.
- [55] D. Nguyen, S. Chilappagari, M. Marcellin, and B. Vasic, "On the construction of structured LDPC codes free of small trapping sets," *IEEE Trans. on Info. Theory*, vol. 58, no. 4, pp. 2280–2302, 2012.
- [56] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Trans. on Info. Theory*, vol. 56, no. 1, pp. 181–201, 2010.
- [57] C. Schlegel and S. Zhang, "On the dynamics of the error floor behavior in (regular) LDPC codes," *IEEE Trans. on Info. Theory*, vol. 56, no. 7, pp. 3248–3264, 2010.
- [58] X. Zhang and P. Siegel, "Quantized iterative message passing decoders with low error floor for LDPC codes," *IEEE Trans. on Commun.*, vol. 62, no. 1, pp. 1–14, 2014.
- [59] S. Tolouei and A. H. Banihashemi, "Lowering the error floor of LDPC codes using multi-step quantization," *Communications Letters*, vol. 18, no. 1, pp. 86–89, 2014.
- [60] K. Andrews, S. Dolinar, D. Divsalar, and J. Thorpe, "Design of low-density parity-check (LDPC) codes for deep-space applications," *IPN Progress Report 42-159*, Nov. 2004.
- [61] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. on Commun.*, vol. 47, no. 5, pp. 673 – 680, May. 1999.

- [62] J. Chen and P. Fossorier, "Density evolution for bp-based decoding algorithms of LDPC codes and their quantized versions," in *Proc. IEEE Global Telecommun. Conf.*, Nov. 2002, pp. 1378 – 1382.
- [63] S. Hemati and A. H. Banihashemi, "Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes," *IEEE Trans. on Commun.*, vol. 54, no. 1, pp. 61–70, 2006.
- [64] H. Xiao, S. Tolouei, and A. H. Banihashemi, "Successive relaxation for decoding of LDPC codes," in *Proc. 24th Biennial Symp. on Commun.*, June 2008, pp. 107–110.
- [65] *Interfaces for the Optical Transport Network (OTN)*, International Telecommunication Union Std., 2009.
- [66] Y. Miyata, W. Matsumoto, H. Yoshida, and T. Mizuochi, "Efficient FEC for optical communications using concatenated codes to combat error-floor," in *Proc. 2008 Optical Fiber Commun. Conf. and Exposition (OFC/NFOEC) and the National Fiber Optic Eng. Conf.*, Feb. 2008, pp. 1 – 3.
- [67] N. Kamiya and S. Shioiri, "Concatenated QC-LDPC and SPC codes for 100 gbps ultra long-haul optical transmission systems," in *Proc. 2010 Optical Fiber Commun. Conf. and Exposition (OFC/NFOEC) and the National Fiber Optic Eng. Conf.*, Mar 2010, pp. 1 – 3.
- [68] D. Chang, F. Yu, Z. Xiao, Y. Li, N. Stojanovic, C. Xie, X. Shi, X. Xu, and Q. Xiong, "FPGA verification of a single QC-LDPC code for 100 gb/s optical systems without error floor down to BER of 10^{-15} ," in *Proc. 2011 Optical Fiber Commun. Conf. and Exposition (OFC/NFOEC) and the National Fiber Optic Engineers Conf.*, Mar 2011, pp. 1 – 3.
- [69] "Stratix IV FPGA," Altera. [Online]. Available: <http://www.altera.com/devices/fpga/stratix-fpgas/stratix-iv/stxiv-index.jsp>
- [70] "Quartus II software," Altera. [Online]. Available: <http://www.altera.com/products/software/quartus-ii/about/qts-performance-productivity.html>
- [71] J. Kim and W. Sung, "Rate-0.96 ldpc decoding VLSI for soft-decision error correction of NAND flash memory," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 5, pp. 1004–1015, May 2014.

- [72] S. Borkar, “Design challenges of technology scaling,” *Micro, IEEE*, vol. 19, no. 4, pp. 23–29, Jul 1999.
- [73] Y. S. Park, D. Blaauw, D. Sylvester, and Z. Zhang, “Low-power high-throughput LDPC decoder using non-refresh embedded DRAM,” *IEEE J. of Solid-State Circuits*, vol. 49, no. 3, pp. 783–794, March 2014.
- [74] X. Chen, J. Kang, S. Lin, and V. Akella, “Hardware implementation of a backtracking-based reconfigurable decoder for lowering the error floor of quasi-cyclic LDPC codes,” *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 58, no. 12, pp. 2931–2943, Dec 2011.
- [75] M. Karimi and A. H. Banihashemi, “Efficient algorithm for finding dominant trapping sets of LDPC codes,” *IEEE Trans. on Info. Theory*, vol. 58, no. 11, pp. 6942–6958, Nov 2012.
- [76] [Online]. Available: <http://cs.anu.edu.au/~bdm/nauty/>