Design of LDPC Decoders for Improved Low Error Rate Performance

by

Zhengya Zhang

B.A.Sc. (University of Waterloo) 2003
M.S. (University of California, Berkeley) 2005

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Fall 2009

Design of LDPC Decoders for Improved Low Error Rate Performance

Abstract

Design of LDPC Decoders for Improved Low Error Rate Performance

by

Zhengya Zhang

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Borivoje Nikolić, Chair

In the past several decades, tremendous progress has been made in both communication theory and practical implementation of communication systems. However, practice often lags the most recent developments in theory possibly for two reasons: the cost of implementation is high, and the practical implementation incurs a non-negligible loss compared to the theoretical bounds. The two objectives of what is theoretically possible and what is achievable by implementation can be better aligned, so theory can be made more relevant and practice can be more powerful and efficient.

A novel emulation-simulation framework is presented on studying the low error rate performance of capacity-approaching low-density parity-check (LDPC) codes decoded using a message-passing algorithm. High-throughput hardware emulation uncovers combinatorial error structures that underpin the error floors. The captured errors are analyzed in functionally equivalent software simulation to illuminate the effects of wordlength, quantization, and algorithm design, thereby extending the theoretical discovery for practical usage.

The emulation-simulation framework further allows the algorithm and implementation to be iteratively refined to improve the error-floor performance of message-passing decoders. A dual quantization scheme is first introduced to reduce the degradation of soft decoding. Then, a reweighted message-passing algorithm is proposed to eliminate local minima caused by the remaining dominant errors. This improved algorithm is realized in a simple post-processor that compensates the message-passing decoding algorithm to achieve the near maximum-likelihood decoding performance. Results are demonstrated by the design of a 5.35 mm^2, 65nm CMOS chip that realizes a grouped parallel architecture to optimize the area and power efficiencies by aggressively scaling down the interconnection overhead. The 47.7 Gb/s LDPC decoder operates without error floor down to the bit error rate level of 10^{-14}.

The iterative emulation-simulation framework and systematic architectural exploration can be extended to other complex systems, thereby enabling the joint optimizations of algorithm, architecture, and implementation.

Professor Borivoje Nikolić
Dissertation Committee Chair

To my parents

Contents

List of Figures

List of Tables

Acknowledgments

I consider myself very fortunate to be able to work with a group of exceptionally talented individuals at Berkeley. First and foremost, my sincere gratitude goes to my advisor, Professor Bora Nikolić, for his support and guidance. I benefited tremendously from his vision and still feel constantly motivated by his own dedication to research. I would like to thank members of my project group, Professor Venkat Anantharam for always upholding high standards and elevating the research to the next level, Professor Martin Wainwright for the most insightful discussions and constant encouragement, Lara Dolecek and Pamela Lee for the hard work and open mind that contributed to the very successful collaboration. I would also like to thank Professor Daniel Tataru for evaluating my research proposal and reviewing this dissertation.

My research was supported in part by Marvell Semiconductor. I have had many constructive meetings with Dr. Zining Wu, Dr. Engling Yeo, and other members of the read channel group. Their technical advice helped guiding this research from the very beginning. This research was also supported by NSF CCF grant no. 0635372, NSF CNS RI grant no. 0403427, as well as the generous donations from Intel Corporation and Infineon Technologies through the University of California MICRO program. The chip fabrication donation was provided by ST Microelectronics. Dr. Pascal Urard and his team at ST Microelectronics offered valuable feedback in reviewing my chip design.

I was based in BWRC for the most part of my Ph.D. career. I consider it a privilege to be associated with the center. The resource and level of collaboration is unmatched. Senior students and staff laid the foundation that made my research possible. I learned how to use the BEE emulation platform from its creators, Chen Chang and Pierre-Yves Droz. I appreciate their effort in listening to me and providing the best solutions. Thanks to Brian Richards for helping with my chip design. I was shielded from all the intricacies of design flows due to his ground work. I was also lucky to have a period of overlap with Dejan Markovic, who shared his wealth of design experience. Many thanks go to Henry Chen for always being patient with my questions and assisting in system emulation and board design. Special thanks to Gary Kelson, Tom Boot, Brenda Farrell, Sue Mellers, Kevin Zimmerman, Brad Krebs, and other staff members for making BWRC such a pleasant place to work.

I would like to express my appreciation to Ruth Gjerde and Mary Byrnes in the graduate office, who helped me navigate through all the complex paperwork and procedures. Thanks to Jennifer Stone and Jessica Budgin for taking care of all the issues with funding of my research.

The friendship and camaraderie with my co-workers in the DCDG group made a very enjoyable six years, during which I crossed paths with Sokratis Vamvakos, Dejan Markovic, Radu Zlatanovici, Bill Tsang, Farhana Sheikh, Liang-Teck Pang, Zheng Guo, Renaldi Winoto, Seng Oon Toh, Ji-Hoon Park, Dusan Stepanovic, Vinayak Nagpal, David Fang, Melinda Ler, Kenneth Duong, Adam Abed, Lauren Jones, Jason Tsai, Milos Jorgovanovic, and Matthew Weiner. Every once a while some of us would graduate and new faces join, but we always kept DCDG a loving family. I still have fond memories of Zheng and I staying up until early mornings finishing project reports, Renaldi and I keeping each other company in the lonely hours before the chip tapeout, as well as numerous retreats, barbecues, group lunches, and graduation dinners. Thanks to Bill, Farhana, and Liang-Teck for the encouragement in the difficult times, Zheng for sharing many thoughts on school and life, Renaldi for being the best critic of my work. My appreciation also goes beyond the group boundary – thanks to Wei-Hung Chen, Stanley Chen, Jing Yang, Mubaraq Mishra, and Simone Gambini for always being supportive.

Finally, my gratitude goes to my parents, Shude and Fenglian, who have been a perpetual source of comfort and encouragement. They taught me the life values and work ethics, which I only learn to appreciate gradually over time. They always valued my honest effort, no matter how minuscule. They provided the best cushion to allow me to recover from each setback and become more determined. I dedicate this work to them for their love and support that made it possible.

Chapter 1

Introduction

Low-density parity-check (LDPC) codes have been demonstrated to perform very close to the Shannon limit when decoded iteratively using a message-passing algorithm [29, 45,46,57]. A wide array of the latest communication and storage systems have chosen LDPC codes as forward error correction in applications including digital video broadcasting (DVB-S2) [25, 53], 10 Gigabit Ethernet (10GBASE-T) [36], broadband wireless access (WiMax) [37], wireless local area network (WiFi) [38], deep-space communications [3], and magnetic storage in hard disk drives [39]. The adoption of the capacity-approaching LDPC codes is, at least in theory, the key to achieving a lower transmission power for a more reliable communication.

There is currently a challenge in implementing high-throughput LDPC decoders with a low area and power on a silicon chip for practical applications [4, 65, 66], thus an LDPC decoder is often considered an additional premium and implemented in systems as an option [37, 38]. LDPC codes are not guaranteed to perform well either. Sometimes the excellent error-correction performance of LDPC codes is only observed up until a moderate bit error rate (BER); at a lower BER, the error curve often changes its slope, manifesting a so-called error floor [54].

With the latest communication and storage systems demanding data rates up to Gb/s, relatively high error floors degrade the quality of service – for example, frequent loss of frames in high-definition video transmission, regular disk failures in magnetic storage, etc. To prevent such degradations, transmission power is raised or a more complex scheme, such as an additional level of error-correction coding [25], is created. These approaches increase the power consumption and complicate the system integration.

This work presents the investigation of error floors of LDPC codes. Exploring these error floors for realistic LDPC codes by software simulation on a general-purpose computer is not practical. Even an optimized decoder implemented in C and executed on a high-end microprocessor provides a peak throughput of only up to the order of 1 Mb/s. Consequently, months of simulation time would be required to collect at least tens of frame errors for a confident estimate of the BER at 10^{-10}. However, the use of field-programmable gate array (FPGA) platforms allows for substantial acceleration in the emulation of LDPC codes [54, 77].

1.1 Scope of Work

This work sheds light on the effects of practical implementations on the error floor levels of some LDPC decoders [77, 79, 80]. Specifically, the understanding on error floors is advanced on the following fronts: 1) the use of the absorbing set objects to quantify how the error counts are affected by wordlength, numerical quantization, and decoding algorithm choices; 2) differentiation of error mechanisms between oscillations and convergence to absorbing sets; 3) differentiation of weak from strong absorbing sets – weak absorbing sets can be eliminated by an optimized decoder implementation, while strong absorbing sets dominate the error floor of even an optimal decoder implementation; 4) proposal of dual quantization and demonstration of approximate algorithms in improving the error floor performance by alleviating weak absorbing sets. High-performance hardware emulation has been applied throughout the investigation to uncover large datasets of error signatures and to verify conjectures.

This work contributes to the solution of the error floors by proposing a post-processing algorithm that utilizes the graph-theoretic structure of absorbing sets [78]. The post-processor carefully adjusts the appropriate messages in the iterative decoding once the decoder enters and remains in the absorbing set of interest. The proposed post-processing approach is based on a message-passing algorithm with selectively-biased messages. As a result, it can be seamlessly integrated with the message-passing decoder. Results show significant performance improvement at low error rates after post-processing even with a short wordlength.

This work advances the state-of-the-art application-specific integrated circuit (ASIC) and FPGA architectures of LDPC decoders [81]. A grouping strategy is applied in localizing irregular wires and regularizing global wires. The optimal parallel architecture depends on the balance between global and local wires, measured in an area expansion metric and a wire length metric respectively. The post-processing algorithm further reduces wiring by wordlength reduction. The post-processor is implemented as a small add-on to each local processing element without adding external wiring, thus the area penalty is kept minimal. Reduced wiring enables a highly parallel decoder design that achieves a very high throughput. Frequency and voltage scaling can be applied to improve power efficiency if a lower throughput is desired.

1.2 Related Work

Methods have been developed through past work on improving the performance of LDPC codes by eliminating short cycles, and by increasing girth and minimum distance of the codes [34, 51, 63, 72]. These methods are effective in lowering the error floors, but the resulting code structures are often irregular, leading to complex decoder implementations. The alternative is to improve the decoding algorithms without modifying the code structure as in [2, 9, 10, 28, 32, 82], where the improved algorithms were evaluated by the analytical technique known as density evolution [57]. Density evolution assumes independent messages being passed during iterative decoding. Though some agreement has been shown by software simulation down to moderate BER levels, the independence assumption is a cause of concern at lower BER levels as the previous analysis disregards the correlation of messages due to cycles. So to reach lower BER levels, FPGA-based emulations were performed in [61, 71] to reveal the error floors. These FPGA platforms conveniently capture the performance of codes, but they do not provide sufficient evidence for the study of error floors.

This work explores practical LDPC decoder design issues using an emulation-simulation approach. This investigation is motivated by Richardson's work on error floors

[54], where he identified and semi-empirically defined a class of trapping sets using hardware emulation. Starting from the same point, some of these earlier findings are confirmed, and moreover, a combinatorial characterization is provided of what is referred to as absorbing sets in terms of the graph structure of the code. For many LDPC codes, the associated factor graphs contain absorbing sets of lower weight than the minimum codeword weight. As a result, the performance of the code in the low error rate region is determined by the distribution and structure of the low-weight absorbing sets, rather than the minimum distance of the code [47,54]. This work is based on the characterization of absorbing sets, which are classified by their structures. Through the analysis of absorbing set profiles, intuitions are provided on why certain quantization choices and decoding algorithms perform better in the error floor region, thereby extending the definition of absorbing sets for practical usage.

To overcome the error floors, past work presented modified message-passing decoding algorithms by appropriately scaling, averaging messages, or reordering message-passing schedules [6,14,41,59]. These modified algorithms were designed without specific considerations of the error structures, thus their effectiveness is limited. In comparison, this work concentrates on the combinatorial structure of the absorbing set in formulating the solution that also minimizes the side effects. The proposed post-processing strategy can be compared to the work by Han and Ryan [31], but note that their bi-mode syndrome-erasure decoding algorithm falls short of resolving the absorbing sets in some codes, where erasure decoding runs into stopping sets (which are defined in [19]) with high probability. The post-processing strategy does not suffer from similar problems because the soft reliability values are retained.

Building high-throughput LDPC decoders has always been challenging. Ever since the very first silicon implementation of the LDPC decoder, high decoding throughput has become the synonym for large area and high power consumption [4]. The challenge lies in the wiring overhead associated with highly parallel decoder designs, resulting in low area utilization due to routing irregularity and congestion. Architectural transformations have been applied to either partition the parallel architecture as in [44], or to parallelize serial architectures as in [43, 60, 65, 66, 73] by exploiting the code structure. The design in [50] adopted a layered schedule that accelerates convergence for a higher throughput at the cost of increasing computational intensity. A novel arithmetic transformation is applied in [17] to enable bit-serialized operations that reduce wiring overhead by a factor of the wordlength (referring to the number of bits representing a message). However, the space for architectural optimization is limited, as a minimum wordlength needs to be kept for an acceptable decoding performance. The post-processing strategy presented by this work provides an excellent decoding performance at a very short wordlength of 4 bits. The wordlength reduction permits a more compact physical implementation.

1.3 Organization

In Chapter 2, the background is provided on the decoding algorithm, the quantization procedure, and decoder architecture of a family of high-performance regular LDPC codes. The architectural choices are presented with the $(2048, 1723)$ Reed-Solomon based LDPC (RS-LDPC) [20] as an example. The LDPC decoder emulator forms the basis of the hardware emulation platform. Error traces are collected from hardware emulations.

In Chapter 3, the error traces are analyzed against the structure of the code to reveal the nature of error floors. In a decoder implementation with a sufficient wordlength, the hard decisions do not change after a number of decoding iterations while some parity checks remain unsatisfied. Such non-codeword errors are attributed to a class of combinatorial structures termed absorbing sets. A series of experiments in Section 3.3 on the $(2048, 1723)$ RS-LDPC code illuminate the fixed-point quantization effects, and then in Section 3.4 the experiments on the $(2209, 1978)$ array-based LDPC code [26] help uncover a collection of different absorbing sets in the error floor region. Methods are developed to improve upon standard quantization approaches and alternative decoder implementations are experimented with in reducing the effects of weak absorbing sets and lowering the error floor.

In Chapter 4, the absorbing-error-inducing channel likelihoods are characterized to demonstrate that most of the absorbing errors occur due to specific patterns in the codeword being subject to noise moderately out in the tail rather than because of noise values in the extreme tails. This intuition motivates the formulation of a message-biasing approach to recover the absorbing errors in a two-step decoder. Tradeoffs are explored in the bias selection and an adaption is proposed to dynamically adjust the bias for the best performance.

In Chapter 5, a high-throughput LDPC decoder is designed following a series of optimization steps. The architecture of the chip is determined based on a set of experiments to explore how each design parameter (architectural grouping, density, pipeline design) affects implementation results (wiring overhead, clock frequency, decoding throughput, area, power). The design parameters are orthogonalized such that each can be determined almost independently. Important design tradeoffs are investigated in more depth: the degree of parallelism versus wiring overhead, the area efficiency versus clock frequency, and the pipeline efficiency versus effective throughput. The architecture choice that optimizes these tradeoffs is adopted in the final decoder design. The decoder chip was fabricated by ST Microelectronics. The chip is measured to be fully functional. The performance and power measurements are presented in the end.

Chapter 2

LDPC Decoder Emulation

Gallager invented low-density parity-check (LDPC) code in his doctoral dissertation in 1960 [29]. It received little attention until the 1990s through the rediscovery of LDPC codes by MacKay [45, 46]. Since then significant advances have been made on the understanding and design of LDPC codes as well as the iterative message-passing decoding algorithms. In particular, irregular LDPC codes can be designed to achieve a performance at rates extremely close to the Shannon limit [56], for example, one LDPC code construction has been demonstrated to perform within 0.0045 dB of the Shannon limit [12], representing a giant leap towards reaching the ultimate channel capacity [16, 55].

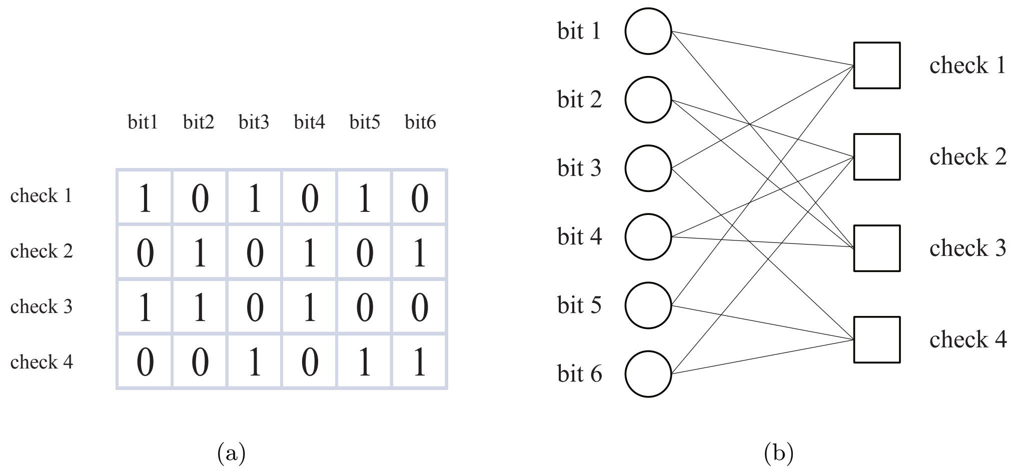
Practical implementations of LDPC decoders immediately followed the theoretical research. The first LDPC decoder in silicon was demonstrated in [4], featuring an impressive 1 Gb/s decoding throughput. This implementation also revealed routing congestion rather than gate count as the bottleneck in high-throughput LDPC decoder designs. Subsequent LDPC decoder implementations reduce the level of parallelism to improve routing [73].

The long block length, largely irregular LDPC codes have gradually lost their appeal due to the difficulty in realizing efficient decoder implementations for a reasonable throughput. The performance-complexity tradeoff propelled the development of structured codes [40,62], which can be efficiently encoded and decoded with reasonably good to very good performance. The majority of the recent communication standards have adopted codes with such structures [36–38].

2.1 LDPC Code and Decoding Algorithm

A low-density parity-check code is a linear block code, defined by a sparse $M \times N$ parity check matrix $\mathbf{H}$ where N represents the number of bits in the code block (block length) and M represents the number of parity checks. In the small example shown in Fig. 2.1a, the first row of the parity-check matrix specifies that bits 1, 3, and 5 have to satisfy even parity constraint, the second row specifies that bits 2, 4, and 6 have to satisfy even parity constraint, and so on. The $\mathbf{H}$ matrix of an LDPC code can be illustrated graphically using a factor graph as in Fig. 2.1b, where each bit is represented by a variable node (shown as a circle) and each check is represented by a factor (check) node (shown as a square). An edge exists between the variable node i and the check node j if and only if $\mathbf{H}(j, i) = 1$.

Consider a simplified communication system block diagram shown in Fig. 2.2, where a binary phase-shift keying (BPSK) modulation and an additive white Gaussian noise (AWGN) channel are assumed. The binary channel bits $\{0, 1\}$ are represented using $\{1, -1\}$ for transmission over the channel. On the receiver side, the analog-to-digital converter samples and digitizes the channel output. The resulting soft information represents each



	bit1	bit2	bit3	bit4	bit5	bit6
check 1	1	0	1	0	1	0
check 2	0	1	0	1	0	1
check 3	1	1	0	1	0	0
check 4	0	0	1	0	1	1

Figure 2.1: Representation of an LDPC code using (a) a parity-check matrix (**H** matrix), and (b) a factor graph.

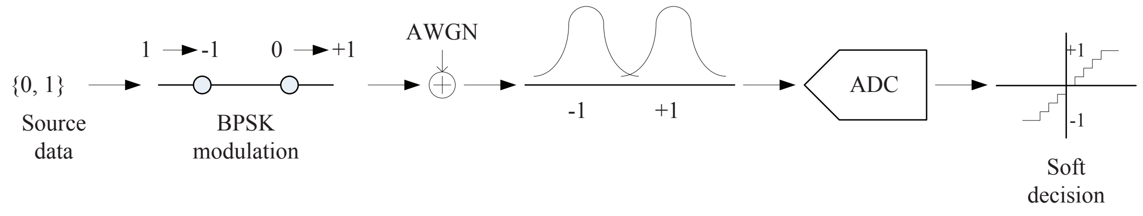


Figure 2.2: Data flow through a simplified communication system (RF front ends are omitted for simplicity).

received bit with a real (quantized) number. The sign part of the soft information represents the hard decision, either 0 or 1; and the magnitude part of the soft information represents the reliability of the hard decision. A decoding algorithm that utilizes both the sign and the reliability information is called soft decoding. Soft decoding outperforms hard-decision decoding, which relies only on the sign.

Low-density parity-check codes are usually iteratively decoded using the belief propagation algorithm, also known as the message-passing algorithm [29]. The highly efficient message-passing algorithm is an important factor that has contributed to the success of LDPC codes in both theoretical studies and practical applications. Suitably-designed LDPC codes have been shown to perform very close to the Shannon limit when decoded using the iterative message-passing algorithm. This algorithm also features an intrinsic parallel scheduling, which makes it very attractive for high-throughput hardware implementations.

The message-passing algorithm operates on a factor graph, where soft messages are exchanged between variable nodes and check nodes. The variable nodes are initialized based on the channel outputs. In the first step of decoding, check nodes receive the initial beliefs from the neighboring variable nodes and in return, send the extrinsic information (information from the neighbors) to each of the variable nodes. In every iteration, each variable node receives new extrinsic information from more distant neighbors and refines its initial decision. The message-passing algorithm is exact when operating on a factor graph that is cycle-free, and in practice, free of short cycles is an important criterion in the construction of good codes. The iterative message-passing algorithm can usually reach convergence within a small number of iterations when operating on graphs containing no short cycles.

The message-passing algorithm can be formulated as follows: in the first step, variable nodes x_i are initialized with the prior log-likelihood ratios (LLR) defined in (2.1) using the channel outputs y_i. This formulation assumes the information bits take on 0 and 1 with equal probability.

$$L^{pr}(x_i) = \log \frac{\Pr(x_i = 0 \mid y_i)}{\Pr(x_i = 1 \mid y_i)} = \frac{2}{\sigma^2} y_i, \tag{2.1}$$

where σ^2 represents the channel noise variance.

The variable nodes send messages to the check nodes along the edges defined by the factor graph. The LLRs are recomputed based on the parity constraints at each check node and returned to the neighboring variable nodes. Each variable node then updates its decision based on the channel output and the extrinsic information received from all the neighboring check nodes. The marginalized posterior information is used as the variable-to-check message in the next iteration.

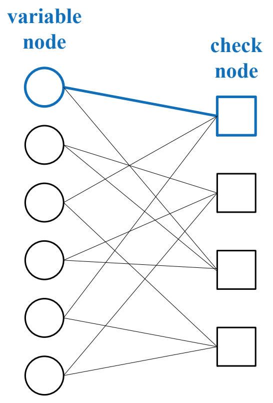
2.1.1 Sum-product Algorithm (SPA)

The sum-product algorithm is a conventional realization of the message-passing algorithm. A simplified illustration of the iterative decoding procedure is shown in Fig. 2.3b. The illustration is for one slice of the factor graph showing a round trip from a variable node to a check node back to the same variable node as highlighted in the Fig. 2.3a. Variable-to-check and check-to-variable messages are computed using equations (2.2), (2.3), and (2.4).

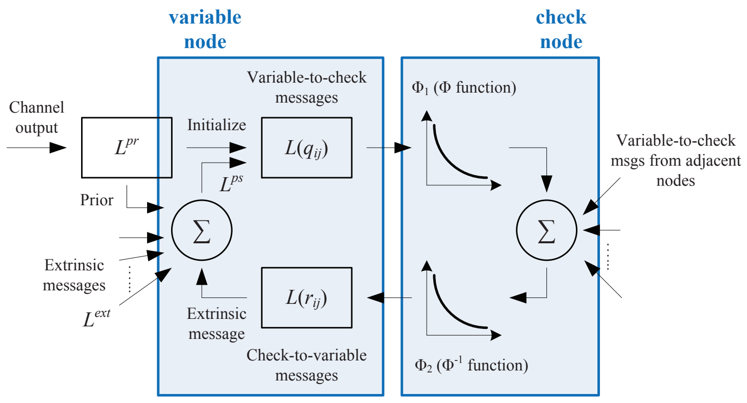
$$L(q_{ij}) = \sum_{j' \in Col[i] \backslash j} L(r_{ij'}) + L^{pr}(x_i), \tag{2.2}$$

$$L(r_{ij}) = \Phi^{-1}\left(\sum_{i' \in Row[j] \backslash i} \Phi\left(|L(q_{i'j})|\right)\right)\left(\prod_{i' \in Row[j] \backslash i} \operatorname{sgn}\left(L(q_{i'j})\right)\right), \tag{2.3}$$

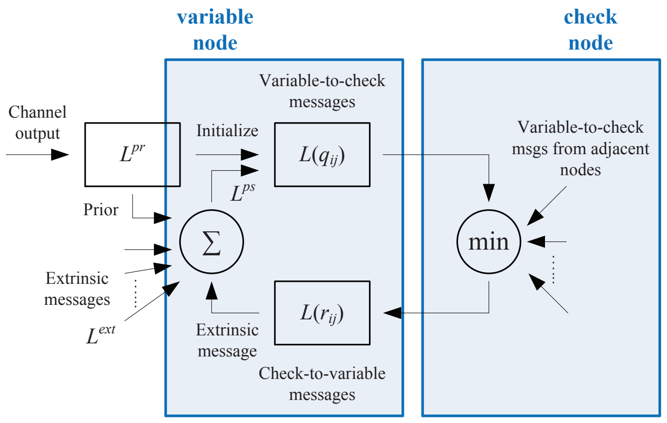
$$\Phi(x) = -\log\left(\tanh\left(\frac{1}{2}x\right)\right), x \geq 0. \tag{2.4}$$



(a)


variable node
check node
Channel output
L^{pr}
Prior
Initialize
Variable-to-check messages
$L(q_{ij})$
L^{ps}
Σ
Extrinsic messages
L^{ext}
Extrinsic message
$L(r_{ij})$
Check-to-variable messages
Φ_1 (Φ function)
Φ_2 (Φ^{-1} function)
Variable-to-check msgs from adjacent nodes

(b)



(c)

Figure 2.3: (a) A factor graph with one slice highlighted. The slice consists of one variable node and one check node. The implementation of the slice is illustrated for (b) a sum-product message-passing decoder and (c) an approximate sum-product message-passing decoder.

The messages q_{ij} and r_{ij} refer to the variable-to-check and check-to-variable messages, respectively, that are passed between the ith variable node and the jth check node. In representing the connectivity of the factor graph, $Col[i]$ refers to the set of all the check nodes adjacent to the ith variable node and $Row[j]$ refers to the set of all the variable nodes adjacent the jth check node.

The posterior LLR is computed in each iteration using (2.5) and (2.6). A hard decision is made based on the posterior LLR as in (2.7).

$$L^{ext}(x_i) = \sum_{j' \in Col[i]} L(r_{ij'}), \tag{2.5}$$

$$L^{ps}(x_i) = L^{ext}(x_i) + L^{pr}(x_i), \tag{2.6}$$

$$\hat{x_i} = \begin{cases} 0 & \text{if } L^{ps}(x_i) \geq 0, \\ 1 & \text{if } L^{ps}(x_i) < 0. \end{cases} \tag{2.7}$$

The iterative decoding algorithm is allowed to run until the hard decisions satisfy all the parity check equations or when an upper limit on the iteration number is reached, whichever occurs earlier.

2.1.2 Approximate Sum-product Algorithm (ASPA)

Equation (2.3) can be simplified by observing that the magnitude of $L(r_{ij})$ is usually dominated by the minimum $|L(q_{i'j})|$ term. As shown in [30] and [28], the update (2.3) can be approximated as

$$L(r_{ij}) = \min_{i' \in Row[j] \backslash i} |L(q_{i'j})| \prod_{i' \in Row[j] \backslash i} \operatorname{sgn}\left(L(q_{i'j})\right). \tag{2.8}$$

Note that equation (2.8) precisely describes the check-node update of the min-sum algorithm. The magnitude of $L(r_{ij})$ computed using (2.8) is usually overestimated and correction terms are introduced to reduce the approximation error. The correction can be either in the form of a normalization factor shown as α in (2.9) [9], an offset shown as β in (2.10) [9], or a conditional offset [82].

$$L(r_{ij}) = \frac{\min_{i' \in Row[j] \backslash i} |L(q_{i'j})|}{\alpha} \prod_{i' \in Row[j] \backslash i} \operatorname{sgn}\left(L(q_{i'j})\right). \tag{2.9}$$

$$L(r_{ij}) = \max\left\{\min_{i' \in Row[j] \backslash i} |L(q_{i'j})| - \beta, 0\right\} \prod_{i' \in Row[j] \backslash i} \operatorname{sgn}\left(L(q_{i'j})\right). \tag{2.10}$$

2.2 Message Quantization and Processing

Practical implementations only approximate the ideal message-passing algorithm. Such approximations are inevitable since real-valued messages can only be approximately represented in a limited wordlength, thus causing saturation and quantization effects, and moreover, the number of iterations is limited, so that the effectiveness of iterative decoding cannot be fully realized.

The approximations are illustrated by considering a pass through the sum-product decoding loop shown in Fig. 2.3b. The channel output is saturated and quantized before it is saved as the prior LLR, L^{pr}. During the first phase of message passing, variable-to-check messages pass through the log-tanh transformation defined in (2.4), then the summation and marginalization, and finally the inverse log-tanh transformation. The log-tanh function is its own inverse, so the two transformations are identical. They are referred to as Φ_1 and Φ_2. The log-tanh function is approximated by discretization. The input and output of the function are saturated and quantized, thus the characteristics of this function cannot be fully captured in finite precision, especially in the regions approaching infinity and zero.

In the second phase of message passing, the extrinsic messages L^{ext} are combined with the prior L^{pr} to produce the posterior probability L^{ps}. The prior, L^{pr}, is the saturated and quantized channel output; the extrinsic message, L^{ext}, is the sum of check-to-variable messages, which originate from the outputs of the approximated Φ_2 function. The messages incur numerical errors, and these errors accumulate, causing a decoder to perform worse than theoretically possible. The deficiencies due to real-valued implementations manifest themselves via performance degradation in the waterfall region, and a rise of the error floor.

The saturation and quantization effects are related to the finite wordlength representation that is used in the processing and storage of data. Two classes of number representations can be used: a more flexible floating-point format which allows the representation of finer resolution and wider range of values but involves more computationally-demanding arithmetic operations, and a compact fixed-point format with a fixed number of digits before and after the radix point. In the case of a high-throughput LDPC decoder, the cost of parallel processing dictates that each processing element be simplified and the fixed-point number format becomes the preferred choice.

The notation Q$m.f$ is used to represent a signed fixed-point number with m bits to the left of the radix point to represent integer values, and f bits to the right of the radix point to represent fractional values. Such a fixed-point representation translates to a quantization resolution of 2^{-f} and a range of $[-2^{m-1}, 2^{m-1} - 2^{-f}]$. Note that there is an asymmetry between the maximum and the minimum because 0 is represented with a positive sign in this number format. Values above the maximum or minimum are saturated, i.e., clipped. The wordlength of this fixed-point number is $m + f$. As an example, a Q4.2 fixed-point quantization translates to a quantization resolution of 0.25 and a range of $[-8, 7.75]$.

In an ASPA implementation (2.8), Φ_1, summation, and Φ_2 are replaced by the minimum operation as shown in Fig. 2.3c. The approximate algorithm introduces errors algorithmically, but it eliminates some numerical saturation and quantization effects by skipping through the log-tanh and the summation operations.

2.3 Structured LDPC Codes

A practical high-throughput LDPC decoder can be implemented in a fully parallel manner by directly mapping the factor graph onto an array of processing elements interconnected by wires, as illustrated in Fig. 2.4a. Under this architecture, each variable node is mapped to a variable node processing element (VN) and each check node is mapped to a check node processing element (CN), such that all messages from variable nodes to check nodes and then in reverse are processed concurrently. Practical high-performance LDPC codes commonly feature block lengths on the order of 1kb and up to 64kb, requiring a large number of VN nodes. The ensuing wiring overhead poses a substantial obstacle towards efficient silicon implementations. The causes of concern are as follows:

1. Each connection between VN and CN consists of multiple wires to support the neces-

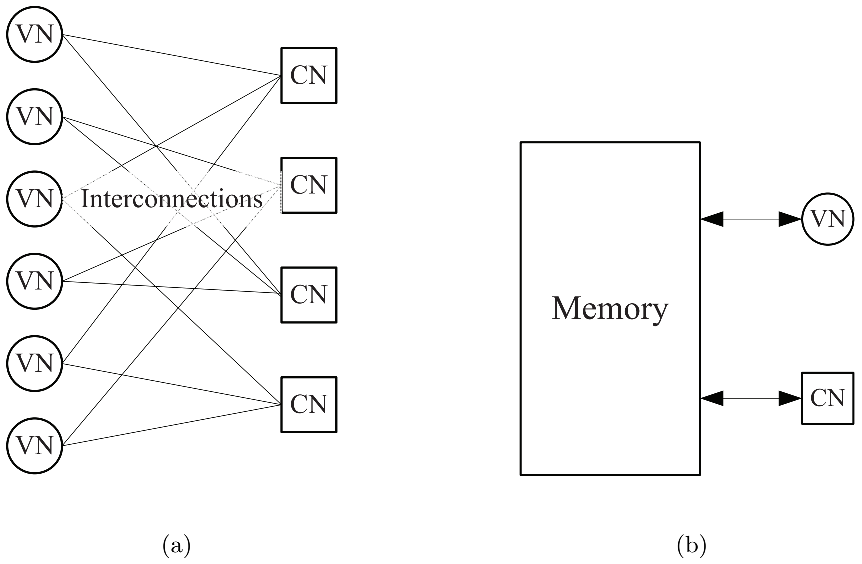


Figure 2.4: Illustration of (a) a parallel decoder architecture, and (b) a serial decoder architecture.

sary wordlength in representing messages. To achieve a good functional performance, wordlength needs to be increased, and so does the number of wires.

2. A large number of VN and CN nodes span a large chip area, and the wires between them are global wires. Global wires are known to suffer from large propagation delays and not scalable with semiconductor technology.

3. Good LDPC codes should resemble a random code with very large block length. Wires supporting the decoders of such codes are necessarily long and irregularly structured, causing difficulty in placement and routing.

On the other hand, a fully serial architecture can be very efficiently constructed. Only one VN and one CN are required and messages can be stored in memory, shown in Fig. 2.4b. Messages are processed sequentially in this architecture, resulting in a very low throughput limited by memory bandwidth. However, this architecture is very flexible and can be easily reconfigured for different codes. More VN and CN nodes could be added to partially parallelize this architecture, but the memory bandwidth limits the level of parallelism and the decoding throughput [74]. A randomly-constructed, or irregular code further complicates the scheduling of a partially parallelized decoder.

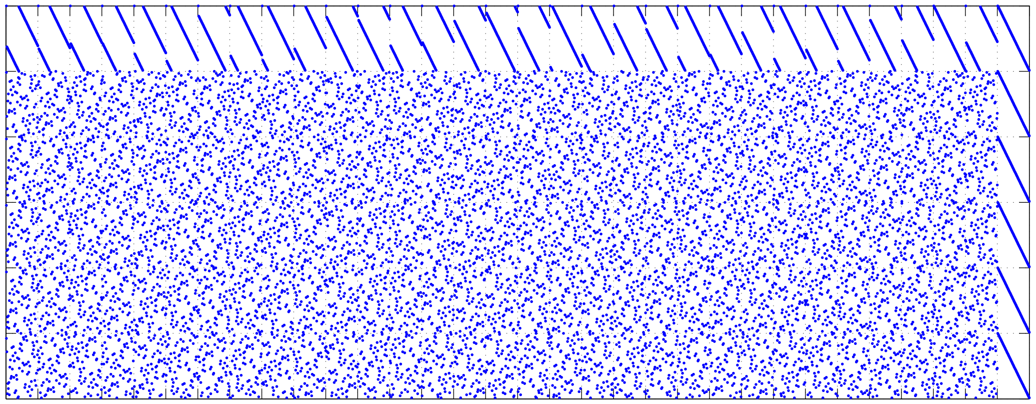
Despite the superior performance of a randomly-constructed, irregular LDPC code, the hardware architecture for the decoders presents difficulties in achieving a high throughput. Structured LDPC codes of moderate block lengths have received more attention in recent research, noticeably the algebraic constructions which are shown to perform within a fraction of dB away from the Shannon limit. Several of these LDPC code constructions, including the Reed-Solomon based codes [20], array-based codes [26], as well as the ones proposed by Tanner *et al.* [62], share the same property that their parity check matrices can be written as a two-dimensional array of component matrices of equal size, each of which is a permutation matrix. Constructions using the ideas of Margulis and Ramanujan [58] have a similar property that the component matrices in the parity check matrix are either permutation or all-zeros matrices. The renditions of a RS-LDPC code and a Ramanujan-Margulis based LDPC code are illustrated in Fig. 2.5a and 2.5b – each 1 in the respective parity-check matrix is shown as a dot and each 0 is shown as a white space. In this family of LDPC codes, the $M \times N$ $\mathbf{H}$ matrix can be partitioned along the boundaries of $\delta \times \delta$ permutation submatrices. For $N = \delta\rho$ and $M = \delta\gamma$, column partition results in ρ column groups and row partition results in γ row groups. This structure of the parity check matrix proves amenable for efficient decoder architectures and recent published standards have adopted LDPC codes defined by such **H** matrices [36–38].

Structured codes open the door to a range of feasible high-throughput decoder architectures ranging from parallelized serial to fully parallel. In a fully parallel architecture, structured codes allow the grouping of VN and CN nodes and the wires between VN and CN nodes of the same group can to be bundled and routed together as shown in Fig. 2.7a for the example **H** matrix in Fig. 2.6. Global wires can be regularized and wiring irregularity can be localized to within the group, thereby significantly reducing the wiring overhead. A serial architecture also benefits from a structured code by effective parallelization: memory can be divided into banks so to avoid access conflicts and decoding schedules can be easily formulated to parallel process among the decoupled code segments. An illustration is shown in Fig. 2.7b.

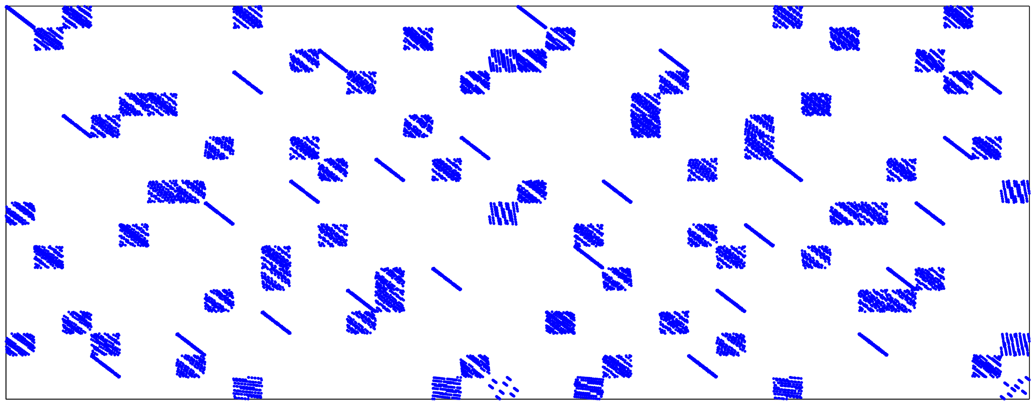
2.4 Emulation-based Investigation

The performance of suitably-designed LDPC codes of large block length can be almost exactly analyzed using techniques such as density evolution and EXIT charts. These techniques assume that the factor graph contains no cycle, and they are based on the asymptotic approximation that the code block length is infinitely long. The assumption and the approximation that form the basis of the analytical techniques do not apply to practical LDPC codes, which usually feature structured parity-check matrices and moderate block lengths on the order of 1kb. Cycles are inevitable in the factor graphs of these codes, though short cycles can be eliminated by suitable code construction strategies.

Software simulation has been used extensively to characterize the performance



(a)

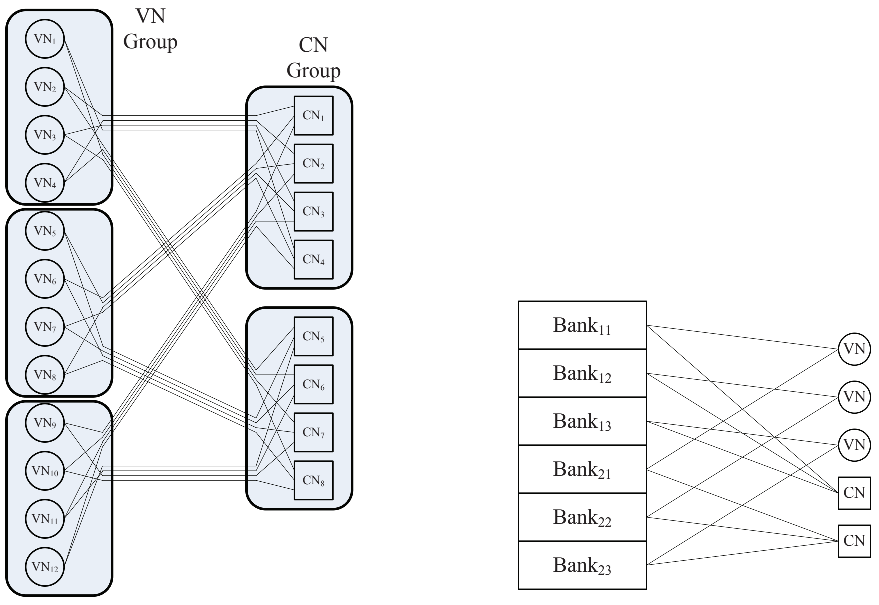


(b)

Figure 2.5: Illustration of parity-check matrices of (a) a $(2048, 1723)$ RS-LDPC code, and (b) a $(4896, 2448)$ Ramanujan-Margulis based LDPC code.

$$\left[\begin{array}{cccc|cccc|cccc}
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
\hline
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}\right]$$

Figure 2.6: A structured parity-check matrix.



(a) (b)

Figure 2.7: (a) An improved parallel architecture by node grouping and wire bundling. (b) A partially parallel architecture by segmenting memory into banks.

of practical LDPC codes. A bit error rate on the order of 10^{-6} to 10^{-8} can be easily achieved on high-performance computing platforms. Such characterizations are sufficient for applications such as most of the wireless standards. However, high-throughput applications, such as wireline, satellite, optical communications, and magnetic storage systems require error free operations below 10^{-10}. The shortage of simulation power and lack of analytical approaches have left a gap in the understanding of practical LDPC codes. The performance uncertainty has prevented or slowed down the adoption of these codes in many high-throughput applications.

An emulation-based design flow is developed to facilitate the investigation of LDPC codes, as seen in Fig. 2.8. The design flow is based on the Berkeley Emulation Engine 2 (BEE2) platform [8]. The BEE2 platform consists of both the FPGA array hardware and the Simulink-based programming paradigm. The message-passing algorithm is first described in a fixed-point reference model in Matlab. The decoder is then constructed in Simulink. Simulink simulations are verified against the Matlab reference model, before mapping to FPGA. More parallel architectures can be implemented on FPGA, providing a throughput at least several orders of magnitude higher than software simulations to reach very low BER levels. The Simulink-based design flow allows the rapid translation from data-flow-based design entry to hardware, enabling iterative designs and refinement.

2.4.1 Decoder Architectures for Emulation

Designing a decoder emulator on FPGA should be differentiated from designing a decoder for practical implementations. Practical implementations aim at high performance (function and throughput) and efficiency (area and power) while satisfying a particular ap-

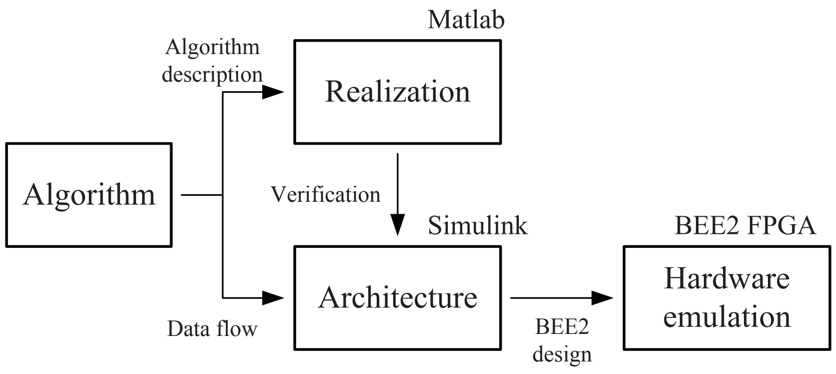


Figure 2.8: Design flow for hardware emulation.

plication requirement, whereas the decoder emulator is designed with resource efficiency and configurability as the primary objectives. The FPGA platform is used as an investigation platform, and as such a large amount of resources on FPGA are dedicated to capturing event traces for analysis, leaving a limited level of parallelism available to the decoder design. The architecture of the decoder should also be very reconfigurable, so that it can be programmed for different codes, decoding algorithms, and capable of operating with a varying number of iterations at different SNR levels.

Two architectures have been used to map the decoder emulators, a canonical architecture and a layered architecture. Both architectures are based on the partial parallelization of the serial architecture, which resembles the designs proposed in [33, 48, 76], but the degree of parallelism is intentionally limited by partitioning the **H** matrix in only one direction (i.e., parallelize among column partitions and process rows serially) to reduce complexity. Each of the partitions is configurable based on the structure of the **H** matrix. Compared to a fully parallel architecture [4], which is not reconfigurable, or a fully serial architecture, which lacks the throughput [74], these architectures represent a tradeoff for the purpose of code emulation.

A $(6,32)$-regular $(2048,1723)$ RS-LDPC code is selected for the illustration of these architectures. This particular LDPC code has been adopted as the forward error correction in the IEEE 802.3an 10GBASE-T standard [36], which governs the operation of 10 Gigabit Ethernet over up to 100 m of CAT-6a unshielded twisted-pair (UTP) cable. The $\mathbf{H}$ matrix of this code contains $M = 384$ rows and $N = 2048$ columns. This matrix can be partitioned into $\gamma = 6$ row groups and $\rho = 32$ column groups of $\delta \times \delta = 64 \times 64$ permutation submatrices.

In the canonical architecture, column partition is applied to divide the decoder into 32 parallel units, where each unit processes a group of 64 bits. Fig. 2.9 illustrates the architecture of the RS-LDPC sum-product decoder. Two sets of memories, $M0$ and $M1$, are designed to be accessed alternately. $M0$ stores variable-to-check messages and $M1$ stores check-to-variable messages. Each set of memories is divided into 32 banks. Each bank is assigned to a processing unit that can access them independently. In a check-to-variable operation defined in (2.3), the 32 variable-to-check messages pass through the log-tanh transformation, and then the check node computes the sum of these messages. The sum is marginalized locally in the processing unit and stored in $M1$. The stored messages pass through the inverse log-tanh transformation to generate check-to-variable messages. In the variable-to-check operation defined in (2.2), the variable node inside every processing unit accumulates check-to-variable messages serially. The sum is marginalized locally and stored in $M0$. This architecture minimizes the number of global interconnects by performing marginalization within the local processing unit.

The canonical architecture realizes the canonical form of the sum-product algo-

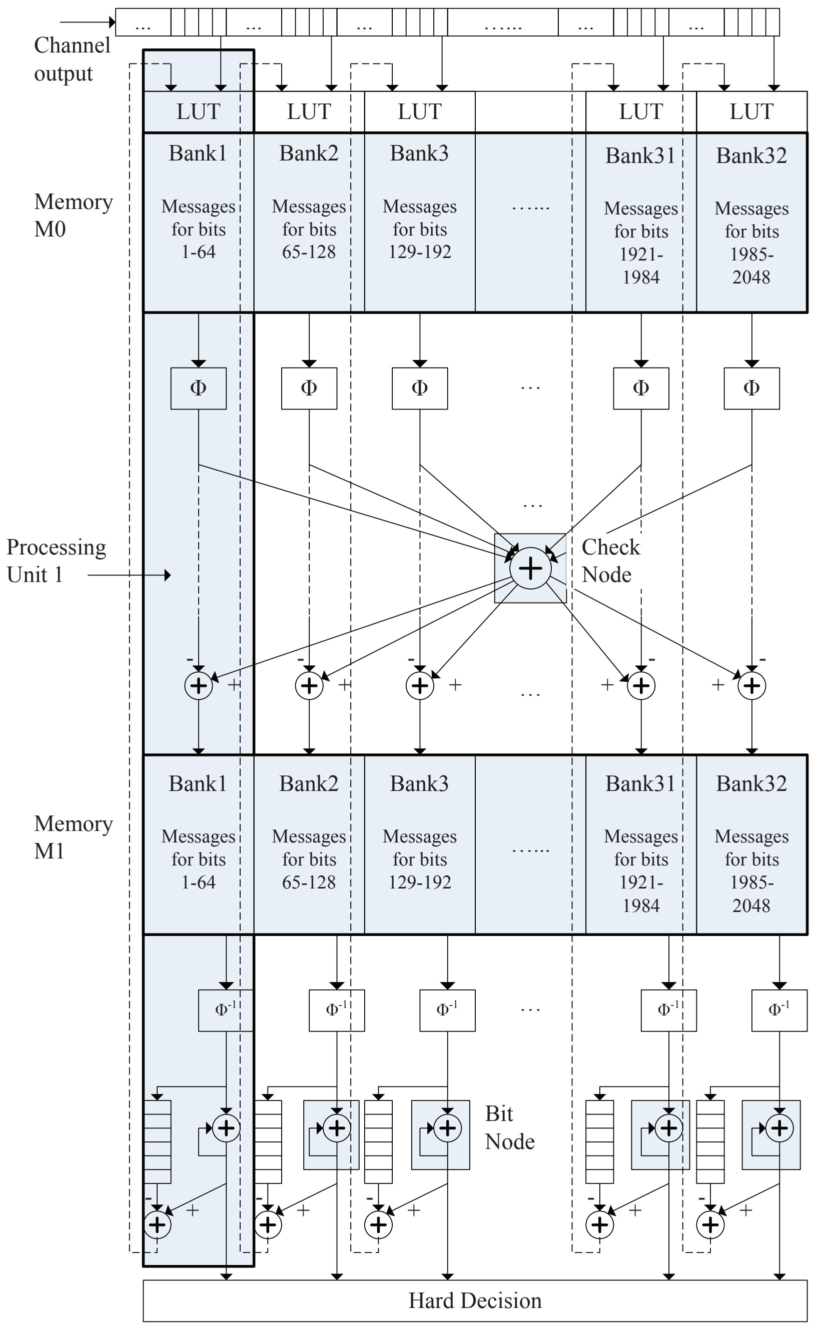


Figure 2.9: A canonical architecture of the (2048,1723) RS-LDPC decoder composed of 32 processing units.

rithm shown in (2.2), (2.3), (2.5), and (2.6). These equations can also be rearranged by taking into account the relationship between consecutive decoding iterations. A variable-to-check message of iteration n can be computed by subtracting the corresponding check-to-variable message from the posterior of iteration $n-1$ as in (2.11), while the posterior of iteration n can be computed by updating the posterior of the previous iteration with the check-to-variable message of iteration n, as in (2.13).

$$L_n(q_{ij}) = L^{ps}_{n-1}(x_i) - L_{n-1}(r_{ij}), \tag{2.11}$$

$$L_n(r_{ij}) = \Phi^{-1}\left(\sum_{i' \in Row[j]\backslash i} \Phi\left(|L_n(q_{i'j}))|\right)\right)\left(\prod_{i' \in Row[j]\backslash i} \operatorname{sgn}\left(L_n(q_{i'j})\right)\right), \tag{2.12}$$

$$L^{ps}_n(x_i) = L^{ps}_{n-1}(x_i) - L_{n-1}(r_{ij}) + L_n(r_{ij}), j \in Col[i]. \tag{2.13}$$

The reformulated sum-product algorithm leads to a check-node centric message-passing schedule without an explicit variable-node operation. When interpreted using the **H** matrix, operations are performed in horizontal layers, thus it is called the layered architecture. The block diagram of the layered architecture is shown in Fig. 2.10 for the $(2048, 1723)$ RS-LDPC code. Only one set of memory $M0$ is required to store the check-to-variable messages and the posterior. In each iteration except the first, the check-to-variable message from the previous iteration is subtracted from the posterior to produce the variable-to-check message as in (2.11). One variable-to-check message from each of the column groups is processed by the check node, and the new check-to-variable message is computed according to (2.12). The new check-to-variable message replaces the old check-to-variable message to update the posterior as in (2.13). Compared to the canonical architecture, the variable-to-check operation is interleaved with the check-to-variable operation in the layered architecture.

Both types of architectures allow efficient mapping of a practical decoder. For example, an RS-LDPC code of up to 8kb in block length can be supported on a Xilinx Virtex-II Pro XC2VP70 FPGA [70]. These architectures are also reconfigurable, so that any member of the LDPC code family described in Section 2.3 can be accommodated. Address lookup tables can be reconfigured based on the **H** matrix. Processing units can be allocated depending on the column partitions, and the memory size can be adjusted to allow variable code rates.

The decoding throughput of both types of architectures is determined by the dimensions of the **H** matrix of the LDPC code. In a high SNR regime, the majority of the received frames can be decoded in one iteration. Therefore, the maximum achievable throughput is approximately $f_{clk}N/(2M)$ for the canonical architecture, and $f_{clk}N/M$ for the layered architecture, where f_{clk} represents the clock frequency. Since pipeline stalls need to be inserted between variable-to-check and check-to-variable operations in a canonical architecture, and between horizontal layers in a layered architecture to resolve data dependencies, the peak throughput attainable in practice is slightly lower than what is quoted above. Note that a characteristic of both types of architectures is that the decoding throughput depends on N/M, which is related to the rate of the code – the higher the code rate, the higher the decoding throughput.

The ASPA decoders can be implemented similarly. Following the approximation

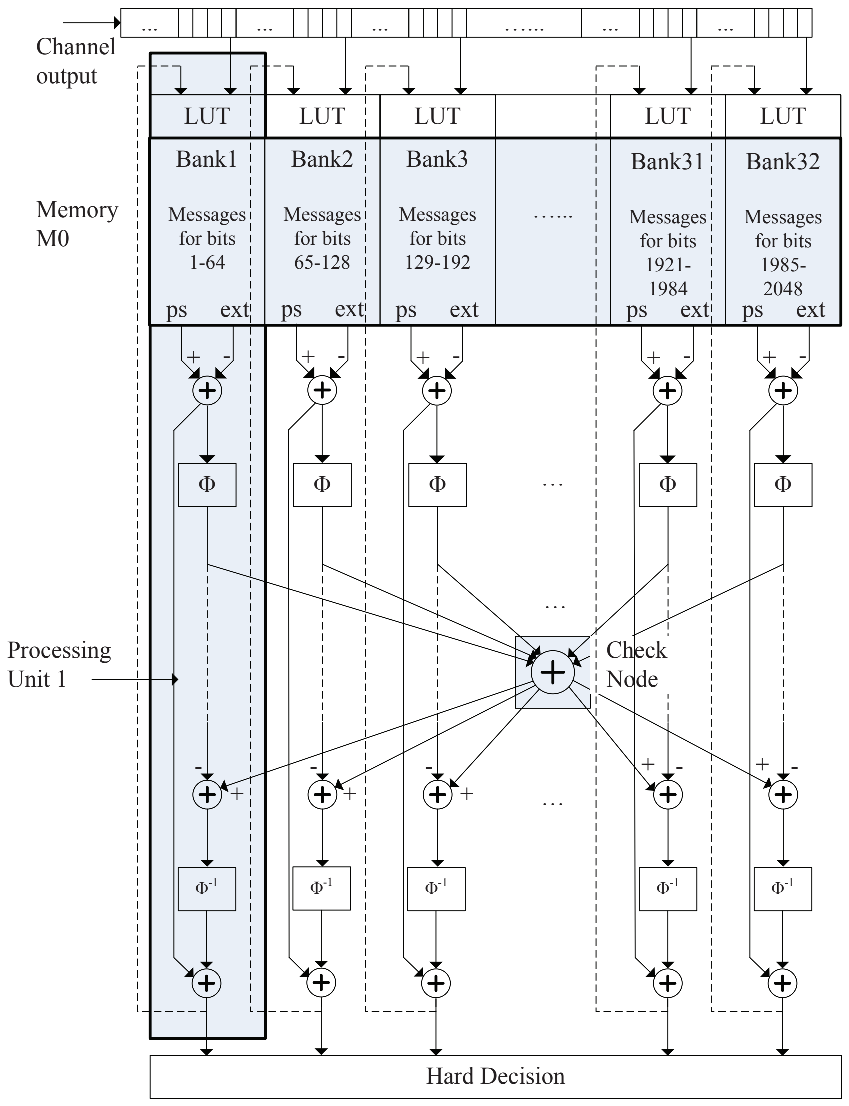


Figure 2.10: A layered architecture of the (2048,1723) RS-LDPC decoder composed of 32 processing units.

(2.8), the lookup tables based on Φ are eliminated and the summation in a check node is replaced by comparisons to find the minimum.

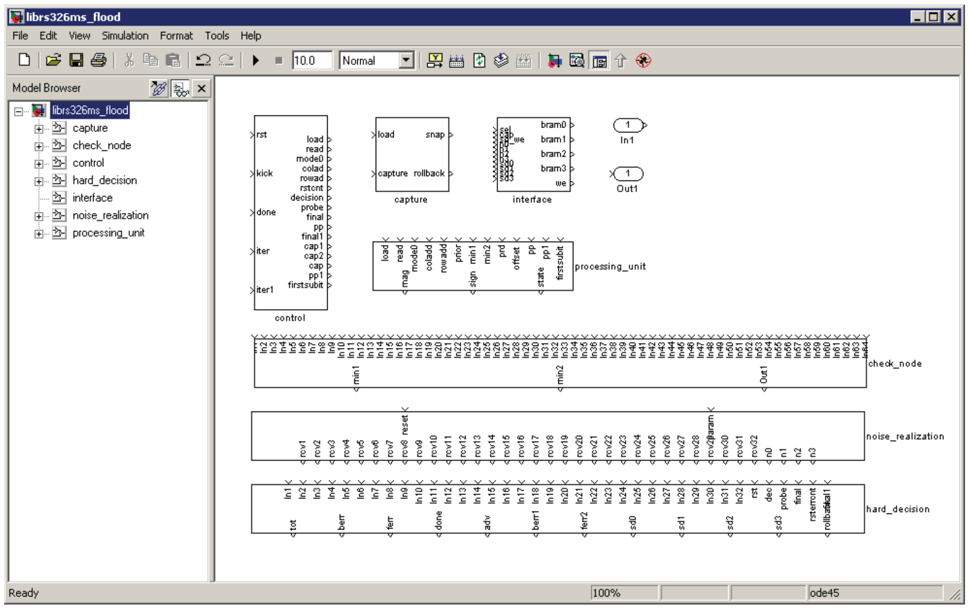
2.4.2 Design Flow

The decoder is hierarchically constructed in a bottom-up manner. The basic component modules, including a processing unit (highlighted in Fig. 2.9 and 2.10), a check node, and a controller, are designed and verified in Simulink. These component modules are parameterized. The processing unit is parameterized by wordlength, quantization, and the submatrix supported. The check node is constructed as an adder tree (in a sum-product algorithm), or a compare-select tree (in an ASPA decoder). The breadth and depth of the tree are determined by the number of partitioned column groups. The controller is parameterized by the check and variable node degrees, column partitions, and submatrix size. These modules are copied to a Simulink design library, as in Fig. 2.11a.

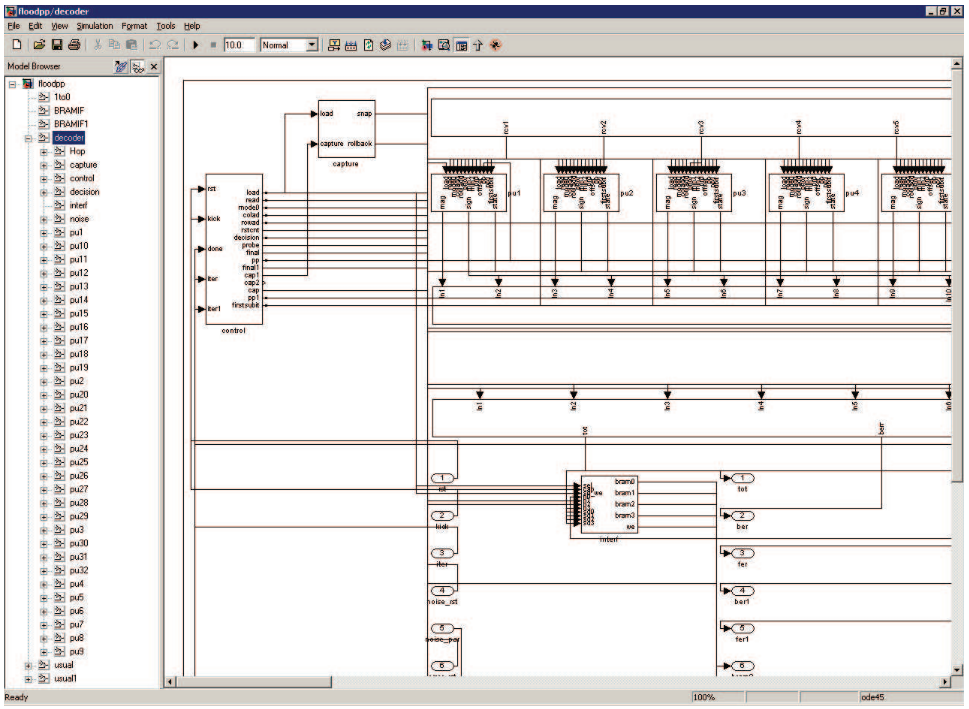
A Matlab script takes as inputs the **H** matrix of the LDPC code, the decoding algorithm, as well as the quantization choice, and then instantiates modules from the Simulink design library. Most importantly, the Matlab script draws the connections between modules based on the **H** matrix. An example design is illustrated in Fig. 2.11b. This approach significantly simplifies the design process and enables the design-time configurability.

2.4.3 Noise Realization

Along with the LDPC decoder, multiple independent additive Gaussian noise generators have been incorporated on the FPGA using the Xilinx AWGN generator [69] to emulate the communication channel. The datasheet specifies that the probability density



(a)



(b)

Figure 2.11: (a) A design library containing component modules. (b) A portion of a complete LDPC decoder design showing instantiated component modules and the interconnections drawn by a Matlab script.

function (PDF) of the noise realization deviates within 0.2% from the ideal Gaussian PDF up to 4.8σ [69]. Questions arise on whether this noise generator would allow the decoding performance to be truthfully characterized down to very low error rate levels. In particular, what is of interest is how much the decoder performance would deviate from the one operating under the ideal AWGN channel. To answer this question, the decoder is treated as a blackbox and inputs causing decoding errors at very low error rate levels are captured. The empirical error probability under the Xilinx noise realization can be compared to the error probabilities under ideal Gaussian channels. The inputs are characterized using quantized (binned) samples, because the decoder operates on quantized inputs. This study consists of the following three steps:

1. Bound the noise distribution

   (a) Characterize the binned noise samples produced by the Xilinx noise generator, $f_X^{Xilinx}(x) = Pr[X = x], x \in S$, where S indicates the sample space, or the set of quantized levels.

   (b) Compute the cumulative mass function (CMF) $F_X^{Xilinx}(x) = Pr(X \leq x), x \in S$. Empirically bound $F_X^{Xilinx}(x)$ by the CMF of two ideal Gaussian distributions, $\mathcal{N}_1 \sim \mathcal{N}(0, \sigma_1)$ and $\mathcal{N}_2 \sim \mathcal{N}(0, \sigma_2)$, as the lower and upper bound respectively, such that $F_X^{\mathcal{N}_1}(x) \leq F_X^{Xilinx}(x) \leq F_X^{\mathcal{N}_2}(x)$ for $x \in S$.

2. Characterize the decoder performance by hardware emulation

   Select an SNR point of interest and run decoder emulations. The SNR point of interest is at the moderate to high-SNR levels where the error floors could occur. Assume all-zeros codeword is transmitted using a BPSK modulation, where the binary channel bits $\{0, 1\}$ are mapped to $\{1, -1\}$ for transmission over the AWGN channel. Capture a set of decoding errors T and perform the following three steps for each error.

(a) Characterize the noise realization causing this error. Assume a code block length of N, for each $i \in N$, compute $F_X^{Xilinx}(x_i)$, where x_i corresponds to the noise sample at bit i and $x_i \in S$. From Step 1, $F_X^{\mathcal{N}_1}(x_i) \leq F_X^{Xilinx}(x_i) \leq F_X^{\mathcal{N}_2}(x_i)$.

(b) The tightness of the bounds can be improved by finding the maximum multiplier m_{1,x_i} and the minimum multiplier m_{2,x_i} that satisfy $m_{1,x_i} F_X^{\mathcal{N}_1}(x_i) \leq F_X^{Xilinx}(x_i) \leq m_{2,x_i} F_X^{\mathcal{N}_2}(x_i)$.

(c) Compute the probability of the decoding error (frame error) under the Xilinx AWGN channel $P_e^{Xilinx} = \prod_{1 \leq i \leq N} F_X^{Xilinx}(x_i)$, and bound it by the decoding error (frame error) probabilities under the ideal AWGN noise channels: $P_e^{\mathcal{N}_1} = \prod_{1 \leq i \leq N} F_X^{\mathcal{N}_1}(x_i)$ and $P_e^{\mathcal{N}_2} = \prod_{1 \leq i \leq N} F_X^{\mathcal{N}_2}(x_i)$, i.e., $M_1 P_e^{\mathcal{N}_1} \leq P_e^{Xilinx} \leq M_2 P_e^{\mathcal{N}_2}$, where $M_1 = \prod_{1 \leq i \leq N} m_{1,x_i}$ and $M_2 = \prod_{1 \leq i \leq N} m_{2,x_i}$.

3. Compute the performance bounds

   The product of multipliers M_1 and M_2 provide empirical measures of how much the frame error rate obtained from hardware emulation deviates from the simulations based on ideal AWGN channels.

A larger T size yields more reliable estimates of the performance bounds. But even with a set of 64 errors collected at the FER of approximately 10^{-10}, intuitions can be gained on how the noise fidelity affects the emulation results. The above procedure is followed in characterizing the $\mathcal{N}(0, 1)$ Xilinx Gaussian noise generator based on 2^{32} samples. The noise

Table 2.1: Characterization of the Xilinx noise generator.

x_i	$F_X^{Xilinx}(x_i)$	$F_X^{\mathcal{N}_1}(x_i)$	$F_X^{\mathcal{N}_2}(x_i)$	m_{1,x_i}	m_{2,x_i}
-4.00	5.31×10^{-5}	5.21×10^{-5}	5.31×10^{-5}	1.0197	1.0000
-3.75	1.43×10^{-4}	1.42×10^{-4}	1.44×10^{-4}	1.0081	0.9910
-3.50	3.65×10^{-4}	3.63×10^{-4}	3.69×10^{-4}	1.0036	0.9887
-3.25	8.78×10^{-4}	8.78×10^{-4}	8.89×10^{-4}	1.0006	0.9876
-3.00	2.00×10^{-3}	2.00×10^{-3}	2.02×10^{-3}	1.0000	0.9889
-2.75	4.30×10^{-3}	4.30×10^{-3}	4.34×10^{-3}	1.0000	0.9906
-2.50	8.73×10^{-3}	8.73×10^{-3}	8.80×10^{-3}	1.0003	0.9925
-2.25	1.67×10^{-2}	1.67×10^{-2}	1.68×10^{-2}	1.0004	0.9940
-2.00	3.04×10^{-2}	3.03×10^{-2}	3.05×10^{-2}	1.0003	0.9951
-1.75	5.21×10^{-2}	5.21×10^{-2}	5.23×10^{-2}	1.0002	0.9961
-1.50	8.47×10^{-2}	8.47×10^{-2}	8.49×10^{-2}	1.0001	0.9970
-1.25	1.31×10^{-1}	1.31×10^{-1}	1.31×10^{-1}	1.0002	0.9980
-1.00	1.91×10^{-1}	1.91×10^{-1}	1.92×10^{-1}	1.0003	0.9988
-0.75	2.67×10^{-1}	2.67×10^{-1}	2.67×10^{-1}	1.0003	0.9993

samples are uniformly quantized in a Q4.2 format, corresponding to a step size of 0.25. Hardware emulation is performed using a Q4.2 sum-product decoder of the $(2048, 1723)$ RS-LDPC code. The resulting CMF $F_X^{Xilinx}(x_i)$ is listed in Table 2.1 (an incomplete table for brevity). This CMF is bounded by the CMFs of two ideal Gaussian distributions, $\mathcal{N}_1 \sim \mathcal{N}(0, 0.997556)$ and $\mathcal{N}_2 \sim \mathcal{N}(0, 0.998776)$, and the associated multipliers m_{1,x_i}, and m_{2,x_i} are listed in Table 2.1.

For each of the 64 errors in T, the product of multipliers M_1 and M_2 are computed. The average products are $M_{1,mean} = 1.178$, $M_{2,mean} = 0.284$, which suggests that the frame error rate under ideal AWGN channels is within a factor of 3.5 above and 0.85 below the results obtained by hardware emulation down to the 10^{-10} FER levels. As what will be described later, in the nonlinear finite-wordlength decoding process based emulations it is observed that the decoder fails to converge at very low error rate levels because of specific patterns of locations in the codeword being subject to noise moderately out in the tail rather than because of noise values in the extreme tails. Thus accuracy of the random number generator in the extreme tail distribution is not of concern in this application, in contrast to what is stated in [42].

2.4.4 Emulation Setup

Block RAMs on the FPGA record the noise realizations and final iterations of soft decisions when decoding fails. With a large number of block RAMs available on modern FPGA devices, a large memory bandwidth and real-time access are possible. For example, the Xilinx Virtex-II Pro XC2VP70 FPGA, featured on the BEE2 platform, contains $5,904$kb of block RAM memory. Assume a 6-bit Q4.2 quantization and a 2kb block length LDPC code. Storing one frame of noise realizations requires 12kb memory. Additional 12kb memory is required to store one iteration of soft decisions. With these assumptions, up to about 100 errors can be captured – each error recording consists of the frame of noise realizations that cause the decoding error and final three decoding iterations of soft decisions. The recordings can be analyzed offline.

An on-chip PowerPC microprocessor controls the decoder by issuing start, stop commands, setting upper limit on the number of decoding iterations, and adjusting the noise variance for different SNR levels. The hardware emulation platform is illustrated in Fig. 2.12. Such a platform allows the characterization of the code and evaluation of practical implementation parameters [77]. Error traces enable the exploration of patterns that cause the decoder to fail.

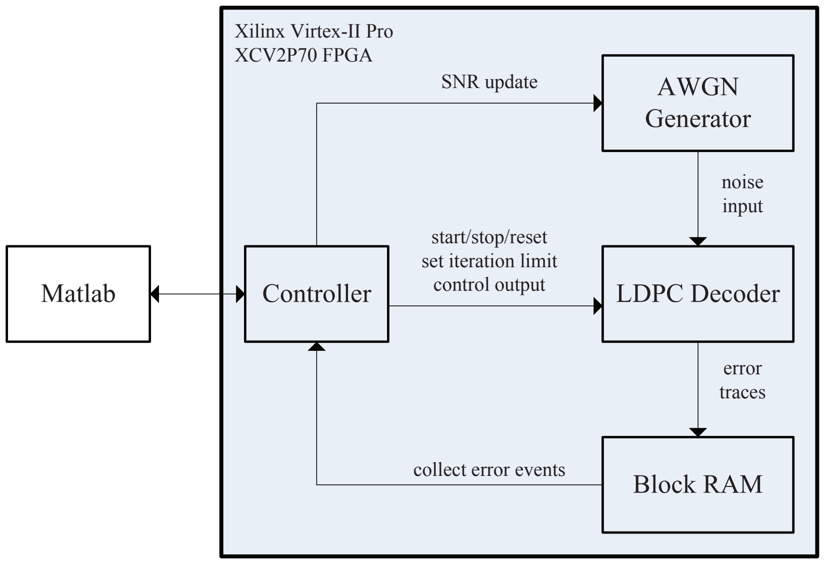


Figure 2.12: An LDPC decoder emulation platform.

The FPGA resource utilization is listed in Table 2.2 for the $(2048, 1723)$ RS-LDPC decoders implemented in the layered architecture with various options: wordlengths of 4, 6, and 8 bits for both the SPA and the ASPA algorithms. These decoders are resource efficient – occupying less than 1/3 of the available slices and 130 of the 328 available block RAMs on a Xilinx Virtex-II Pro XC2VP70 FPGA. The Xilinx AWGN noise generator can be incorporated at the cost of approximately 900 slices. The remaining resource on-chip can be used to store error traces. A decoder implemented this way achieves a peak throughput of 480 Mb/s using a 100 MHz clock. Hardware emulation of this LDPC decoder extends the BER curve below 10^{-10} within hours. For comparison, an optimized implementation of the same decoder in C provides a peak throughput of only 260 kb/s on an Intel Xeon 2.4 GHz microprocessor.

Table 2.2: FPGA resource utilization of the (2048,1723) RS-LDPC decoder designs based on the layered architecture.

	Available[1]	4-bit wordlength			6-bit wordlength			8-bit wordlength		
		SPA	ASPA	w/noise[2]	SPA	ASPA	w/noise[2]	SPA	ASPA	w/noise[2]
Number of Slice Flip Flops	66,176	7,997	7,964	9,153	8,737	8,830	10,017	9,477	9,696	10,881
Number of 4 input LUTs	66,176	7,580	7,763	8,386	8,955	9,671	10,296	10,426	11,318	11,946
Number of Occupied Slices	33,088	7,247	7,322	8,199	8,459	8,810	9,706	9,536	9,961	10,828
Total Number 4 input LUTs	66,176	9,988	10,171	11,252	11,843	12,559	13,645	13,794	14,686	15,778
Number of Block RAMs	328	130	130	134	130	130	134	130	130	134

[1] Available resource and its allocation are based on the Xilinx Virtex-II Pro XC2VP70 FPGA.

[2] An ASPA decoder with the Xilinx AWGN noise generator.

2.4.5 Related Work

FPGA platforms have been commonly used in studying decoder designs, mainly for two purposes – verifying hardware architecture and accelerating code simulation. Decoders of different architectures can be compared by the resource usage and the maximum achievable clock frequency on an FPGA; hardware emulation also accelerates code simulation [61, 71], allowing codes to be quickly evaluated and decoder implementations to be compared under practical settings.

The emulation platform of this work is constructed for studying decoder failure mechanisms. A similar hardware emulation platform in [54] returns only the final state of the decoding, which limits its usefulness in analyzing error mechanisms. The platform described in this work records the noise realizations, as well as the iteration-by-iteration states that lead to a decoding error. Recording the noise realizations and the decoding states in soft information requires a large amount of free resource available on chip, which explains that less than 1/3 of the on-chip resource is allocated to the decoder itself, while all the remaining is reserved for error capturing.

Chapter 3

Investigation of Error Floors

Both the wordlength and the number of decoding iterations are important design parameters that determine the area, power, and performance of an LDPC decoder. In particular, a short wordlength and a small number of iterations are always desirable in practical implementations. As an illustration, the frame error rate (FER) and the bit error rate versus the signal-to-noise ratio are plotted in Fig. 3.1 showing the effect of iteration number on the performance of a 6-bit Q4.2 fixed-point implementation of the $(2048, 1723)$ RS-LDPC sum-product decoder. More iterations result in better performance, although the gain becomes marginal after 50 iterations. So as to minimize the effect of iteration number and to isolate the error events caused by fixed-point implementations, up to 200 iterations are performed.

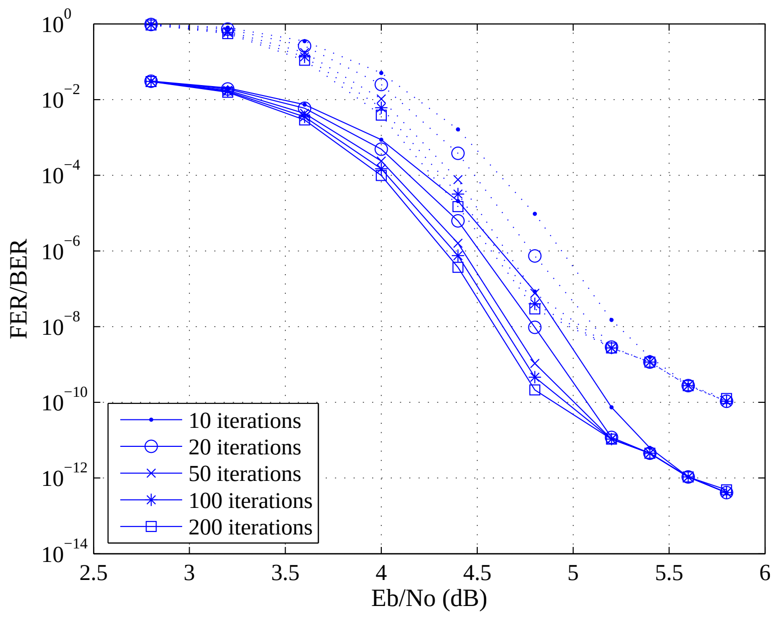
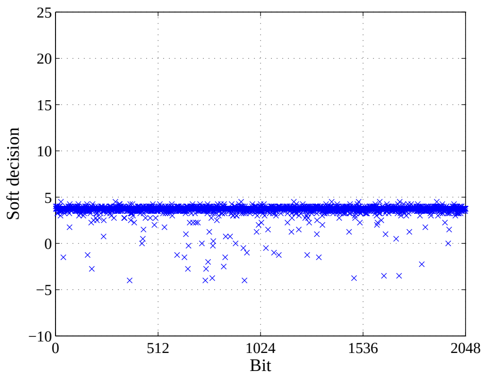


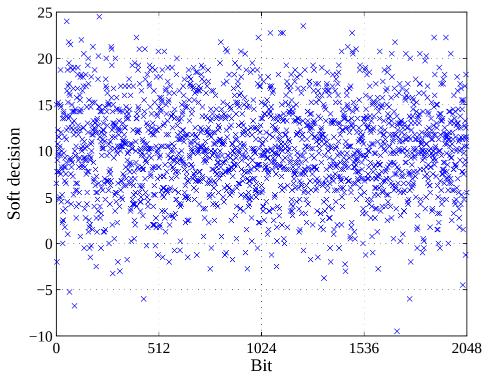
Figure 3.1: FER (dotted lines) and BER (solid lines) performance of the Q4.2 sum-product decoder of the (2048,1723) RS-LDPC code using different number of decoding iterations.

3.1 Characterization of Error Events

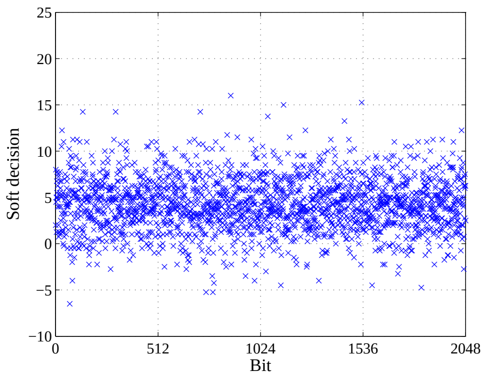
Hardware emulations reveal that the errors dominating the error floors do not resemble the errors occur in the waterfall region of the BER-SNR curve. Errors in the waterfall region are mostly random-like errors with bit error count greater than the minimum distance of the code. At higher SNR levels where the error floors occur, random-like errors are very rare, and instead the dominant errors exhibit rather pronounced characteristics, either oscillatory or absorbing. Both these types of errors appear to start with a small number of bits that are received incorrectly. An oscillation error is illustrated in Fig. 3.2 showing the soft decisions after each decoding iteration. The horizontal axis is for each of the 2048 bits in the code block, and the vertical axis is the soft decision each bit assumes after an iteration of decoding. For simplicity of illustration, it is assumed that all-zeros



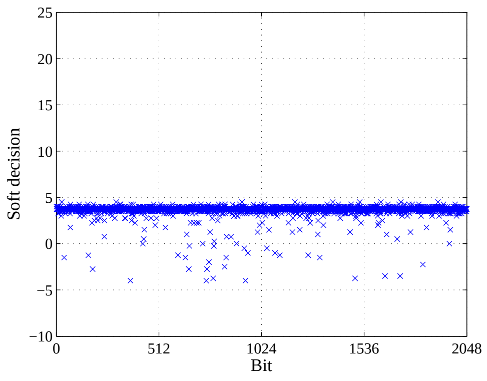
(a) Iteration i



(b) Iteration $i+1$



(c) Iteration $i+2$



(d) Iteration $i+3$

Figure 3.2: Illustration of the oscillation error based on soft decisions from four consecutive decoding iterations.

codewords are transmitted using a BPSK modulation and the binary channel bits $\{0, 1\}$ are mapped to $\{1, -1\}$ for transmission over the channel. Thus the positive soft decisions can be interpreted as correct decisions and the negative decisions as the incorrect ones. An oscillation error appears to be unstable under the message-passing decoding: the number of incorrect bits increases to a certain level before it falls in a periodic fashion. Examples of the bit error counts illustrating the oscillatory behavior are given in Table 3.1.

Absorbing errors behave differently. These errors also start with a small number

Table 3.1: Examples of bit error counts in the final 12 iterations of decoding.

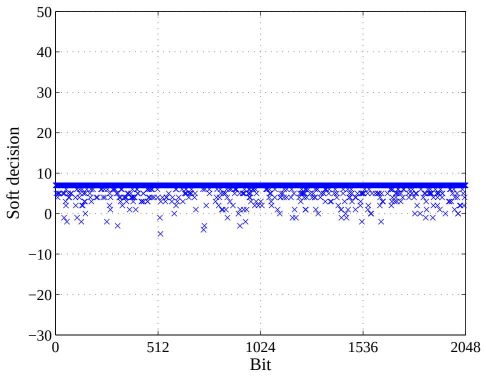
Iteration #	189	190	191	192	193	194	195	196	197	198	199	200
Error 1	23	68	137	23	72	143	24	69	136	24	75	121
Error 2	28	61	125	28	60	149	28	61	125	28	60	149
Error 3	28	52	148	28	49	138	28	49	128	27	51	139

of bits that are received incorrectly. But the small number of bits gradually stabilize among themselves, such that the incorrect bits become more and more incorrect and correct bits become more and more correct, as illustrated in Fig. 3.3, which appears to be a local minimum state that absorbs the message-passing algorithm, thus it is called an absorbing error.

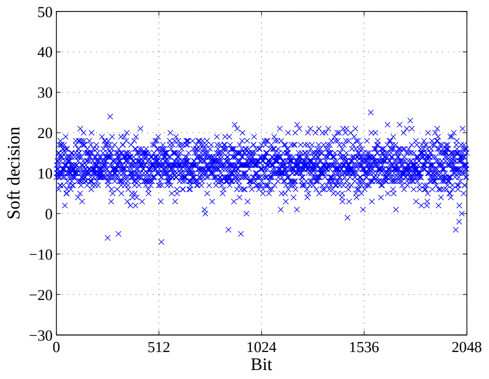
As demonstrated here, hardware emulation provides a convenient replacement for software simulations. Very low error rate performance of practical LDPC decoders can be characterized. Iteration-by-iteration soft decision capture further allows the classification of errors that cause the decoder to fail. Hardware emulation can also be utilized as a step of an iterative design loop, as shown in Fig. 3.4. The input noise realizations that cause the decoding errors are captured and plugged in a functionally-equivalent decoder in Matlab to replicate each failure case. Such a feedback error-replicating simulation, when correlated with code structure and decoding algorithm, could possibly yield in-depth understanding of the decoding errors.

3.2 Absorbing Set

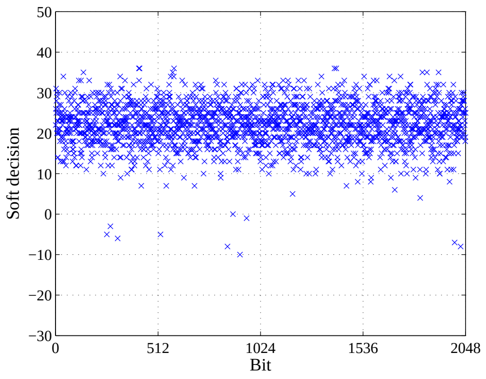
Feedback simulation is applied in investigating decoding errors. Oscillation errors are unstable and can be interpreted as a consequence of the dynamics of quantized message



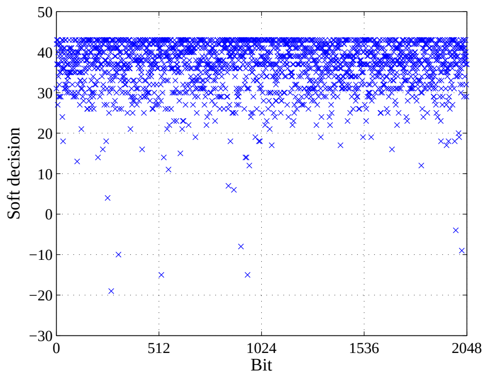
(a) Channel outputs



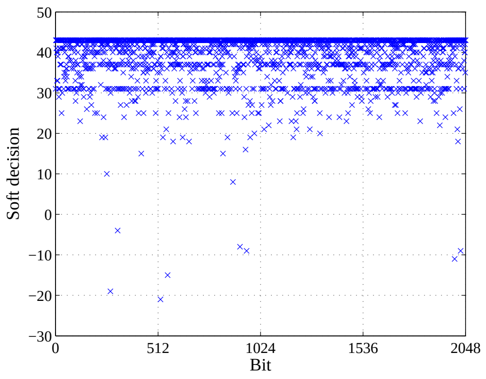
(b) Iteration 1



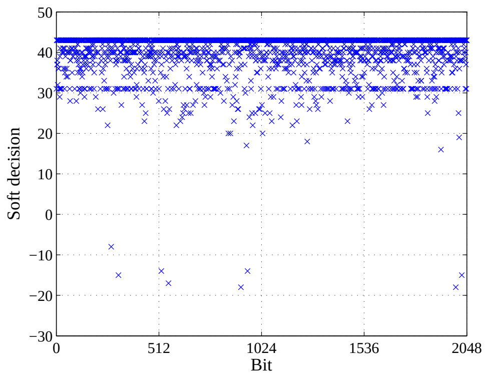
(c) Iteration 2



(d) Iteration 3



(e) Iteration 4



(f) Iteration 5

Figure 3.3: Illustration of the process for a message-passing decoder to enter an absorbing state.

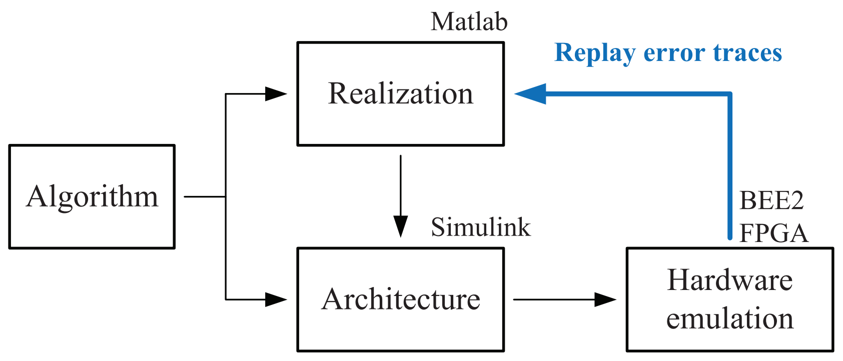
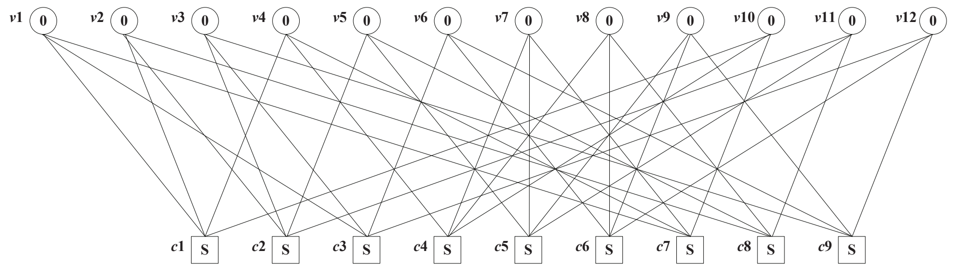


Figure 3.4: Hardware emulation being used in a feedback loop.

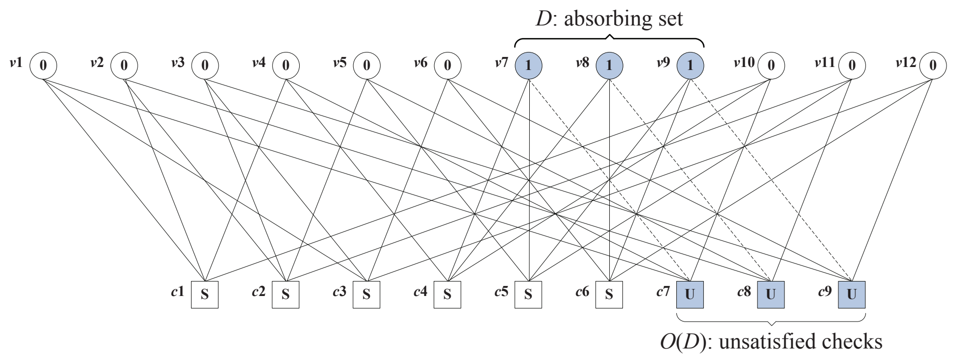
exchange, while absorbing errors are the local minimum states and can be analyzed exactly.

Absorbing errors turn out to be related to the structure of the code under message-passing decoding. For simplicity, hard-decision decoding can be used to illustrate the case. The failure mechanisms of hard-decision and soft decoding are the same in principle as far as absorbing errors are concerned. The starting point is a small factor graph illustrated in Fig. 3.5a that is associated with a small LDPC code. Assume the all-zeros code being transmitted, then all the even parity checks are satisfied. Suppose noise is injected to the transmitted data, such that a subset of the bits are received incorrectly, e.g., the bits corresponding to variable nodes v_7, v_8, and v_9 now assume the incorrect value of 1 as shown in Fig. 3.5b. These incorrectly received bits cause some of the parity checks to be unsatisfied.

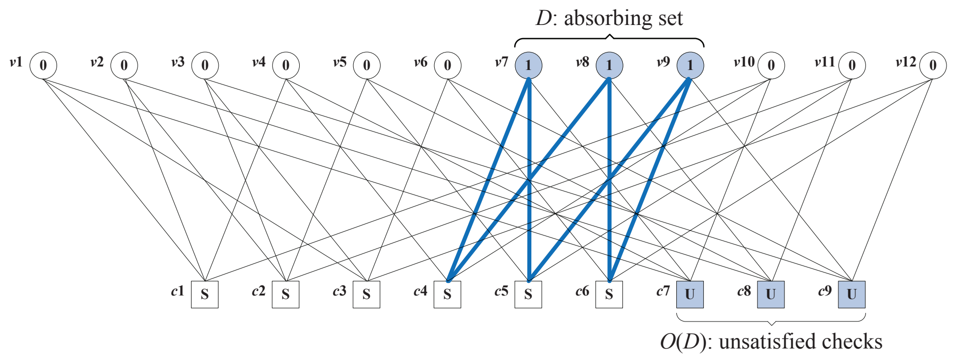
This set of incorrectly received bits constitute an absorbing set, such that all the variable nodes in the absorbing set assume the incorrect value of 1 and all the variable nodes outside the absorbing set assume the correct value of 0. With this setup, every incorrect bit in the absorbing set receives two messages from the satisfied check nodes telling it to


v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12
0
c1 c2 c3 c4 c5 c6 c7 c8 c9
S

(a) All-zeros codeword is transmitted


D: absorbing set
v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12
0
1
c1 c2 c3 c4 c5 c6 c7 c8 c9
S
U
O(D): unsatisfied checks

(b) Message-passing decoder is absorbed


D: absorbing set
v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12
0
1
c1 c2 c3 c4 c5 c6 c7 c8 c9
S
U
O(D): unsatisfied checks

(c) A cycle in the factor graph

Figure 3.5: Illustration of a (3,3) fully absorbing set.

keep the incorrect decision, and it receives one message from the unsatisfied check node telling it to correct the decision. But two is more than one, and the incorrect bit cannot be recovered, thus the message-passing decoder is absorbed.

An absorbing set is fundamentally related to the cycles in the factor graph. A cycle is highlighted in Fig. 3.5c, which connect variable nodes of the absorbing set through some satisfied check nodes. An absorbing error occurs because the cycle in the graph reinforces the incorrect bits among themselves.

Absorbing sets [22, 23, 77] provide a valuable characterization of absorbing errors. In order to define an absorbing set, let $G = (V, F, E)$ be the bipartite graph associated with a parity-check matrix $\mathbf{H}$, such that the set V corresponds to the columns of $\mathbf{H}$, the set F corresponds to the rows of $\mathbf{H}$, and $E = \{e(i, j)|\mathbf{H}(j, i) = 1\}$. Such a graph $G_{\mathbf{H}}$ is commonly referred to as the Tanner or factor graph of the parity check matrix $\mathbf{H}$ of a code [27, 67]. For a subset D of V, let $\mathcal{O}(D)$ be the set of neighboring vertices of D in F with odd degree with respect to D. With this setup we have the following.

Given an integer pair (a, b), an (a, b) absorbing set is a subset D of V of size a, with $\mathcal{O}(D)$ of size b, and with the property that each element of D has strictly fewer neighbors in $\mathcal{O}(D)$ than in $F \setminus \mathcal{O}(D)$. We say that an (a, b) absorbing set D is an (a, b) fully absorbing set, if in addition, all variable nodes in $V \setminus D$ have strictly fewer neighbors in $\mathcal{O}(D)$ than in $F \setminus \mathcal{O}(D)$.

Related notions have been previously introduced in the literature in the attempt to characterize the behavior of the message-passing decoding algorithms when they do not converge to a codeword, such as stopping sets [19], near-codewords [47], and trapping sets [54]. A fully absorbing set, as defined above, can be understood as a special type of near-codeword or trapping set, one which is stable under the bit-flipping decoding algorithm [29]. An example of an (a, b) fully absorbing set with $a = 3$ and $b = 3$ is given in Fig. 3.5b.

The notion of the absorbing set is being used in this work to resolve the ambiguity in the definitions of objects for describing the error floors. The original definition of the trapping set by Richardson is intuitive, but semi-empirical and decoder-dependent. As a result, three different types of errors could be associated with trapping sets [68]: fixed patterns, oscillatory patterns, and random-like patterns. Subsequent work defined trapping set as a fixed point of the decoder [11]. In contrast, the absorbing set is defined as a combinatorial object, and is decoder independent. Oscillations and random-like errors could be disassociated from absorbing set errors. The combinatorial definition of absorbing set only depends on the structure of the Tanner graph, and therefore the relevant absorbing sets can be systematically enumerated [22, 23]. This exact enumeration of the absorbing sets under iterative decoding can be viewed as being equivalent to identifying the weight enumerator polynomial under maximum likelihood decoding. As such, the absorbing sets of the smallest weight rather than smallest distance codewords determine the performance in the error floor region. In particular, the count of relevant absorbing sets is a key component in developing accurate error floor predictions using importance sampling [24].

Trapping sets are defined in [15] and [52] as any length-n bit vector denoted by a pair (a, b), where a is the Hamming weight of the bit vector and b is the number of unsatisfied checks. An absorbing set could be understood as a special type of such trapping set where each variable node is connected to strictly more satisfied than unsatisfied checks.

The satisfied versus unsatisfied notion in the absorbing set definition explains how a fully absorbing set is stable under bit-flipping operations; the implication on practical decoder designs is significant.

Another related structure is an (a, b) elementary trapping set [15, 52], which is defined as a trapping set for which all check nodes in the induced subgraph have either degree one or two, and there are exactly b degree-one check nodes. Here again, the primary contrast with absorbing sets is the stability of absorbing sets under bit-flipping operations, implied by their definition. The notion of absorbing set can also be refined further by imposing restrictions on vertex and check degree profiles, as done, for instance, in Section 3.4.

The definition of absorbing set uses the bit-flipping decoding algorithm, which is a hard-decision decoding algorithm. The analysis of a soft decoding algorithm is more involved as the variable node decision is not simply based on comparing number of satisfied versus unsatisfied messages and the prior information also plays a role. The finite wordlength quantization further complicates the problem. However, the following empirical observations are made, which will be elaborated in the following sections.

- When the variable nodes in the absorbing set are initialized with very noisy inputs as prior likelihoods, even a floating-point decoder would be absorbed. Therefore an absorbing set can be stable under an ideal soft message-passing decoding, and it is not simply an implementation-induced phenomenon.

- Finite-wordlength soft decoders are non-ideal soft decoders due to the clipping and quantization effects. Messages tend to saturate after a few iterations (especially at high SNR levels), which causes the degeneration of these soft decoders. In the extreme case when all the soft messages are saturated to a fixed level, a finite-wordlength soft decoder becomes a hard-decision decoder and absorbing errors can happen more easily.

3.3 Reed-Solomon-based LDPC Code

The Reed-Solomon based LDPC codes (RS-LDPC) [20] are regular, structured LDPC codes, with the girth being at least 6. A (γ, ρ)-regular RS-LDPC code is constructed by γ cosets (with each symbol mapped to a binary symbol location matrix) of a one-dimensional subcode that is itself based on a codeword of a $(\rho, 2, \rho - 1)$ shortened Reed-Solomon code of length ρ, dimension 2, and minimum distance $\rho-1$. The minimum distance of this code is at least $\gamma + 1$ for odd γ and $\gamma + 2$ for even γ. The detailed code construction and properties can be found in [20].

A $(6, 32)$-regular $(2048, 1723)$ Reed-Solomon-based LDPC code is being investigated by feedback simulation for the possible link between error events and the wordlength and quantization of the decoder implementation. The error profile of the baseline design is first characterized. The Matlab reference model of the decoder is then improved by tuning the wordlength and quantization. The improved design is subsequently mapped to FPGA for fast emulation and a complete new set of error profiles can be obtained for verification. The iterative investigation loop is illustrated in Fig. 3.6. This approach allows the errors to be classified and the design to be improved based on each class of error events.

The RS-LDPC decoder is implemented using wordlengths of 5, 6, 7 bits, following Q3.2, Q3.3, Q4.2, and Q5.2 uniform quantization schemes. The FER and BER versus

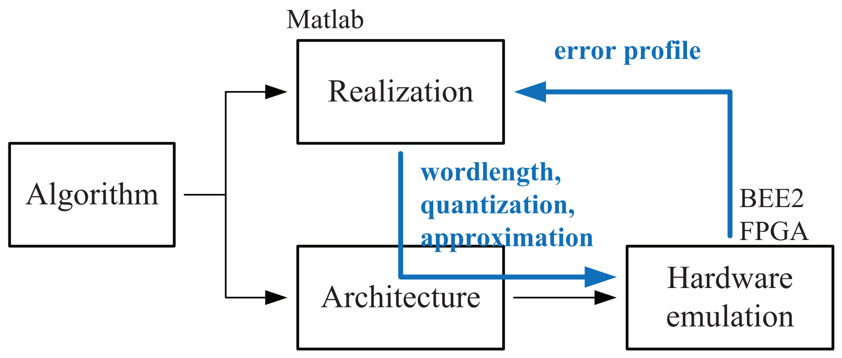


Figure 3.6: Iterative improvement cycle by hardware emulation and feedback simulation.

SNR curves are shown in Fig. 3.7. In all the following experiments, an all-zeros codeword is transmitted and the sum-product algorithm is employed to decode the codeword. The final 3 iterations are recorded when the decoder fails to converge to a codeword after 200 iterations. Absorbing errors are observed in cases when the decoder fails to converge and the hard decisions of all bits remain the same for the final iterations; and oscillation errors are observed in cases when the decoder fails to converge and the hard decisions of some bits fluctuate with each iteration. The statistics of the error events are listed in Table 3.2 for comparison.

3.3.1 Wordlength and Quantization Effects

In the 5-bit Q3.2 fixed-point implementation, most of the errors in the error floor region display an oscillatory behavior and a small number of absorbing errors caused by $(8, 8)$ fully absorbing sets. The oscillatory behavior can be attributed to the dynamics of the message exchange in which a small number of bits propagate incorrect messages through their neighboring unsatisfied checks. These in turn make some of their other neighboring

Table 3.2: Error statistics of (2048,1723) decoder implementations using 200 iterations.

SNR (dB)	Errors	5-bit (Q3.2)	6-bit (Q3.3)	6-bit (Q4.2)	7-bit (Q5.2)
5.2	Errors collected[1]	142	125	94	46
	(8,8) absorbing sets	18	117	92	45
	Oscillations	116	6	0	0
5.4	Errors collected[1]	56	49	44	40
	(8,8) absorbing sets	8	40	42	37
	Oscillations	47	8	0	0
5.6	Errors collected[1]	51	42	22	33
	(8,8) absorbing sets	8	27	20	30
	Oscillations	41	12	0	0
5.8	Errors collected[1]	52	27	14	20
	(8,8) absorbing sets	6	18	13	16
	Oscillations	44	8	0	0

[1] The total number of frames is not uniform for different SNR levels and quantization choices – more input frames were emulated for higher SNR levels and longer-wordlength quantizations. The number of errors collected is divided by the total number of frames to produce the FER plots in Fig. 3.7.

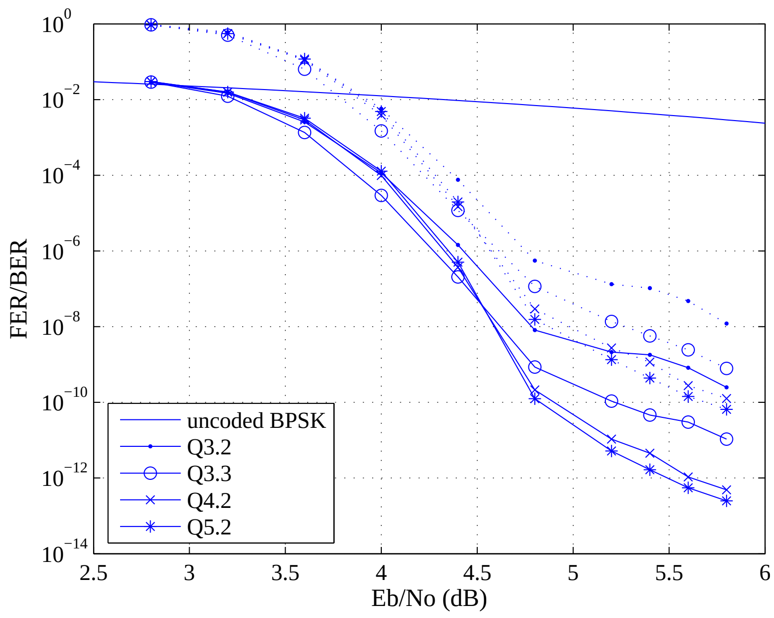


Figure 3.7: FER (dotted lines) and BER (solid lines) performance of the (2048,1723) RS-LDPC sum-product decoder with Q3.2, Q3.3, Q4.2, and Q5.2 fixed-point quantization using 200 iterations.

bits admit incorrect values, which are propagated further to more bits. As the number of incorrect bits increases, so do their neighboring checks, which means that after about two steps there is a sufficient number of unsatisfied checks to enforce the correct values. As a result, the total number of incorrect bits decreases again.

Using the 5-bit Q3.2 uniform quantization, reliable (large-valued) prior LLRs outside the range $[-4, 3.75]$ are clipped, causing underestimation. Variable nodes with underestimated prior LLRs become vulnerable to influence from extrinsic messages. The situation is aggravated by limited resolution (two fractional bits for a resolution of 0.25). As seen in Fig. 3.8 for a Q3.2 quantization of the Φ_1 function, any input $x \geq 3$ produces an output of 0. In the identical Φ_2 function that follows, an input of 0 produces an output of 3.75. The two back-to-back Φ functions cause saturation (overestimation) of extrinsic messages

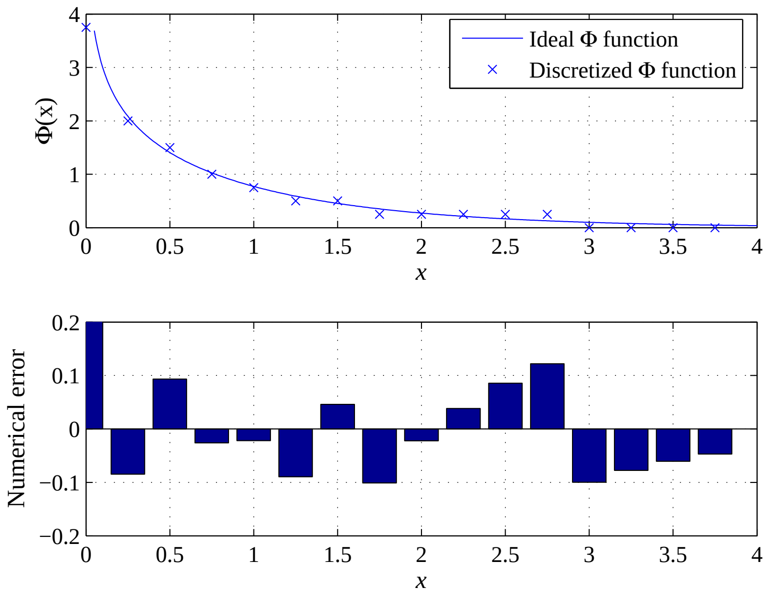


Figure 3.8: Discretization of the Φ function using a Q3.2 uniform quantization and the resulting numerical errors.

in $[3, 3.5]$ and clipping (underestimation) of extrinsic messages greater than 4.0.

At high SNR levels where the majority of the prior LLRs are received correctly with very high reliability, a short wordlength causes excessive clipping of the reliable prior LLRs. And due to the saturation and clipping effects of the two log-tanh functions, both reliable and some less reliable messages are clipped or saturated to a fixed level. Even some weakly incorrect extrinsic messages are capable of exerting the same magnitude of influence as very strongly correct extrinsic messages and priors, which necessarily encourages error propagation and the oscillation errors become more likely.

A 6-bit wordlength allows one more bit for quantization. The extra bit can be allocated either to resolution or range increase. An increased resolution reduces the over-estimation error of less reliable extrinsic messages and limits error propagation. This is

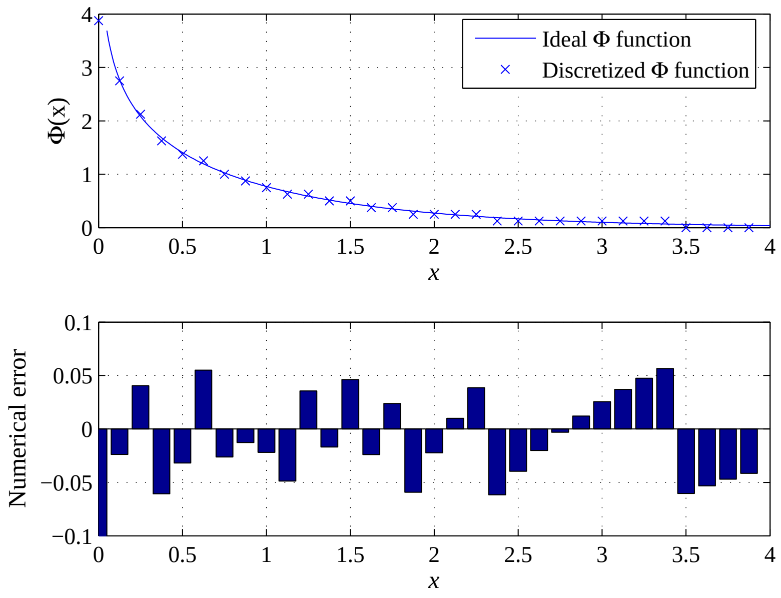


Figure 3.9: Discretization of the Φ function using a Q3.3 uniform quantization and the resulting numerical errors.

demonstrated by the Q3.3 quantization shown in Fig. 3.9. The majority of the errors in the Q3.3 decoder are due to $(8, 8)$ fully absorbing sets and only a small number of errors are due to oscillations. Alternatively, the extra bit can be allocated for range, as in a Q4.2 implementation. A higher range allows reliable prior LLRs to obtain stronger representations, thus stabilizing the respective variable nodes to prevent oscillations.

The 7-bit Q5.2 implementation further improves the error floor performance. All errors collected in Q4.2 and Q5.2 implementations are absorbing errors, and the overwhelming majority of which exhibit the $(8, 8)$ absorbing set structure.

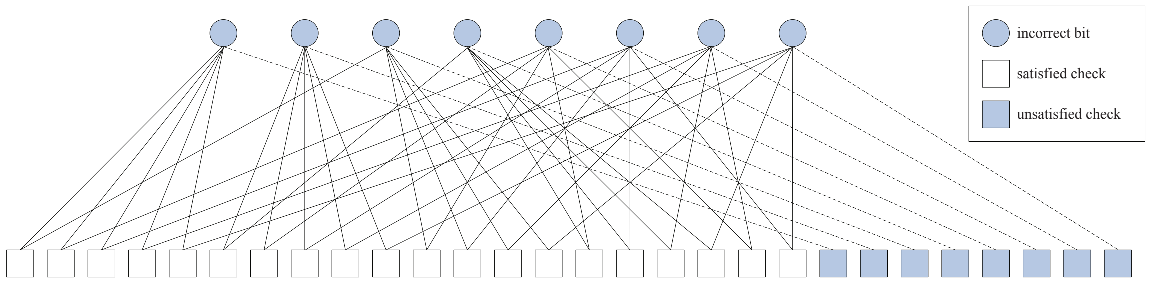


Figure 3.10: Illustration of the subgraph induced by the incorrect bits in an (8,8) fully absorbing set.

3.3.2 Absorbing Set Characterization

As previously discussed, almost all encountered absorbing set errors are of $(8, 8)$ type, all of which are fully absorbing. They share the same structure in which these eight variable nodes participate in a total of twenty-eight checks. Of these, twenty checks are connected with degree-two to the eight variable nodes. Since the girth of the code is at least six [20], these variable node pairs are all different. The remaining eight checks are each connected to a different variable node in the absorbing set. The illustration of such configuration is provided in Fig. 3.10. Although only a subgraph is drawn, all the $(8, 8)$ sets are indeed fully absorbing sets. The validity of the $(8, 8)$ absorbing error is also verified experimentally by simulating a floating-point decoder for channel realizations with very noisy inputs in precisely eight bits that constitute an absorbing set, and observing that even the floating-point decoder cannot successfully decode such realizations.

Even though this special $(8, 8)$ configuration is intrinsic to the code, and hence implementation-independent, its effect on BER is highly implementation-dependent. In particular, when the wordlength is finite, the effect of the absorbing sets can be exacerbated. This effect is demonstrated in the difference between the performance of the Q4.2 and Q5.2

decoders in the error floor region, whereby in the former case the number of absorbing set failures is higher, leading to a relatively higher error floor.

3.3.3 Finite Number of Decoding Iterations

The number of decoding iterations is usually limited in practice, as it determines the latency and throughput of the system. In the practical high-throughput implementations, the maximum number of iterations for the LDPC decoder is limited to less than ten.

Fig. 3.1 shows that a good performance in the waterfall region can be achieved with as few as ten iterations. The loss in performance in the waterfall region is due to an insufficient number of iterations for the decoding to converge. The ten-iteration BER curve eventually overlaps with the 200-iteration in the error floor region. Analysis of the failures in this region confirms that the $(8, 8)$ fully absorbing set, the dominant cause of error floors in the 200-iteration decoder, causes the ten-iteration decoder to fail as well. This result suggests that in the high SNR region, the absorbing process usually happens very quickly and the absorbing structure emerges in full strength within a small number of decoding iterations. Non-convergent errors, however, become negligible in the error floor region.

3.4 Array-based LDPC code

Array-based LDPC codes [26] are regular LDPC codes parameterized by a pair of integers (p, γ), where $\gamma \leq p$, p is an odd prime. The $\mathbf{H}$ matrix ($\mathbf{H}_{p,\gamma}$) is given by

$$\mathbf{H}_{p,\gamma} = \begin{bmatrix} I & I & I & \cdots & I \\ I & \sigma & \sigma^2 & \cdots & \sigma^{p-1} \\ I & \sigma^2 & \sigma^4 & \cdots & \sigma^{2(p-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I & \sigma^{\gamma-1} & \sigma^{(\gamma-1)2} & \cdots & \sigma^{(\gamma-1)(p-1)} \end{bmatrix}$$

where σ denotes a $p \times p$ permutation matrix of the form

$$\sigma = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

To demonstrate the applicability of hardware emulation approach in identifying potentially important results, the finite-wordlength decoders of a $(5, 47)$-regular $(2209, 1978)$ array-based LDPC code is studied [80]. The class of array-based LDPC codes is known to perform well under iterative decoding [26]. The **H** matrix of this code can be partitioned into 5 row groups and 47 column groups of 47×47 permutation submatrices. Note that the regular structure of the **H** matrix is well suited for the emulation platform. The following experiments are performed with the wordlength fixed to 6 bits. Unless specified otherwise, a maximum of 200 decoding iterations is allowed so as to isolate the quantization effect from the iteration number effect. Using a Q4.2 quantization in a sum-product decoder yields the results shown in Fig. 3.11.

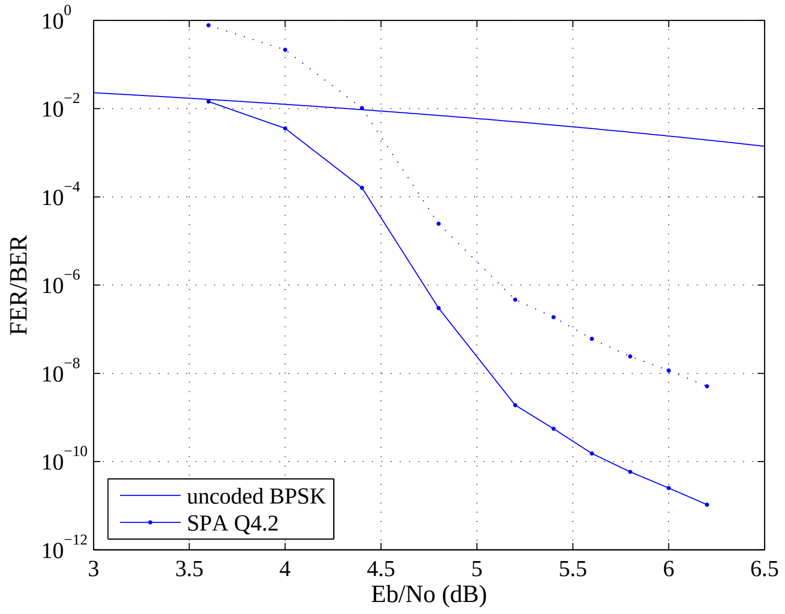


Figure 3.11: FER (dotted lines) and BER (solid lines) performance of a (2209,1978) array-based LDPC code using 200 decoding iterations.

Based on our emulation results, the failures in the error floor region are entirely due to absorbing sets. The statistics of the frequently observed absorbing sets are listed in Table 3.3. The structure of the dominant $(4, 8)$ absorbing set is illustrated in Fig. 3.12. To facilitate further discussions, the notation $(p : q)$ is introduced to describe the connectivity of a variable node with p connections to satisfied check nodes and q connections to unsatisfied check nodes. In the $(4, 8)$ absorbing set, each variable node in the absorbing set has a $(3 : 2)$ connection. All the other absorbing sets listed in Table 3.3 contain variable nodes with $(4 : 1)$ and $(5 : 0)$ connections.

Variable node decisions are based on both the extrinsic information and the prior information, as in equation (2.6). Numerical representations of extrinsic and prior information affect the dynamics of the message-passing algorithm. Two types of tradeoffs can be

Table 3.3: Absorbing set profile of (2209,1978) Q4.2 sum-product decoder implementations.

Sat level[1]	SNR (dB)	Errors collected[2]	(4,8)	(5,9)	(6,8)	(7,9)	(8,6)	(8,8)	(9,5)
3.5	5.4	185	50	22	34	17	9	13	2
	5.6	121	39	12	36	9	8	4	
	5.8	104	50	15	11	6		1	
	6.0	50	32	5	5	4			
4.0	5.6	247	65	12	56	22	24	20	4
	5.8	191	83	20	35	17	6	3	2
	6.0	91	68	4	12	2	3		
	6.2	36	25	4	3		3	1	
4.5	5.8	111	38	3	30	7	8	4	4
	6.0	70	38		12	5	6	1	2
	6.2	24	19		2	1		1	

[1] Saturation level of the Φ_2 function.

[2] The total number of frames is not uniform for different SNR levels, quantization, and algorithm choices. The number of errors collected is divided by the total number of frames to produce the FER plots in Fig. 3.13.

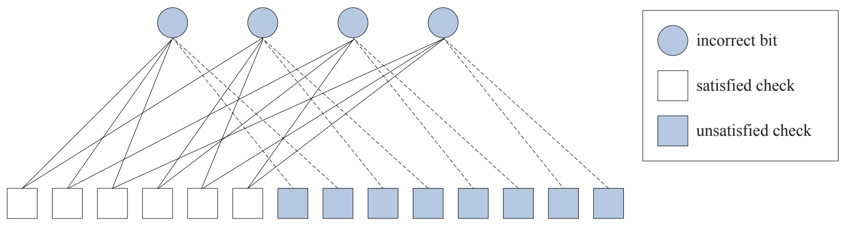


Figure 3.12: Illustration of the (4,8) absorbing set.

made, how to weigh extrinsic information versus prior information, and how to differentiate extrinsic messages among themselves. The following experiments are performed to explore these tradeoffs while keeping the wordlength constant to maintain the implementation complexity.

3.4.1 Strength of Extrinsic Messages

The clipping level of extrinsic messages determines the strength of the most reliable extrinsic messages. Adjusting the clipping level of extrinsic messages affects the weight of extrinsic information compared to the prior information – a lower clipping level underweights the extrinsic information and vice versa.

In a sum-product decoder, extrinsic messages are simply the outputs of the Φ_2 function. Moving the saturation level of the Φ_2 function (i.e., $\Phi_2(0)$) adjusts the strength of the most confident extrinsic messages. Fig. 3.13 shows that stronger extrinsic messages lower the error floor but worsen the performance in the waterfall region.

This observation can be intuitively explained as strong extrinsic messages permitting stronger extrinsic effect on the variable node decisions. The extrinsic effect can be favorable – the correct bits exert strong influences on the incorrectly-received bits, and the effect can be unfavorable – the incorrectly-received bits exert strong influences on the correct bits. The SNR level determines whether the favorable influences overpower the adverse influences or the opposite. In a low-SNR waterfall region, more bits are received incorrectly. A large number of strong adverse influences tend to encourage excessive error propagation, which prevents the convergence. At a high SNR level, very few bits are received incorrectly. Strong extrinsic messages allow the favorable influences to significantly outnumber

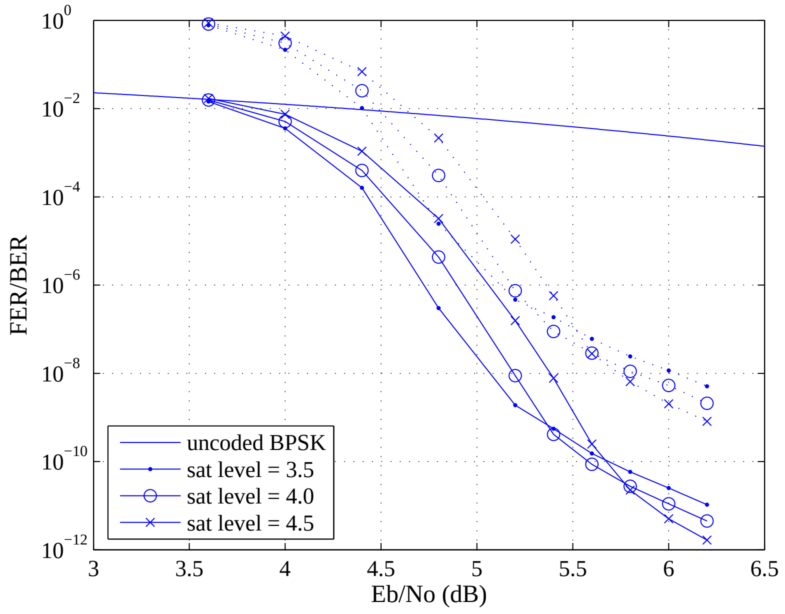


Figure 3.13: The effect of adjusting the strength of extrinsic messages in a Q4.2 uniform quantized sum-product decoder implementation using 200 decoding iterations.

and overpower the adverse influences, which makes it more difficult to enter an absorbing state.

The above describes the average behavior of a message-passing decoder. An absorbing set is a special configuration at the high SNR level where seemingly satisfied checks gather enough adverse influences that outnumber the favorable influences, therefore strengthening extrinsic messages uniformly is not likely to change an absorbing configuration. This conjecture is verified by strengthening the extrinsic messages and observe the failure cases in the error floor region. Partial lists of the absorbing sets are shown in Table 3.3. The $(4, 8)$ absorbing set remains the dominant cause of error floors when the extrinsic messages are strengthened.

Increasing the weight of extrinsic messages slows down the convergence speed, as

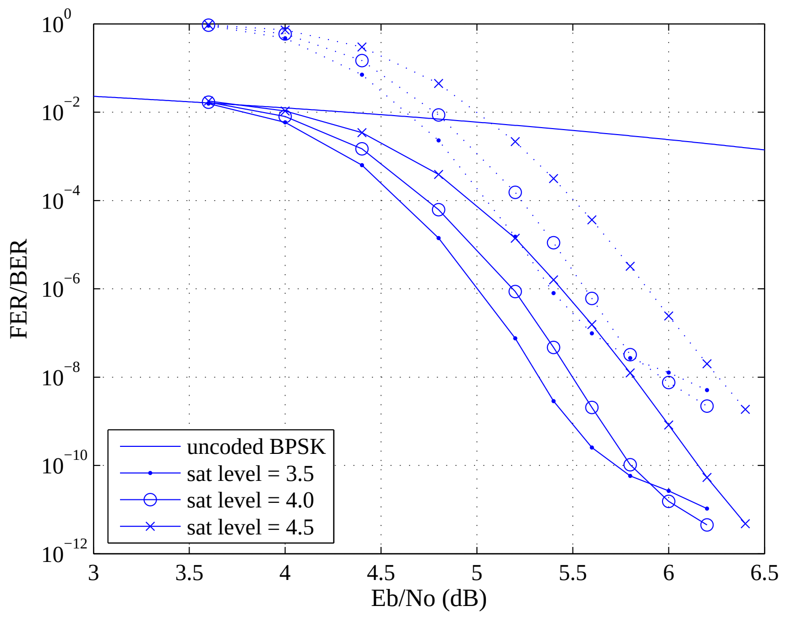


Figure 3.14: The effect of adjusting the strength of extrinsic messages in a Q4.2 uniform quantized sum-product decoder implementation using 10 decoding iterations.

evidenced in Fig. 3.14. If very few decoding iterations are permitted, the performance gap between various decoders appears to be more significant in the waterfall region.

3.4.2 Differentiation among Extrinsic Messages

As the decoder starts to converge, the variable-to-check messages usually grow larger, as their certainty increases. In this regime, the sum-product decoder is essentially operating on the lower right corner of the Φ_1 curve and subsequently on the upper left corner of the Φ_2 curve as highlighted in Fig. 3.15. These corners are referred to as the operating regions of the Φ_1 and Φ_2 functions. A more accurate representation of extrinsic messages requires more output levels of the Φ_2 function in its operating region, which also necessitates high-resolution inputs to the Φ_2 function. These requirements can be both satisfied if the

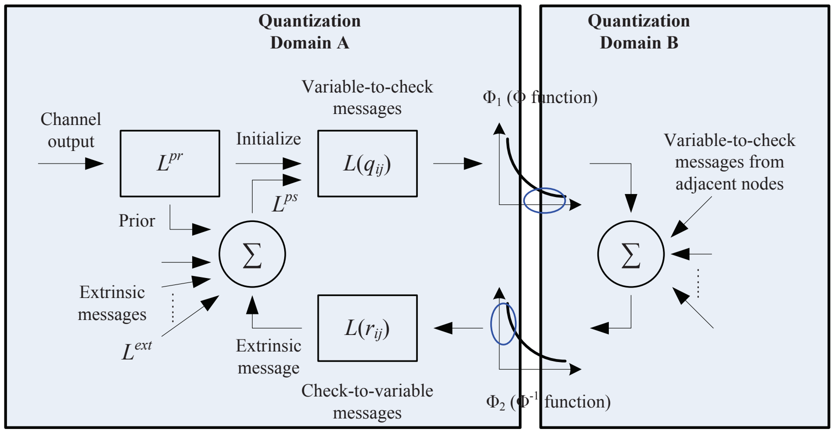
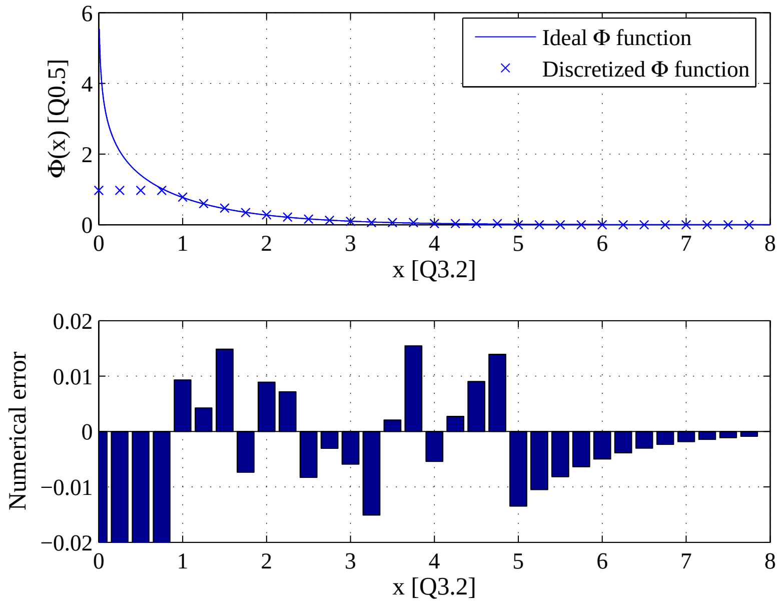


Figure 3.15: A sum-product decoder with two quantization domains (the operating regions of Φ_1 and Φ_2 functions are circled).

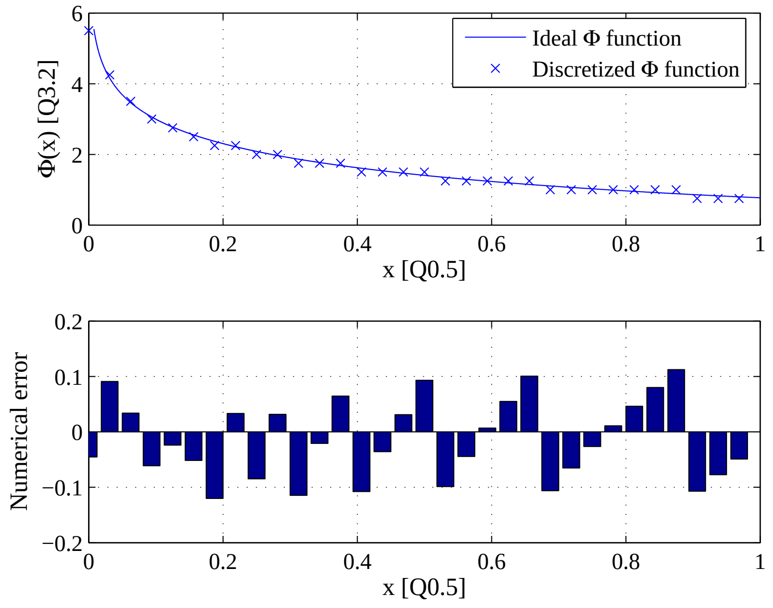
quantization scheme is designed to have two quantization domains illustrated in Fig. 3.15. For instance, suppose that Domain A uses a Q4.2 quantization whereas Domain B uses a quantization with a higher resolution, such as a Q1.5 quantization. The 6-bit wordlength is preserved to maintain the same implementation complexity. The functions Φ_1 and Φ_2 separate the two domains. The input to Φ_1 is in a Q3.2 quantization and the output of Φ_1 is in a Q0.5 quantization. The Φ_2 function assumes the opposite quantization assignment. This scheme is referred to as dual quantization, since the quantization levels are tailored to the operating region within each domain. There is no increase in hardware complexity for implementing this scheme.

In a Q4.2/1.5 dual quantization scheme, the discretization of two Φ functions are shown in Fig. 3.16a and 3.16b. Note that the numerical error incurred is small in the operating regions of the Φ_1 function (lower right corner) and the Φ_2 function (upper left corner).

Fig. 3.17 shows that the Q4.2/1.5 dual quantization results in better performance



(a) Discretization of Φ_1 function



(b) Discretization of Φ_2 function

Figure 3.16: Discretization of log-tanh functions.

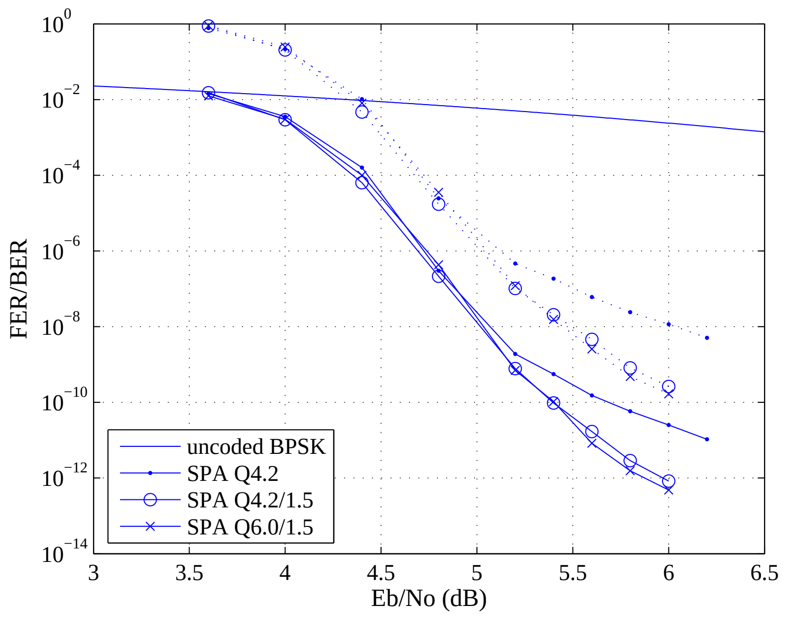
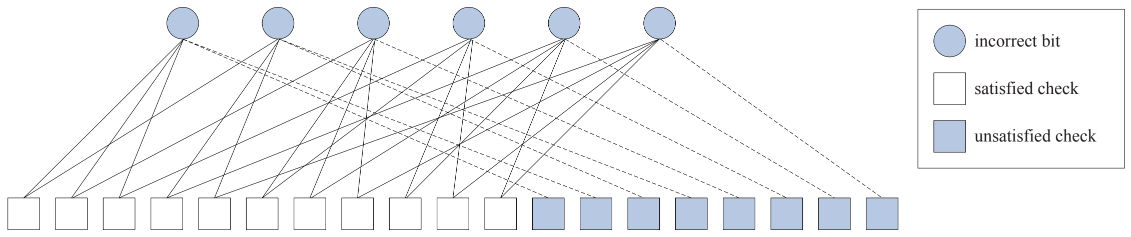


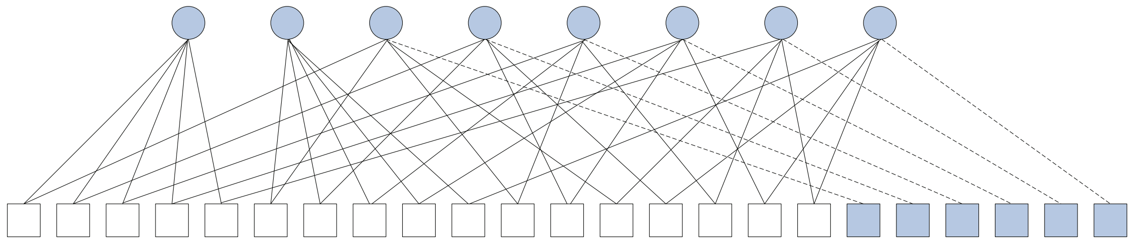
Figure 3.17: FER (dotted lines) and BER (solid lines) performance of a (2209,1978) array-based LDPC code using 200 decoding iterations.

than the Q4.2 quantization in both the waterfall and the error floor regions. The performance advantage of the Q4.2/1.5 dual quantization is attributed to more levels in the operating regions of the Φ_1 and Φ_2 functions, which enable a more accurate representation of the extrinsic messages. Reliable extrinsic messages could potentially obtain a stronger representation than the less reliable extrinsic messages, so that the error propagation is limited and the absorbing set errors become less likely.

The $(4, 8)$ and $(5, 9)$ absorbing sets, observed in the Q4.2 quantization, are much less frequent when decoding using the dual quantization scheme, and the error floor is now dominated by $(6, 8)$ and $(8, 6)$ absorbing sets as shown in Table 3.4. All of the collected $(6, 8)$ and $(8, 6)$ sets are fully absorbing, with configurations illustrated in Fig. 3.18a and 3.18b. The $(6, 8)$ absorbing set consists of two variable nodes with $(3 : 2)$ connections and four



(a)



(b)

Figure 3.18: Illustration of (a) the (6,8) absorbing set, and (b) the (8,6) absorbing set.

variable nodes with $(4:1)$ connections. The $(8,6)$ absorbing set consists of only variable nodes with $(4:1)$ and $(5:0)$ connections. Both the $(4:1)$ and the $(5:0)$ configurations are more stable as absorbing sets than the $(3:2)$ configuration, for which reason the $(6,8)$ and $(8,6)$ absorbing sets are considered stronger than the $(4,8)$ absorbing set.

3.4.3 Representation of Channel Likelihoods

For practical SNR levels, a Q4.2 quantization scheme does not offer enough range to capture the input signal distribution. Moreover, it clips correct priors and incorrect priors disproportionately. By selecting a Q6.0 quantization in Domain A, an increased input range is accepted, which permits correct priors to assume stronger values without being clipped excessively. Variable nodes backed by stronger correct priors cannot be easily attracted to

Table 3.4: Absorbing set profile of (2209,1978) decoder implementations.

Algorithm & Quantization	SNR (dB)	Errors collected	(4,8)	(5,9)	(6,8)	(7,9)	(8,6)	(8,8)	(9,5)	(10,4)	(10,6)
SPA Q4.2 (sat level = 3.5)	5.4	185	50	22	34	17	9	13	2		
	5.6	121	39	12	36	9	8	4			
	5.8	104	50	15	11	6		1			
	6.0	50	32	5	5	4					
SPA Q4.2/1.5	5.4	149			16	3	57	9	17	4	3
	5.6	87			21	5	33	8	7		2
	5.8	42	1		6	2	15	8	2	2	2
	6.0	21	2		8		7	2		1	
SPA Q6.0/1.5	5.4	133	1		28	7	16	12	3	1	1
	5.6	66		1	29	5	12	12			
	5.8	38			17	2	7	6	1	1	
	6.0	13			9	2	1				
ASPA Q4.2	5.6	221			2		91	5	36	14	7
	5.8	59				1	30	1	13	3	
	6.0	22					15	1	3	1	
ASPA β=1 Q4.2	5.4	307			6	2	143	17	38	16	12
	5.6	243			6	2	122	13	40	16	9
	5.8	58			1		35	1	8	4	2
	6.0	18			2		9	3	2		1

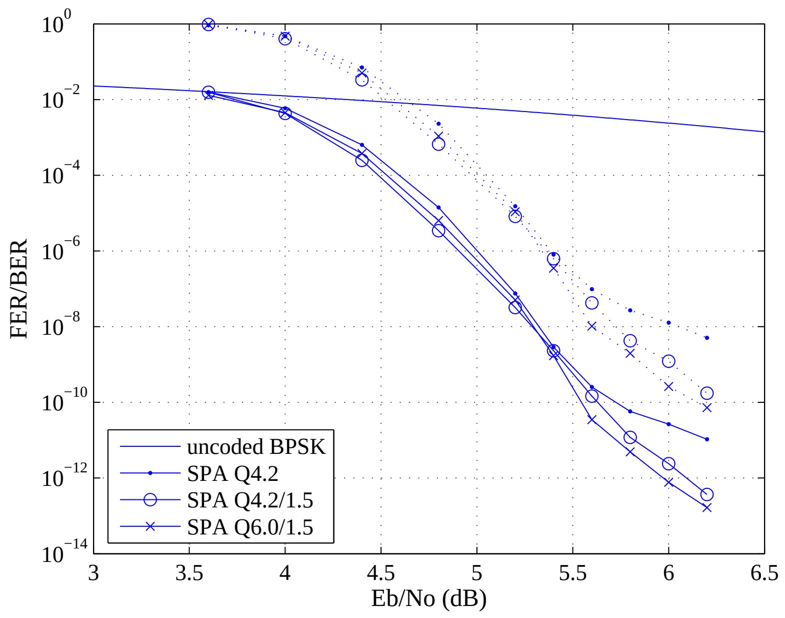


Figure 3.19: FER (dotted lines) and BER (solid lines) performance of a (2209,1978) array-based LDPC code using 10 decoding iterations.

an absorbing set, thus the probability of absorbing set errors is reduced. Statistics in Table 3.4 show that the $(6, 8)$ and $(8, 6)$ sets remain to be dominant. The error floor performance of the Q6.0/1.5 dually-quantized decoder improves by at least a factor of two over the Q4.2/1.5 performance.

The dual quantization scheme allows the extrinsic messages and prior messages to differentiate among themselves by assuming more accurate representations. Compared to the previous approach of uniformly increasing the weight of extrinsic messages versus the weight of prior message, the dual quantization scheme achieves a better performance without sacrificing convergence speed. Fig. 3.19 shows that the dually-quantized decoders perform better in both the waterfall region and the error floor region in as few as 10 decoding iterations.

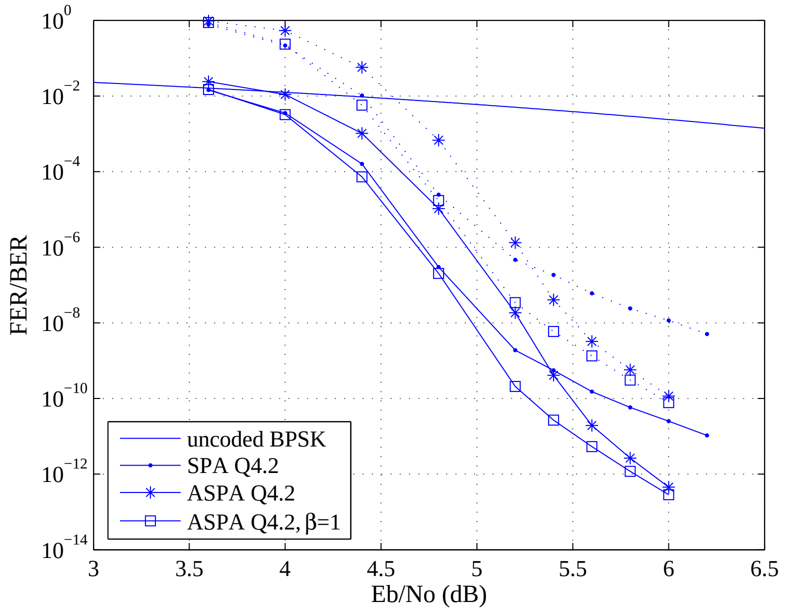


Figure 3.20: FER (dotted lines) and BER (solid lines) performance of ASPA decoders of (2209,1978) array-based LDPC code using 200 decoding iterations.

3.4.4 Approximate Sum-product Decoding

By using the approximate sum-product algorithm (2.8) to bypass Φ_1, summation, and Φ_2 altogether, saturation and quantization errors incurred in the log-tanh processing are eliminated. The Q4.2 sum-product decoder of the $(2209, 1978)$ array-based LDPC code is simplified using the approximation (2.8). The performance of the Q4.2 ASPA decoder is illustrated along with its sum-product counterpart in Fig. 3.20. In the waterfall region, the ASPA decoder incurs nearly 0.2 dB of performance loss due to overestimation errors; however, it performs better in the error floor region. The error floor is dominated by $(8, 6)$ and $(9, 5)$ fully absorbing sets, which both consist of only variable nodes with $(4 : 1)$ and $(5 : 0)$ connections. Lower-weight weak absorbing sets $(4, 8)$ and $(5, 9)$ are eliminated and even instances of $(6, 8)$ and $(7, 9)$ absorbing sets are reduced, as evidenced in Table 3.4.

The lackluster error floor performance of a conventional sum-product decoder compared to an ASPA decoder is largely due to the estimation of the two log-tanh functions. As in the case of the oscillatory behavior, a finite-wordlength quantization of the log-tanh functions causes underestimations of reliable messages and overestimations of unreliable messages. As a result, the reliability information is essentially lost, and soft decoding degenerates to a type of hard-decision decoding where the decisions are based entirely on majority counting. Such a decoding algorithm is susceptible to weak absorbing sets because it disregards the reliability information. In contrast, the approximate sum-product algorithm is better in maintaining the reliability information, so that it is not easily attracted to weak absorbing sets.

The ASPA decoder can be improved using a correction term [9]. An offset of $\beta = 1$ is selected to optimize the decoder performance. Such a decoder is implemented as in Fig. 3.21. The performance of the offset-corrected decoder is illustrated in Fig. 3.20, where both the waterfall and the error floor performance are improved. The absorbing set profile shows that the $(8, 6)$ and $(9, 5)$ fully absorbing sets determine the error floor.

With reduced iteration count, the ASPA decoder incurs almost 0.5 dB of performance loss. However, the loss can be easily compensated after applying the offset correction to reduce the overestimation of extrinsic messages. In ten iterations, the performance of the offset-corrected ASPA decoder easily surpasses the sum-product decoder as shown in Fig. 3.22.

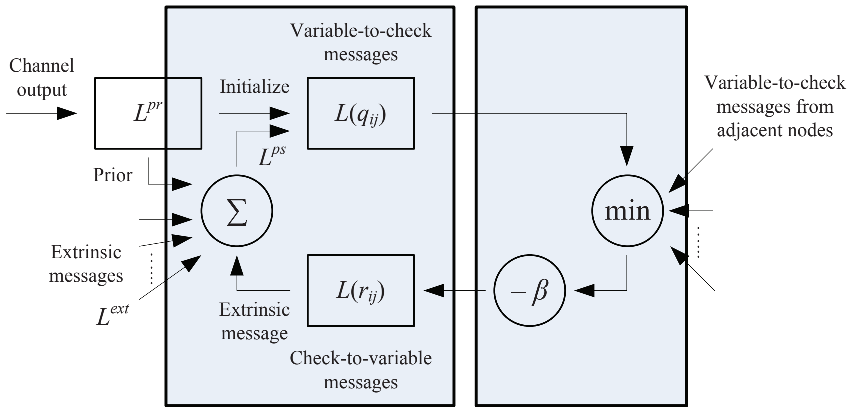


Figure 3.21: An ASPA decoder with offset correction.

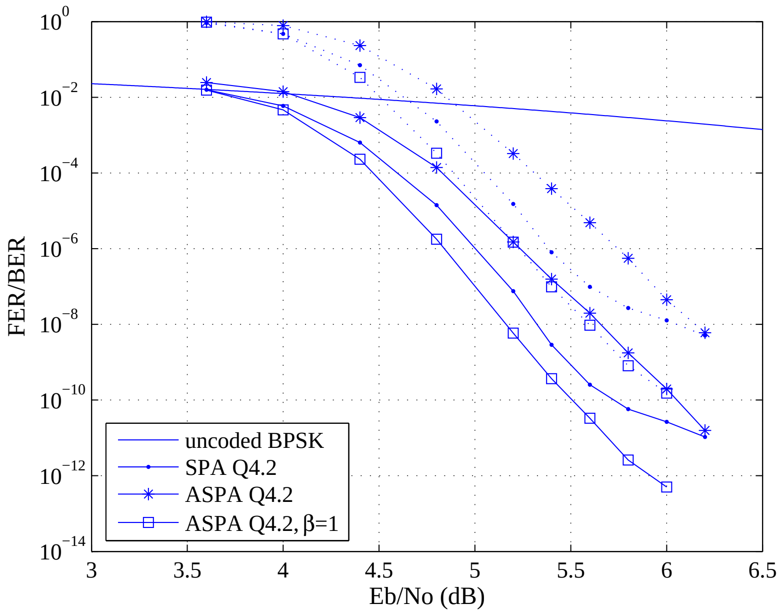


Figure 3.22: FER (dotted lines) and BER (solid lines) performance of ASPA decoders of (2209,1978) array-based LDPC code using 10 decoding iterations.

3.4.5 Dominant Absorbing Sets

In previous discussions of the $(2209, 1978)$ array-based LDPC code, the configurations of $(4, 8)$, $(6, 8)$, $(8, 6)$, and $(9, 5)$ fully absorbing sets have been described. Two simple ways to characterize these sets are by weight and by stability. Everything else being equal, low-weight absorbing sets appear much more frequently when decoding fails. This phenomenon is more pronounced in higher SNR levels. The stability of an absorbing set is related to the structure of the set and the connectivity of the factor graph. In the $(2209, 1978)$ array-based LDPC code, the $(8, 6)$ and $(9, 5)$ absorbing sets are stronger or more stable, as it is more difficult to escape such absorbing configurations. In general, the ratio a/b provides clues to how stable an (a, b) absorbing set is – the higher the a/b ratio, the more stable the (a, b) absorbing set. Low-weight absorbing sets and strong absorbing sets are of greater importance because they dominate the error floors.

In suboptimal decoder implementations where severe message saturations can occur, such as the Q4.2 sum-product implementation, the performance is dictated by low-weight weak absorbing sets, which lead to an elevated error floor. The implementations can be improved using the dual quantization or the approximate sum-product algorithm to reduce the adverse effects of message saturation and quantization. However, the underlying message-passing algorithm is a local algorithm when operating on graphs containing cycles. Despite a lower error floor achieved with these improved approaches, the floor still remains and it is eventually defined by the strong absorbing sets.

Chapter 4

Reweighted Message Passing

Similar to the array-based LDPC code, the low error rate performance of the RS-LDPC code can be improved using the ASPA decoder with offset correction. The performance of the offset ASPA decoder is superior to the SPA decoder of the same wordlength, as shown in Fig. 4.1. Despite the extra coding gain and lower error rate performance of the offset ASPA decoder, the error floor emerges at a BER level of 10^{-11}, which still renders this implementation unacceptable for 10GBASE-T Ethernet that requires an error-free operation down to the BER level of 10^{-12} [36]. The $(8, 8)$ fully absorbing set discussed in Section 3.3 underpins the error floors of both the SPA decoder and the offset ASPA decoder of the $(2048, 1723)$ RS-LDPC code.

Brute-force performance improvement requires an even longer wordlength, though the performance gain with each additional bit of wordlength diminishes as the wordlength increases over 6 bits. Alternative decoding algorithms have been proposed in literature, such as scaled decoder [14], averaged decoder [14, 41], and reordered decoding schedules [6, 59].

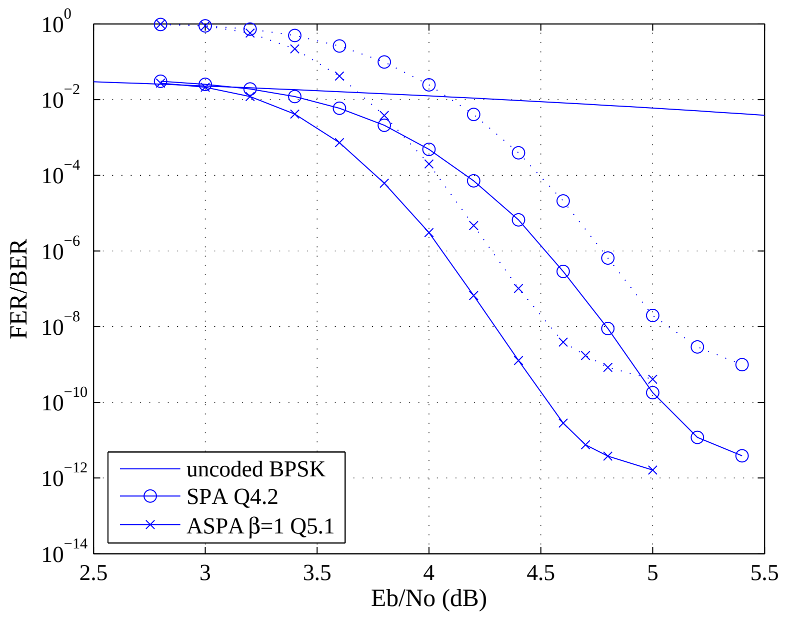


Figure 4.1: FER (dotted lines) and BER (solid lines) performance of a (2048,1723) RS-LDPC code using 20 decoding iterations.

A scaled decoder is a form of the sum-product decoder with check-to-variable messages scaled down by a factor [14]. Empirical evidence shows an improvement of the error floor by an appropriate choice of the scaling factor. An averaged decoder improves the decoder performance at high SNR levels by averaging the messages over multiple iterations to slow down the convergence to a trapping state [41]. The scheme is based on a heuristic indicator of the emergence of error traps as a sudden magnitude change in the values of certain variable messages. Another heuristic approach is an "informed" decoder that processes messages in the order of the largest residuals, defined as the magnitude change of check-to-variable messages between consecutive iterations [6]. This informed schedule accelerates the updates of low-reliability variable nodes and it was conjectured that this schedule would target the variable nodes that belong to the trapping set. Alternatively, the messages can be processed in the order of the most reliable check nodes [59]. The idea is to reinforce the decoder with the reliable data before processing the less reliable ones.

The common drawback of all the above approaches is that they are formulated without regard to the error structure, therefore they are only capable of improving the average behavior of the decoder and the effect on error floor is not significant. Further improvement should rely on adapting the message-passing algorithm to combat the effect due to absorbing sets. In [31], Han and Ryan proposed a bi-mode decoder – in the first mode, the regular sum-product decoding is performed for the decoder to reach the point of failure, then it switches to the second mode for post-processing, which is activated only when the syndrome weight of the error matches the set of syndrome weights of the target trapping sets or oscillating sets. Post-processing is carried out in two steps: 1) starting from the unsatisfied check nodes as roots, the set of the variable node neighbors are flagged as erasures, 2) iterative erasure decoding is performed to resolve the erasures. The bi-mode decoder works remarkably well when the erasure set contains no stopping set [19], such as a regular $(3, 6)$ rate-1/2 $(2640, 1320)$ Margulis code [47,58] and a rate-0.3 $(640, 192)$ QC code designed using the approaches outlined in [34, 63]. However, the erasure set can be large for some codes (due to a large number of unsatisfied checks and a high check node degree), and the bi-mode erasure decoding cannot avoid running into stopping sets. An example is the $(2048, 1723)$ RS-LDPC code under consideration, where the size of the erasure set is 256 with respect to the errors caused by the dominant $(8, 8)$ absorbing set.

The bi-mode erasure decoding algorithm can be improved. An algorithm improvement loop is formulated as shown in Fig. 4.2, relying on the hardware emulation

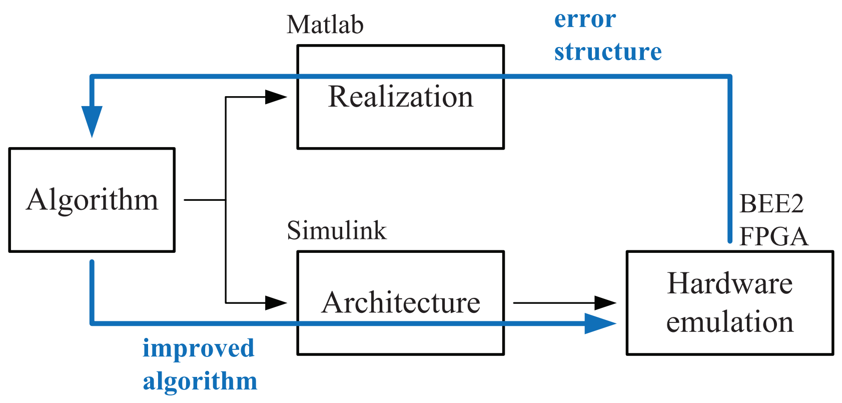
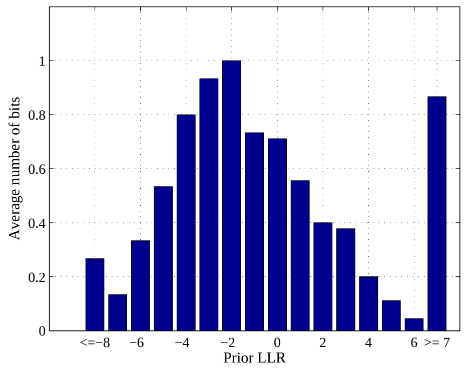


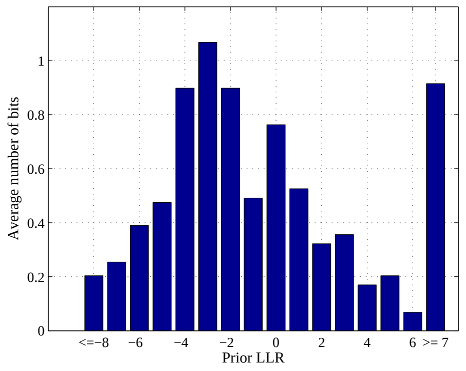
Figure 4.2: Algorithm improvement based on hardware emulation.

infrastructure and feedback simulations. The statistics of the error-inducing channel outputs collected through hardware emulations are of the most interest as they shed light on certain “weaknesses” of the absorbing error mechanism. An improved algorithm is designed to exploit the weaknesses.

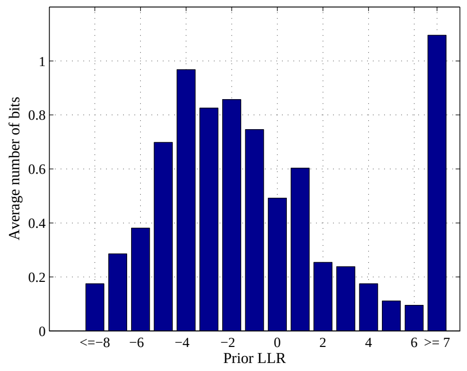
Whenever an $(8, 8)$ absorbing error occurs, the emulation system records the prior LLRs causing the error. Averaged over a large number of errors, the distribution of prior LLRs of the variable nodes that belong to the absorbing set can be captured, as illustrated for four different SNR levels in Fig. 4.3. In these plots, the y-axis shows the average number of bits in the $(8, 8)$ absorbing set that assume each of the prior LLR value displayed on the x-axis. The center lobe of the distribution concentrates in the moderate tail region (LLR values in $[-4, 0]$), confirming that absorbing errors are mostly due to noise moderately out in the tail rather than noise values in the extreme tails.



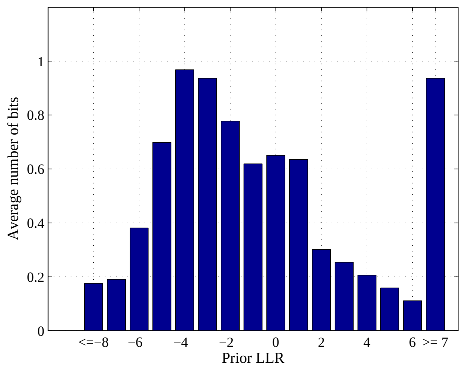
(a) SNR = 4.6 dB (averaged over 45 samples)



(b) SNR = 4.8 dB (averaged over 59 samples)



(c) SNR = 4.9 dB (averaged over 63 samples)



(d) SNR = 5.0 dB (averaged over 63 samples)

Figure 4.3: Prior LLR distribution of the bits that belong to the (8,8) absorbing set. Results are based on a Q4.0 offset ASPA decoder of the (2048,1723) RS-LDPC code.

4.1 Message Biasing

A post-processing method is described below, based on the combinatorial structure of the absorbing set. In the following discussion, it is assumed that the all-zeros codewords are used in transmission. Using the LLR definition in (2.1) and the hard decision rule in (2.7), the prior LLRs are nonnegative if received correctly, and the posterior LLRs are nonnegative if decoded correctly.

For simplicity, a $(3, 3)$ absorbing set is used for illustration shown in Fig. 3.5b, where each bit in the absorbing set is connected to two satisfied checks (these checks are falsely satisfied because their neighboring bits are not all correct) and one unsatisfied check. The message from the unsatisfied check attempts to correct the wrong bit decision, as opposed to the messages from two falsely satisfied checks that reinforce the wrong bit decision.

An intuitive way in which to escape this absorbing state is to perform the following type of post-processing: reduce the reliability of the messages from falsely satisfied checks, and increase the reliability of the message from the unsatisfied check. This selective biasing method alleviates the joint effect of a large number of incorrect messages. As an illustration, consider Fig. 4.4, where variable node $v_7(v_7 \in D)$ is connected to falsely satisfied checks c_4 and c_5, as well as unsatisfied check c_7. We selectively bias messages $L(r_{v7c4})$ and $L(r_{v7c5})$ by reducing their reliabilities and bias the message $L(r_{v7c7})$ by increasing its reliability, so that $L(r_{v7c7})$ reduces (or overcomes) the joint effect of $L(r_{v7c4})$ and $L(r_{v7c5})$, thus reversing the wrong decision at v_7.

Despite the simplicity of this approach, its exact form is not implementable because

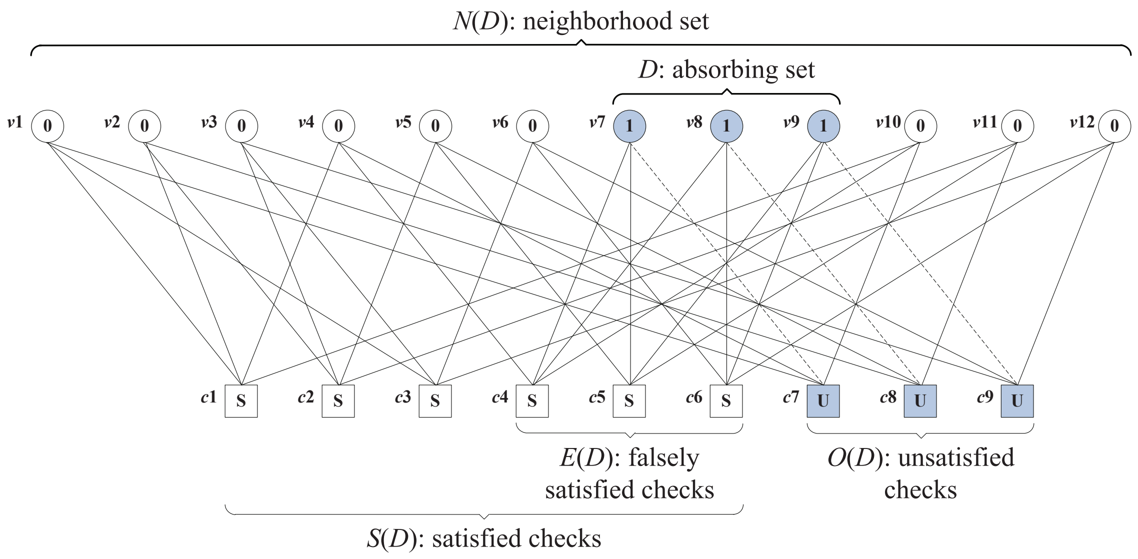


Figure 4.4: Illustration of a (3,3) fully absorbing set with falsely satisfied checks and neighborhood set labeled.

the falsely satisfied checks cannot be separated from the correctly satisfied checks in a message-passing decoding process, and the post-processor needs to be further refined.

4.1.1 Relaxed Selectivity

Following the absorbing set definition in Section 3.1, let $\mathcal{E}(D)$ be the set of neighboring vertices of D in F with even degree with respect to D. Let the neighborhood set $N(D)$ be the set of neighboring variable nodes to the unsatisfied checks in $\mathcal{O}(D)$. Let $\mathcal{S}(D)$ be the set of neighboring satisfied check nodes to the variable nodes in $N(D)$. Then $\mathcal{S}(D)$ is composed of both the set of falsely satisfied checks $\mathcal{E}(D)$ and the set of correctly satisfied checks. The example $(3, 3)$ absorbing set is annotated and shown in Fig. 4.4.

The set of unsatisfied checks $\mathcal{O}(D)$ can be identified in each iteration of message-passing decoding, but no knowledge of the absorbing set D can be inferred besides that D is

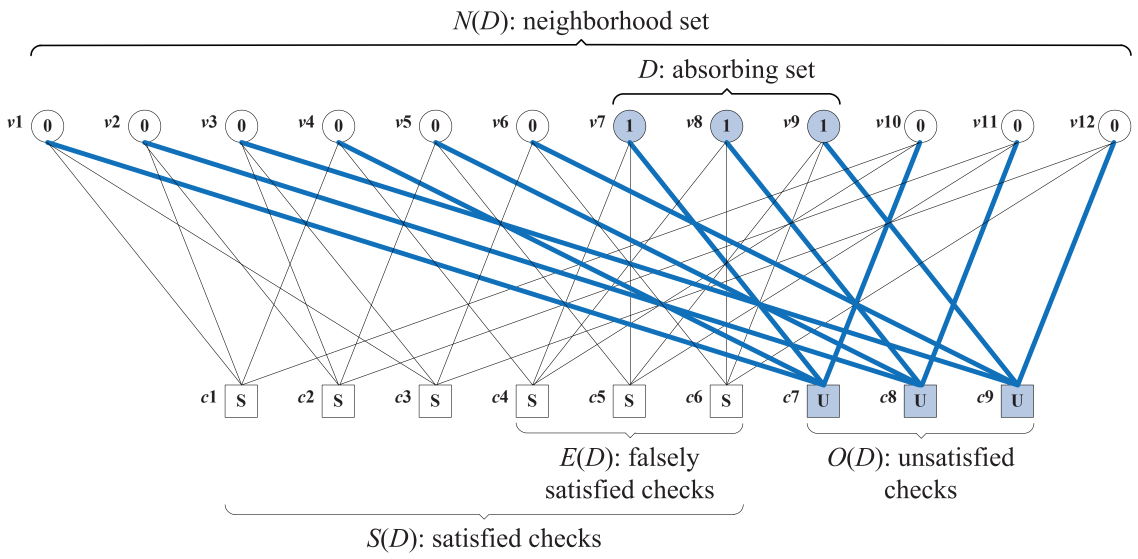


Figure 4.5: Perturbation is introduced by biasing the messages. (Thick blue lines indicate strengthened messages from check nodes to variable nodes and black lines indicate weakened messages from check nodes to variable nodes.)

a subset of $N(D)$. Thus we relax the selectivity and increase the reliabilities of all messages from check nodes in $\mathcal{O}(D)$ to variable nodes in $N(D)$ and decrease the reliabilities of all messages from check nodes in $\mathcal{S}(D)$ to variable nodes in $N(D)$. As a result, in the $(3,3)$ absorbing set illustrated in Fig. 4.4, each bit in D receives two weak messages from falsely satisfied checks and one strong message from the unsatisfied check, easing the absorbing state as expected. However, the relaxed selectivity causes each (correct) bit in $N(D) \setminus D$ to receive one strong message from the unsatisfied check and two weak messages from the satisfied checks as shown in Fig. 4.5. The side effect is undesired, as it can cause the correct bits to reverse their values.

The biasing of the reliabilities needs to be carefully tuned, so that the incorrect bits within the absorbing set can be corrected, while the negative side effects are minimized. Two scenarios are depicted: for a bit in an absorbing set, e.g. $v_7 \in D$, and for a bit in the neighborhood set but outside of the absorbing set, e.g. $v_6 \in N(D) \setminus D$. The posterior LLRs of bits v_7 and v_6 are computed as in (4.1) and (4.2). The "+" and "−" signs indicate strengthened and weakened reliabilities. Assume uniform strengthening and weakening, referring to uniformly increase reliabilities to a fixed level, L_{strong}, in performing strengthening, and uniformly reduce reliabilities to a fixed level, L_{weak}, when performing weakening. Then the sum of extrinsic messages received at v_7 and v_6 are equal in magnitude but opposite in sign.

$$\begin{aligned} L^{ps}(v_7) &= L^{-}(r_{v7c4}) + L^{-}(r_{v7c5}) + L^{+}(r_{v7c7}) + L^{pr}(v_7) \\ &= (L_{strong} - 2L_{weak}) + L^{pr}(v_7). \end{aligned} \tag{4.1}$$

$$\begin{aligned} L^{ps}(v_6) &= L^{-}(r_{v6c3}) + L^{-}(r_{v6c6}) + L^{+}(r_{v6c9}) + L^{pr}(v_6) \\ &= -(L_{strong} - 2L_{weak}) + L^{pr}(v_6). \end{aligned} \tag{4.2}$$

Let the biasing offset $\epsilon = L_{strong} - 2L_{weak}$. Selecting a larger positive offset ϵ helps recovering the bits in the absorbing set, but also spreads more errors to the correct bits in the neighborhood set. An optimal selection of the ϵ value should be preferably small enough to control the spreading of errors to the correct bits while still capable of correcting the absorbing set errors. The selection criteria can be determined based on the prior LLRs.

4.1.2 Two-step Decoding

In the following, the above analysis is reformulated for the $(8, 8)$ absorbing set that dominates the error floor performance of the $(2048, 1723)$ RS-LDPC code. The cardinality

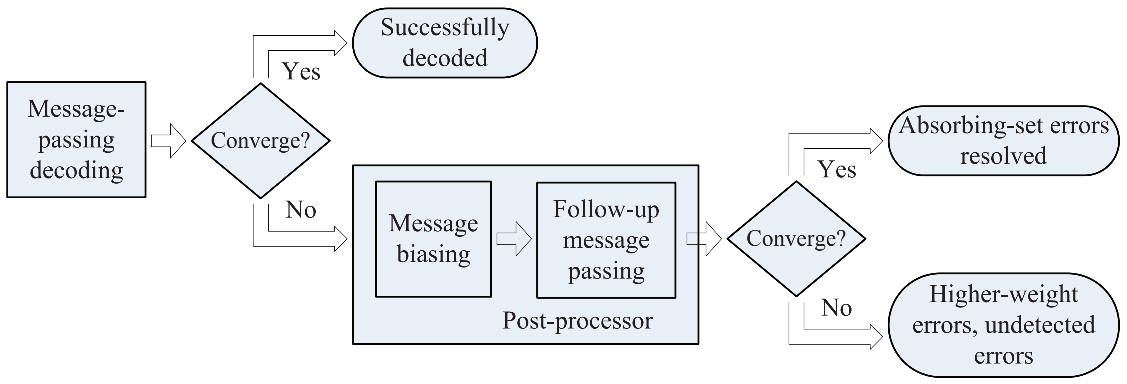


Figure 4.6: A two-step decoder composed of a regular decoder and a post-processor.

of the neighborhood set $N(D)$ is 256. Each bit in $N(D)$ is connected to five satisfied checks and one unsatisfied check. The following decoding steps are performed:

1. Regular message-passing decoding

   Run for a fixed number of iterations. If decoding fails to converge, continue with the next step.

2. Post-processing

   (a) Message biasing: apply uniform strengthening to all messages from check nodes in $\mathcal{O}(D)$ to variable nodes in $N(D)$ and uniform weakening to all messages from check nodes in $\mathcal{S}(D)$ to variable nodes in $N(D)$.

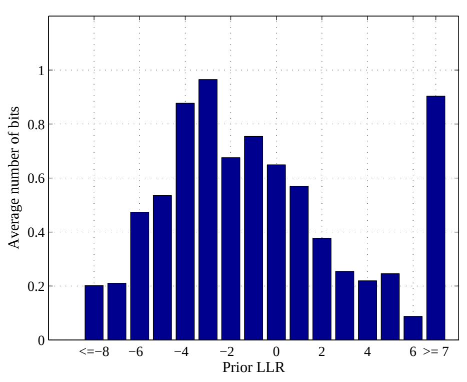
   (b) Follow up with a small number of iterations of regular message-passing decoding until post-processing converges or declare failure.

The flow chart describing the above steps is illustrated in Fig. 4.6.

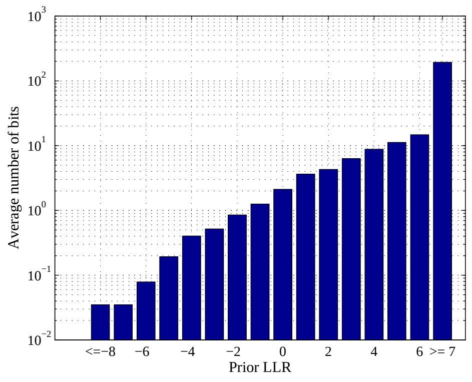
The offset ϵ can be defined as $\epsilon = L_{strong} - 5L_{weak}$ with respect to the $(8, 8)$ absorbing set, because each bit in the absorbing set is connected to five satisfied checks and one unsatisfied check. By hardware emulation, 114 $(8, 8)$ absorbing set errors have been recorded together with the associated prior LLRs at an SNR level of 4.8 dB. The prior LLR distribution of bits belonging to the absorbing set D and bits belonging to the set $N(D) \setminus D$ are shown in Fig. 4.7a and 4.7b.

Based on Fig. 4.7a, the prior LLRs of over half of the bits in the $(8, 8)$ absorbing sets are greater than -1; thus setting $\epsilon = 1$, for instance, solves at least half of the errors and weakens the rest. Fig. 4.7b shows that the prior LLRs of the overwhelming majority of the bits in $N(D) \setminus D$ are very positive. Setting $\epsilon = 1$ causes only 2.2% of the bits in $N(D) \setminus D$ (5 to 6 bits on average) to reverse their values and the remaining bits in $N(D) \setminus D$ are stable. Therefore, it is possible to select a small offset value ϵ to correct at least a reasonable fraction of the incorrect bits in the absorbing set, and affect negatively only a few correct bits.

After message biasing is applied, we follow up with a few more iterations of regular message-passing decoding to further break up the absorbing set and recover the few incorrectly flipped bits. An absorbing set usually collapses quickly after a fraction of bits in the absorbing set are corrected and the rest of the bits are weakened – the reinforcements between the bits of the absorbing set are significantly reduced and the absorbing set structure is resolved in a domino fashion.



(a) Prior LLR distribution of bits in D, $|D| = 8$



(b) Prior LLR distribution of bits in $N(D) \setminus D$, $|N(D) \setminus D| = 248$

Figure 4.7: Prior LLR distribution based on 114 (8,8) absorbing error traces. Results are obtained using a Q4.0 offset ASPA decoder of the (2048,1723) RS-LDPC code at SNR = 4.8 dB.

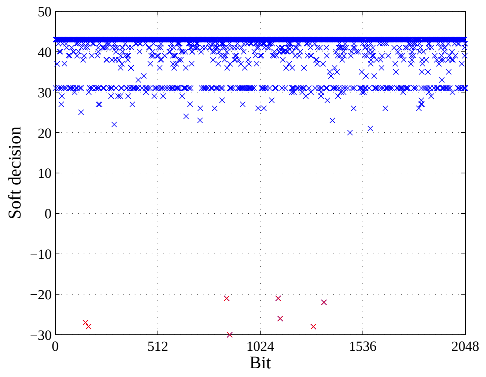
4.1.3 Offset Selection

Messages are often saturated after just a few decoding iterations, and the strengthening operation in post-processing is not necessary as the messages have reached the maximum reliability by the time post-processing starts. Since only the weakening operation is needed, the offset can be reformulated as $\epsilon \approx L_{max} - 5L_{weak}$, where L_{max} corresponds to the maximum value that the quantization allows. In a Q4.0 uniform quantization, $L_{max} = 7$. Setting $L_{weak} = \{0, 1, 2\}$ yields $\epsilon \approx \{7, 2, -3\}$. The iteration-by-iteration illustration of the soft decisions during post-processing are shown in Fig. 4.8 for $L_{weak} = 0$, Fig. 4.9 for $L_{weak} = 1$, and Fig. 4.10 for $L_{weak} = 2$. These three cases are demonstrated using the same initial absorbing state with eight bits of an (8,8) absorbing set assuming extremely incorrect values after 20 iterations of regular message-passing decoding. The initial absorbing states are shown in Fig. 4.8a, 4.9a, and 4.10a.

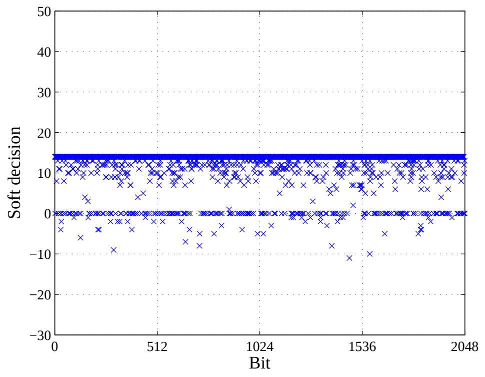
Setting $L_{weak} = 0$ introduces a large offset of $\epsilon = 7$. The strong bias allows all eight bits in the absorbing set to be corrected immediately after the bias is applied. In the meantime, a strong bias enables the bits in the absorbing set to easily propagate errors to force 51 correct bits to admit wrong values, as in Fig. 4.8b. In the first follow-up iteration, most of these incorrect bits can be recovered, though the errors continue to propagate to more neighboring bits, as in Fig. 4.8c. Since the girth of the code is six [20], in the second follow-up iteration, the errors propagate back to the bits that belong to the absorbing set, forcing five of the eight bits in the absorbing set to revert to the wrong values as in Fig. 4.8d. After a few more iterations, the decoder re-enters the same absorbing state that it starts with.

Setting $L_{weak} = 1$ reduces the bias offset to $\epsilon = 2$. With a weaker bias, only four of the eight bits that belong to the absorbing set are corrected immediately after the bias is applied. The weaker bias also restricts the error propagation to only seven correct bits, as in Fig. 4.9b. The error propagation continues in the first follow-up iteration but in a much limited scale to only affect one correct bit, while seven of the eight bits in the absorbing set are recovered, as in Fig. 4.9c. In the second follow-up iteration, the message-passing decoding converges to the correct all-zeros codeword, as in Fig. 4.9d.

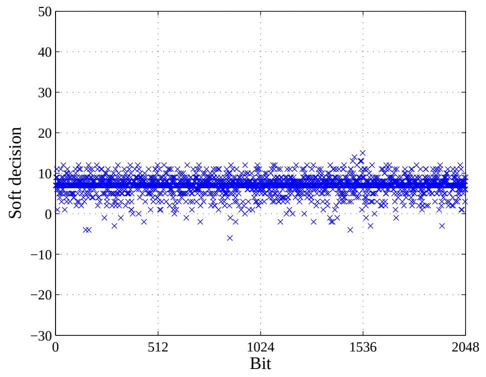
Setting $L_{weak} = 2$ further reduces the bias offset to $\epsilon = -3$. A very weak bias causes no error propagation, but neither does it help recover the bits in the absorbing set, as shown in Fig. 4.10b. The reinforcement among the bits that belong to the absorbing set remains in place and attracts the message-passing decoder back to the same state very quickly, as in Fig. 4.10c and 4.10d.



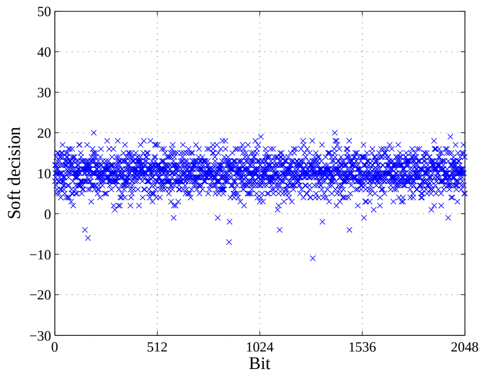
(a) Absorbing state



(b) Message biasing

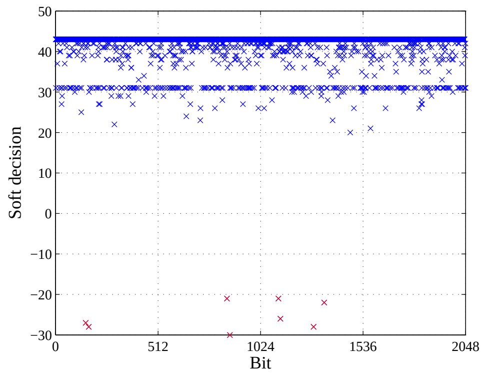


(c) Follow-up iteration 1

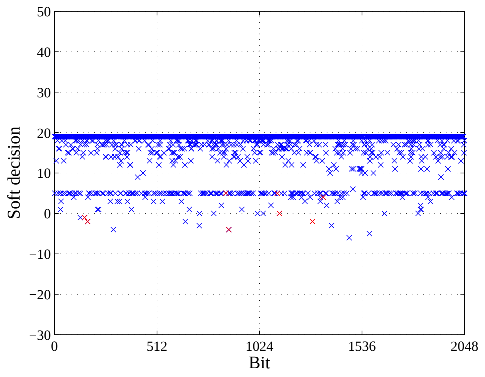


(d) Follow-up iteration 2

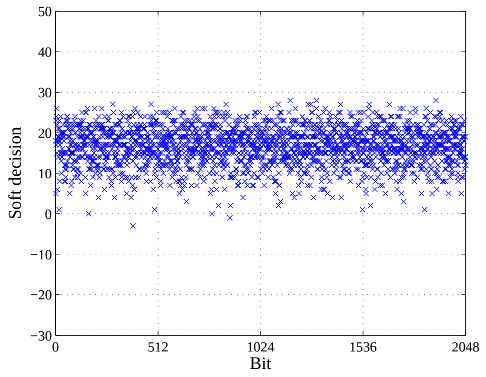
Figure 4.8: Soft decisions at each iteration of post-processing with $L_{weak} = 0$.



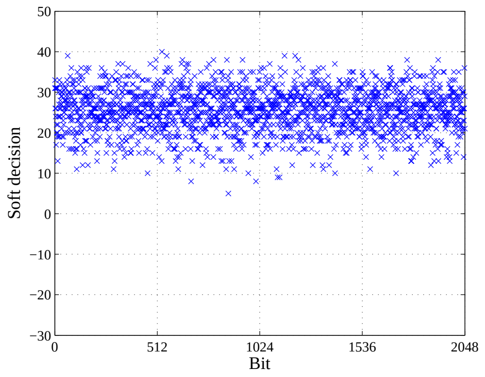
(a) Absorbing state



(b) Message biasing

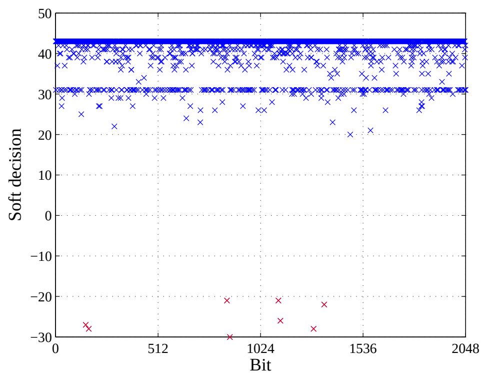


(c) Follow-up iteration 1

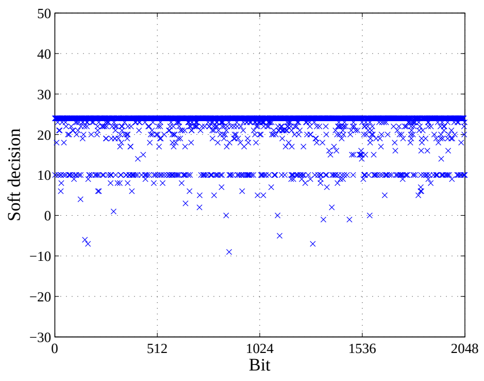


(d) Follow-up iteration 2

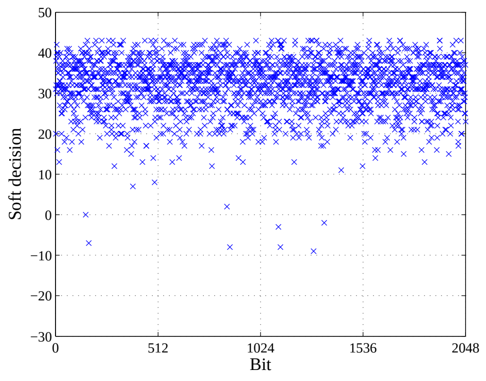
Figure 4.9: Soft decisions at each iteration of post-processing with $L_{weak} = 1$.



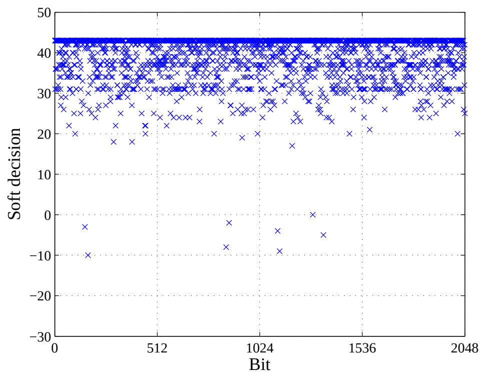
(a) Absorbing state



(b) Message biasing



(c) Follow-up iteration 1



(d) Follow-up iteration 2

Figure 4.10: Soft decisions at each iteration of post-processing with $L_{weak} = 2$.

Table 4.1: Bit error count after post-processing.

SNR (dB)	Before post-proc	$L_{weak} = 0$	$L_{weak} = 1$	$L_{weak} = 2$
4.6	10.75	13.62	26.31	21.15
4.7	8.56	9.55	25.74	10.50
4.8	8.11	9.04	20.78	8.00
5.0	8.01	8.00	–	8.00

The effect of message biasing can be identified by the change of bit error count after post-processing, which is listed in Table 4.1. In a regular message-passing decoder, the average bit error count of a frame error converges to approximately 8 at high SNR levels, indicating the dominance of $(8, 8)$ absorbing sets in determining the error floor. Applying a large ϵ, i.e., $L_{weak} = 0$, causes error propagation and the possible re-convergence to the same $(8, 8)$ absorbing set, thus the average bit error count still approaches 8 at high SNR levels. On the other hand, applying a small ϵ, i.e., $L_{weak} = 2$, has no significant effect on the absorbing state, and the average bit error count remains at 8 after post-processing. The optimal level of message biasing is at $L_{weak} = 1$, which allows the injection of a sufficient amount of noise to the absorbing state to push the decoder out of the local minimum. After the $(8, 8)$ absorbing errors are removed, the average bit error count increases.

The BER and FER performance are shown in Fig. 4.11. Post-processing with either $L_{weak} = 0$ or $L_{weak} = 2$ lowers the error floor, but the floor still remains due to the uncorrected $(8, 8)$ absorbing errors. In comparison, after applying $L_{weak} = 1$, the dominance of $(8, 8)$ absorbing errors is removed and no error floor is observed below the BER level of 10^{-12}.

For a numerical example, 117 absorbing errors were collected at the SNR level of 4.8

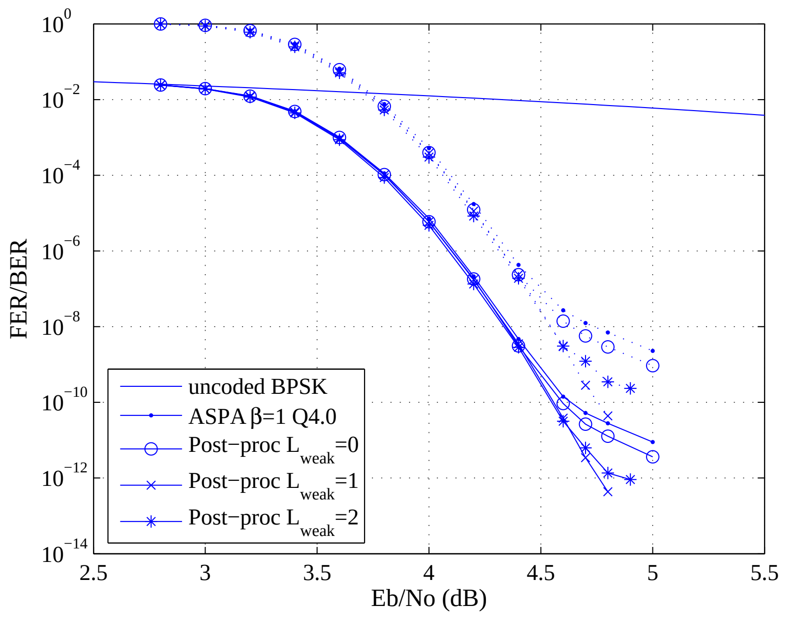


Figure 4.11: FER (dotted lines) and BER (solid lines) performance of a (2048,1723) RS-LDPC code using 20 decoding iterations followed by post-processing with $L_{weak} = 0, 1, 2$.

dB. The number of unresolved absorbing errors after post-processing displays a "U"-shaped relationship with the magnitude of ϵ, as shown in Fig. 4.12. Finding the optimal ϵ for any given code is nontrivial as it depends on the absorbing set structure and the quantization choice, however, the "U"-shaped curve can be exploited by run-time adaptation. A small ϵ is applied initially, which limits the error propagation to the correct bits. If the small ϵ does not result in convergence, a slightly larger ϵ is applied. The procedure continues until the convergence is reached or a failure is declared. Adaptive biasing eliminates the need of deciding the optimal ϵ beforehand, and it is shown to perform as well as applying the optimal offset ϵ, resulting in all the 117 absorbing errors being successfully corrected in this example.

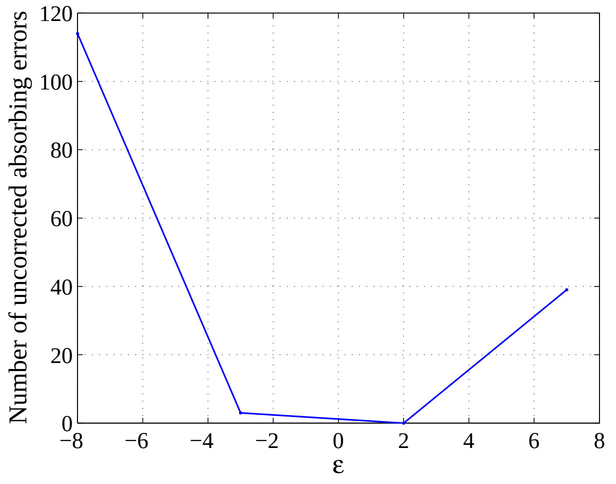


Figure 4.12: The effect of message bias offset ϵ on the post-processing results.

4.1.4 Absorbing Region Analysis

Despite the good case presented above, the reweighted message passing algorithm is not guaranteed to work in all other LDPC codes. Running hardware emulation on each code and trying different offsets for post-processing can be prohibitively expensive. A more promising approach is to perform deterministic estimates of the absorbing region [21].

Given a decoder characterized by its decoding algorithm, quantization, and maximum number of decoding iterations, its associated absorbing region of a given absorbing set is the set of input vectors for which the decoder converges to the absorbing set [21]. Absorbing region is channel-independent and can be deterministically estimated. In [21], the absorbing region is approximated by its projection on a two-dimensional space. Intuitively, absorbing region serves as an indicator of the error probability due to a particular absorbing set. This indicator is verified by experiments showing that the absorbing region shrinks with an improved quantization. Furthermore, the lower bound obtained from the absorbing region analysis shows good agreement with hardware emulation and importance sampling for different quantization choices, channel models, and absorbing sets [21].

A similar absorbing region analysis can be performed to find the effectiveness of post-processing when the dominant absorbing set is known. If the absorbing region shrinks after post-processing, it is a good indication that the reweighted message passing algorithm is successful. The analysis is particularly useful in offset selection: different offset values can be applied in post-processing and the performance improvement can be quantized.

4.2 Emulation Results

The Q4.0 offset ASPA decoder performs worse than the Q5.1 offset decoder and the error floor is elevated by an order of magnitude, as shown in Fig. 4.13. After applying post-processing, The error floor is eliminated down to the BER level of 10^{-12}, which surpasses even the longer-wordlength Q5.1 decoder. The post-processor requires minimal overhead: the 4-bit wordlength is maintained and the approximate sum-product algorithm is unchanged. A small block of logic is added to perform the weakening operation on the corresponding messages after the decoder stalls.

The error count and weight statistics are shown in Table 4.2. The average bit error count per decoding failure converges to 8 at higher SNR levels, signifying the effect of $(8, 8)$ absorbing sets in determining the error floor performance. The average error weight is larger after post-processing, because the lower weight $(8, 8)$ absorbing sets are resolved and only the higher weight errors remain. In addition to $(8, 8)$ absorbing sets, the message biasing scheme successfully resolved $(7, 12)$ and $(11, 6)$ absorbing sets, which demonstrates

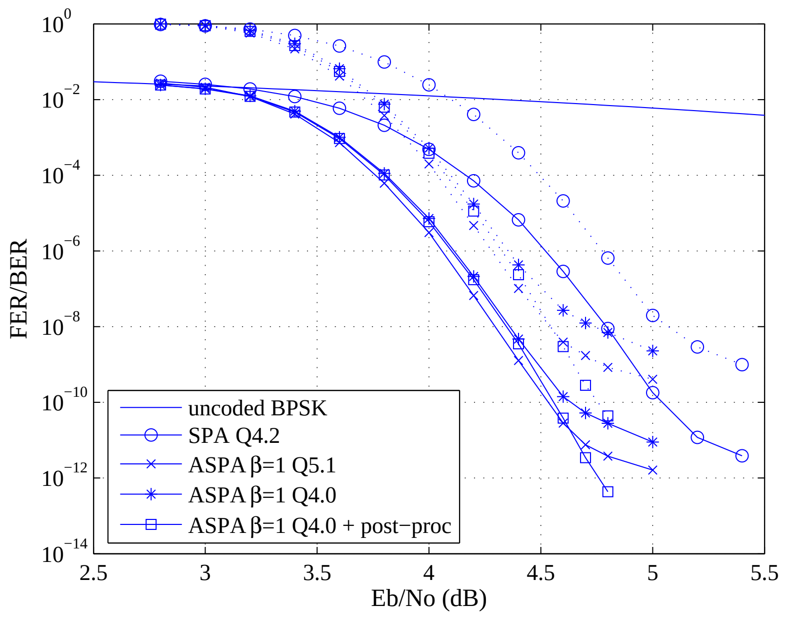


Figure 4.13: FER (dotted lines) and BER (solid lines) performance of a (2048,1723) array-based LDPC code using 20 decoding iterations, which demonstrates the effectiveness of the post-processing approach.

the general applicability of the solution. Approximately 3 iterations are required for the message biasing scheme to finally converge. The extra iterations for post-processing can be easily accommodated due to faster convergence rates at a higher SNR level and infrequent invocation of the post-processor.

Table 4.2: Error statistics before and after post-processing.

SNR (dB)	Before post-processing		After post-processing		Iteration count[2]
	Errors[1]	Average weight	Errors[1]	Average weight	
4.6	928	10.75	102	26.31	2.9
4.7	2578	8.55	58	25.02	3.0
4.8	1603	8.12	10	20.30	2.7
5.0	485	8.01	0	–	2.5

[1] The total number of frames is not uniform for different SNR levels – more input frames were emulated for higher SNR levels. The number of errors collected is divided by the total number of frames to produce the FER plots in Fig. 4.13.

[2] Number of iterations include one iteration for message biasing and iterations of follow-up message-passing decoding to reach convergence.

Chapter 5

Decoder Chip Implementation

The intrinsically-parallel message-passing decoding algorithm relies on the message exchange between variable processing nodes (VN) and check processing nodes (CN) in the graph defined by the **H** matrix. A direct mapping of the interconnection graph causes large wiring overhead and low area utilization. In the first silicon implementation of a fully parallel decoder, Blanksby and Howland reported that the size of the decoder was determined by routing congestion and not by the gate count [4]. Even with optimized floor plan and buffer placement technique, the area utilization rate is only 50%.

Architectures with lower degrees of parallelism can be attractive, as the area efficiency can be improved. In [44], the **H** matrix is partitioned: partitions are time-multiplexed and each partition is processed in a fully parallel manner. Pipeline registers are inserted in the routing paths to reduce congestion and improve maximum frequency. With structured codes, the routing can be further simplified. Examples include the decoders for DVB-S2 [65, 66], where the connection between memory and processors is realized using Barrel shifters. A more compact routing scheme, only for codes constructed with circulant **H** matrices, is to fix the wiring between memory and processors while rotating data stored in shift registers [73]. The more generic and most common partially-parallel architecture is implemented in segmented memories to increase the access bandwidth and the schedules are controlled by lookup tables. Architectures constructed this way permit reconfigurability, as demonstrated by a WiMAX decoder [60].

Solely relying on architecture transformation could be limiting in producing the optimal designs. Novel schemes have been proposed in achieving the design specification with no addition (or even a reduction) of the architectural overhead. In [50], a layered decoding schedule was implemented by interleaving check and variable node operations in order to speed up convergence and increase throughput. This scheme costs additional processing and a higher power consumption. In [17], a bit-serial arithmetic reduces the number of interconnects by a factor of the wordlength, thereby lowering the wiring overhead in a fully parallel architecture. This bit-serial architecture was demonstrated for a small LDPC code with a block length of 660. More complex codes can still be difficult due to the poor scalability of global wires.

In this work, a systematic design flow is adopted, which takes into account both architectural and algorithmic solutions. An illustration is shown in Fig. 5.1. The functional specification of the decoder chip is entered in the Simulink environment through a design flow developed in the Berkeley Wireless Research Center. Chip synthesis and physical design are performed using the Integrated Systems Environment for Configurable Technologies and ASICs (INSECTA) [18]. INSECTA encompasses a collection of scripts that wrap

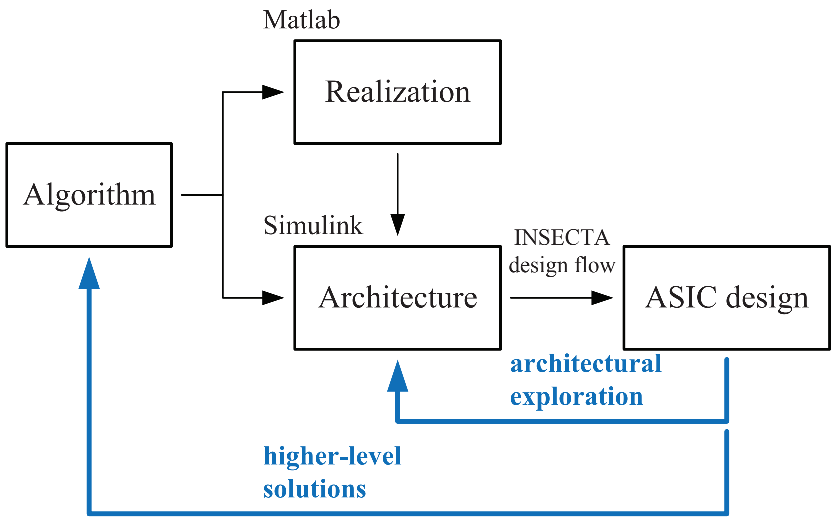


Figure 5.1: Design optimization loop involving both architectural and algorithmic solutions.

around a number of commercial tools: Xilinx System Generator that replaces a set of predefined Simulink blocks with register transfer level (RTL) description, Synopsys Design Compiler for RTL synthesis to map RTL to gate-level (standard cells) circuit description, and Synopsys Astro for placement and routing to generate chip layout for fabrication. The unified Simulink-based entry point enables design reuse – the same design that has been verified through extensive hardware emulation can be made to application-specific integrated circuit (ASIC) without any change. Even though designs for emulation and ASIC do not share a common set of objectives in performance and efficiency as described in Section 2.4.1, the design library of the component blocks could be shared in constructing different architectures to suit each set of objectives.

5.1 Architectural Design

A procedure is described here in designing a high-throughput decoder architecture for the $(6, 32)$-regular $(2048, 1723)$ RS-LDPC code. A high decoding throughput requires a high degree of parallelism and a large memory access bandwidth. The previously described strategy in Section 2.3 is adopted in grouping the VN and CN nodes and bundling the wires between the nodes. Wiring irregularities are sorted within the group, as illustrated in Fig. 5.2b for the example **H** matrix in Fig. 5.2a. Wire sorting can be viewed as a routing operation and the operation on each submatrix can be viewed as a separate routing operation. The fully parallel architecture with all the routers expanded is shown in Fig. 5.2b.

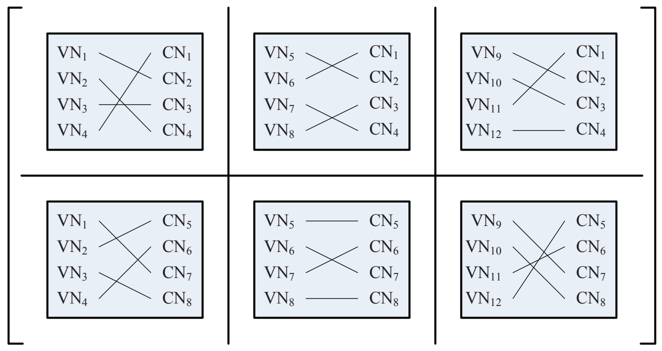
5.1.1 Wiring Overhead

Even with node grouping and wire bundling, the fully parallel architecture might not be the most efficient for a complex LDPC decoder. To reduce the level of parallelism, individual routers are combined and routing operations are time-multiplexed. Fig. 5.2c shows how the two routers in every column are combined, resulting in a one-dimensional 3-way parallel architecture. Router combining leads to the creation of local units, shown as variable node groups (VNG) and check node groups (CNG) in Fig. 5.2c, that encapsulate irregular local wiring, and wires outside of local units are regular and structured.

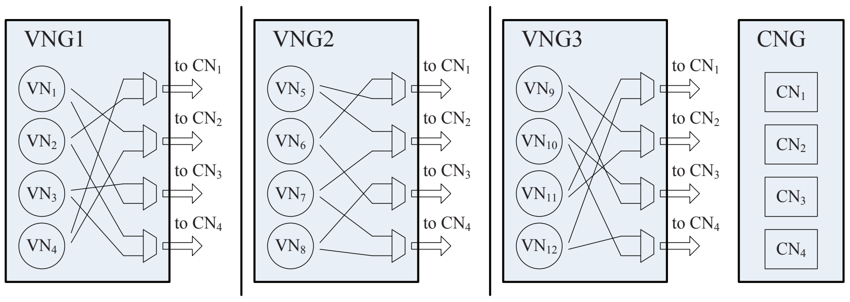
The number of local units determines the level of parallelism. A less parallel design uses fewer local units, but each one needs to be more complex as it needs to encapsulate more irregular wiring to support multiplexing; a highly parallel design uses more local units

$$
\begin{array}{cccc|cccc|ccccc}
VN_1 & VN_2 & VN_3 & VN_4 & VN_5 & VN_6 & VN_7 & VN_8 & VN_9 & VN_{10} & VN_{11} & VN_{12} & \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & CN_1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & CN_2 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & CN_3 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & CN_4 \\
\hline
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & CN_5 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & CN_6 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & CN_7 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & CN_8
\end{array}
$$

(a) A simple structured **H** matrix



(b) The fully parallel architecture



(c) A 3VNG-1CNG parallel architecture

Figure 5.2: Architectural mapping and transformation.

and each one is simpler, but the amount of global wiring, though regular and structured, would increase accordingly. Tradeoff exists between the global wiring overhead and local wiring overhead. Local wiring is relatively cheaper but when the degree of parallelism is lowered to a certain level, the multiplexing complexity increases rapidly, and the local wiring overhead becomes dominant. On the other hand, regular global wiring is manageable but a large volume of even regular global wires still makes it impossible for efficient placement and routing.

To explore the optimal level of parallelism targeting a lower wiring overhead, the area expansion factor (AEF) is defined in (5.1)

$$\text{AEF} = \frac{\text{area of the complete system}}{\text{total area of stand-alone component nodes}} \tag{5.1}$$

The numerator of AEF is the area of the assembled system after placement and routing and the denominator represents the sum of area of component nodes. i.e., VN and CN. Based on this definition, AEF is a convenient metric for global wiring overhead. An AEF of 1 indicates zero global wiring overhead, which is the case for a fully serial architecture. As the degree of parallelism increases, more and more component nodes are assembled and extra space is allocated for wiring. As a result, the AEF would increase accordingly. AEF is a reliable metric for global wiring overhead for two reasons:

1. AEF accounts for wire buffering and gate upsizing in assembling a complete system. Another common measure of the wiring overhead is the cell-to-core ratio (also known as density or utilization ratio), referring to the ratio of standard cell area to core area. A standard cell is a group of transistors and internal interconnect structures, which

provides a logic or storage function. A high cell-to-core ratio is not necessarily a good indication of a low global wiring overhead. Techniques in reducing excessive global wire delays, such as wire buffering and gate upsizing, increases the cell-to-core ratio, which could be mistaken for a low global wiring overhead. In comparison, AEF bases the denominator on the stand-alone component nodes, thus wire buffering and gate upsizing are reflected in a higher AEF value.

2. AEF measures global wiring overhead. The total wire length and count reported by the placement and routing tool do not make the distinction between global and local wires.

For the $(6, 32)$-regular $(2048, 1723)$ RS-LDPC $\mathbf{H}$ matrix under consideration, a few selected architectures were investigated, listed in Table 5.1 with increasing degrees of parallelism from top to bottom. The AEF of the designs is plotted in Fig. 5.3 with the horizontal axis displaying the decoding throughput (for simplicity, assume throughput is a linear function of the maximum clock frequency). The AEF curve shows an upward trend with increasing degrees of parallelism. The middle segment of the curve from the 16VNG-1CNG architecture to the 32VNG-1CNG architecture appears to be flat. Designs positioned in the flat segment achieve a balance of throughput and area – doubling the throughput from the 16VNG-1CNG architecture to the 32VNG-1CNG architecture requires almost twice as many processing nodes, but the AEF remains almost constant, so the core area doubles. In the region where the AEF is constant, the average global wiring overhead is constant and it is advantageous to increase the degree of parallelism for a higher throughput.

Table 5.1 also shows that density is not a reliable measure of the wiring overhead.

Table 5.1: Architectural selection based on synthesis, place and route results in the worst-case corner.

Architecture	VN	CN	Freq(MHz)	Density	AEF	Wire length(m)
8VNG-1CNG	512	64	400	91.01%	1.331	20.343
16VNG-1CNG	1024	64	400	91.21%	1.471	24.614
32VNG-1CNG	2048	64	400	84.17%	1.465	30.598
64VNG-2CNG	4096	128	350	86.84%	1.738	65.504

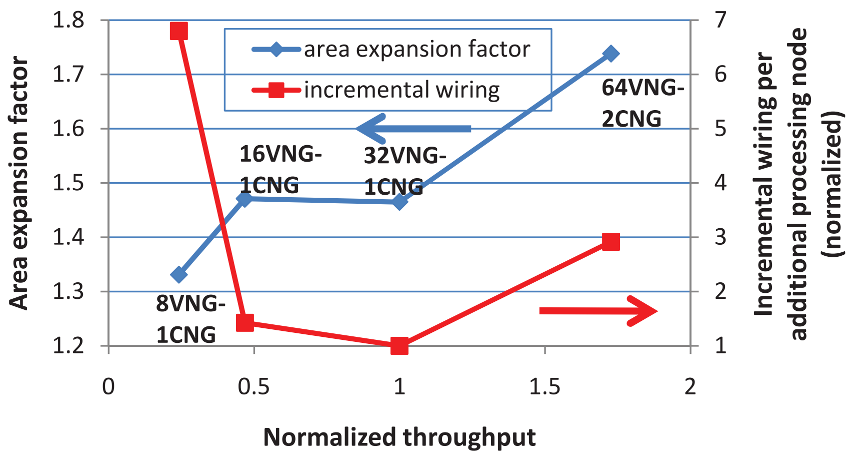


Figure 5.3: Architectural optimization by the area expansion metric.

For example, the 64VNG-2CNG architecture has a higher density than the 32VNG-1CNG architecture, but the total on-chip signal wire length is more than twice longer.

The AEF plot suggests a more serial architecture, e.g., 8VNG-1CNG, as it incurs the lowest average global wiring overhead. However, the total on-chip signal wire length of the 8VNG-1CNG architecture is still significant – an indication of the excessive local wiring in supporting time-multiplexing. The incremental wiring overhead per additional processing node is plotted in Fig. 5.3. As the degree of parallelism increases from 8VNG-1CNG, the local wiring decreases more quickly while the global wiring increases slowly, resulting in a decrease in the incremental wiring overhead. The incremental wiring overhead eventually reaches the minimum with the 32VNG-1CNG architecture. This minimum corresponds to the balance of local wiring and global wiring. Any further increase in the degree of parallelism causes a significant increase in the global wiring overhead (suggested by the increase of AEF), and the rise of the incremental wiring overhead.

The 32VNG-1CNG architecture is selected for final implementation to achieve the balance of throughput and area and the balance of local and global wires.

5.1.2 Density

Density measures area efficiency, and a high density is desirable in practice. The optimal density target depends on the tradeoff between routability and wire distance. A lower-density design can be easily routed, but it occupies a larger core area and wires need to travel longer distances from point to point. On the other hand, a high-density design cannot be routed easily, and the clock frequency needs to be reduced as a compromise. The density choice of the 32VNG-1CNG architecture is explored. An initial density is specified for the allocation of white space in placement and routing. The physical design tool performs iterative trial placement, cell sizing, placement relocation, trial routing, wire buffering, parasitic extraction, and rerouting. The clock tree is inserted in the process of physical design. The density of the final design is usually higher than the initial density due to the addition of a clock tree, wire buffers, and possible cell upsizing.

The target clock frequency of the 32VNG-1CNG decoder architecture is set to 400 MHz based on worst-case corner operating condition. Table 5.2 shows that timing closure can be achieved with initial densities of 70% to 80%. The total signal wire length decreases with increasing density due to the shorter wire distances even with increasing wire counts.

Table 5.2: Density selection based on synthesis, place and route results in the worst-case corner.

Initial density	Final Density	Frequency(MHz)	Wire length(m)	Wire count(,000)
70%	74.43%	400	31.884	9,071
75%	79.73%	400	31.868	9,270
80%	84.17%	400	30.598	9,295
85%	90.48%	360	32.118	10,258
90%	96.29%	350	31.431	10,308

An initial density above 80% results in routing difficulty and the maximum clock frequency has to be reduced to accomodate longer propagation delays. To maximize density without sacrificing timing, an 80% initial density is selected.

5.2 Functional Design

5.2.1 Component Nodes

The 32VNG-1CNG decoder architecture consists of $2,048$ VN nodes, representing the majority of the chip area. The block diagram of the VN node for the offset ASPA decoder is illustrated in Fig. 5.4. Each VN node sends six variable-to-check messages and receives six returning messages from check nodes per decoding iteration. The six messages are sent and received sequentially. (Refer to the the pipeline chart in Fig. 5.6 for the timing diagram.) Three storage elements are allocated: the posterior memory which accumulates the check-to-variable messages, the extrinsic memory which stores the check-to-variable messages in a shift register, and the prior memory which contains the prior LLRs.

Each of the $2,048$ VN nodes participates in the operations in six horizontal rows. In each operation, a variable-to-check message is computed by subtracting the corresponding

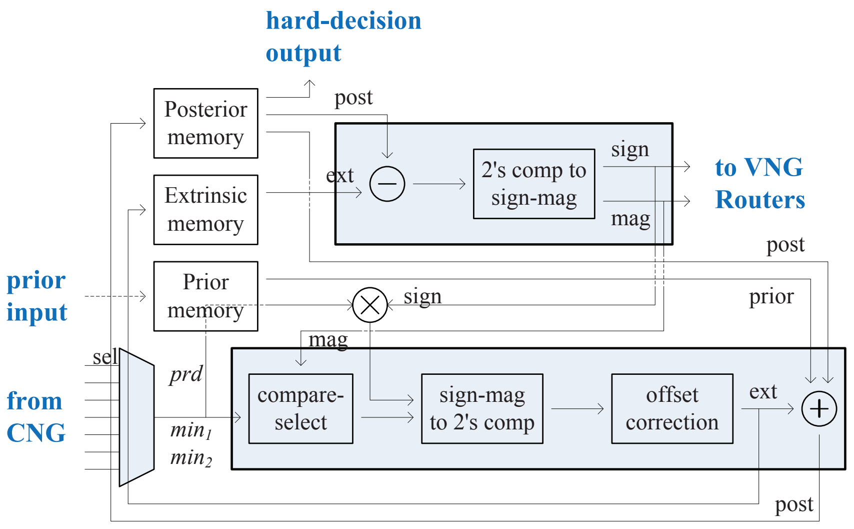


Figure 5.4: VN node design for an offset ASPA decoder.

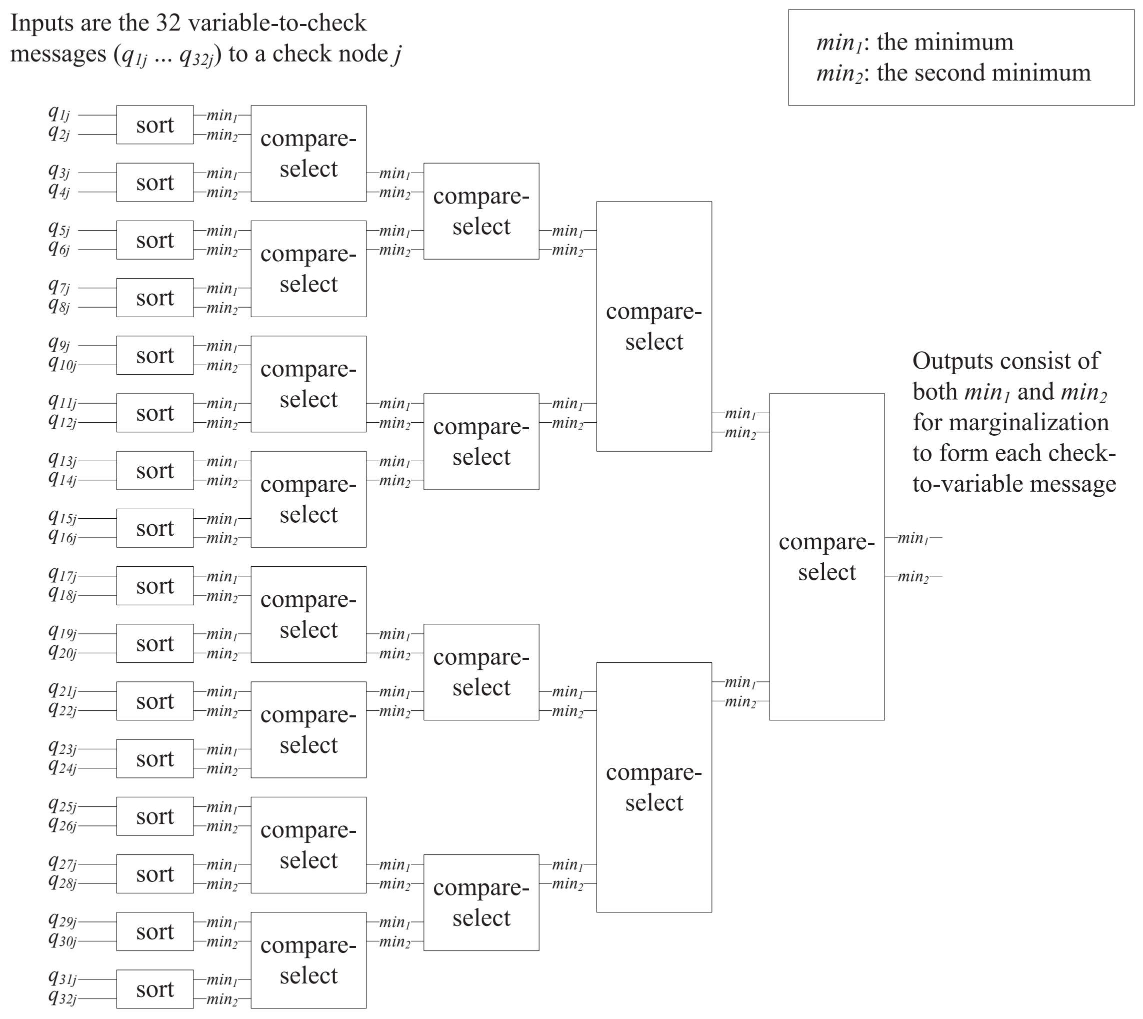
check-to-variable message (of the previous iteration) from the posterior (of the previous iteration) as in equation (2.11). The variable-to-check message is converted to the sign-magnitude form before it is sent to the VNG routers destined for a CN node. The returning messages to the VN node could be from one of the six CN nodes. A multiplexer selects the appropriate message based on a schedule. The check node operation described in equation (2.8) is divided to two parts: for the first part, the CN node computes the minimum (min_1) and the second minimum (min_2, $min_2 \geq min_1$) among all the variable-to-check messages received from the neighboring VN nodes, as well as the product of the signs (prd) of the variable-to-check messages. The min_1, min_2, and prd are bundled and routed to the neighboring VN nodes; for the second part, the VN node completes the check-to-variable operation with the compare-select described in equation (5.2), followed by the conversion to the two's complement format and the offset correction. The resulting check-to-variable message is accumulated serially to form the posterior as in equation (2.13). The hard decisions can be extracted from the posterior in every iteration.

$$L_n(r_{ij}) = \begin{cases} prd \cdot \mathrm{sgn}(L_n(q_{ij}))min_1 & \text{if } |L_n(q_{ij})| \neq min_1, \\ \\ prd \cdot \mathrm{sgn}(L_n(q_{ij}))min_2 & \text{if } |L_n(q_{ij})| = min_1. \end{cases} \tag{5.2}$$

The VNG routers follow the structure shown in Fig. 5.2c with 64 6 : 1 multiplexers. The CN node is designed as a compare-select tree shown in Fig. 5.5. The 32 input variable-to-check messages are sorted in pairs, followed by four stages of 4-to-2 compare-selects. The outputs min_1, min_2, and product of signs, prd (omitted in Fig. 5.5) are buffered and broadcast to the 32 VNG blocks. In placing the compare-select tree on-chip, the first-level sort and compare-selects are distributed very close to the respective VNG blocks. The routing from the first-level to later stages of the compare-select tree covers longer distances and incurs more delays. The final broadcast routing from CNG to VNG incurs even longer delays due to long wire lengths. Considerations are taken in the pipeline design to account for the extra wire delays in balancing the pipeline stages.

5.2.2 Pipeline

An example 7-stage pipeline is designed as in Fig. 5.6. Each box in the chart represents a pipeline stage and pipeline registers are inserted at stage boundaries. One stage is allocated for the VN node in preparing variable-to-check message, and one stage for the delay through the VNG routers. Three stages are dedicated to the compare-select tree in the CN node – one for the sorting and the first-level compare-select, one for the following two levels of compare-select, and one for the final compare-select as well as the fanout. Two stages are set aside for processing the return messages from the CN node – one



Inputs are the 32 variable-to-check messages (q_{1j} ... q_{32j}) to a check node j
min_1: the minimum
min_2: the second minimum
q_{1j} ... q_{32j}
sort
min_1
min_2
compare-select
Outputs consist of both min_1 and min_2 for marginalization to form each check-to-variable message

Figure 5.5: CN node design for an offset ASPA decoder.

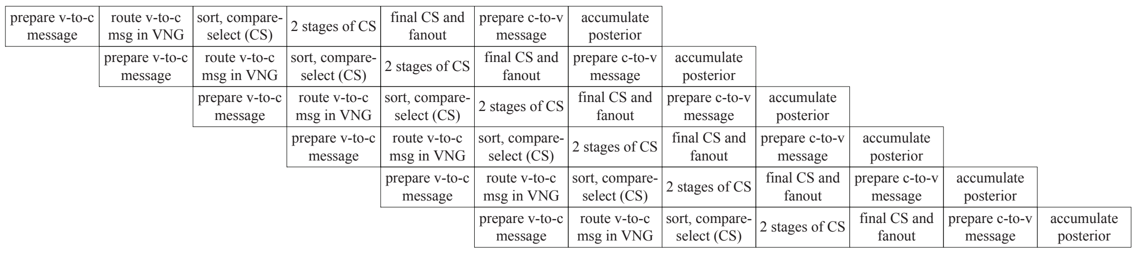


Figure 5.6: Pipeline design of the 32VNG-1CNG decoder.

for preparing the check-to-variable message and one for accumulating the check-to-variable message.

The 7-stage pipeline is designed to minimize the clock period. Trial placement and routing are performed to identify the critical paths and characterize the global wiring delays. The clock period is set such that it accommodates the longest wire delay and the wire's driving or receiving gate delay with a sufficient margin. The pipeline stages are balanced, each targeting the set clock period. A deeper pipeline design would require wire pipelining and an increase in area and power due to additional pipeline registers.

With the 7-stage pipeline and the minimum 2.5-ns clock period for the CMOS technology being used, the final decoding throughput is computed in (5.3). For 10 decoding iterations, a throughput of 6.827 Gb/s can be achieved, which meets the standard requirement.

$$\begin{aligned}\text{Decoding throughput per iteration} &= \frac{\text{Code block length}}{\text{Pipeline latency per iteration} \times \text{Clock period}} \\ &= \frac{2,048 \text{ bits}}{12 \text{ cycles} \times 2.5 \text{ ns/cycle}} = 68.27 \text{ Gb/s}. \qquad (5.3)\end{aligned}$$

In a conventional message passing schedule, data dependency exists between consecutive iterations. The two-iteration pipeline stack is shown in Fig. 5.7. A 6-cycle stall is inserted between iterations to resolve data dependencies such that the posterior can be fully updated in the current iteration before the next iteration starts. The pipeline stall means that the first VC stage (refer to Fig. 5.7) of iteration $i + 1$ has to wait for 6 cycles for the last PS stage of iteration i to complete. During the stall cycles, the pipeline stages are not performing useful work, so the efficiency is lower. The efficiency is reduced even more if a turbo decoding schedule (also known as a layered schedule) [49] or a shuffled schedule [75] is applied to such a pipeline, where data dependency arises between layers within an iteration. If pipeline stalls are inserted to resolve the dependency, the efficiency is degraded to as low as 1/7, thus defeating the purpose of a slightly higher convergence rate achieved with these schedules. Therefore, a conventional message passing schedule is adopted.

Now consider the consequence of simply eliminating the pipeline stalls as in Fig. 5.8: the first layer of variable-to-check messages (referring to the messages processed in the first cycle of a decoding iteration, corresponding to the first layer, or row, of the pipeline chart) would not account for the last five layers of check-to-variable messages from the previous iteration; the second layer of variable-to-check messages would disregard the last four layers of check-to-variable messages, and so on. This incomplete message passing would slow down the convergence. For the maximum convergence speed, a pipeline design with stalls is chosen.

The pipeline efficiency is formally defined in (5.4). To increase the pipeline efficiency, the number of pipeline stages can be reduced at the cost of a slower clock frequency. Examples of a 3-stage pipeline and a 5-stage pipeline are illustrated in Fig. 5.9. The

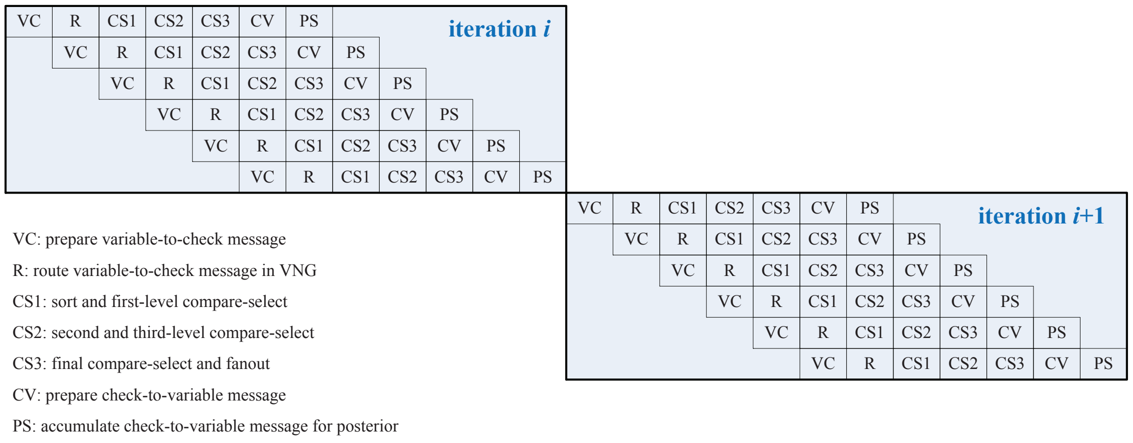


Figure 5.7: Two-iteration pipeline chart with pipeline stalls.

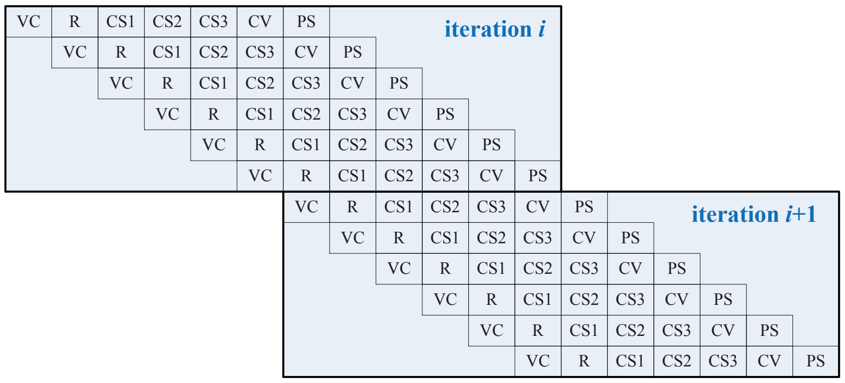
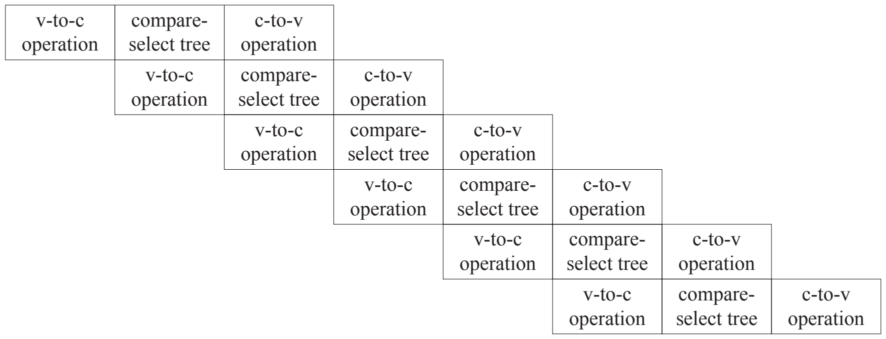
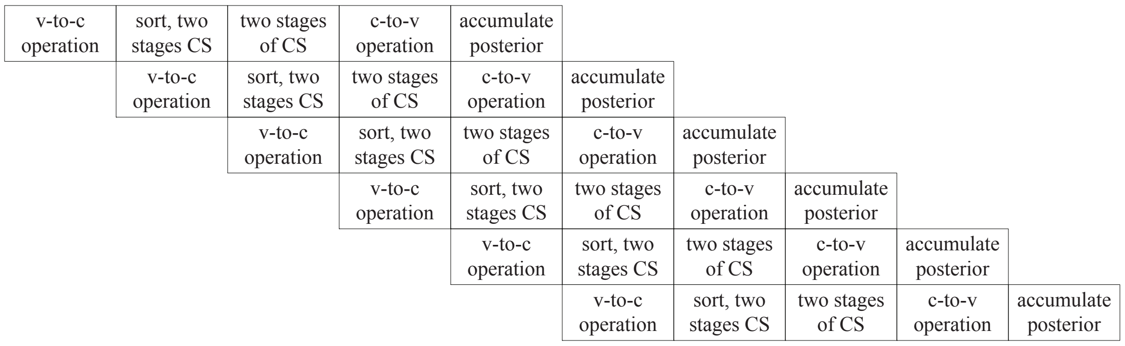


Figure 5.8: Two-iteration pipeline chart without stalls.



(a) A 3-stage pipeline



(b) A 5-stage pipeline

Figure 5.9: Pipelines with shorter latency.

pipeline efficiency and the maximum clock frequency are listed in Table. 5.3. The comparison shows that a 3 or 5-stage pipeline improves the pipeline efficiency, but the maximum clock frequency has to be reduced drastically to accommodate longer delays in a pipeline stage. So to achieve the maximum decoding throughput, the 7-stage pipeline is adopted in the current design.

$$\text{Pipeline efficiency} = 1 - \frac{\text{pipeline stalls per decoding iteration}}{\text{pipeline latency per decoding iteration}} \tag{5.4}$$

Table 5.3: Pipeline designs.

Pipeline	Latency[1](cycles)	Efficiency	Frequency(MHz)	Throughput[2](Gb/s)
3-stage	8	75%	150	3.84
5-stage	10	60%	300	6.14
7-stage	12	50%	400	6.83

[1] Latency of one decoding iteration.

[2] Throughput with 10 decoding iterations at the maximum clock frequency.

5.2.3 Complete System

The block diagram of the complete decoder is shown in Fig. 5.10. The decoder is hierarchically constructed using a Matlab script. The VN and CN nodes are first designed and verified. The RTL code of the nodes are produced by the Xilinx System Generator and then "blackboxed". The higher-level VNG and CNG blocks are constructed with these blackboxes. Hierarchical grouping affects the efficiency and the quality of synthesis, placement and routing. The hierarchical boundaries are drawn such that they align with the pipeline boundaries. Each block is self-contained and remain relatively "untouched" when it is assembled to a higher-level block.

The decoder supports automated functional testing by incorporating AWGN noise generators and error collection. AWGN noise is implemented by the Box-Muller algorithm that transforms two independent uniform random variables to a unit Gaussian random variable. The uniform random variables are produced by linear feedback shift registers (LFSR). The unit Gaussian random noise is scaled by pre-stored multipliers to emulate an AWGN channel at a particular SNR level. The automated functional testing assumes either all-zeros or all-ones codeword. The output hard decisions are compared to the expected codeword,

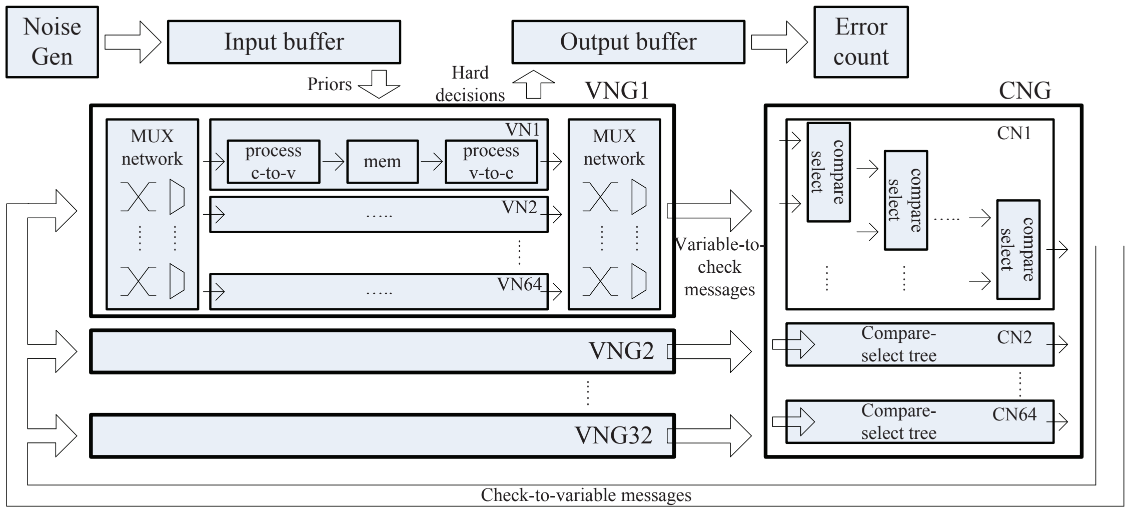


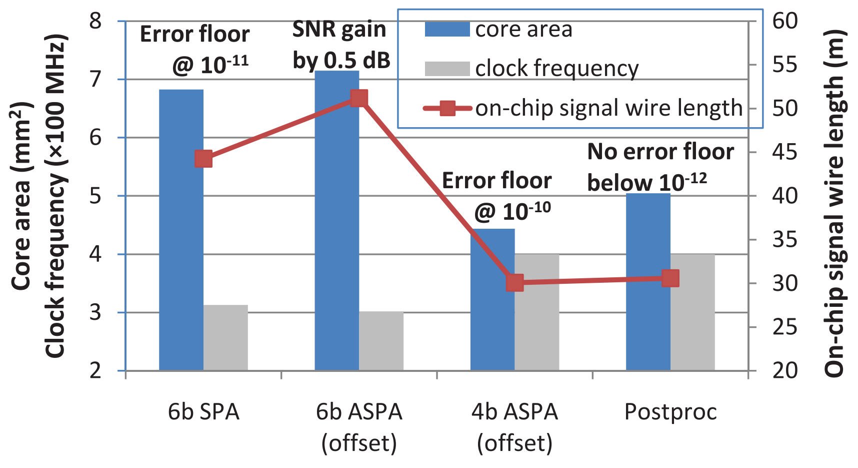
Figure 5.10: The decoder implementation using the 32VNG-1CNG architecture.

and the number of bit and frame mismatches are accumulated in the error counter. The automated functional testing enables real-time decoder testing without the need of waiting for external input or inspecting output externally.

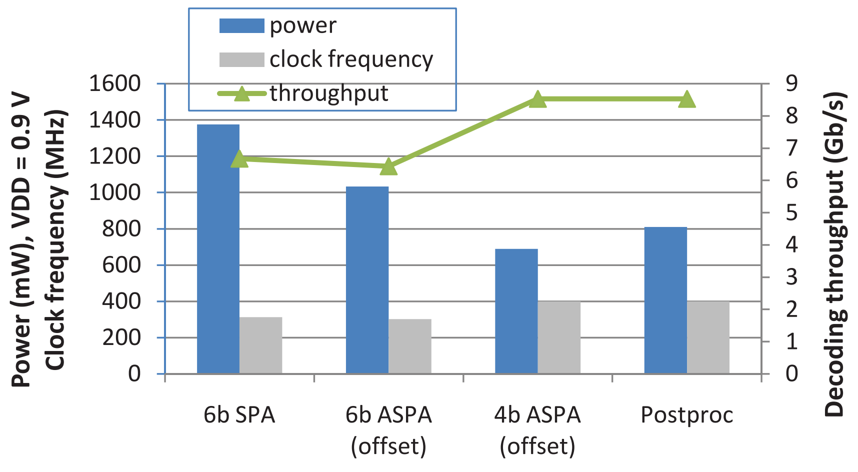
5.2.4 Area and Power Optimizations

Steps of area, performance, power, and throughput improvements are illustrated in Fig. 5.11a and 5.11b. The baseline design is a 6-bit SPA decoder. It occupies 6.83 mm^2 of core area in a 65nm CMOS technology and it consumes 1.38 W of power to deliver the 6.68 Gb/s throughput (assuming 8 decoding iterations) at the maximum 310 MHz clock frequency (at a 0.9 V supply voltage and the worse-case operating condition). This implementation incurs an error floor at a BER level of 10^{-11}. To achieve a lower BER, extra SNR needs to be spent due to the error floor.

By the hardware emulation results from Fig. 4.13, the conventional 6-bit SPA



(a) Area and performance improvement



(b) Power and throughput improvement

Figure 5.11: Steps of improvement evaluated on the 32VNG-1CNG architecture using synthesis, place and route results in the worst-case corner.

decoder is replaced by a 6-bit offset ASPA decoder to gain 0.5 dB in SNR. The core area increases to 7.15 mm^2 due to the differences in the CN node design. The CN node is implemented as an adder tree in an SPA decoder and a compare-select tree in an ASPA decoder (illustrated in Fig. 5.5). While the area of a 6-bit compare-select is comparable to a 6-bit adder, every compare-select block maintains both min_1 and min_2 for the purpose of marginalization. The resulting 11-bit message composed of min_1, min_2, and prd needs to be routed to the VN nodes, and in comparison only 6 bits are required in a SPA decoder. The additional routing overhead is reflected in the 15.6% increase in wiring, leading to a slightly lower maximum clock frequency of 300 MHz and a reduced decoding throughput of 6.44 Gb/s (assuming 8 decoding iterations).

Despite the wiring and area increase, the offset ASPA decoder consumes less power at 1.03 W (at the maximum clock frequency of 300 MHz). The power saving is attributed to the check node operation characteristics – an adder tree consumes more dynamic power than a compare-select tree due to a higher activity factor. At high SNR levels or when decoding approaches convergence, the majority of the variable-to-check messages are saturated and the wires driven by the compare-select blocks do not switch frequently, resulting in a lower power consumption.

To further reduce the area and power, the wordlength of the offset ASPA decoder is reduced from the conventional 6 bits to 4 bits. The reduction of wordlength cuts the total wire length by 41.2%, shrinks the core area by 37.9% down to 4.44 mm^2. With a reduced wiring overhead, the maximum clock frequency can be raised to 400 MHz, while consuming only 690 mW – a 33.3% power reduction. The decoding throughput is increased to 8.53

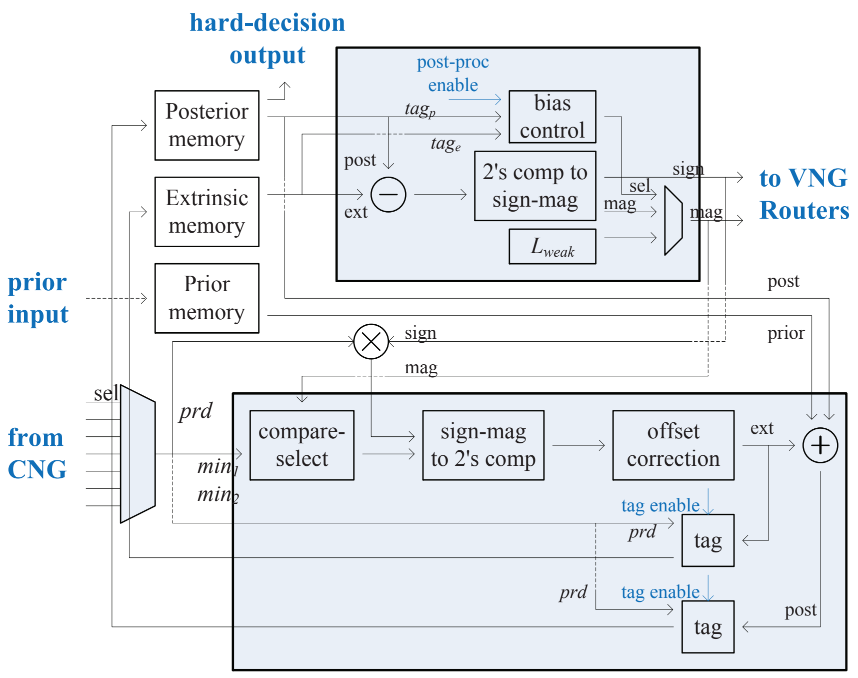


Figure 5.12: VN node design for post-processing.

Gb/s.

Wordlength reduction results in better designs by all measures except the functional performance. The error floor is elevated by an order of magnitude, as seen in Fig. 4.13. To fix the error floor, a post-processor is added to the 4-bit ASPA decoder. The post-processor is designed as an add-on to the VN node without contributing to the wiring external to the VN node. The block diagram of the new VN node is shown in Fig. 5.12. Post-processing works in three phases: pre-biasing, biasing, and follow-up. The operations are described as follows.

1. Pre-biasing phase (one iteration before post-processing): tag is enabled (refer to Fig. 5.12). If a parity check is not satisfied, as indicated by *prd*, the check-to-variable messages originating from the check node are tagged, and the neighboring variable

nodes are also tagged by marking the posterior of these variable nodes. The 1-bit tags cost small additional storage.

2. Biasing phase: post-proc is enabled (refer to Fig. 5.12). Tags of both the posterior and the check-to-variable messages are inspected, such that if a variable node in the neighborhood set sends a message to a satisfied check node, the magnitude of this variable-to-check message is weakened.

3. Follow-up phase: post-proc is disabled (refer to Fig. 5.12). Regular message passing decoding is performed for a few more iterations to clean up the errors after message biasing.

The post-processor increases the core area by 13.7% to 5.05 mm^2 and the power consumption by 17.6% to 810 mW. Overall wiring overhead only increases by 1.7%, thus the majority of the area and power increase is attributed to the extra logic and storage in the VN node. Since the wiring overhead is almost constant, the maximum clock frequency can be maintained, and the decoding throughput is kept constant at 8.53 Gb/s.

To further increase the decoding throughput, an early termination scheme is implemented on-chip to detect early convergence by monitoring whether all the check nodes are satisfied and if so, the decoder can immediately proceed to the next input frame. The early termination scheme eliminates idle cycles and the processing nodes are kept busy all the time. The throughput gain becomes significant at high SNR levels. For example, at an SNR level of 5.5 dB, convergence can be achieved in approximately 1.47 iterations on average. Taking into account one additional iteration to detect convergence, a total of 2.47 iterations are required. Therefore the average throughput can be improved to 27.68 Gb/s

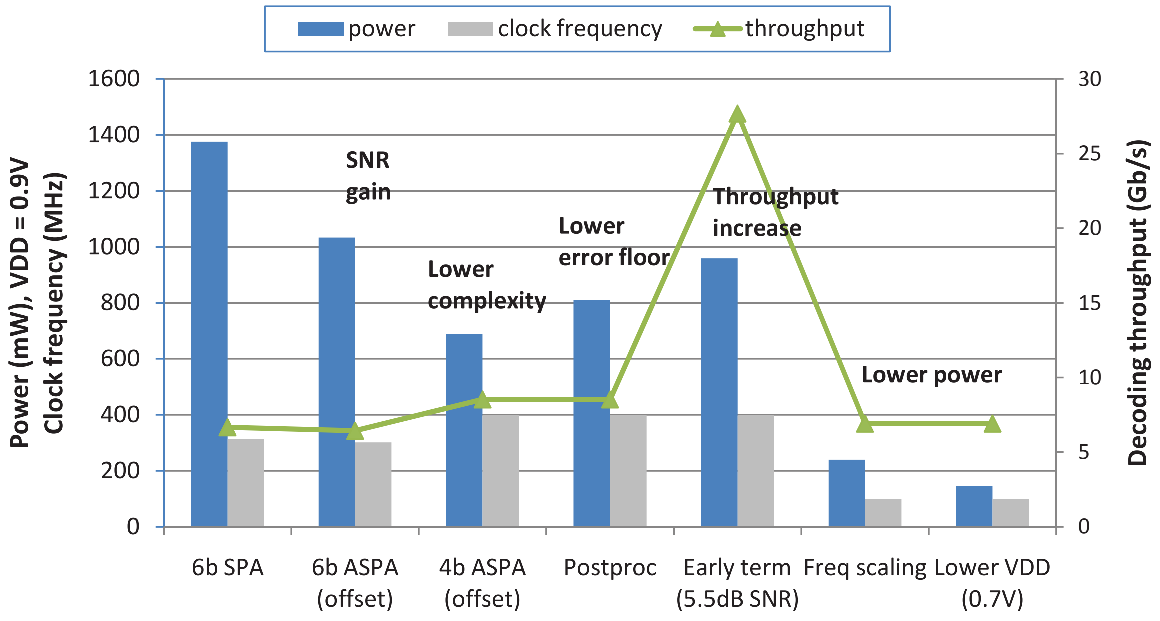


Figure 5.13: Power reduction steps with results from synthesis, place and route in the worst-case corner.

as shown in Fig. 5.13, assuming a 400 MHz clock frequency in the worse-case corner. With early termination, the power consumption increases by 18.4% to 960 mW due to a higher activity factor.

With a much higher throughput, the clock frequency and supply voltage can be aggressively scaled down to reduce the power consumption. To reach the required throughput of 6.67 Gb/s, the clock frequency can be scaled from 400 MHz to 100 MHz and the supply voltage can be scaled from 0.9 V to 0.7 V to reduce the power consumption by almost 85% to 145 mW.

The decoding throughput quoted for an early-termination-enabled decoder is an average throughput at a specific SNR point; a maximum iteration limit is still imposed to prevent running an excessive number of iterations due to the occasional failures. A

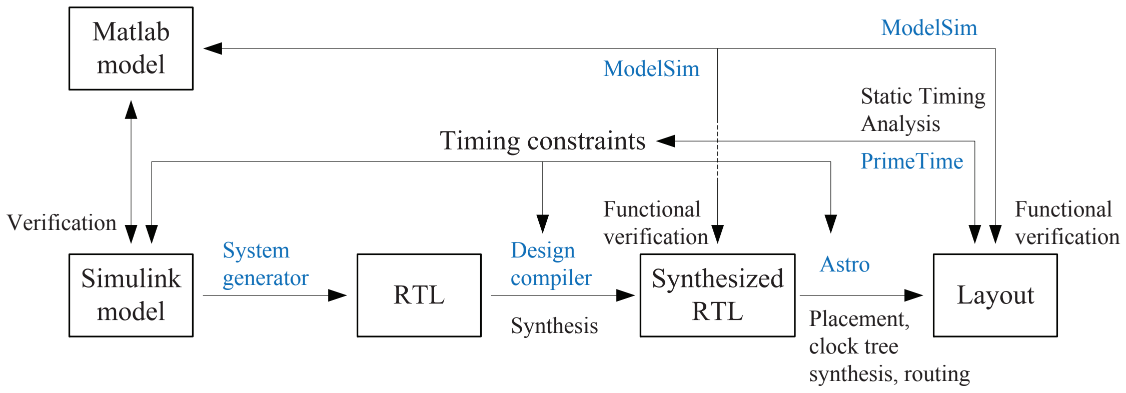


Figure 5.14: Chip design flow with timing and functional verifications.

higher maximum iteration limit calls for a larger input and output buffering to provide the necessary timing slacks. A detailed analysis can be performed to determine the optimal buffer length for a performance target [5].

5.2.5 Timing Constraints and Verification

Functional equivalence and timing consistency are maintained in every step of the design flow as shown in Fig. 5.14. The Simulink model is verified rigorously through hardware emulation. Emulation results are checked against the Matlab model through feedback simulations. A set of timing constraints is applied at different stages of the chip design flow, described in Table 5.4. More abstract constraints are applied at the early stage of the flow and the constraints become more elaborate towards later stages.

The design in Simulink is based on the initial estimation of the clock period and the pipeline stage division. The Simulink model is converted to the RTL model, which is then synthesized using Synopsys Design Compiler. More elaborate timing constraints are applied in the synthesis stage, including the constraints on clock latency, uncertainty, and

Table 5.4: Application of timing constraints in the design flow.

Flow step	Timing constraints
Simulink model	Clock: period and pipeline stage division
RTL synthesis	Clock: latency, uncertainty, transition I/O: input and output delay Wire delay: estimated
Initial placement	Clock: latency, uncertainty, transition I/O: input and output delay Wire delay: extracted
Clock tree insertion	Clock: clock tree insertion delay, skew, transition, uncertainty
Placement optimization	Clock: propagated clock tree I/O: input and output delay Wire delay: extracted
Routing	Clock: propagated clock tree, hold time constraints I/O: input and output delay Wire delay: extracted
Post-layout	Clock: propagated clock tree, hold time constraints I/O: input and output delay Wire delay: extracted (STAR-RCXT)

transition. Input and output delays are also factored into consideration. These constraints, coupled with wiring delay estimation in the synthesis process, tightens the window in each clock period that can be allocated to gate delays. Initial clock period is adjusted and pipeline boundaries are fine tuned to balance the stages. The procedure can be iterated. In producing the mapped gate-level (standard cells) RTL, a timing annotation file is generated which specifies the delay through each wire and standard cell instance. The gate-level RTL is verified for functional equivalence in Mentor ModelSim. The model of the on-chip Gaussian noise generator is implemented in Matlab, so the noise sequence can be reproduced in Matlab simulation. Random test vectors are generated at different SNR points and the correspondence is checked between the Matlab reference model and the functional simulation in ModelSim.

Synopsys Astro performs placement with the objective of optimizing timing while minimizing area. Following the initial placement, clock tree is inserted according to the timing specification. The clock signal is first brought in from the input pads of the chip to approximately the center of the chip. It then branches out in a tree structure to drive the clocked elements. Clock insertion delay accounts for the propagation delay from the edge of the chip to the center of the chip, as well as the subsequent fanout and propagation delays through the tree stages. The worst-case clock insertion delay in a large design can be in excess of one clock period, and a significant amount of clock skew needs to be budgeted. The structure of the clock tree, such as the fanout and location of each intermediate node, is also dependent on the clock transition time specification. The optimal clock transition time should be set to allow for the most efficient fanout in achieving the minimal propagation time, and fanout-4 can be used as a rule of thumb. Placement is further optimized after clock tree insertion with extracted timing characteristics, including clock tree insertion delay, skew, and transition time.

Accurate wire delays can be extracted in placement and routing. The discrepancies between estimated timing in the early stage of the design and extracted timing in the later stage of the design are resolved by adjusting the clock period and re-balancing the pipeline stages, a procedure known as register retiming. For example, only one compare-select operation is allocated in the final pipeline stage of the CN node to make a sufficient timing margin for the wire delay from CN to VN. Unlike in smaller and well-contained designs, register retiming in the LDPC decoder design cannot be easily performed by software tools. Instead, retiming is done manually, relying on heuristics and iterative cycles to resolve the timing bottlenecks.

The final chip layout is extracted to obtain the parasitic capacitance and resistance using Synopsys Star-RCXT. The parasitics are back-annotated in the post-layout RTL for timing verification using Synopsys PrimeTime – a static timing analysis (STA) tool. The functional equivalence is verified in ModelSim with back-annotated timing against the Matlab reference model.

5.3 Chip Implementation

The decoder is implemented in 65nm 7-metal low-power CMOS technology. An initial density of 80% is used in placement and routing to produce the final density of 84.5% in a 5.35 mm^2 core. The decoder occupies 5.05 mm^2 of area, while the remaining 0.30 mm^2

is dedicated to AWGN noise generation, error collection, and I/O compensation for process, voltage, and temperature conditions.

The chip microphotograph is shown in Fig. 5.15, featuring a dimension of 2.556×2.608 mm for a chip area of 6.67 mm^2. The nominal core supply voltage is 1.2 V and I/O supply voltage is 1.8 V. The core is surrounded by 137 I/O pads, with one pair of high-speed low-voltage differential signaling (LVDS) pads feeding the clock signal differentially from an external source, and the rest being standard digital pads. 24 pairs of the pads carry core VDD and GND supplies and 9 pairs carry I/O supplies. The supply pads are interlaced in the pad ring at regular intervals. The number of core supply pairs is determined by the core current consumption to reduce the voltage drop (IR drop) and supply noise ($L\frac{di}{dt}$ noise). To further reduce the voltage drop and supply noise, on-chip metal layers, M6 and M7, are entirely dedicated to core VDD and GND routing in a dense mesh structure. The number of I/O supply pads is determined by the current consumption of the I/O pads and their switching activities.

The chip is packaged in a 144-pin Ceramic Pin Grid Array (PGA) package. A test board, shown in Fig. 5.16, is designed to provide the appropriate supplies to the chip. The clock signal is generated externally using an Agilent E4438C ESG Vector Signal Generator [1] and plugged to the test board through an SMA connection. The input clock is converted to balanced differential signals on board and the common mode is shifted to 0.9 V before they are routed to the LVDS receivers on-chip. The input (output) signals are routed to (from) the test board through an LVDS interface, known as the Z-Dok connector [64].

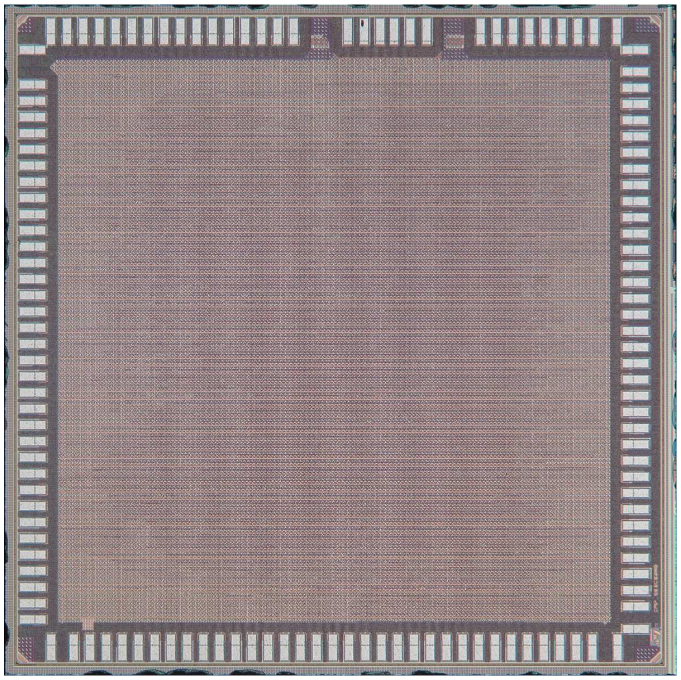


Figure 5.15: Chip microphotograph.

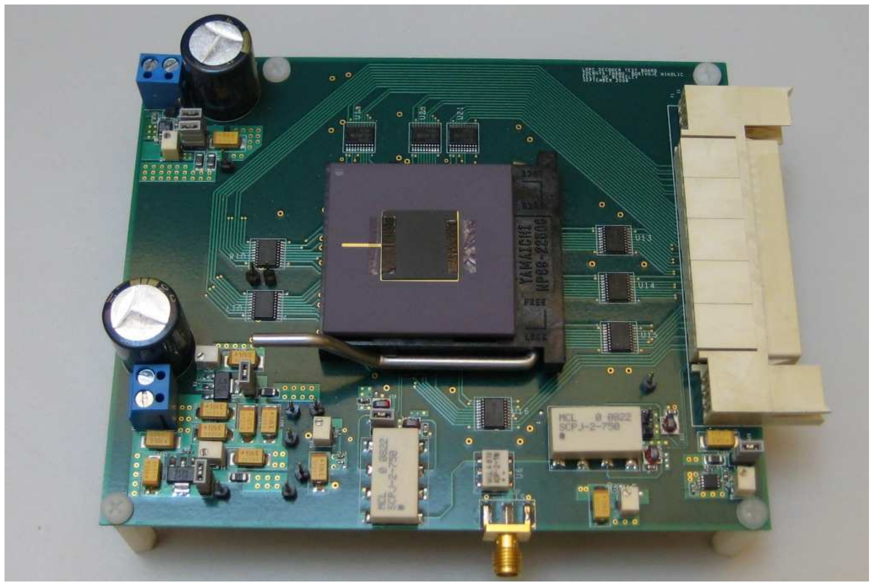


Figure 5.16: Printed circuit board hosting the packaged chip.

5.3.1 Chip Testing Setup

A versatile, internally-developed FPGA board, known as the interconnect breakout board (IBOB) [7], is programmed to be the equivalent logic analyzer that can be attached to the chip test board through the Z-Dok connector. The input clock signal is split into two, one drives the chip, and the other is routed through the Z-Dok connection to drive the IBOB. The chip clock and the IBOB clock are provided by the same external clock source. In addition, the phase of the IBOB clock can be tuned in 90 ° increments for phase alignment. The IBOB operates at a 2.5 V supply voltage; the common mode of the IBOB clock signal needs to be shifted to 1.25 V and all the inputs and outputs of FPGA board also need to be voltage shifted to remain compatible.

The IBOB can be programmed in the same way as the BEE2 system with the Simulink-based design entry. The board hosts a Xilinx Virtex II Pro XC2VP50 FPGA [70], which contains two PowerPC microprocessors on-chip. Registers and block RAMs in the FPGA can be address-mapped and accessed through the PowerPC microprocessor. In the simplest setup, the registers can be programmed on the FPGA to connect to the corresponding Z-Dok pins. A subsequent write operation (to the register) functions as an input to the chip under test and a read operation (from the register) functions as an output from the chip under test. This simplest form is used in automated testing, where the control signals (start, load, reset) and configuration (limit on iteration count, SNR level, limit on the number of input frames) are set via the PowerPC microprocessor. The progress of decoding (number of input frames processed) can be monitored by polling the corresponding registers. Decoding results (bit error count and frame error count) are automatically collected by the decoder chip and can be read from the respective registers on the FPGA.

In a more elaborate testing scheme, the FPGA is programmed to generate the input data (scanned in). A functionally-equivalent LDPC decoder (of a much lower throughput due to resource limitation) is programmed on the FPGA, which runs concurrently with the decoder chip. The output from the chip (through output scan chains) is compared to the on-FPGA emulation to check for errors. This elaborate testing scheme enables more flexibility of operating on any codeword, however the decoder needs to be paused in waiting for scan-in and scan-out to complete loading and unloading, resulting in a much lower decoding throughput.

5.3.2 Functional Measurements

Automated functional testing has been used to collect error counts at a range of SNR levels to generate the waterfall curve. Early termination is adopted to boost the decoding throughput while the maximum iteration limit is set to 20 for regular decoding. Without post-processing, the waterfall curve displays a change of slope below the BER of 10^{-11}. After enabling post-processing, the error floor is lowered. An excellent error correction performance is measured below the BER of 10^{-14}, as shown in Fig. 5.17. The measured waterfall curve matches the performance obtained from hardware emulation shown in Fig. 4.13 with extended BER by more than two orders of magnitude at high SNR levels. The post-processor suppresses the error floor by eliminating the absorbing errors, which is evidenced in Table 5.5. In fact, five of the seven unresolved errors at the highest SNR point (5.2 dB) measured are due to undetected errors – errors that are valid codewords, but not the intended codeword. It was empirically discovered that the minimum number of bitwise

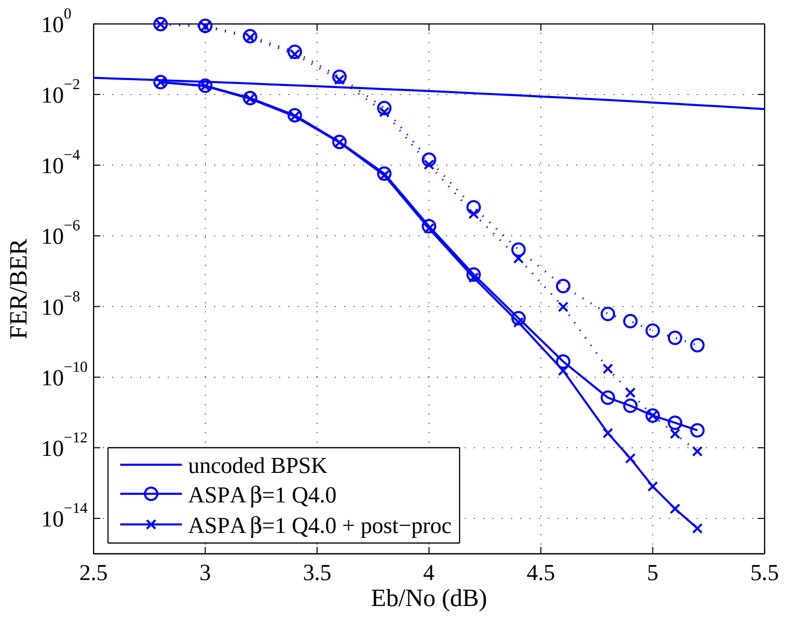


Figure 5.17: Measured FER (dotted lines) and BER (solid lines) performance of the decoder chip using a maximum of 20 decoding iterations.

decoding errors of an undetected error (or minimum distance) is 14 for the $(2048, 1723)$ RS-LDPC code. The eventual elimination of absorbing errors and the emergence of weight-14 undetected errors indicate the near maximum-likelihood decoding performance.

5.3.3 Power Measurements

The decoder chip operates at a maximum clock frequency of 700 MHz with the nominal 1.2 V supply, which delivers a throughput of 47.7 Gb/s. The throughput is measured at an SNR level of 5.5 dB with early termination enabled on-chip. To achieve the required 6.67 Gb/s of throughput for 10GBASE-T Ethernet, the chip can be frequency and voltage scaled to operate at 100 MHz in a 0.7 V supply, resulting in a power dissipation of 144 mW. At the maximum allowed supply voltage of 1.32 V, a decoding throughput of 53.3

Table 5.5: Error statistics based on chip measurements.

SNR (dB)	Before post-processing		After post-processing	
	Errors	Average weight	Errors	Average weight
4.8	3396	8.72	95	30.66
4.9	4229	8.23	40	28.20
5.0	4553	8.08	18	20.22
5.1	5714	8.04	11	15.36
5.2	7038	8.01	7	13.43

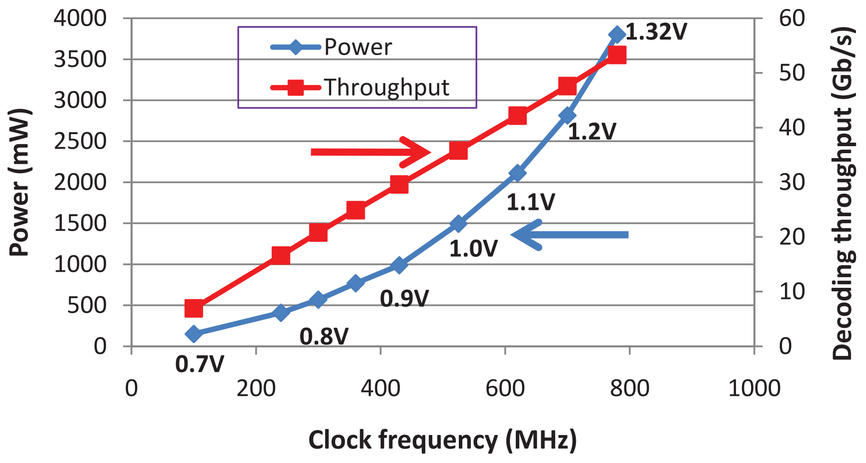


Figure 5.18: Frequency and power measurement results of the decoder chip.

Gb/s is achieved at the clock frequency of 780 MHz.

The maximum clock frequency and decoding throughput are measured at each supply voltage level. The measurements are performed by fixing the supply voltage while ramping up the clock frequency until the FER and BER performance start to deviate. The power consumption and decoding throughput are plotted against the clock frequency in Fig. 5.18. Quadratic power savings can be achieved by the simultaneous voltage and frequency scaling. It is therefore more power efficient to operate at the lowest supply voltage and

Table 5.6: Chip features.

Technology	ST 65nm low-power CMOS	
Core area	2.316×2.311 mm	
Chip area	2.556×2.608 mm	
Area utilization	80% initial, 84.5% final	
Clock Frequency	100 MHz	700 MHz
Supply Voltage	0.7 V	1.2 V
Power Consumption	144 mW	2.80 W
Decoding Throughput[1]	6.67 Gb/s	47.7 Gb/s
Decoding Latency[2]	960 ns	137 ns
Energy Efficiency	21.5 pJ/bit	58.7 pJ/bit

[1] At SNR = 5.5 dB with early termination.

[2] Assumes an 8-iteration decoding limit for regular decoding at a low SNR point. The worst latencies are longer if the post-processing is accounted for.

clock frequency to deliver the required throughput within this range of operation.

The features of the decoder chip are summarized in Table 5.6. At the nominal supply voltage and the maximum 700 MHz of clock frequency, the decoder experiences the worst latency of 137 ns assuming an 8-iteration regular decoding limit, or 206 ns if an additional 4-iteration post-processing is accounted for. The energy per coded bit reaches 58.7 pJ/bit. At the 100 MHz clock frequency and a 0.7 V supply voltage, the worst latency is 960 ns (or 1440 ns with a 4-iteration post-processing), but the energy per coded bit is reduced to 21.5 pJ/bit. These implementation results compare favorably to the state-of-the-art high-throughput LDPC decoder implementations.

Chapter 6

Conclusion

Flexible, high-throughput architectures are proposed to allow the mapping of a family of high performance LDPC decoders on an emulation platform. This emulation platform has been demonstrated to be capable of capturing low BER traces down to 10^{-12} for a $(2048, 1723)$ RS-LDPC code and a $(2209, 1978)$ array-based LDPC code.

6.1 Advances

Error traces were analyzed to show that a class of combinatorial structures known as absorbing sets ultimately determine the error floor performance of these LDPC codes. This study also established the connection between fixed-point quantization choices and the error floor performance of a sum-product decoder: in a low-resolution implementation, the dominant cause of the error floor is oscillatory behavior, which can be corrected with an increase in resolution, or, more effectively, an increase in range, whereas absorbing sets dominate error floors in a high-range implementation and are due to the code construction.

Investigations based on the $(2209, 1978)$ array-based LDPC code allows further isolation of weak from strong absorbing sets. The conventional quantization schemes applied to the sum-product decoder can be suboptimal, thus allowing weak absorbing sets of relatively small size to dominate, thereby leading to an elevated error floor. The proposed dually-quantized sum-product decoder improves the estimation of log-tanh functions, and the approximate sum-product decoder eliminates the log-tanh functions altogether. Both approaches mitigate the effects of weak absorbing sets and lower the error floor even with a small number of decoding iterations.

Even with these improved approaches, strong absorbing sets ultimately determine the error floor performance of some LDPC codes. The well-defined structure of absorbing sets motivated the design of a post-processor to tackle them using the message-passing algorithm. The proposed message biasing scheme boosts the reliabilities of messages from unsatisfied checks and weakens the reliabilities of messages from the satisfied checks, so that the bits of the absorbing set can reverse the wrong values. The offset value ϵ is set small enough to minimize the negative impact on the correct bits and the optimal ϵ can be reached through run-time adaptation. A good low error rate decoding performance can be achieved with post-processing due to the elimination of the absorbing errors. The post-processing algorithm converges quickly, usually within 3 or 4 iterations.

The efficient implementation of the post-processor allows it to be easily padded on an existing decoder with no change to the scheduling and only a minor fix to the decoding algorithm. The post-processor further shortens the minimum wordlength required to achieve a good decoding performance, thereby improving the area and power efficiencies. A grouping strategy is applied in the architectural design to divide wires into global wires and local wires, such that all wiring irregularities are kept within local groups and all global wires are kept regular and structured. The optimal architecture lies in the point where the incremental wiring overhead per additional processing element reaches the minimum, which coincides with the balance point between area and throughput. The decoder with this architecture is synthesized, placed and routed to achieve a 84.5% density without sacrificing the maximum clock frequency. The message-passing decoding is scheduled based on a 7-stage pipeline to deliver a high effective throughput.

The optimized decoder architecture, when aided by an early termination scheme, achieves a maximum 47.7 Gb/s decoding throughput at the nominal supply voltage. The high throughput capacity allows the voltage and frequency to be scaled to realize quadratic power savings. Automated functional testing with real-time noise generation and error collection extends the BER measurements below 10^{-14}, where no error floor is observed. Measurements show that the dominant absorbing errors are completely eliminated, and the decoding performance approaches the maximum-likelihood decoder as the majority of the residual errors are due to the minimum-distance codewords.

6.2 Future Work

Intuitions gained from this work enable further advancement of LDPC decoder designs for a wide range of applications. An example is the LDPC decoding for digital magnetic recording, where the AWGN channel assumption no longer holds. The magnetic storage system utilizes partial-response signaling in combination with maximum-likelihood sequence detection (PRML) [13]. The channel characteristics can alter the significance of each type of decoding error, and the effect on the error floor level is yet to be determined.

The error-floor-lowering post-processing approach was derived based on heuristics in this work. A better understanding can be developed to both qualitatively and quantitatively justify the effectiveness of this approach. An interesting direction is to apply a deterministic technique, such as the absorbing region analysis [21], in selecting the optimal message bias and estimating performance bounds.

The decoder chip design experience can be extended to many other high-throughput applications, including data storage, optical communications, and high-speed wireless. One drawback of the current design is the lack of reconfigurability, which becomes essential in an application such as the 60-GHz short-range wireless communications [35], where three different LDPC codes have been included in one standard. How to maintain a low power consumption and a high decoding throughput while accommodating multiple codes is the key to these applications.

A hardware emulation platform has been applied extensively in this work for the iterative exploration of problems that are out-of-reach by conventional simulations. Such a design platform can be successfully exploited in building complex communication, signal processing, and computation systems. The unified emulation and chip design flow enforces the implementation compatibility, so every solution obtained from investigation can be realized in practice. The gap between theoretical investigation and practical implementation can be effectively bridged.

Bibliography

[1] E4438C ESG Vector Signal Generator. Agilent Technologies. [Online]. Available: http://www.home.agilent.com/agilent/product.jspx?nid=-536902340.536880956.00

[2] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," *IEEE Communications Letters*, vol. 6, no. 5, pp. 208–210, May 2002.

[3] K. S. Andrews, D. Divsalar, S. Dolinar, J. Hamkins, C. R. Jones, and F. Pollara, "The development of turbo and LDPC codes for deep-space applications," *Proceedings of the IEEE*, vol. 95, no. 11, pp. 2142–2156, Nov. 2007.

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

[5] G. Bosco, G. Montorsi, and S. Benedetto, "Decreasing the complexity of LDPC iterative decoders," *IEEE Communications Letteres*, vol. 9, no. 7, pp. 634–636, Jul. 2005.

[6] A. I. V. Casado, M. Griot, and R. D. Wesel, "Informed dynamic scheduling for belief-

propagation decoding of LDPC codes," in *Proc. IEEE International Conference on Communications*, Glasgow, UK, Jun. 2007, pp. 932–937.

[7] IBOB. Center for Astronomy Signal Processing and Electronics Research. [Online]. Available: http://casper.berkeley.edu/wiki/IBOB

[8] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: a high-end reconfigurable computing system," *IEEE Design and Test of Computers*, vol. 22, no. 2, pp. 114–125, Mar. 2005.

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

[10] J. Chen and M. P. C. Fossorier, "Density evolution for two improved BP-based decoding algorithms of LDPC codes," *IEEE Communications Letters*, vol. 6, no. 5, pp. 208–210, May 2002.

[11] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasic, "Error floors of LDPC codes on the binary symmetric channel," in *Proc. IEEE International Conference on Communications*, Istanbul, Turkey, Jun. 2006, pp. 1089–1094.

[12] S. Chung, G. D. Forney, Jr., T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Letters*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[13] R. D. Cideciyan, F. Dolivo, R. Hermann, W. Hirt, and W. Schott, "A PRML system

for digital magnetic recording," *IEEE Journal on Selected Areas in Communications*, vol. 10, no. 1, pp. 38–56, Jan. 1992.

[14] C. A. Cole and S. G. Wilson, "Message passing decoder behavior at low error rates," *IEEE Transactions on Communications*, submitted for publication.

[15] C. A. Cole, S. G. Wilson, E. K. Hall, and T. R. Giallorenzi, "A general method for finding low error rates of LDPC codes," *IEEE Transactions on Information Theory*, submitted for publication.

[16] D. J. Costello, Jr. and G. D. Forney, Jr., "Channel coding: the road to channel capacity," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1150–1177, Jun. 2007.

[17] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Power reduction techniques for LDPC decoders," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 8, pp. 1835–1845, Aug. 2008.

[18] W. R. Davis, N. Zhang, K. Camera, D. Markovic, T. Smilkstein, M. J. Ammer, E. Yeo, S. Augsburger, B. Nikolić, and R. W. Brodersen, "A design environment for high-throughput low-power dedicated signal processing systems," *IEEE Journal of Solid State Circuits*, vol. 37, no. 3, pp. 420–431, Mar. 2002.

[19] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[20] 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," *IEEE Communications Letters*, vol. 7, no. 7, pp. 317–319, Jul. 2003.

[21] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolić, and M. J. Wainwright, "Predicting error floors of LDPC codes: deterministic bounds and estimates," *IEEE Journal on Selected Areas in Communications*, to be published.

[22] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolić, "Analysis of absorbing sets for array-based LDPC codes," in *Proc. IEEE International Conference on Communications*, Glasgow, UK, Jun. 2007, pp. 6261–6268.

[23] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolić, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Transactions on Information Theory*, 2009, to be published.

[24] L. Dolecek, Z. Zhang, M. J. Wainwright, V. Anantharam, and B. Nikolić, "Evaluation of the low frame error rate performance of LDPC codes using importance sampling," in *Proc. IEEE Information Theory Workshop*, Lake Tahoe, CA, Sep. 2007, pp. 202–207.

[25] *ETSI Standard TR 102 376 V1.1.1: Digital Video Broadcasting (DVB) User guidelines for the second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*, ETSI Std. TR 102 376, Feb. 2005.

[26] J. L. Fan, "Array codes as low-density parity-check codes," in *Proc. International Symposium on Turbo Codes and Related Topics*, Brest, France, Sep. 2000, pp. 543–546.

[27] G. D. Forney, Jr., "Codes on graphs: normal realizations," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.

[28] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, “Reduced complexity iterative decoding of low-density parity check codes based on belief propagation,” *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, May 1999.

[29] R. G. Gallager, *Low-Density Parity-Check Codes.* Cambridge, MA: MIT Press, 1963.

[30] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[31] Y. Han and W. E. Ryan, “LDPC decoder strategies for achieving low error floors,” in *Proc. Information Theory and Applications Workshop*, San Diego, CA, Jan. 2008.

[32] J. Heo and K. M. Chugg, “Optimization of scaling soft information in iterative decoding via density evolution methods,” *IEEE Transactions on Communications*, vol. 53, no. 6, pp. 957–961, Jun. 2005.

[33] D. E. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes,” in *Proc. IEEE Workshop on Signal Processing Systems*, Austin, TX, Oct. 2004, pp. 107–112.

[34] X. Hu, E. Eleftheriou, and D. Arnold, “Progressive edge-growth Tanner graphs,” in *Proc. IEEE Global Communications Conference*, San Antonio, TX, Nov. 2001, pp. 995–1001.

[35] IEEE 802.15 WPAN Task Group 3c (TG3c) Millimeter Wave Alternative PHY. IEEE. [Online]. Available: http://www.ieee802.org/15/pub/TG3c.html

[36] *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.

[37] *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*, IEEE Std. 802.16e, Feb. 2006.

[38] *IEEE Draft Standard for Information Technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment : Enhancements for Higher Throughput*, IEEE Std. 802.11n/D2.00, Feb. 2007.

[39] A. Kavčić and A. Patapoutian, "The read channel," *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1761–1774, Nov. 2008.

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

[41] S. Ländner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes," in *Proc. IEEE International Wireless Communications and Mobile Computing Conference*, Maui, HI, Jun. 2005, pp. 630–635.

[42] D. U. Lee, W. Luk, J. D. Villasenor, and P. Y. K. Cheung, "A Gaussian noise generator for hardware-based simulations," *IEEE Transactions on Computers*, vol. 53, no. 12, pp. 1523–1534, Dec. 2004.

[43] C. Liu, S. Yen, C. Chen, H. Chang, C. Lee, Y. Hsu, and S. Jou, "An LDPC decoder chip based on self-routing network for IEEE 802.16e applications," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 3, pp. 684–694, Mar. 2008.

[44] H. Liu, C. Lin, Y. Lin, C. Chung, K. Lin, W. Chang, L. Chen, H. Chang, and C. Lee, "A 480Mb/s LDPC-COFDM-based UWB baseband transceiver," in *Proc. IEEE International Solid-State Circuits Conference*, San Francisco, CA, Feb. 2005, pp. 444–445.

[45] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[46] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[47] D. J. C. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, pp. 97–104, Oct. 2003.

[48] M. M. Mansour and N. R. Shanbhag, "Low-power VLSI decoder architectures for LDPC codes," in *Proc. International Symposium on Low Power Electronics and Design*, Monterey, CA, Aug. 2002, pp. 284–289.

[49] ——, "Turbo decoder architectures for low-density parity-check codes," in *Proc. IEEE Global Communications Conference*, Taipei, Taiwan, Nov. 2002, pp. 1383–1388.

[50] ——, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, Mar. 2006.

[51] Y. Mao and A. H. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," in *Proc. IEEE International Conference on Communications*, Helsinki, Finland, Jun. 2001, pp. 41–44.

[52] O. Milenkovic, E. Soljanin, and P. Whiting, "Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles," *IEEE Transactions on Information Theory*, vol. 53, no. 1, pp. 39–55, Jan. 2007.

[53] A. Morello and V. Mignone, "DVB-S2: the second generation standard for satellite broad-band services," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 210–227, Jan. 2006.

[54] T. Richardson, "Error floors of LDPC codes," in *Proc. Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Oct. 2003, pp. 1426–1435.

[55] T. Richardson and R. Urbanke, "The renaissance of Gallager's low-density parity-check codes," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 126–131, Aug. 2003.

[56] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

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

[58] J. Rosenthal and P. O. Vontobel, “Construction of LDPC codes based on Ramanujan graphs and ideas from Margulis,” in *Proc. Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Oct. 2000, pp. 248–257.

[59] V. Savin, “Iterative LDPC decoding using neighborhood reliabilities,” in *Proc. IEEE International Symposium on Information Theory*, Nice, France, Jun. 2007, pp. 221–225.

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

[61] L. Sun, H. Song, Z. Keirn, and B. Kumar, “Field programmable gate array (FPGA) for iterative code evaluation,” *IEEE Transactions on Magnetics*, vol. 42, no. 2, pp. 226–231, Feb. 2006.

[62] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, “LDPC block and convolutional codes based on circulant matrices,” *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[63] T. Tian, C. R. Jones, J. D. Villasenor, and R. D. Wesel, “Selective avoidance of cycles in irregular LDPC code construction,” *IEEE Transactions on Communications*, vol. 52, no. 8, pp. 1242–1247, Aug. 2004.

[64] Z-DOK Product Line Information. Tyco Electronics. [Online]. Available: http://www.tycoelectronics.com/catalog/cinf/en/c/21138/0?RQS=

[65] P. Urard, L. Paumier, V. Heinrich, N. Raina, and N. Chawla, “A 360mW 105Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes enabling satellite-

transmission portable devices," in *Proc. IEEE International Solid-State Circuits Conference*, San Francisco, CA, Feb. 2008, pp. 310–311.

[66] P. Urard, E. Yeo, L. Paumier, P. Georgelin, T. Michel, V. Lebars, E. Lantreibecq, and B. Gupta, "A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes," in *Proc. IEEE International Solid-State Circuits Conference*, San Francisco, CA, Feb. 2005, pp. 446–447.

[67] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Linköping University, Linköping, Sweden, 1996.

[68] H. Xiao and A. H. Banihashemi, "Estimation of bit and frame error rates of finite-length low-density parity-check codes on binary symmetric channels," *IEEE Transactions on Communications*, vol. 55, no. 12, pp. 2234–2239, Dec. 2007.

[69] Xilinx Additive White Gaussian Noise. Xilinx Corporation. [Online]. Available: http://www.xilinx.com/products/ipcenter/DO-DI-AWGN.htm

[70] Xilinx Virtex Series FPGA. Xilinx Corporation. [Online]. Available: http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/

[71] L. Yang, H. Liu, and R. Shi, "Code construction and FPGA implementation of capacity approaching low error-floor LDPC decoder," *IEEE Transactions on Circuits and Systems-I: Regular Papers*, vol. 53, no. 4, pp. 892–904, Apr. 2006.

[72] M. Yang, W. E. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate irregular LDPC codes," *IEEE Transactions on Communications*, vol. 52, no. 4, pp. 564–571, Apr. 2004.

[73] E. Yeo and B. Nikolić, "A 1.1-Gb/s 4092-bit low-density parity-check decoder," in *Proc. IEEE Asian Solid-State Circuits Conference*, Hsinchu, Taiwan, Nov. 2005, pp. 237–240.

[74] E. Yeo, B. Nikolić, and V. Anantharam, "Iterative decoder architectures," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 132–140, Aug. 2003.

[75] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, Feb. 2005.

[76] T. Zhang and K. K. Parhi, "A 54 Mbps (3,6)-regular FPGA LDPC decoder," in *Proc. IEEE Workshop on Signal Processing Systems*, Antwerp, Belgium, Sep. 2001, pp. 25–36.

[77] Z. Zhang, L. Dolecek, B. Nikolić, V. Anantharam, and M. Wainwright, "Investigation of error floors of structured low-density parity-check codes by hardware emulation," in *Proc. IEEE Global Communications Conference*, San Francisco, CA, Nov. 2006, pp. 1–6.

[78] Z. Zhang, L. Dolecek, B. Nikolić, V. Anantharam, and M. J. Wainwright, "Lowering LDPC error floors by postprocessing," in *Proc. IEEE Global Communications Conference*, New Orleans, LA, Nov. 2008, pp. 1–6.

[79] ——, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Transactions on Communications*, to be published.

[80] Z. Zhang, L. Dolecek, M. Wainwright, V. Anantharam, and B. Nikolić, "Quantization

effects of low-density parity-check codes," in *Proc. IEEE International Conference on Communications*, Glasgow, UK, Jun. 2007, pp. 6231–6237.

[81] Z. Zhang, B. Nikolić, V. Anantharam, and M. J. Wainwright, "A 47 Gb/s LDPC decoder with improved low error rate performance," in *Proc. Symposium on VLSI Circuits*, Kyoto, Japan, Jun. 2009, pp. 286–287.

[82] 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 Transactions on Communications*, vol. 53, no. 4, pp. 549–554, Apr. 2005.