Algorithms and Architectures for Efficient Low Density Parity Check (LDPC) Decoder Hardware

By

TINOOSH MOHSENIN
B.S. (Sharif University of Technology) 1999
M.S. (Rice University) 2004

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL and COMPUTER ENGINEERING

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Bevan M. Baas, Chair

Shu Lin

Venkatesh Akella

Rajeevan Amirtharajah

Committee in charge

2010

Abstract

Many emerging and future communication applications require a significant amount of high throughput data processing and operate with decreasing power budgets. This need for greater energy efficiency and improved performance of electronic devices demands a joint optimization of algorithms, architectures, and implementations.

Low Density Parity Check (LDPC) decoding has received significant attention due to its superior error correction performance, and has been adopted by recent communication standards such as 10GBASE-T 10 Gigabit Ethernet. Currently high performance LDPC decoders are designed to be dedicated blocks within a System-on-Chip (SoC) and require many processing nodes. These nodes require a large set of interconnect circuitry whose delay and power are wire-dominated circuits. Therefore, low clock rates and increased area are a common result of the codes' inherent irregular and global communication patterns. As the delay and energy costs caused by wires are likely to increase in future fabrication technologies new solutions dealing with future VLSI challenges must be considered.

Three novel message-passing decoding algorithms, Split-Row, Multi-Split and Split-Row Threshold are introduced, which significantly reduce processor logical complexity and local and global interconnections. One conventional and four Split-Row Threshold LDPC decoders compatible with the 10GBASE-T standard are implemented in 65 nm CMOS and presented along with their tradeoffs in error correction performance, wire interconnect complexity, decoder area, power dissipation, and speed. For additional power saving, an adaptive wordwidth decoding algorithm is proposed which switches between a 6-bit *Normal Mode* and a reduced 3-bit *Low Power Mode* depending on the SNR and decoding iteration.

A 16-way Split-Row Threshold with adaptive wordwidth implementation achieves improvements in area, throughput and energy efficiency of 3.9x, 2.6x, and 3.6x respectively, compared to a MinSum Normalized implementation, with an SNR loss of 0.25 dB at BER = 10^{-7}. The decoder occupies a die area of 5.10 mm^2, operates up to 185 MHz at 1.3 V, and attains an average throughput of 85.7 Gbps with early-termination. Low power operation at 0.6 V gives a worst case throughput of 9.3 Gbps–above the 6.4 Gbps 10GBASE-T requirement, and an average power of 31 mW.

To my lovely husband and my parents for their love and support

Acknowledgments

The road to finish my PhD has been enriching, fun and challenging. Getting here would have not been possible without the support that I have received from many. The list is too long to recall every single name to put down on paper as I am writing these lines. I would like to take this chance though to thank certain individuals who have been influential specifically on the process of completion of this dissertation.

First and foremost, my sincere gratitude goes to my advisor and mentor, Professor Bevan Baas for his valuable guidance and advice throughout my PhD study. Besides knowledge, he provided me the confidence I always needed to persist throughout the hardest times. Like a father, his kindness and support have greatly inspired me and shall substantially influence the rest of my career. I am grateful to Professor Shu Lin for introducing me to the subject of this work, helping me with the theory, for being supportive and critical to my work and elevating this research to the next level. I would like to thank Professor Venkatesh Akella for his constant support and advice throughout my PhD study and specially for the past few months of finishing my work. Many thanks to Professor Rajeevan Amirtharajah for his enormous support and advice during my faculty interview preparation and for reviewing my dissertation. Also thanks to Dr. Soheil Ghiasi for his advice during my faculty interview preparation and for evaluating my research proposal.

I would like to thank Intel Corporation, Intellasys Corporation, National Science Foundation (grant No. 0430090 and CAREER Award 0546907), UC MICRO, ST Microelectronics, SRC, and UC Davis Faculty Research Grant, for their generous financial donations to our research. Also, thanks to Jean-Pierre Schoellkopf for giving me permission to use ST Microelectronics libraries and Pascal Urard for his valuable advices on my work.

Particularly, I would like to thank my great friend Dean for his endless support, his sleepless nights working on papers with me and for being so patient with me during past few years. I could not finish this path without him. I would like to express my appreciation to my friends Qin and Lan for their significant help with LDPC code constructions and teaching me coding theory. Many thanks to Houshmand for being such a great friend and his hard work and support during past few months to finish our last journal paper.

My special thanks goes to my dearest friends Elham, Ladan, Farinaz and Sepideh for their constant support, advice and encouragement which helped me keep going during difficult times. I also would like to thank my other friends who have turned my experience in Davis to a memorable one. They include but are not limited to: Kaveh, Mehdi, Mahnoosh, Shadi, Neda, Sharegheh, Ladan, Arash, Pouya, Faramarz, Navid, Mina, Marjan, Matin, Mohammad, Sumei, Liping, Jovana, Milena, Zhibin, Xiaoheng, Jon, Trevin, Aaron, Lucas, Bin, Anh, Travis, Justin, Stanley, Stevan, Frank, Maggie and Jenny.

I would like to express my deep appreciation to my beloved husband, Arash to whom this dissertation is dedicated. I cannot overstate the importance of such a supportive family to my academic career. Ultimately I owe many thanks to my parents and my sister Farnoosh whose endless supports and sacrifices have been the source of my courage and persistence all the time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Communication systems are becoming a standard requirement of every computing platform from wireless sensors, mobile telephony, and server class computers. Local and cellular wireless communication throughputs are expected to increase to hundreds of Mbps and even beyond 1 Gbps [15, 38, 51]. With this increased growth for bandwidth comes larger system integration complexity and higher energy consumption per packet. Low power design is therefore a major design criterion alongside the standards' throughput requirement as both will determine the quality of service and cost.

Error correction plays a major role in communication and storage systems to increase the transmission reliability and achieve a better error correction performance with less signal power. Low Density Parity Check (LDPC) code was first developed in 1962 [25] as an error correction code that allowed communication over noisy channels possible near the Shannon limit. With advancements in VLSI, LDPC codes have recently received a lot of attention because of their superior error correction performance when decoded iteratively using a message passing algorithm [45]. As the result, they have been adopted as the forward error correction method for many recent standards such as digital video broadcasting via satellite (DVB-S2) [5], the WiMAX standard for microwave communications (802.16e) [3], the G.hn/G.9960 standard for wired home networking [2], and the 10GBASE-T standard for 10 Gigabit Ethernet (802.3an) [4].

1.1 Challenges and Related Work

So far the emerging 10GBASE-T standard has not been adopted as quickly as predicted into the data center infrastructures because of the power constraints [76]. The power consumption of the 10GBASE-T PHY layer (more specifically the receiver, whose implementation is left open by the 802.3an standard [4]) has become difficult to reduce [60]. Of particular concerns is the LDPC decoder, which must have high throughputs in order for the system to achieve the necessary 10Gb Ethernet bandwidth. For recently fabricated software defined radio system-on-chips (SoC), the LDPC decoders dissipate 14% to 23% of the total average power (PHY+MAC) [15, 38]. Interestingly, the only comparable block on the SoCs with similar power dissipation numbers are the control processors, which consume 18% to 31% of the total chip power. As a result, LDPC decoders are likely to be a major part of a baseband processor's power consumption as communication standards increase their throughput requirements and error correction code complexity (for increased coding gain, spectral efficiency, etc.).

While there has been much research on LDPC decoders and their VLSI implementations, designing a high throughput decoder with low power and small silicon area is still a challenge. LDPC decoder design choices are most often considered at the architecture level and are categorized into two domains: "full-parallel" and "partial-parallel". Full-parallel is a direct implementation of the message-passing algorithm with every computational unit and interconnection between them realized in hardware. Partial-parallel decoders use pipelining, large memory resources and shared computational blocks to deal with the inherent communication complexity and massive bandwidth.

Since the number of operations achievable per cycle is larger with full-parallel decoders, their energy efficiencies and throughput are theoretically the best [18, 51]. Figure 1.1 shows the throughput of reported ASIC designs (measured or post-layout implementations) versus year for full-parallel and partial-parallel decoders. The tick marks along the right side of the plot indicate the maximum throughput requirement for five popular standards. All decoders for DVB-S2, 802.16e and 802.11n standards which require reconfigurable hardware to support different code lengths and code rates are partial-parallel. As shown in the plot,

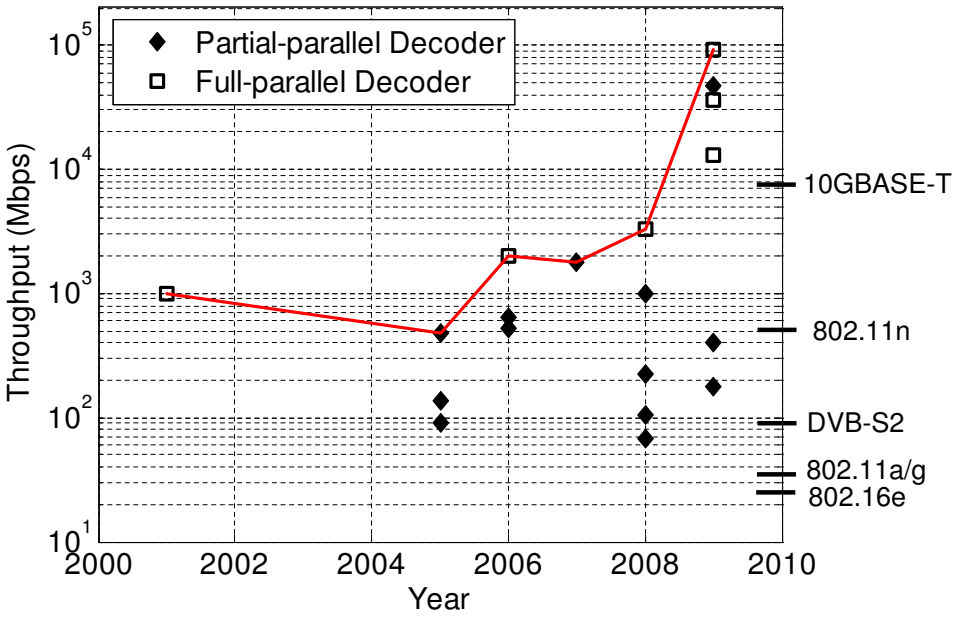


Figure 1.1: Throughput of reported full-parallel and partial-parallel LDPC decoder ASIC implementations versus year

although there are not many reported full-parallel decoder implementations, their throughput is in general higher than that of the partial-parallel decoders. Full-parallel decoders which require many processing nodes typically suffer from large wire-dominated circuits operating at low clock rates due to large critical path delays caused by the codes' inherently irregular and global communication patterns.

Global wires, which make up the long interconnections between distant circuit blocks are increasingly common in system-on-chip designs, and are the predominant source of circuit delay. The common solution to deal with global wire delay is to partition the wire into segments and place repeaters (i.e. buffers) between each segment to decrease the overall delay [61]. This delay optimization will require even more additional buffers as CMOS continues to scale, which consumes more power and causes routing congestion because of the added vias between buffer pins and wire segments [65]. Unlike transistors and local wires, global wires have not reaped the benefits of scaling. The ideal scaling of both wire types results in no RC delay reduction in local wiring while global wiring receives a 50% RC delay increase per year [61]. While feature size scales by 0.7, global wire capacitance, resistivity, and RC delay scale by 0.9, 1.1, and 2.4, respectively, and the contribution of global wire capacitance to dynamic power consumption will be expected to have a 1.1 times

Watt per GHz-cm^2 increase for every generation [31, 65]. [1] Using repeaters also has the unfortunate drawbacks of added power consumption, and added vias between metal layers to and from the buffer and wire segments, which makes the routing problem harder [64].

There have been recent studies to reduce routing congestion and wire delay of the full-parallel decoder implementations through bit-serial communication [18], wire partitioning [59], and floorplan optimization [7, 82]. Bit-serial and wire partitioning use microarchitectural techniques, while floorplan optimization use physical layout back-end techniques. In the bit-serial method [18], messages are transmitted serially in multiple cycles. Although the proposed work results in higher clock frequency, the number of clock cycles required to transmit messages is increased, which overall results in low decoding throughput and energy efficiency.Wire partitioning offers an improvement in reducing differences in communication path delays by pipelining wires, however the clock tree and additional registers increase overall power. For example, in a one frame-per-iteration design, throughput was improved by only 1.3× however, clock power was increased by a factor of 1.8×, which resulted in worse power and negligible (∼1pJ/bit) or no energy improvements [59]. In floorplan optimization a grouping strategy is used to localize irregular wires and regularize global wires. While this method might be applicable for code sizes $<$ 2 Kbits, it requires large backend effort and floorplan design time and also depends on the code structure. Both microarchitecture and floorplan techniques require careful implementation in order to reach the optimal efficiency. In contrast, algorithm techniques trade off error correction performance for improved throughput and power.

1.2 Contributions

1. This work contributes to the solution of interconnect complexity of LDPC decoders by introducing nonstandard decoding algorithms based on SPA and MinSum called "Split-Row" [49], "Multi-Split" [50] and "Split-Row Threshold" [56, 53], which reduce data dependency and inter-processor message passing. The Split-Row and Multi-Split achieve this through partitioning the links needed in the message-passing algorithm,

[1] Inter-buffer distances decrease by a factor of $s\sqrt{s}$, where s is the scaling factor for the transistor. Thus the distance between buffers is shrinking by 0.586 for every generation ($s = 0.7$).

and localize communication. A minimal amount of information is transferred amongst partitions to ensure computational accuracy while reducing global communication. The Split-Row Threshold algorithm largely gains back the loss in error performance of Split-Row by adding an additional form of information based on a comparison with a threshold value (T). Based on this comparison a "threshold enable" bit is sent between partitions. With Split-Row Threshold higher levels of partitioning are possible with a modest SNR loss of 0.05-0.3 dB, when compared to MinSum Normalized.

2. This work investigates the impact of LDPC code matrix properties (e.g. code length and check node degree), and Split-Row-type algorithm characteristics such as the level of partitioning, correction factor value, threshold value and fixed-point representation on the error correction performance [52, 55]. Theories were verified using empirical simulations on approximately 120 networked computers.

3. This work contributes to the solution of the physical layout implementation of full-parallel LDPC decoders [55]. A block partitioning method is used for the layout implementation that naturally comes from Split-Row algorithm. Each block is independently implemented whose internal wires are all relatively short. The blocks are interconnected by a small number of wires. This results in denser, faster and more energy efficient circuits. In addition, it effectively reduces back-end engineering time and effort for full-parallel architectures with large codes (e.g. 2 Kbits to 64 Kbits) and high check node degrees. The benefits of the algorithms and the levels of partitioning on silicon area, clock speed and power are investigated with the results of several place-and-routed standard-cell decoder designs. All decoders were designed using a standard cell flow up to the layout-level just before GDS extraction.

4. This work proposes an adaptive wordwidth algorithm to achieve additional power savings by minimizing unnecessary bit toggling of decoding messages while maximizing bit error performance. The new algorithm takes advantage of data input patterns during the decoding process using Split-Row Threshold. The adaptive micro-architecture is implemented as a small add-on to each processing element with minimal area cost. Depending on the SNR and decoding iteration, different low power settings are deter-

mined to find the best trade off between bit error performance and energy consumption of the decoder chip.

1.3 Organization

The dissertation is organized as follows: Chapter 2 gives an overview of LDPC codes, message passing algorithm, and decoder architectures. The (6,32)-regular (2048,1723) RS-LDPC code [19], which is adopted by 10GBASE-T standard [4] is used as an example for architecture comparisons. Chapter 3 introduces Split-Row and Multi-Split decoding methods, along with an optimal value of correction factor and bit error performance comparisons for different codes. It also describes the hardware implementation of Multi-Split decoder for the 10GBASE-T LDPC code. Chapter 4 studies the Split-Row ability to reduce routing congestion in layout. Then it introduces a low hardware cost modification to Split-Row called Split-Row Threshold that improves error performance while maintaining the former's routing reduction benefits. It investigates the optimum value of threshold (T) and the bit error performance of 10GBASE-T LDPC code using Split-Row Threshold with multiple partitioning. It also presents detailed analysis of post-layout results of full-parallel 10GBASE-T LDPC decoders that implement Split-Row Threshold. Chapter 5 introduces an adaptive wordwidth power reduction method which reduces bit toggling of messages based on the SNR and decoding iteration. Three different implementation methods along with their bit error performance results and details of their architecture are presented. The best trade off between bit error performance and energy consumption of the decoder chip is found for the post-layout implementations of 10GBASE-T LDPC decoders that implement the algorithm.

Chapter 2

Background

2.1 LDPC Codes and Message Passing Decoding Algorithm

LDPC codes are defined by an $M \times N$ binary matrix called the parity check matrix H. The number of columns, N, defines the code length. The number of rows in H, M, defines the number of parity check constraints for the code. The information length K is $K = N - M$ for full-rank matrices, otherwise $K = N - rank$. Column weight W_c is the number of ones per column and row weight W_r is the number of ones per row.

LDPC codes can also be described by a bipartite graph or Tanner graph [69]. The parity check matrix and the corresponding Tanner graph of a $(W_c = 2, W_r = 4)$ $(N = 12, K = 7)$ LDPC code are shown in Fig. 2.1. The rank of the matrix is 5, therefore the information length K is 7. In the graph, there are two sets of nodes: check nodes and variable nodes. Each column of the parity check matrix corresponds to a variable node in the graph represented by V. Each row of the parity check matrix corresponds to a check node in the graph represented by C. There is an edge between a check node C_i and a variable node V_j if the position (i, j) in the parity check matrix is 1, or $H(i, j) = 1$. For example, the first row of the matrix corresponds to C_1 in the Tanner graph which is connected to V_3, V_5, V_8 and V_{10} variable nodes. A variable node which is connected to a check node is called the neighbor variable node. Similarly, a check node that is connected to a variable node is called the neighbor check node. Total number of edges (connections)

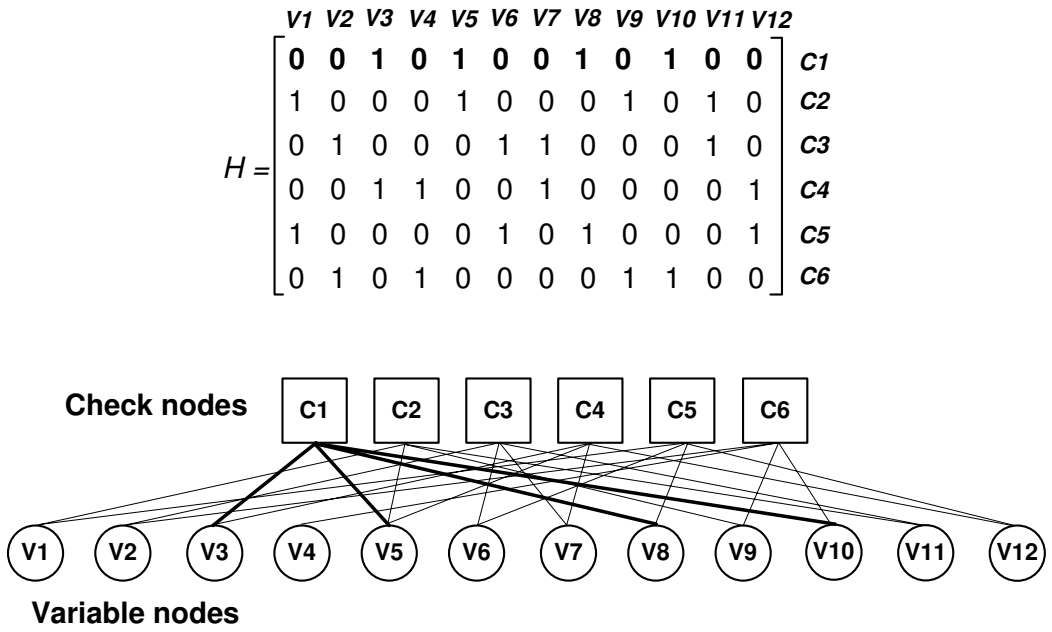


Figure 2.1: Parity check matrix (upper) and Tanner graph (lower) representation of a 12 column (N), 6 row (M), column weight 2 (W_c), row weight 4 (W_r), LDPC code with information length 7 (K).

between variable node and check nodes is $N \times W_c$ or $M \times W_r$. For clearer explanations, in this work, we examine cases where H is *regular* and thus W_r and W_c are constants. For VLSI implementation examples, we use a (6,32)-regular (2048,1723) RS-LDPC code [19] which has been adopted for the forward error correction in the IEEE 802.3an 10GBASE-T standard [4]. The 384×2048 H matrix of the code has a $W_r = 32$ and $W_c = 6$. There are $M = 384$ check nodes and $N = 2048$ variable nodes in its graph representation.

The iterative message-passing algorithm is the most widely used method for practical decoding [39, 44] and its basic flow is shown in Fig. 2.2. After receiving the corrupted information from an AWGN channel (λ), the algorithm begins by processing it and then iteratively corrects the received data. First, all check node inputs are initialized to "0" and then a *check node update* step (i.e. row processing) is done to produce α messages. Second, the variable node receives the new α messages, and then the *variable node update* step (i.e. column processing) is done to produce β messages. This process repeats for another iteration by passing the previous iteration's β messages to the check nodes. The algorithm finally terminates when it reaches a maximum number of decoding iterations ($Imax$) or a valid

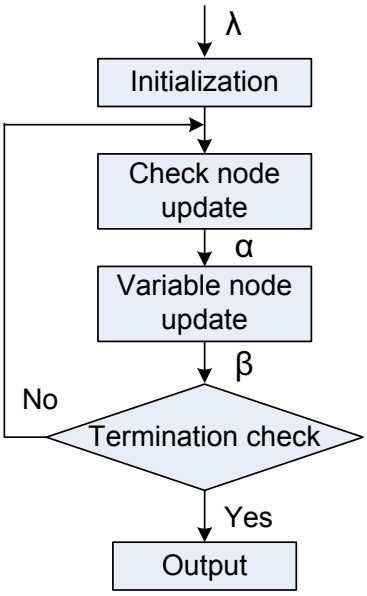


Figure 2.2: Flow diagram of an iterative message-passing decoding algorithm.

code word is detected. Sum-Product (SPA) [43] and MinSum (MS) [22] are near-optimum decoding algorithms which are widely used in LDPC decoders.

2.1.1 Sum Product Algorithm (SPA)

We assume a binary code word $(x_1, x_2, ..., x_N)$ is transmitted using a binary phase-shift keying (BPSK) modulation. Then the sequence is transmitted over an additive white Gaussian noise (AWGN) channel and the received symbol is $(y_1, y_2, ..., y_N)$.

We define $V(i)\backslash j$ as the set of variable nodes connected to check node C_i excluding variable node j. Similarly, we define the $C(i)\backslash j$ as the set of check nodes connected to variable node V_i excluding check node j. For example in Fig. 2.1, $V(1) = \{V_3, V_5, V_8, V_{10}\}$ and $V(1)\backslash 3 = \{V_5, V_8, V_{10}\}$. Also $C(1) = \{C_2, C_5\}$ and $C(1)\backslash 2 = \{C_5\}$. Moreover, we define the following variables which are used throughout this paper.

λ_i is defined as the information derived from the log-likelihood ratio of received symbol y_i,

$$\lambda_i = ln\left(\frac{P(x_i = 0|y_i)}{P(x_i = 1|y_i)}\right) \tag{2.1}$$

α_{ij} is the message from check node i to variable node j. This is the check node processing output.

β_{ij} is the message from variable node j to check node i. This is the variable node processing output.

SPA decoding can be summarized in these four steps:

1. *Initialization:* For each i and j, initialize β_{ij} to the value of the log-likelihood ratio of the received symbol y_j, which is λ_j. During each iteration, α and β messages are computed and exchanged between variable nodes and check nodes through the graph edges according to the following steps numbered 2–4.

2. *Row processing or check node update:* Compute α_{ij} messages using β messages from all other variable nodes connected to check node C_i, excluding the β information from V_j:

$$\alpha_{ijSPA} = \prod_{j' \in V(i) \backslash j} sign(\beta_{ij'}) \times \phi \left(\sum_{j' \in V(i) \backslash j} \phi(|\beta_{ij'}|) \right) \tag{2.2}$$

   where the non-linear function $\phi(x) = -\log\left(\tanh \frac{|x|}{2}\right)$. The first product term in Eq. 2.2 is the parity (sign) bit update and the second product term is the reliability (magnitude) update.

3. *Column processing or variable node update:* Compute β_{ij} messages using channel information (λ_j) and incoming α messages from all other check nodes connected to variable node V_j, excluding check node C_i.

$$\beta_{ij} = \lambda_j + \sum_{i' \in C(j) \backslash i} \alpha_{i'j} \tag{2.3}$$

4. *Syndrome check and early termination:* When variable node processing is finished, every bit in variable node j is updated by adding the channel information (λ_j) and α messages from neighboring check nodes.

$$z_j = \lambda_j + \sum_{i' \in C(j)} \alpha_{i'j} \tag{2.4}$$

From the updated vector, an estimated code vector $\hat{X} = \{\hat{x_1}, \hat{x_2}, ..., \hat{x_N}\}$ is calculated by:

$$\hat{x_i} = \begin{cases} 1, & \text{if } z_i \leq 0 \\ 0, & \text{if } z_i > 0 \end{cases} \tag{2.5}$$

If $H \cdot \hat{X}^T = 0$, then $\hat{X}$ is a valid code word and therefore the iterative process has converged and decoding stops, this is called *early termination*. Otherwise the decoding repeats from step 2 until a valid code word is obtained or the number of iterations reaches a maximum number, $Imax$, which terminates the decoding process.

2.1.2 MinSum Algorithm (MS)

MinSum simplifies the SPA check node update equation, which replaces the computation of the non-linear $\phi()$ function by a min() function. The MinSum check node update equation is given as:

$$\alpha_{ijMinSum} = Sfactor \times \underbrace{\prod_{j' \in V(i) \backslash j} \text{sign}(\beta_{ij'})}_{\text{Sign Calculation}} \times \underbrace{\min_{j' \in V(i) \backslash j} (|\beta_{ij'}|)}_{\text{Magnitude Calculation}} \tag{2.6}$$

where each α_{ij} message is generated using the β messages from all variable nodes $V(i)$ connected to check node C_i as defined by H (excluding V_j). Note that a normalizing factor $Sfactor$ is included to improve error performance, and so this variant of MinSum is called "MinSum Normalized" [11, 10]. Because check node processing requires the exclusion of V_j while calculating the min() for α_{ij}, it necessitates finding both the first and second minimums ($Min1$ and $Min2$, respectively). In this case min() is more precisely defined as follows:

$$\min_{j' \in V(i) \backslash j} (|\beta_{ij'}|) = \begin{cases} Min1_i \ , & \text{if } \ j \neq \text{argmin}(Min1_i) \\ Min2_i \ , & \text{if } \ j = \text{argmin}(Min1_i) \end{cases} \tag{2.7}$$

where,

$$Min1_i = \min_{j \in V(i)} (|\beta_{ij}|) \tag{2.8}$$

$$Min2_i = \min_{j'' \in V(i) \backslash \text{argmin}(Min1_i)} (|\beta_{ij''}|) \tag{2.9}$$

Moreover, the term $\prod \text{sign}(\beta_{ij'})$ is actually an XOR of sign bits, which generate the final sign bit that is concatenated to the magnitude $|\alpha_{ij}|$, whose value is equal to the min() function given in Eq. 2.7.

The MinSum equations themselves do not cause the difficulties of implementing LDPC decoders since, from an outward appearance, the core kernel is simply an addition for the variable node update, and an XOR plus comparator tree for the check node update. Rather, the complexities are caused by the large number of nodes and interconnections as defined by H in tandem with the message-passing algorithm. Recall that the 10GBASE-T H matrix has 2048 variable nodes, V_j, with each one connected to 6 check nodes, $C(j)$, and 384 check nodes, C_i, with each C_i connected to 32 variable nodes, $V(i)$. As a result, we have $M \times W_r + N \times W_c = 384 \times 32 + 2048 \times 6 = 24576$ connections, where check nodes send, as an aggregate, 12288 β messages to the variable nodes, and the variable nodes, as an aggregate, send 12288 α messages to the check nodes *per iteration.*

In summary, for a single iteration of the message-passing algorithm, the 10GBASE-T LDPC code requires a total of $M \times W_r = 12288$ total check node update computations and $N \times W_c = 12288$ total variable node update computations, as well as pass these updated results for a total of $M \times W_r + N \times W_c = 24576$ unique messages.

2.2 LDPC Decoder Architectures

2.2.1 Full-parallel Decoders

Full-parallel decoders directly map each row and each column of the parity check matrix H to a different processing unit, while all these processing units operate in parallel [7, 54, 50, 18]. All M check nodes, N variable nodes, and their associated $M \times W_r + N \times W_c$ total connections are implemented. Thus, a full-parallel decoder will have $M + N$ check and variable node processors, and $M \times W_r + N \times W_c$ global interconnections. A full-parallel 10GBASE-T LDPC decoder will require 2432 processors which are interconnected by a set of $24576 \times b$ global wires and their associated wire buffers (i.e. repeaters), where b is the number of bits in the datapath. So optimizing the fixed-point format becomes an important design parameter in reducing chip costs, not only by optimizing logic area but

also interconnect complexity.

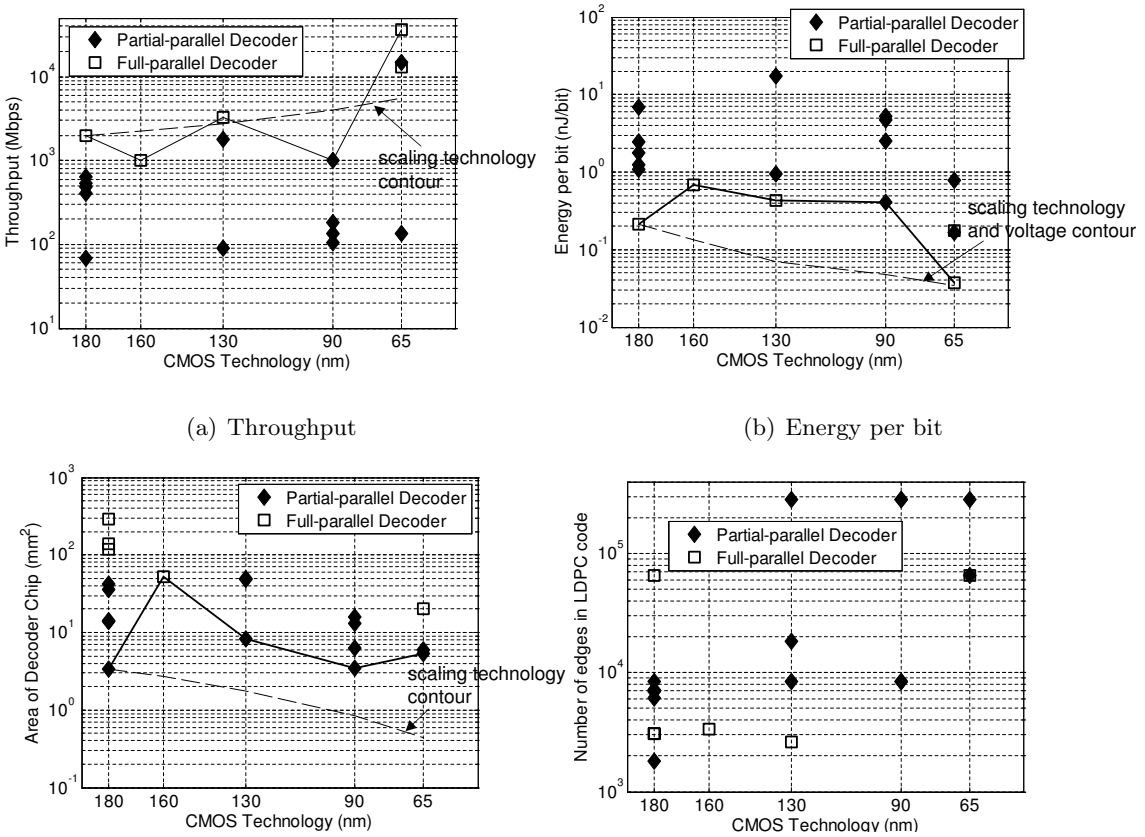
In general, while full-parallel decoders have larger area and capacitance, and lower operating frequencies than their partial-parallel counterparts (to be discussed shortly in the next subsection), they typically only need a single cycle per message-passing iteration; thus, full-parallel decoders are inherently more energy efficient [18].

2.2.2 Serial and Partial-parallel Decoders

In contrast to full-parallel decoders, serial decoders have one processing core and one memory block. If the processing core can calculate either a single check or variable node update per cycle, then the memory must store all $M \times W_r + N \times W_c$ messages. With this architecture a 10GBASE-T serial LDPC decoder will require a $24,576 \times b$-bit memory. A 5-bit datapath would require the memory to store 122,880 bits or 15.36 KB. To increase performance we can alternatively make the processing core larger and calculate several checks and variables per cycle and thus reduce the memory requirement, but this requires a multi-port SRAM which increases SRAM area significantly [42] and building large high-performance SRAMs in deep-submicron technologies results in sizable leakage currents [57].

Although much smaller than full-parallel decoders, serial decoders have much lower throughputs and larger latencies. Partial-parallel designs [47, 79, 16, 41, 84, 9, 13, 81] ideally try to find a balance between the two extremes by partitioning H into rowwise and columnwise groupings such that a set of check node and variable node updates can be done per cycle. Block-structured [70] and quasi-cyclic [39, 21] codes are very well suited for partial-parallel decoder implementations. The parity check matrix of these codes consists of square sub-matrices, where each sub-matrix is either a zero matrix or a permuted identity. This structure makes the memory address generation for partial-parallel decoders very efficient and many communication standards such as DVB-S2, 802.11n, 802.16e and 10GBASE-T use this structure.

Figure 2.3 (a) and (b) show the decoding throughput and the energy dissipation per bit of the decoders versus CMOS technology, respectively. In order to fairly compare throughput and energy dissipation, implementations with an early termination scheme are



(a) Throughput

(b) Energy per bit

(c) Silicon area

(d) Number of edges (connections) in code

Figure 2.3: Throughput, energy dissipation per bit, silicon area and number of edges (check node and variable node connections in Tanner graph) of reported LDPC decoder ASIC implementations versus CMOS technology. For throughput and energy plots the implementations with early termination scheme are excluded. Also for the area plot, full-parallel implementations with reduced routing schemes such as Split-Row [50], Split-Row Threshold [54] and bit-serial [17] methods are excluded for a fair comparison. The idealized contour in the throughput plot is obtained through linear scaling with technology (S); in the energy plot it is obtained through linear scaling with technology and quadratic scaling with voltage (V); and in the area plot it is obtained through quadratic scaling with technology.

excluded. A curve is shown connecting data points that have the maximum throughput and minimum energy per given technology in Fig. 2.3 (a) and (b), respectively. As shown in the figures, in general, most partial-parallel decoders have lower decoding throughput and higher energy dissipation than full-parallel decoders in each technology. However, as shown in Fig. 2.3 (c) (where the curve connects the smallest die area per given technology) full-parallel decoders have larger circuit area than partial-parallel decoders. Also note that, in general, the number of edges in LDPC codes, which is an indication of code complexity,

has increased as technology advances (Fig. 2.3 (d)).

2.3 Current Research on LDPC Decoders

Current research on LDPC decoders has focused on efficient code design, decoding algorithm, and VLSI implementation to meet the demands for current applications. These requirements are: very low error floor, hardware reconfigurability, small silicon area, very high throughput and high energy efficiency.

2.3.1 Structured LDPC Codes

LDPC codes by nature have a very random structure that makes them very inefficient for hardware implementations. A new class of hardware efficient codes are called Quasi-Cyclic (QC) [21, 36, 12] or block-structured LDPC codes [70] and have shown comparable error performance as randomly structured codes. These codes have both encoding [37] and decoding advantage over other types of LDPC codes. The parity check matrix of these codes consists of square sub-matrices, where each sub-matrix is either a zero matrix or a permuted identity. An example is shown in Fig. 2.4, which defines a matrix with $n \times b$ columns which is the code length and $m \times b$ rows with $b \times b$ submatrices. This structure makes the memory address generation for partial-parallel decoders very efficient and many communication standards such as DVB-S2, 802.11n and 802.16e and 10GBASE-T use this structure.

2.3.2 Error Floor Reduction

Although message passing decoding for LDPC codes have shown a very good error performance, most LDPC codes have a major drawback known as an error floor and this is when the error performance curve's slope drop suddenly becomes shallow [62]. Usually this happens when a small number of check sums are not satisfied because of a very small number of errors. A trapping set is defined as a set of variable nodes that are connected to a small number of odd degree check nodes [32, 62, 27]. If errors happen on variable nodes in the trapping set, the messages from such a small number of check nodes are most

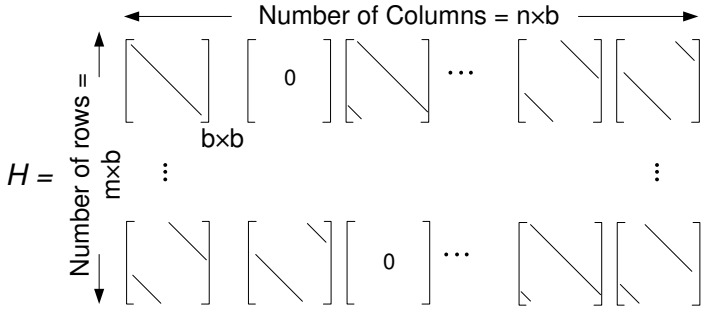


Figure 2.4: Parity check matrix of a quasi-cyclic code consisting of $b \times b$ columns and $m \times b$ rows, with $n \times b$ permuted identity submatrices.

probably not sufficient to correct these errors, which can result in an error floor. Current studies to lower the error floor has focused on better code construction techniques, code concatenation with conventional codes such as Reed-Solomon or BCH and decoding-based strategies. The latter consists of two-stage decoding. The first stage is usually the regular message passing scheme, the second stage is performed only if the iterative decoding fails to correct the errors after some iterations. The recent proposed post-processing methods perform a message biasing scheme [83] on check nodes or bit flipping on selective variable nodes [32]. Both of these schemes are followed by at least another iteration of a regular message passing scheme.

2.3.3 Reconfigurable Decoder Design

A generic architecture for a reconfigurable decoder is shown in Fig. 2.5, which maps each submatrix or multiple submatricies to a memory block or register file and connects them to variable and check node processors through a reconfigurable routing scheme. A controller generates addresses for memory access and defines the interconnections for different modes. Overlapped check node and variable node processing [14], also known as Turbo decoding message passing (TDMP) [46] or Layer decoding [28], is used for Quasi-Cyclic codes to enhance the throughput [47, 40, 66]. Depending on the code structure it may require reordering row and columns of the parity check matrix for efficient address generation [66]. To reduce the area and power consumption, block-serial scheduling is used [33] and register files are proposed [66]. For further power reduction, shared processors and memory blocks

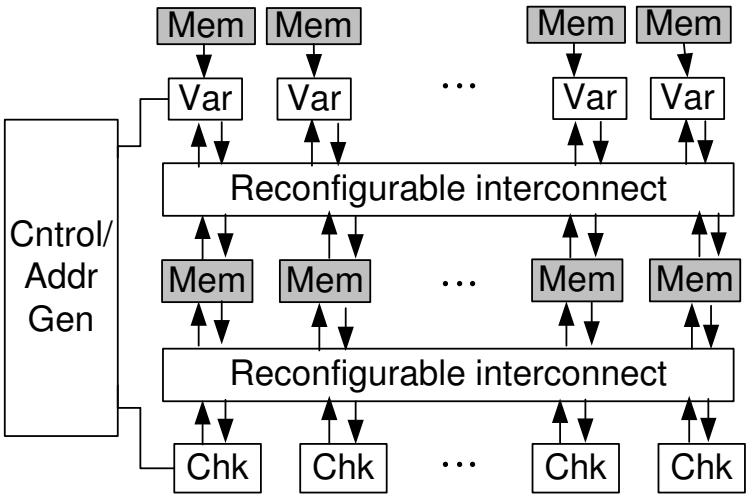


Figure 2.5: A generic reconfigurable decoder architecture

that are not used are deactivated [78].

2.3.4 Routing Congestion Reduction

Full-parallel decoders can potentially have the highest throughput and energy efficiency but because of high routing congestion caused by long global wires between processors they are not efficient to build. Several works tried to address this problem by reducing routing congestion and wire delay of the full parallel decoder implementations through bit-serial communication [18], wire partitioning [59], and floorplan optimization [7, 82]. Bit-serial and wire partitioning use microarchitectural techniques, while floorplan optimization use physical layout backend techniques. In the bit-serial method [18], messages are transmitted serially in multiple cycles. Although the proposed work results in higher clock frequency, the number of clock cycles required to transmit messages is increased, which overall results in low decoding throughput and energy efficiency.Wire partitioning offers an improvement in reducing differences in communication path delays by pipelining wires, however the clock tree and additional registers increase overall power. For example, in a one frame-per-iteration design, throughput was improved by only 1.3× but clock power was increased by a factor of 1.8×, which resulted in worse power and negligible (∼1pJ/bit) or no energy improvements [59]. In floorplan optimization a grouping strategy is used to localize irregular

wires and regularize global wires. While this method might be tangible for codes sizes < 2 Kbits, it requires large backend effort and floorplan design time and also depends on the code structure. Both microarchitecture and floorplan techniques require careful implementation in order to reach the optimal efficiency. This work presents algorithm techniques to reduce interconnect complexity which tradeoff error correction performance for improved throughput and power.

Chapter 3

Split-Row Decoding Method

3.1 Proposed Split-Row Decoding Method

The Split-Row decoding method is proposed to facilitate hardware implementations capable of: high-throughput, high hardware efficiency, and high energy efficiency.

In the Split-Row algorithm, check node processing is partitioned into two blocks, where the check node processing in each partition is performed using only the input messages contained within its own partition, plus one cross-partition sign bit. This stands in contrast to standard decoding where check node processing requires the passing of all check node processor input data across the entire row of the parity check matrix. As an illustration, Fig. 3.1 (a) shows a parity check matrix highlighting the processing of the first check node using a standard decoding (SPA or MinSum) method. The check node C_1 is shown at the bottom and connects to four variable nodes. The Split-Row method is shown in Fig. 3.1 (b) where the check node processing is divided into two parallel check node processing blocks and the check nodes connect to only two variable nodes within each partition.

In the simplest possible split implementation of not passing any information between partitions, a significant error performance loss results. Thus, a sign bit is passed between check node processor halves with a single wire. These are the only wires between partitions. A block diagram of the Split-Row decoder with sign wires between the two halves is shown in Fig. 3.2.

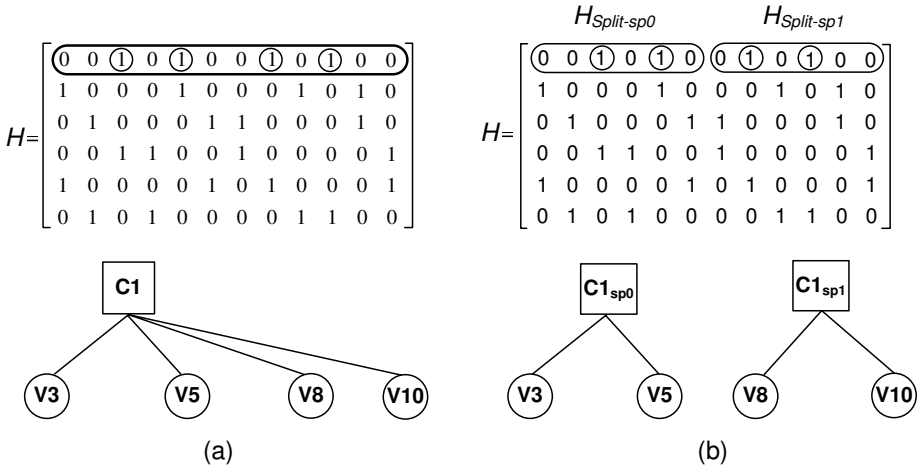


Figure 3.1: The parity check matrix example highlighting the first check node processing (row processing) step using (a) standard decoding (SPA or MinSum) and (b) Split-Row decoding. The check node C_1 and its connected variable nodes are shown for each method.

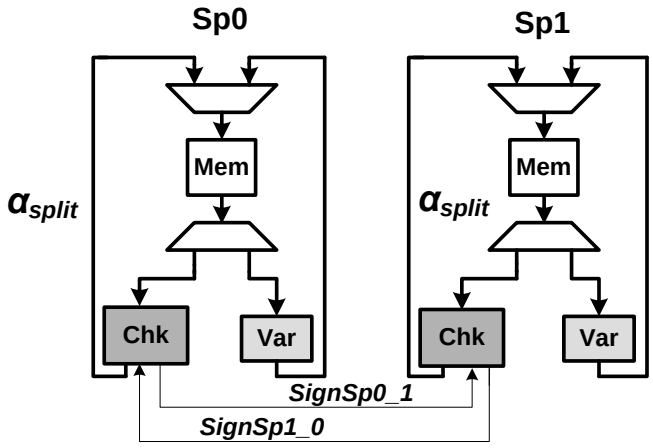


Figure 3.2: Block diagram of the proposed Split-Row decoder

This architecture has two major benefits: 1) it decreases the number of inputs and outputs per check node processor, resulting in many fewer wires between check node and variable node processors, and 2) it makes each check node processor much simpler because the outputs are a function of fewer inputs. These two factors make the decoder smaller, faster, and more energy efficient. In the following subsections, we show that Split-Row introduces some error into the magnitude calculation of the check node processing outputs, and that the error can be largely compensated with a correction factor.

3.1.1 SPA Split

From a mathematical point of view, all steps are similar to the SPA decoder except the check node processing step. In each half of the Split-Row decoder's check node operation, the parity (sign) bit update is the same as in the SPA decoder, because the sign is passed between halves. The magnitude part (the second product term) is updated using half of the messages in each check node of the parity check matrix and this leads to an accuracy loss. We denote the parity check matrix H divided into half column wise by H_{Split}. $V_{Split}(i)\backslash j$ denotes the set of variable nodes in each half of the parity check matrix connected to check node C_i, excluding variable node j. For example, in Fig. 3.1 (b) in the left half matrix, $H_{Split-sp0}$, $V_{Split}(1) = \{V_3, V_5\}$ and $V_{Split}(1)\backslash 3 = \{V_5\}$. Therefore, modifying Eq. 2.2 using half of the messages yields:

$$\alpha_{ijSPASplit} = \prod_{j' \in V(i)\backslash j} \text{sign}(\beta_{ij'}) \times \phi\left(\sum_{j' \in V_{Split}(i)\backslash j} \phi(|\beta_{ij'}|)\right) \tag{3.1}$$

If the β input messages for a Split-Row decoder and an SPA decoder are the same in a particular decoding step, then,

1. $\alpha_{ijSPASplit}$ and α_{ijSPA} have the same sign, and

2. $|\alpha_{ijSPASplit}| \geq |\alpha_{ijSPA}|$.

Since the sign values are passed between each half, the proof of the first assertion is straightforward. The proof of the second assertion comes from the fact that ϕ is a positive function

and therefore the sum of half of the positive values is less than or equal to the sum of all:

$$\sum_{j' \in V_{Split}(i) \backslash j} \phi(|\beta_{ij'}|) \leq \sum_{j' \in V(i) \backslash j} \phi(|\beta_{ij'}|) \tag{3.2}$$

Also $\phi(x)$ is a decreasing function, therefore the following inequality holds:

$$\phi\left(\sum_{j' \in V_{Split}(i) \backslash j} \phi(|\beta_{ij'}|)\right) \geq \phi\left(\sum_{j' \in V(i) \backslash j} \phi(|\beta_{ij'}|)\right) \tag{3.3}$$

And we obtain:

$$|\alpha_{ijSPASplit}| \geq |\alpha_{ijSPA}| \tag{3.4}$$

To reduce the difference between $\alpha_{SPASplit}$ and α_{SPA}, $\alpha_{SPASplit}$ values are multiplied by a correction factor $Sfactor$ less than one according to:

$$\alpha_{ijSPASplit} = Sfactor \times \prod_{j' \in V(i) \backslash j} \text{sign}(\beta_{ij'}) \times \phi\left(\sum_{j' \in V_{Split}(i) \backslash j} \phi(|\beta_{ij'}|)\right) \tag{3.5}$$

3.1.2 MinSum Split

Similarly, in the MinSum Split decoder the sign bit is computed using the sign bit of all messages across the whole row of the parity check matrix. The magnitude of a message in each half is computed using the minimum of the messages in each half.

$$\alpha_{ijMinSumSplit} = \prod_{j' \in V(i) \backslash j} \text{sign}(\beta_{ij'}) \times \min_{j' \in V_{split}(i) \backslash j}(|\beta_{ij'}|) \tag{3.6}$$

It is clear that the minimum value among half of the messages is equal to or larger than the minimum value of all messages. Therefore, we obtain:

$$|\alpha_{ijMinSumSplit}| \geq |\alpha_{ijMinSum}|. \tag{3.7}$$

To reduce the difference between $\alpha_{MinSumSplit}$ and α_{MinSum}, $\alpha_{MinSumSplit}$ values are multiplied by a correction factor $Sfactor$ less than one according to:

$$\alpha_{ijMinSumSplit} = Sfactor \times \prod_{j' \in V(i) \backslash j} \text{sign}(\beta_{ij'}) \times \min_{j' \in V_{split}(i) \backslash j}(|\beta_{ij'}|) \tag{3.8}$$

The correction factor $Sfactor$ is relatively easily obtained empirically through simulations and examples are given in Section VI.

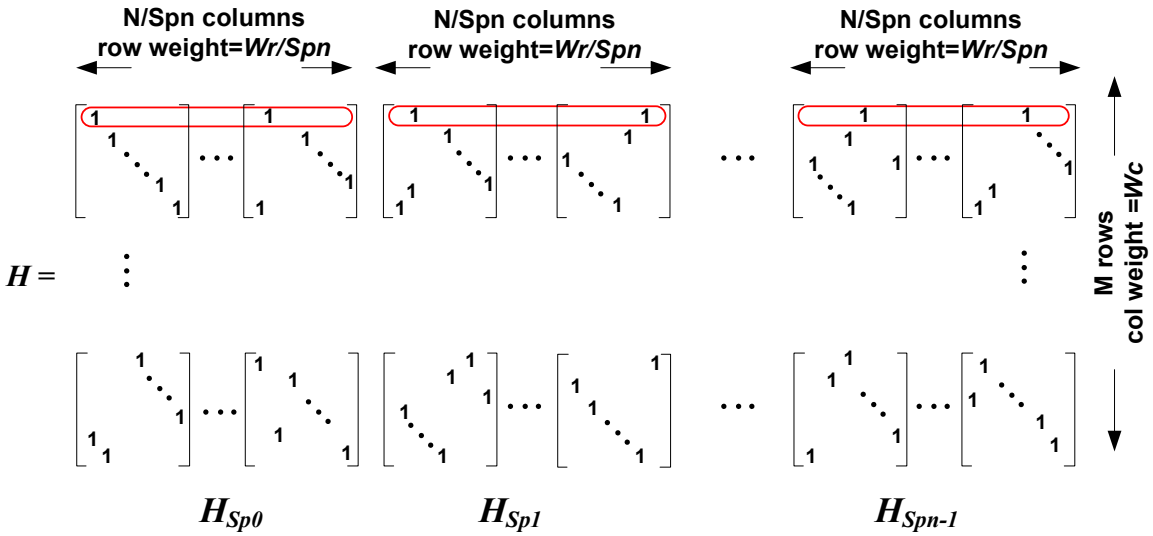


Figure 3.3: The parity check matrix of a (W_c, W_r) (N, K) permutation-based LDPC code highlighting the first check node processing operation with *Spn*-way splitting (Multi-Split) method.

3.2 Multi-Split Decoding Method

To further reduce interconnect and decoder complexity, the Multi-Split method partitions matrix rows into *Spn* multiple blocks (called *Split-Spn*). This requires new circuits to correctly process sign bits among multiple blocks and is especially beneficial for regular permutation-based high row-weight ($W_r \geq 16$) decoders. A Multi-Split parity check matrix highlighting the check node processing operation is shown in Fig. 3.3. We denote each partition of the parity check H which is divided into *Spn* partitions column wise by $H_{Spk}, k = 0, ..., n-1$. As shown in the figure, in check node processing (check-node update) there are only W_r/Spn nodes to be processed in each partition, resulting in even less complex processing and wire interconnect in each partition.

In each partitioned Multi-Split row operation, the parity (sign) bit update is the same as in the SPA decoder, since it uses sign bits from across the row of the matrix. The magnitude part is updated using the messages of its own partition of the parity check matrix. Similar to the Split-Row method, the magnitude part of the check node processor output, α, is larger than that of the SPA decoder. Therefore the error performance can be improved by multiplying the $\alpha_{Split\text{-}Spk}$ values with a correction factor $Sfactor$ less than

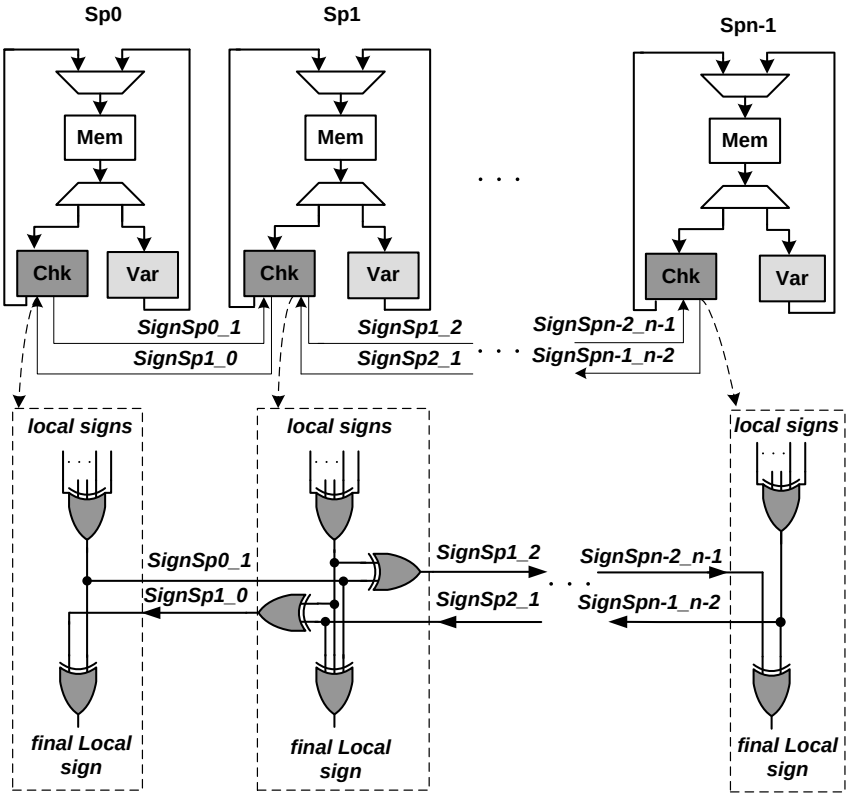


Figure 3.4: Multi-Split decoder with *Spn*-way splitting method, highlighting inter-partition *sign* wires and the simplified logic for implementation of the *sign* bit in each check node processor.

one. Modifying Eq. 2.2 using the messages in each partition yields:

$$\alpha_{ijSPASpk} = Sfactor \times \prod_{j' \in V(i) \backslash j} \text{sign}(\beta_{ij'}) \times \phi \left(\sum_{j' \in V_{Spk}(i) \backslash j} \phi(|\beta_{ij'}|) \right) \tag{3.9}$$

$V_{Spk}(i) \backslash j$ denotes the set of variable nodes in each partition of the parity check matrix (H_{Spk}) which connects to check node C_i, excluding variable node j.

Similarly, in MinSum Multi-Split, check node processor outputs are normalized with a correction factor $Sfactor$:

$$\alpha_{ijMinSumSpk} = Sfactor \times \prod_{j' \in V(i) \backslash j} \text{sign}(\beta_{ij'}) \times \min_{j' \in V_{Spk}(i) \backslash j} (|\beta_{ij'}|) \tag{3.10}$$

Figure 3.4 shows how the sign bits are calculated according to Eq. 3.9 or Eq. 3.10. The top half of the figure shows a block diagram of a decoder using an *Spn*-way splitting method. A small number of sign wires pass between decoder partitions. The bottom

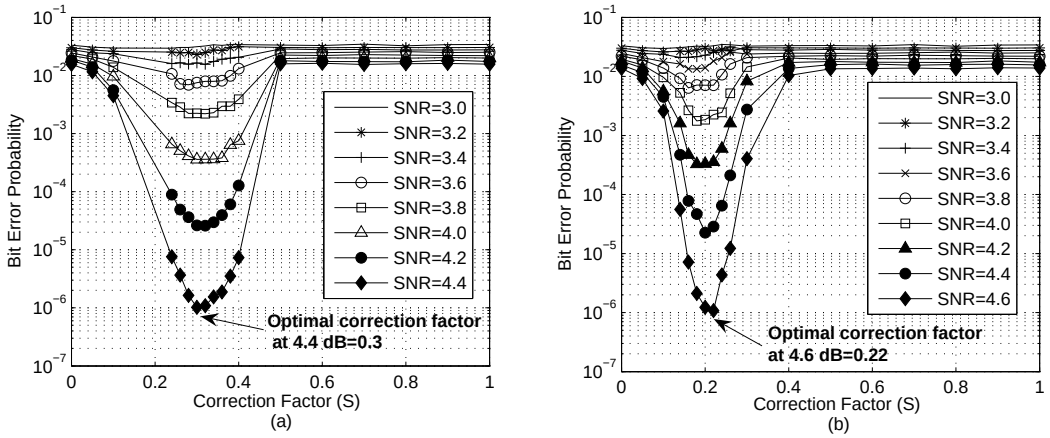


Figure 3.5: Determination of correction factor ($Sfactor$) for a (6,32) (2048,1723) RS-based LDPC code using (a) MinSum Split-2 and (b) MinSum Split-4 decoders. The optimal correction factor variations with the SNR values are very small with the average value of 0.3 for Split-2 and 0.19 of Split-4.

half shows the sign logic inside each check node processor. Local sign bits are generated inside each block simultaneously, resulting in lower latencies. The final local sign bits are calculated using 1 or 2 bits from adjacent blocks and are used to generate sign bits for output messages.

3.3 Correction Factor and Bit Error Performance Results

3.3.1 Split-Row Correction Factors

Finding the optimal correction factor for the Split-Row algorithm that results in the best error performance requires complex analysis such as density evolution [63]. For simplicity and to account for realistic hardware effects, the correction factors presented in this paper are determined empirically based on bit error rate (BER) results for various SNR values and numbers of decoding iterations.

As the number of partitions increases, a smaller correction factor should be used to normalize the error magnitude of check node processing outputs in each partition. This is because for SPA Multi-Split, as the number of partitions increases, the summation on the left side of Eq. 3.2 decreases in each partition and since $\phi(x)$ is a decreasing function,

(N, K)	(W_c, W_r)	average optimal correction factor $Sfactor$				
		SP-2	SP-4	SP-6	SP-8	SP-12
(1536,770)	(3,6)	0.45	*	-	*	*
(1008,507)	(4,8)	0.35	-	*	-	*
(1536,1155)	(4,16)	0.4	0.25	*	-	*
(8088,6743)	(4,24)	0.4	0.27	0.22	-	-
(2048,1723)	(6,32)	0.3	0.19	*	0.15	*
(16352,14329)	(6,32)	0.4	0.25	*	0.17	*
(8176,7156)	(4,32)	0.4	0.24	*	0.17	*
(5248,4842)	(5,64)	0.35	0.25	*	0.2	*
(5256,4823)	(6,72)	0.35	0.2	0.18	0.15	0.14

Table 3.1: Average optimal correction factor $Sfactor$ for different constructed regular codes. The asterisk ("*") indicates that the row weight of the code is not evenly divisible by that level of splitting. The dash ("-") indicates that the row weight for that level of splitting is very small and error performance loss is therefore significant ($\geq$0.7 dB).

the summation on the left side of Eq. 3.3 becomes larger which results in larger magnitude check node processing outputs in each partition. For MS Multi-Split, except for the partition which has the global minimum, the difference between local minimums in most other partitions and the global minimum becomes larger as the number of partitions increases. Thus, the average check node processor output magnitude gets larger as the number of partitions increases and a smaller correction factor is required to normalize the check node processing outputs in each partition.

Achieving an absolute minimum error performance would require a different correction factor for each check node processor output—but this is impractical because it would require knowledge of unavailable information such as check node processor inputs in other partitions. Since significant benefit comes from the minimization of communication between partitions, we assume a constant correction factor for all row processing outputs. This is the primary cause of the error performance loss and slower convergence rate of Split-Row.

Figure 3.5 plots the error performance of a (6,32) (2048,1723) RS-based LDPC code [19] when decoded with (a) MinSum Split-2 and (b) MinSum Split-4 methods versus correction factors for various SNR values with a maximum of 15 decoding iterations. As shown in the figures, there is a strong dependence of the error performance on the correction factor magnitudes. The optimum correction factors are different for different SNR values, although the variations are very small. In MinSum Split-2 the optimum correction factors

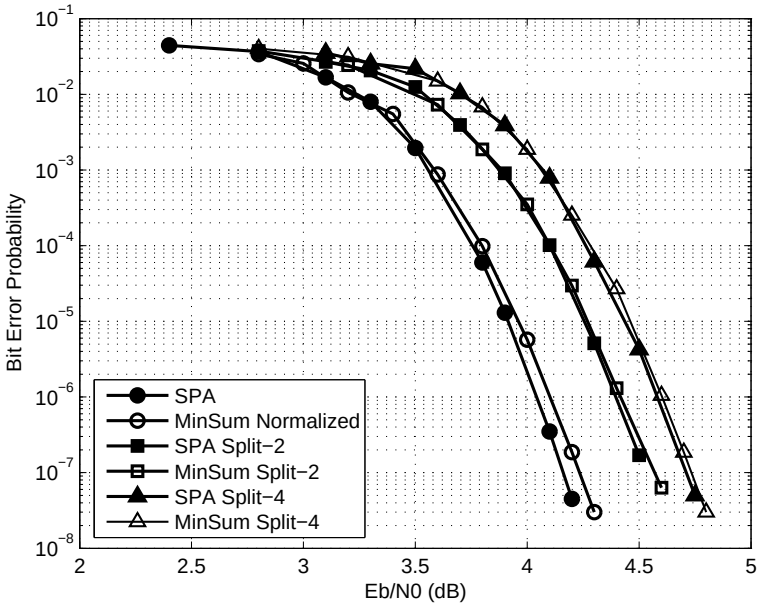


Figure 3.6: BER performance of the (6,32) (2048,1723) code using the Multi-Split method in SPA and MinSum decoders with optimal correction factors.

for SNR ranges of 3.4–4.4 dB are between 0.28–0.32 with an average of 0.3 and a variation from the mean of ±0.016 (±5%). In MinSum Split-4 the optimum correction factor is in the range 0.16–0.22 with an average of 0.19 and a variation of ±0.02 (±11%). Similar analysis was performed for various maximum numbers of decoding iterations and simulation results indicate the optimum correction factors remain the same as the values shown in Fig. 3.5.

Since the error performance improvements are small (≤0.07 dB) if a decoder used multiple correction factors for different SNR values, we use the average value as the correction factor for the error performance simulations in this paper.

Table 3.1 summarizes the average optimal correction factors for different regular constructed permutation-based codes with various levels of splitting. Multi-Split is specially beneficial for regular high row weight codes. Correction factors decrease in magnitude as the level of splitting increases; but the correction factor varies little across the wide variety of codes for the same level of splitting.

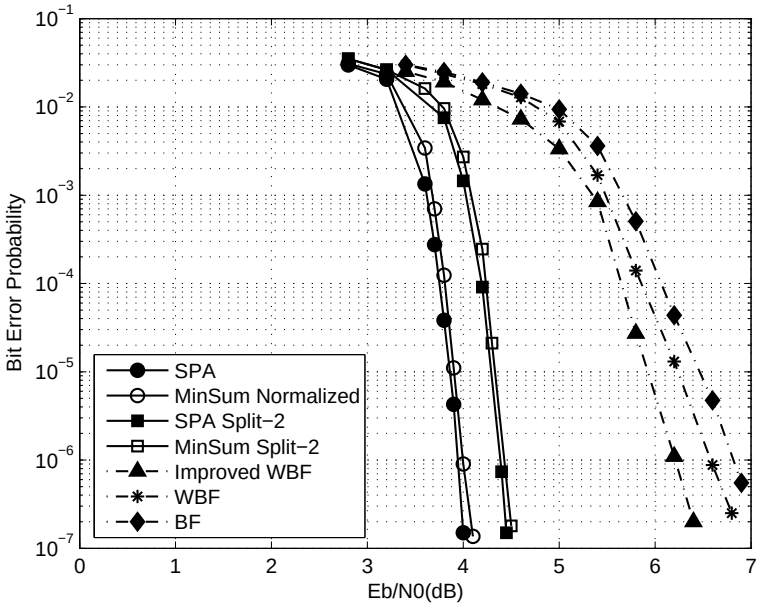


Figure 3.7: Error performance comparison with different decoding algorithms for a (4,32) (8176,7156) QC-LDPC code

3.3.2 Error Performance Results

All simulations assume an additive white Gaussian noise channel with BPSK modulation. BER results presented here were made using simulation runs with more than 100 error blocks each and with a maximum of 15 iterations ($Imax = 15$) or were terminated early when a zero syndrome was detected for the decoded codeword.

Figure 3.6 shows the bit error rate (BER) for the same code using standard, Split-2 and Split-4 decoding with SPA and MinSum algorithms. The error performance of SPA Split-2 is approximately 0.35 dB away from SPA and SPA Split-4 is 0.2 dB away from SPA Split-2 at BER = 10^{-7}. The BER curves show that when using the same level of splitting in SPA and MinSum decoders, the error performance of MinSum Split is about 0.05 dB away from the SPA Split decoder.

Figure 3.7 shows the error performance of the (4,32) (8176,7156) Euclidean geometry-based QC-LDPC code [12] using standard and Split-2 decoding in SPA and MinSum algorithms. Also shown are reduced complexity hard decision algorithms: Bit-Flipping (BF) [25], weighted Bit-Flipping (WBF) [34], and improved WBF [80]. Split-2 performs approximately 0.5 dB away from the SPA and MS decoders and attains around 1.7 dB gain

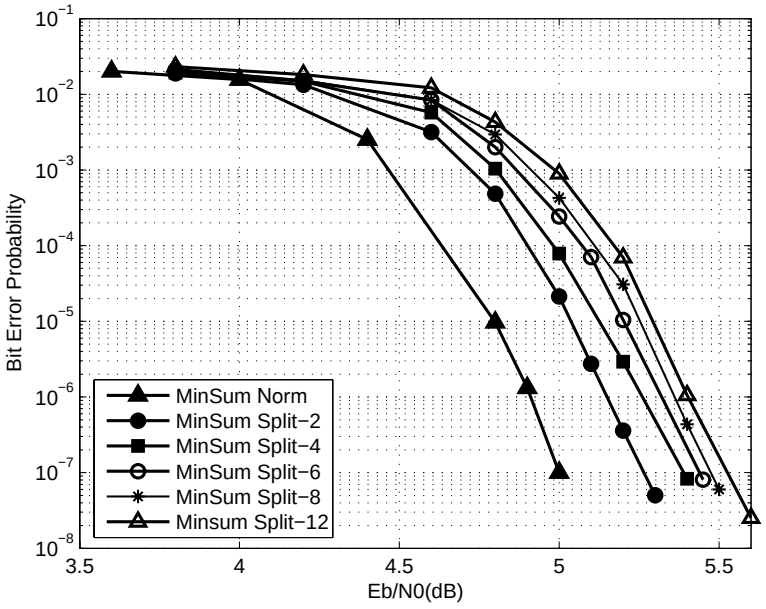


Figure 3.8: BER performance of a (6,72) (5256,4823) QC-LDPC code using various MinSum decoders with different levels of splitting and near-optimal correction factors.

over improved WBF at BER $= 2 \times 10^{-7}$.

With high row-weight codes, the H matrix can be split into more blocks reducing the decoder complexity even further. As an example, Fig. 3.8 shows the error performance for a (6,72) (5256,4823) code with row weight 72 using MinSum normalized algorithm with different levels of splitting and with near-optimal correction factors. Changing from MinSum to Split-2 loses about 0.25 dB. Then, from Split-2 through Split-4, 6, 8, and all the way to Split-12 results in small degradations of less than 0.1 dB in each step, and the total from Split-2 to Split-12 is only 0.3 dB loss at BER $= 10^{-7}$.

3.4 Full-Parallel MinSum Multi-Split Decoders

Figure 3.9 shows the top level block diagram of a full-parallel decoder for an N-bit and M-parity check code using the Multi-Split method. The decoder uses a two-phase check node and variable node processing method. The pipelined decoder is partitioned into Spn sub-blocks where the check node processors are interconnected by $2M$ sign wires

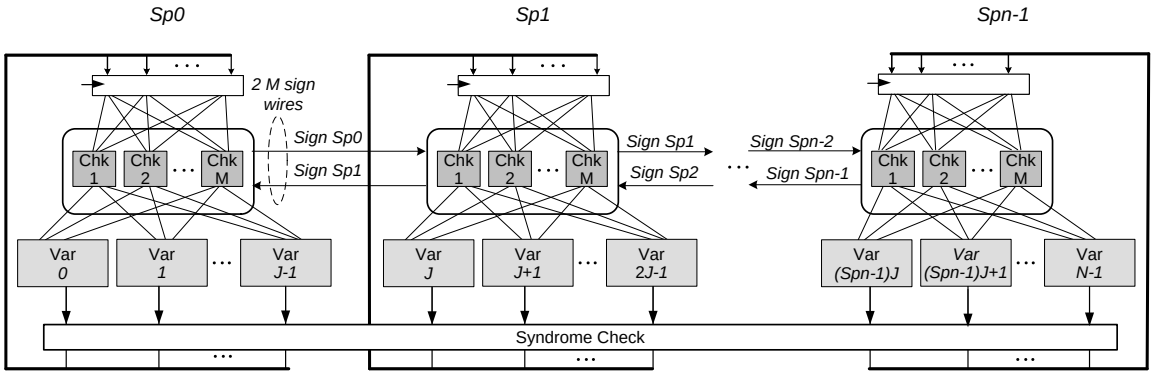


Figure 3.9: Top level block diagram of a full-parallel decoder corresponding to an $M \times N$ parity check matrix, using Split-Row with Spn partitions. The inter-partition $Sign$ signals are highlighted. $J = N/Spn$, where N is the code length.

(each check node processor sends its sign bit and receives a sign bit from the check node processor in the neighboring partition). Each partition consists of M check node processors and N/Spn variable node processors. The α messages from each check node processor are routed to variable node processors according to the parity check matrix structure. Similarly, β messages are sent to corresponding check node processors after syndrome check and after being synchronized by the global clock signal.

The check node processor block diagram for the MinSum Multi-Split decoder in partition Spk is shown in Figure 3.10. Each row processor has W_r/Spn inputs (β) and W_r/Spn outputs (α) (instead of the W_r inputs and outputs in MinSum and SPA). The inputs arrive in sign-magnitude format in parallel. The XOR circuits on top generate the sign bit for each α_i message and the output sign ($SignSpk$) to the nearest neighbors. In partition Spk, sign bits from partitions $Spk - 1$ and $Spk + 1$ are received, XORed with the local sign bit resulting $SignSpn$, which is sent to partition $Spk + 1$ and $Spk - 1$. The sign bit for α_i is the 1-bit multiplication of all neighboring sign bits and the sign bit of local β messages, excluding β_i. The Min block finds the smallest magnitude ($Min1$) and the second smallest magnitude ($Min2$) among all β messages in the row of the partition. It also outputs the location of $Min1$ ($IndexMin1$). For each output (α_i), the Mux selects $Min2$ if ($IndexMin1$=$Index\alpha_i$), otherwise it selects $Min1$ [26].

Figure 3.11 shows the block diagram of the variable node processor. Similar to MinSum normalized, the messages from check node processors (check nodes) are multiplied

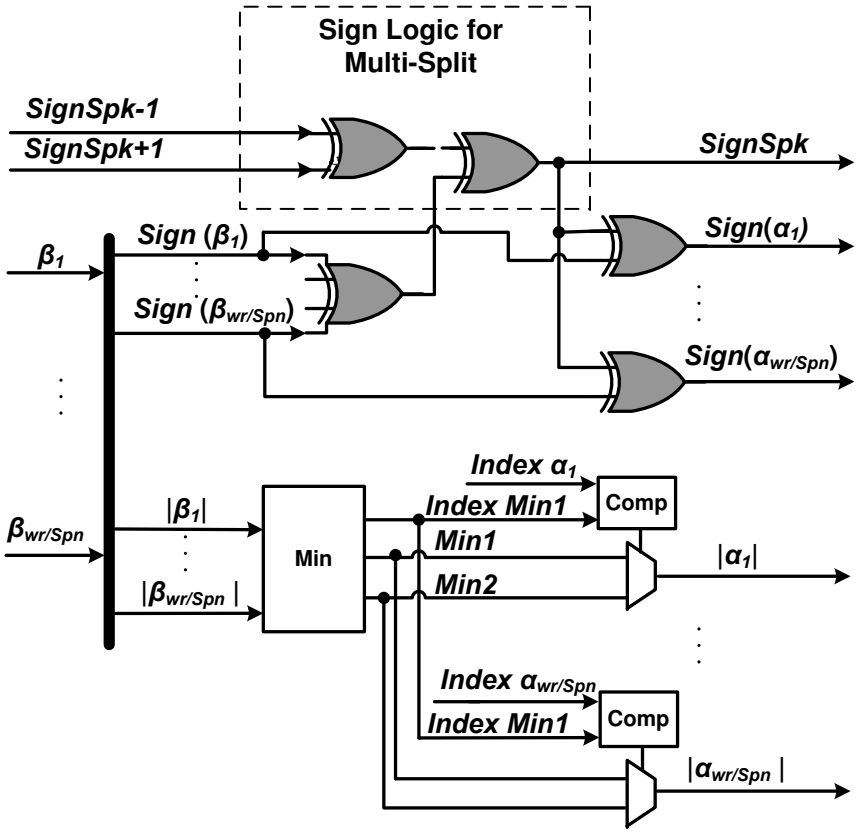


Figure 3.10: Check node processor block diagram of MinSum Multi-Split for partition Spk, with sign logic on top and magnitude calculation of α at the bottom.

by the correction factor $Sfactor$. This correction scheme can be implemented with shift registers if the correction factor is a power of 2, or can be implemented using lookup tables—both have small circuit area and complexity. The number of inputs and outputs to each variable node processor (W_c, which is the column weight of parity check matrix) and their processing are the same as standard decoding (SPA or MS). After being converted from sign-magnitude to 2's complement, α and λ are added together to generate β according to variable node processing equation Eq. 2.3. β messages are then converted to sign-magnitude for use in check node processing in the next iteration.

3.5 Decoder Implementation Example and Results

To precisely quantify the benefits of the Split-Row and Multi-Split algorithms when built into hardware, we have implemented three MinSum full-parallel decoders for

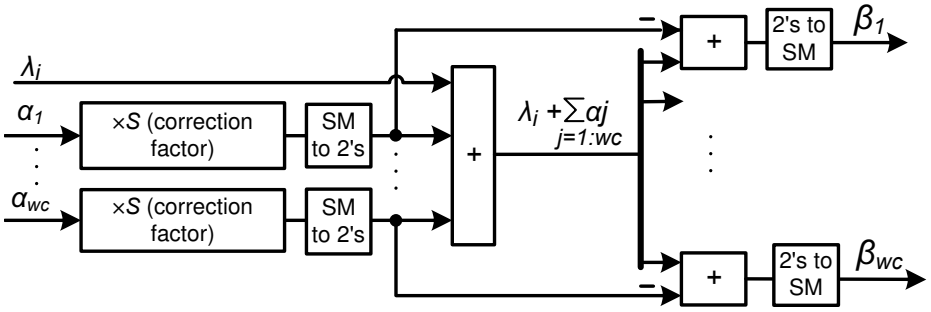


Figure 3.11: Variable node processing unit block diagram

Code length, No. of columns (N)	2048
Information length (K)	1723
Parity check equations, No. of rows (M)	384
Row weight (W_r)	32
Column weight (W_c)	6
Size of permutations	64

Table 3.2: Summary of the key parameters of the implemented (6,32) (2048,1723) 10GBASE-T LDPC code

the (2048,1723) 10GBASE-T code using MinSum normalized, Split-2 and Split-4 methods. The decoders were developed using Verilog to describe the architecture and hardware, synthesized with Synopsys Design Compiler, and placed and routed using Cadence SOC Encounter. All designs were created in ST Microelectronics' 65 nm, 1.3 V low-leakage, seven-metal layer CMOS.

The parity check matrix of the (2048,1723) code has 384 rows and is composed of 6×32 sub-matrices. Each sub-matrix is a 64×64 permutation. Table 3.2 summarizes the code parameters that specify all three decoder implementations.

The full-parallel decoder maps each variable node to one variable node processor and each check node to one check node processor. Figure 3.12 shows the mapping block diagrams for full-parallel decoders using (a) MinSum normalized, (b) Split-2 and (c) Split-4 architectures. The MinSum normalized decoder has 384 row and 2048 variable node processors corresponding to the parity check matrix dimensions M and N, respectively. As seen in Fig. 3.1 and further described in Sec. VII, the split architectures reduce the number of interconnects by reducing the number of columns per sub-block by a factor of $1/Spn$.

Thus, in each Split-2 sub-block there are again 384 row processors (though simplified), but only 1024 variable node processors. For Split-4 there are only 512 variable node processors in each sub-block. The area and speed advantage of a Multi-Split decoder is significantly higher than in a MinSum normalized version due to the benefits of smaller and relatively lower complexity partitions, each of which communicate using short and structured sign passing wires.

3.5.1 Effects of Fixed-point Number Representation

One of the major issues when realizing decoder architectures is the choice of number representation. Although software simulations can show the trade-off in error performance due to quantization noise, it does not give any indication of the hardware costs. Figure 3.14 compares the error performance of floating-point and 5-bit fixed-point implementations of MinSum normalized, MinSum Split-2 and MinSum Split-4 decoders. The MinSum fixed-point is less than 0.1 dB away from MinSum floating point. The error performance loss in Split-2 and Split-4 fixed-point is about 0.15 dB compared to their floating-point equivalents. Because of this reasonable loss in error performance compared to floating-point and its greatly reduced hardware costs, we focus only on trade-offs of fixed-point word widths.

Although there have been several studies on the quantization effects in LDPC decoders [85], [10], as a base overview of the effects of word length in a decoder's datapath we will uniformly change the word widths of the λ, α and β messages. For a fixed-point datapath width of q bits, the majority of the decoder's hardware complexity can be roughly estimated by the wires going to and from variable and check node processors. For M check node processors, the total number of word busses that pass α messages is $M \times W_r$, while N variable node processors that pass β messages require $N \times W_c$ messages. Therefore, the total number of global communication wires is $q \times (M \times W_r + N \times W_c)$. Increasing the word width of the datapath from a 5-bit to 6-bit fixed-point representation—4.1 and 4.2 formats, respectively—increases the number of global wires by $M \times W_r + N \times W_c$. However, the

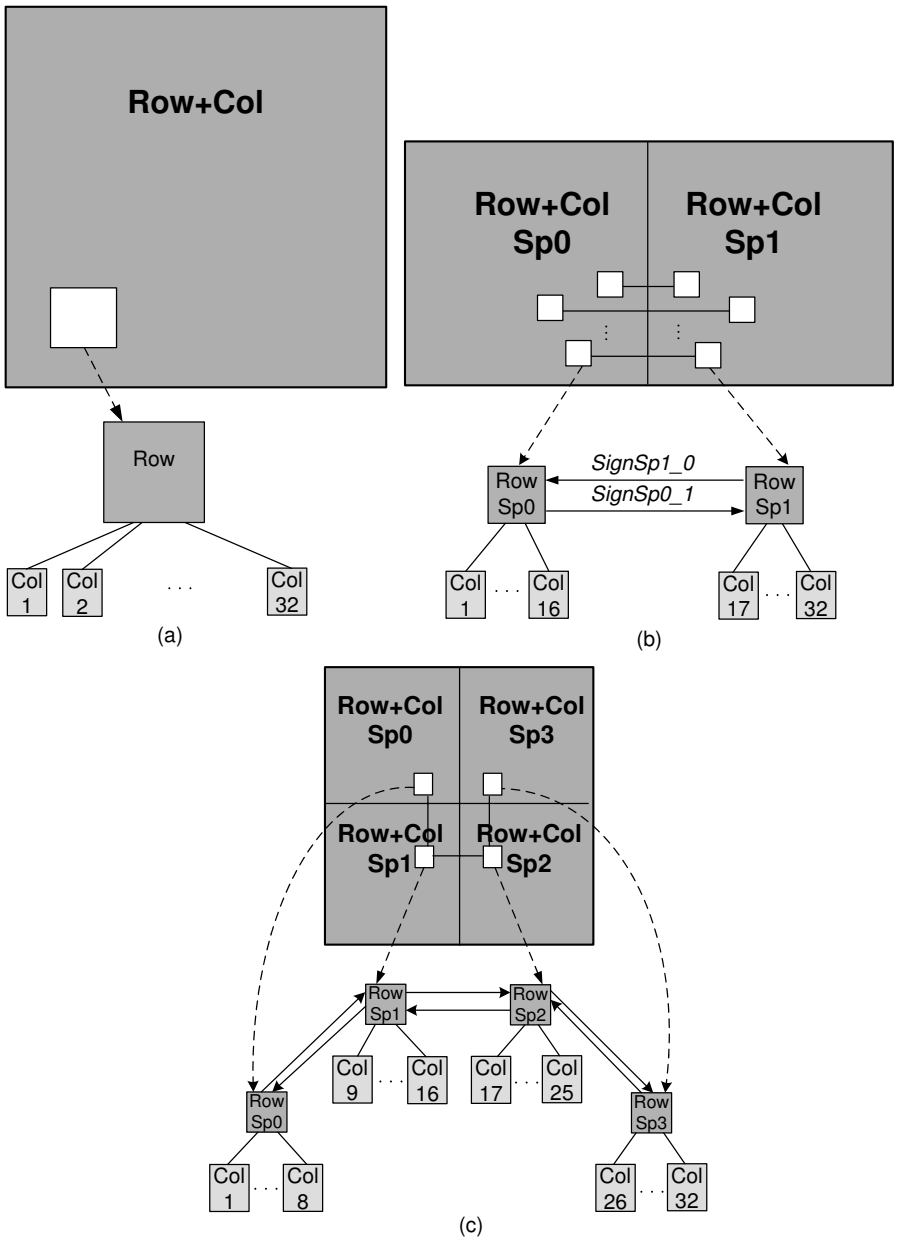


Figure 3.12: Mapping a full-parallel decoder with (a) MinSum normalized (b) Split-2 and (c) Split-4 decoding methods for the (6,32) (2048,1723) code

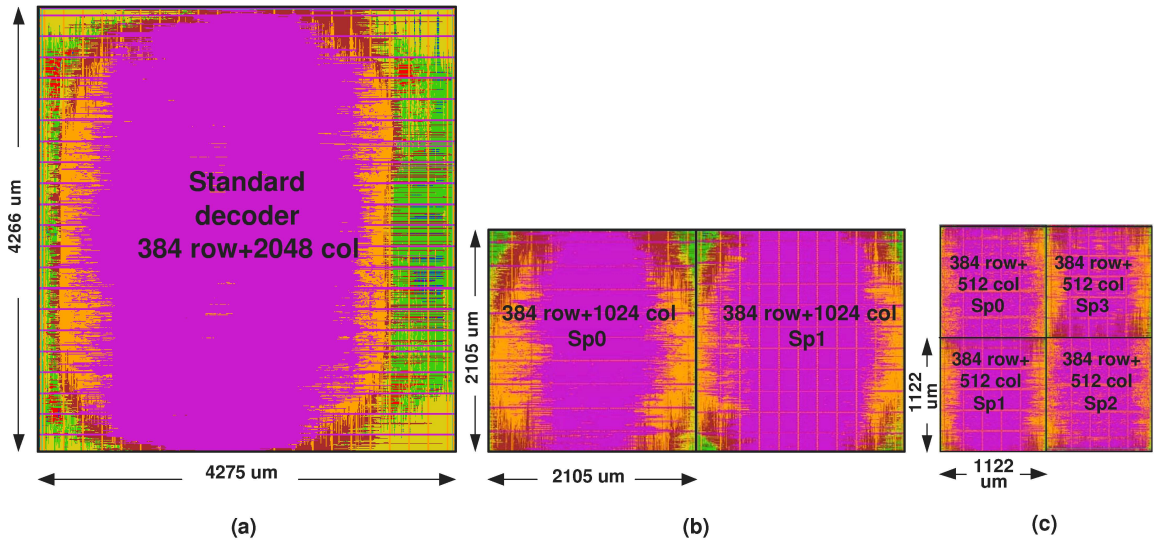


Figure 3.13: Final layout of (a) MinSum normalized, (b) MinSum Split-2 and (c) MinSum Split-4 decoder chips, shown approximately to scale

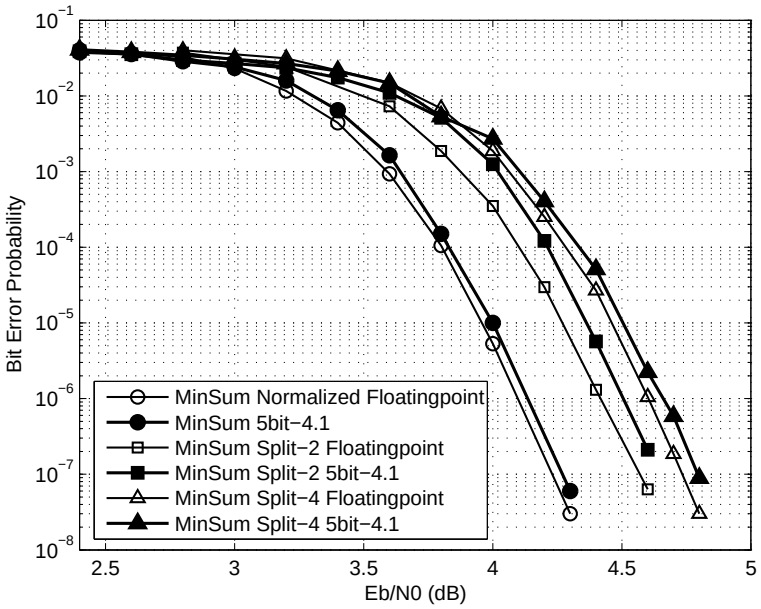


Figure 3.14: BER performance of a (6,32) (2048,1723) LDPC code with floating-point and fixed-point 5-bit 4.1 implementations of MinSum normalized, MinSum Split-2 and MinSum Split-4 with optimal correction factors

	MinSum normalized	Split-2 MinSum	Split-4 MinSum
CMOS fabrication process		65 nm CMOS, 1.3 V	
Area utilization	38%	50%	**85%**
Average wire length (μ)	175.2	115.5	73.8
Area per sub-block (mm^2)	20	6.9	1.5
Total layout area (mm^2)	20	13.8	**6.1**
% area for check node processors	13.2%	19.2%	41.3%
% area for variable node processors	8.0%	11.6%	26.0%
% area for registers and clock tree	16.8%	19.2%	17.7%
% area without standard cells	62.0%	50.0%	15.0%
Maximum clock rate (MHz)	59	110	**146**
Power dissipation (mW)	1941	2179	1889
Throughput @$Imax = 15$ (Gbps)	8.1	15.0	**19.9**
Energy per bit @$Imax = 15$ (pJ/bit)	241	145	**95**
Avg. iterations @ BER $= 3 \times 10^{-5}$ ($Iavg$)	3.8	4.8	4.9
Throughput @$Iavg$ (Gbps)	31.8	46.9	**61.0**
Energy per bit @$Iavg$ (pJ/bit)	61	46	**31**

Table 3.3: Comparison of the three full-parallel decoders implemented in 65 nm CMOS for a (6,32) (2048,1723) code. All area values are for final placed and routed layout. Maximum number of iterations $Imax = 15$.

complexity caused by additional wires is not a simple linear relationship. When designed in a chip, every additional wire results in a super-linear increase in circuit area and delay [7].

On the other hand, using wider fixed-point words improves the error performance. BER simulations show an approximate 0.07-0.09 dB improvement in all three decoders when using 6-bit words (4.2) instead of 5-bit words (4.1). To achieve this improved performance for MinSum normalized with one additional bit, the number of wires increases by $M \times W_r + N \times W_c$, but for Multi-Split the increase is only $M \times W_r + (N/Spn) \times W_c$ per block. Synthesis results for a 6-bit implementation of Split-2 and Split-4 show that the row and variable processors have a 12% and 8% area increase respectively, without any reduction in clock rate, compared to a 5-bit implementation using the same constraints. Thus, the error performance loss of the Split-2 and Split-4 decoders can be reduced by using a larger fixed-point word with a small area penalty.

3.5.2 Area, Throughput and Power Comparison

Figure 4.8 shows the final chip layouts of the (a) MinSum normalized, (b) MinSum Split-2 and (c) MinSum Split-4 decoders, and Table 5.3 summarizes their post-layout results. In Split-2 and Split-4, although the number of check node processors increases with higher splitting levels (they are replicated *Spn* times), Fig. 4.8 highlights the fact that total chip size is actually reduced with these Split decoders.

To achieve a fair comparison between all three architectures, a common CAD tool design flow was adopted. The synthesis, floorplan, and place and route stages of the layout were automated with minimal designer intervention.

Since Split-Row reduces check node processor area and eliminates significant communication between row and variable node processors (causing them to operate as smaller nearly-independent groups), layout becomes much more compact and automatic place and route tools can converge towards a better solution in a much shorter period of time.

As shown in Table 5.3, Split-4 achieves a high area utilization (the ratio of standard cell area to total chip area) and a short average wire length compared to the MinSum normalized decoder whose many global row and variable node processor interconnections force the place and route tool to spread standard cells apart to provide sufficient space for routing.

As an additional illustration, Table 5.3 provides a breakdown of the basic contributors of layout area, which shows the dramatic decrease in *% area without standard cells* (i.e., chip area with only wires) with an increased level of splitting.

The critical path delay in Split-4 is about 2.3 times shorter than that of MinSum normalized. Place and route timing analysis and extracted delay/parasitic annotation files (i.e., SDF) show that the critical path delay is composed primarily of a long series of buffers and wire segments. Some buffers have long RC delays due to large fanouts of their outputs. For the MinSum decoder, the sums of interconnect delays caused by buffers and wires (intrinsic gate delay and RC delay) is 13.1 ns. In Split-2 and Split-4, the total interconnect delays are 5.1 ns and 6.2 ns, respectively, which are 2.6 and 6 times smaller than that of MinSum. Thus, Split-4's speedup over MinSum normalized is due in part to

its simplified check node processing, but the major contributor is the significant reduction in variable/check node processor interconnect delay.

To summarize Split-Row's benefits, the Split-4 decoder occupies 6.1 mm^2, which is 3.3 times smaller than MinSum normalized. It runs at 146 MHz and with 15 iterations it attains 19.9 Gbps decoding throughput which is 2.5 times higher, while dissipating 95 pJ/bit—a factor of 2.5 times lower than MinSum normalized.

Although it is not possible to exactly quantify the benefit of chip area reductions, chip silicon area is a critical parameter in determining chip costs. For example, reducing die area by a factor of 2 results in a die cost reduction of more than 2 times when considering the cost of the wafer and die yield [61]. Other chip production costs such as packaging and testing are also significantly reduced with smaller chip area.

At a supply voltage of 0.79 V, the Split-4 decoder runs at 47 MHz and achieves the minimum 6.4 Gbps throughput required by the 10GBASE-T standard [4]. Power dissipation is 226 mW at this operating point. These estimates are based on measured data from a chip that was recently fabricated on the exact same process and operates correctly down to 0.675 V [71].

3.5.3 Wire Statistics

Figure 3.15 shows the wire length distribution of (a) MinSum normalized, (b) MinSum Split-2, and (c) MinSum Split-4 decoders. Compared to the MinSum decoder, the longest wire in Split-2 and Split-4 is 1.9 times and 3.6 shorter, respectively. The average wire length in Split-2 and Split-4 is about 1.5 and 2.4 times shorter, respectively, than the MinSum decoder.

The total number of sign-passing wires between sub-blocks in the Multi-Split methods is $2(Spn - 1)M$. For these decoders where $M = 384$, the sign wires in Split-2 are only 0.12% of the total number of wires and in Split-4 they are only 0.30% of the total.

The source of Multi-Split's benefits are now clear: the method breaks row processors into multiple blocks whose internal wires are all relatively short. These blocks are interconnected by a small number of sign wires. This results in denser, faster and more

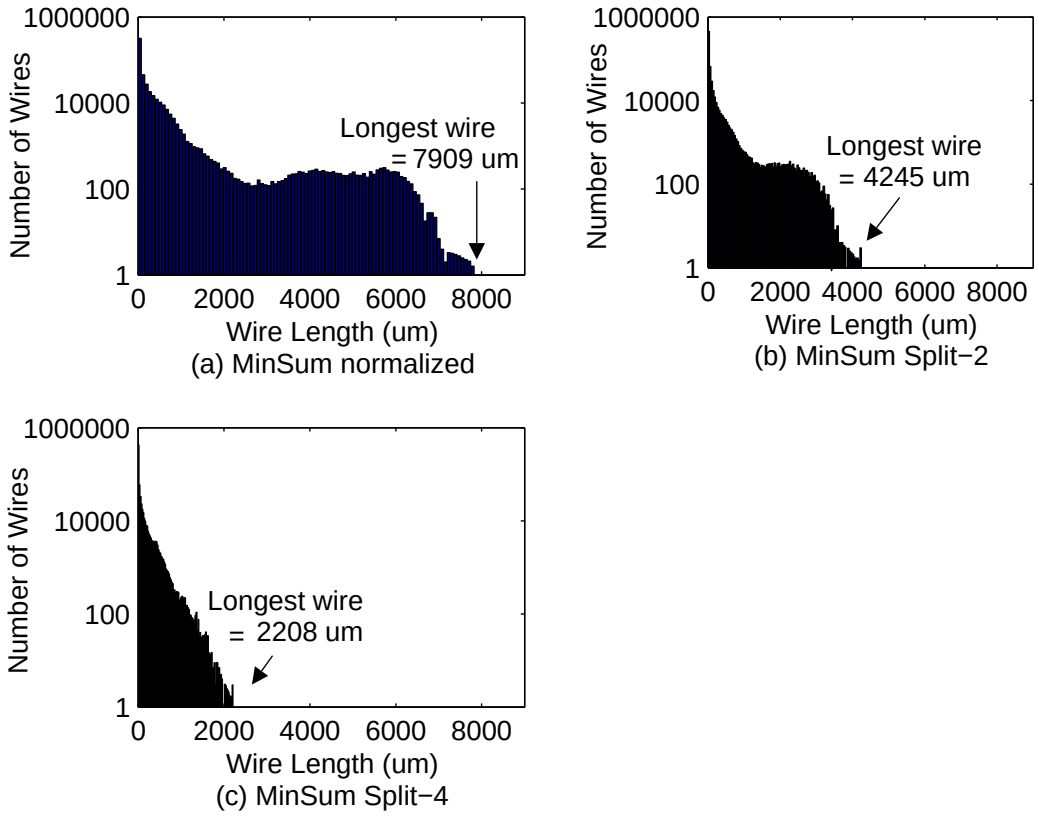


Figure 3.15: Wire length distribution for (a) MinSum normalized, (b) MinSum Split-2 and (c) MinSum Split-4 decoders

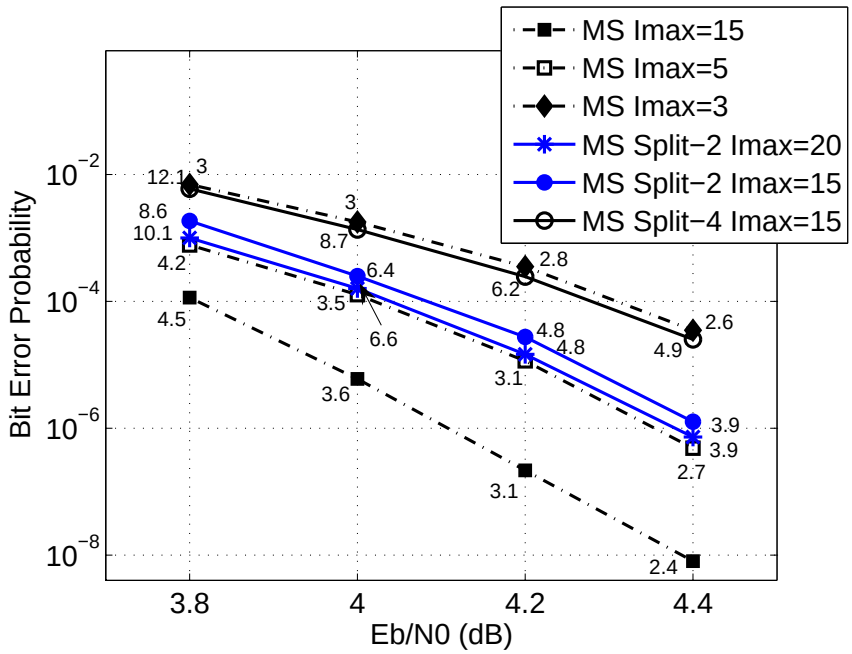


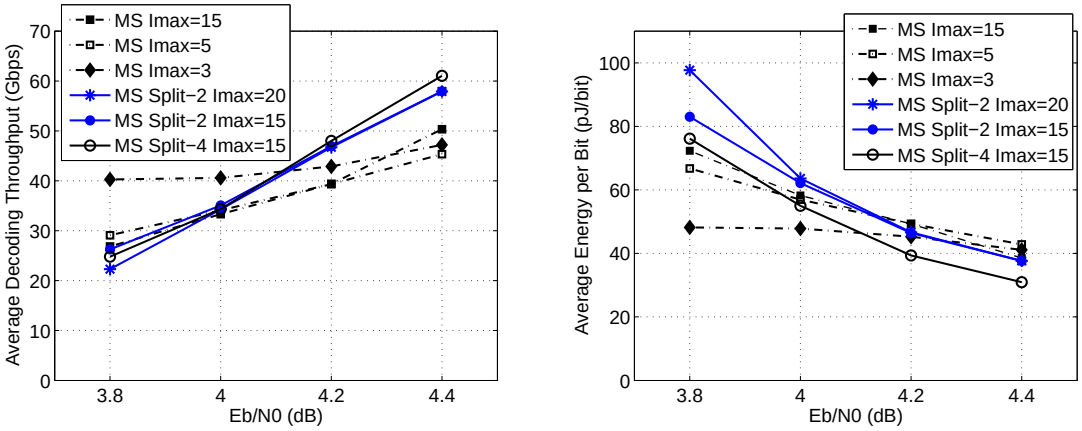
Figure 3.16: Error performance of the MinSum normalized, MinSum Split-2 and MinSum Split-4 decoders for (2048,1723) code with various maximum number of iterations (*Imax*). The average number of decoding iterations is shown at every simulation point.

energy efficient circuits.

3.5.4 Analysis of Maximum and Average Numbers of Decoding Iterations

The maximum number of decoding iterations strongly affects the best case error performance, the maximum achievable decoder throughput, and the worst case energy consumption. Fortunately, the majority of frames require only a few decoding iterations to converge (specially at high SNRs). By detecting early decoder convergence, throughput and energy can potentially improve significantly while maintaining the same error performance. Early convergence detection is done by a syndrome check circuit [18, 66] which checks the decoded bits every cycle (see Fig. 3.9) and terminates the decoding process when convergence is detected. Decoding of a new frame can begin if one is available.

Post-layout results show that the syndrome check block for a (2048,1723) code occupies only approximately 0.1 mm^2 and its maximum delay is 2 ns. By adding a pipeline stage for the syndrome check, the block's delay does not add at all to the critical path delay



(a) (b)

Figure 3.17: (a) Average decoding throughput and (b) average energy dissipation per bit in MinSum normalized, MinSum Split-2 and MinSum Split-4 decoders as a function of SNR and the average decoding iteration for different maximum numbers of iterations (*Imax*).

of the decoder.

Figure 3.16 compares the BER of MinSum normalized, MinSum Split-2, and MinSum Split-4 when decoding the (2048,1723) code with a maximum number of decoding iterations (*Imax*). At every simulation point the average number of iterations resulting in convergence is also shown. At the same BER and with an identical *Imax* setting, the average number of iterations in Split-2 is 1.2 to 1.4 times larger than MinSum normalized, and Split-4 is 1.3 to 1.5 times larger. Despite requiring more decoding iterations per block, the Split-2 and Split-4 decoders achieve a throughput 1.5 and 1.9 times higher and energy dissipation 1.3 and 2.0 times lower, respectively, when compared to the MinSum decoder at BER $= 3 \times 10^{-5}$. These data are detailed in Table 5.3.

Figure 3.17 shows the (a) average throughput and (b) average energy dissipation per bit for the same group of decoders shown in Fig. 3.16 as a function of SNR (at different *Imax*). The variance across SNR is caused by a varying number of decoding iterations to achieve convergence (see Fig. 3.16). For throughput results, we assume that there is always a new frame to be decoded upon request.

It is interesting to compare decoders at the same BER. From Fig. 3.16, Split-2 at $Imax = 20$ and MinSum normalized at $Imax = 5$ both have nearly the same BER. But the Split-2 implementation has 1.2 to 1.3 times higher throughput while consuming 1.1

times lower energy for SNR values larger than 4.1 dB. Similarly, Split-4 at $Imax = 15$ and MinSum normalized at $Imax = 3$ have nearly equal BER, but Split-4 has 1.1 to 1.3 times greater throughput and 1.1 to 1.4 times lower energy dissipation for SNR values larger than 4.1 dB.

In summary, with the same maximum number of decoding iterations ($Imax$) and at the same BER, the average number of decoding iterations ($Iavg$) of Split-2 and Split-4 are larger than that of MinSum normalized, but they still have larger throughput and energy efficiency at high SNR values. The maximum number of decoding iterations for MinSum normalized can be lowered until it obtains the same BER as Split-2 and Split-4. Even when MinSum normalized operates with a much lower number of iterations, Split-2 and Split-4 have higher throughput and energy efficiencies for most SNR values. In addition, Split-2 and Split-4 require 1.4 times and 3.3 times smaller circuit area, respectively, than the MinSum normalized decoder.

3.6 Summary

The proposed Split-Row and Multi-Split algorithms are viable approaches for high throughput, small area, and low power LDPC decoders, with a small error performance degradation that is acceptable for many applications—especially in mobile designs that typically have severe power and cost constraints. The method is especially well suited for long-length regular codes and codes with high row weights. Compared to standard (MinSum and SPA) decoding, the error performance loss of the method is about 0.35–0.65 dB for the implemented (2048,1723) code, depending on the level of splitting.

The proposed algorithm and architecture break check node processors into multiple blocks whose internal wires are all relatively short. These blocks are interconnected by a small number of sign wires whose lengths are almost zero. The result is decoders with denser, faster and more energy efficient circuits.

We have demonstrated the significant benefits of the splitting methods by implementing three decoders using MinSum normalized, MinSum Split-2, and MinSum Split-4 for the 2048-bit code used in the 10GBASE-T 10 Gigabit ethernet standard. Post-layout

simulation results show that the Split-4 decoder is 3.3 times smaller, attains 2.5 times higher throughput, and dissipates 2.5 times less energy per bit compared to a MinSum normalized decoder while performing 0.55 dB away from MinSum normalized at BER $= 5 \times 10^{-8}$ with 15 decoding iterations.

Using early termination circuits, the average number of decoding iterations in the Split-4 decoder is about 1.3 times larger than that of the MinSum normalized decoder. With early termination enabled, the Split-4 decoder's throughput is 1.9 times higher and its energy dissipation per bit is 2.0 times lower compared to the MinSum decoder at BER $= 3 \times 10^{-5}$.

Increasing the number of decoding iterations and increasing the fixed-point word width reduces the error performance loss in the Split-2 and Split-4 decoders. With a maximum of 20 decoding iterations, the error performance loss of the Split-2 decoder is reduced to 0.25 dB compared to MinSum normalized while it still achieves times higher throughput and occupies smaller circuit area.

Chapter 4

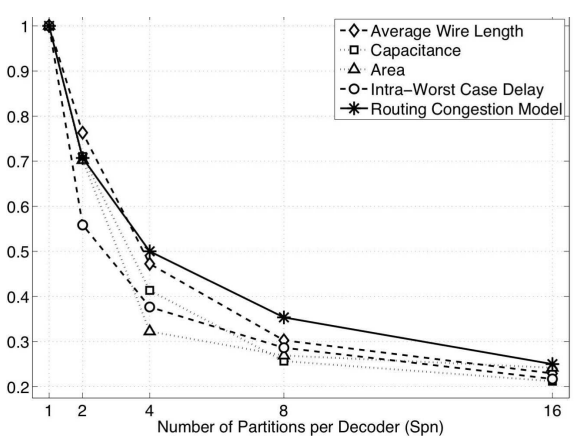
Split-Row Threshold Decoding Method

In order to improve the error correction performance of Split-Row, this chapter presents a low cost algorithm called Split-Row Threshold. We first study the Split-Row ability to reduce routing congestion in layout.

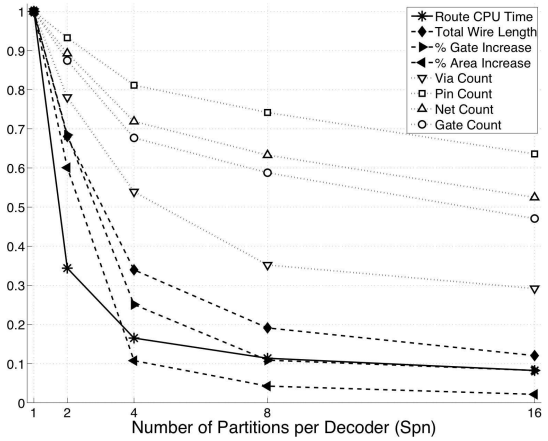
4.1 Routing Congestion Reduction with Split-Row

In theory, for Spn equally dimensioned matrices, we have $1/Spn$ times the number of computations and $1/Spn$ times the number of messages for each decoder partition per iteration. But since we still have Spn times the number of decoder partitions, then it initially appears that the interconnect, memory, and logic complexity should in fact be the same. However, it has been shown that, for full-parallel decoders, Split-Row provides significant increase in area utilization over other message-passing decoders that implement large row weight LDPC codes [18, 50, 49, 42]. This result is reasonable if we consider the physical role that interconnect complexity plays in the design of VLSI systems.

Given an LDPC decoder with equal Spn decoder partitions, each with $1/Spn$ logic and memory resources, the core area A_c per partition is proportional to Spn. A_c quantifies the silicon area used exclusively for logic and memory (e.g. standard cells, SRAMs,



(a) Normalized ratios of per partition properties: *average wire length*, *capacitance*, *area* and *worst case delay*, compared with the routing congestion model ($g_{Spn} = 1/\sqrt{Spn}$)



(b) Normalized ratios of whole decoder properties: *total wire length*, *percentage gate count & area increase* (i.e. A_c vs. A_{pl}) and *via/pin/net/gate count*, compared with routing CPU time

Figure 4.1: Physical indicators of interconnection complexity over five Spn-decoders ($Spn = 1, 2, 4, 8, 16$) normalized to the the case where $Spn = 1$ (i.e. MinSum). A 5-bit datapath (1-bit sign, 4-bit magnitude) is used for all five decoder implementations.

etc.). Routing congestion is defined as: $g = T_d/T_s$, where T_d is the number "tracks demanded", and T_s is the number of "tracks supplied" [65]. Tracks are the lengthwise (or widthwise) wires drawn assuming constant metal pitch and width. The maximum number of tracks for one metal layer is proportional to the length (or width) of A_c. Therefore, if we assume that the layout is square[1], $T_s \propto \sqrt{A_c}$, which provides a measure of the maximum routing resources available. Given that $T_d \propto (M \times (W_r/Spn) + (N/Spn) \times W_c)$ (i.e. the communication requirements) per partition, then as the decoder partition's A_c is reduced by $1/Spn$, the routing congestion for an LDPC decoder with Spn partitions is: $g_{Spn} \propto (1/Spn)/\sqrt{1/Spn} = 1/\sqrt{Spn}$. In other words, g_{Spn} is the ratio of interconnect complexity over a given chip dimension.

Figure 4.1(a) plots g_{Spn} with the per partition capacitance, area, average wire length, and worst-case (intra-partition) delay of five post-layout 10GBASE-T LDPC decoder implementations with $Spn = 1, 2, 4, 8, 16$. (Note that $Spn = 1$ represents a decoder using MinSum Normalized.) Clearly our routing congestion model ($g_{Spn} = 1/\sqrt{Spn}$) fol-

[1]If the layout is rectangular then we have two routing congestion numbers, g_{width} and g_{length}, and the same analysis is done for each. For simplicity we assume a square layout.

low these metrics closely. For all five decoders, we use a 5-bit datapath (1-bit sign and 4-bit magnitude). Since Split-Row eliminates the wires for check node magnitudes between partitions, its impact of routing congestion reduction is even more significant when implementing larger datapath widths. On the other hand, using a smaller datapath width results in significant error performance loss when compared to a floating point implementation. Usually a 4 to 6-bit datapath results in near floating point error performance.

Cadence's SoC Encounter CAD tool's computational complexity, given by its routing CPU time, is a real measure of routing congestion on their algorithms' ability to converge to a solution that satisfies the design rules, timing, etc. Since total wire length can be used as a pessimistic measure of routing congestion [65], then it should behave closely with the route CPU time. Figure 4.1(b) verifies this, and in fact, the percentage area (as well as gate count) increase as compared to the original core (synthesis/core area and gate count) also follow this trend. Via count also matches with the rate of change in total wire length, albeit at a different magnitude. Notice that gate, pin, and net counts do not increase rapidly until $Spn = 2$, which indicate that increases in wire buffering changes more dramatically starting at the $Spn = 2$ data point. Since the gate, pin, and net counts represent the connectivity needed by the design and is not decreasing as much as compared to total wire length for $Spn > 2$ this means that the amount of tracks needed (T_d) is decreasing faster than the silicon complexity. Therefore, both post-layout (A_{pl}) and core (A_c) areas will converge with increasing Spn: $\lim_{Spn\to\infty} A_{pl}(Spn) = A_c(Spn) \equiv A_c$.

In conclusion, theoretical results show that Split-Row can reduce routing congestion by a factor of $1/\sqrt{Spn}$. Actual implementation results of decoders at the post-layout step bore out this conclusion.

4.2 Split-Row Threshold Decoding Method

4.2.1 Split-Row Error-performance

Because Split-Row is a general modification of the message-passing algorithm, it can be used with both SPA and MinSum [52]. In addition, Split-Row, like SPA or MinSum,

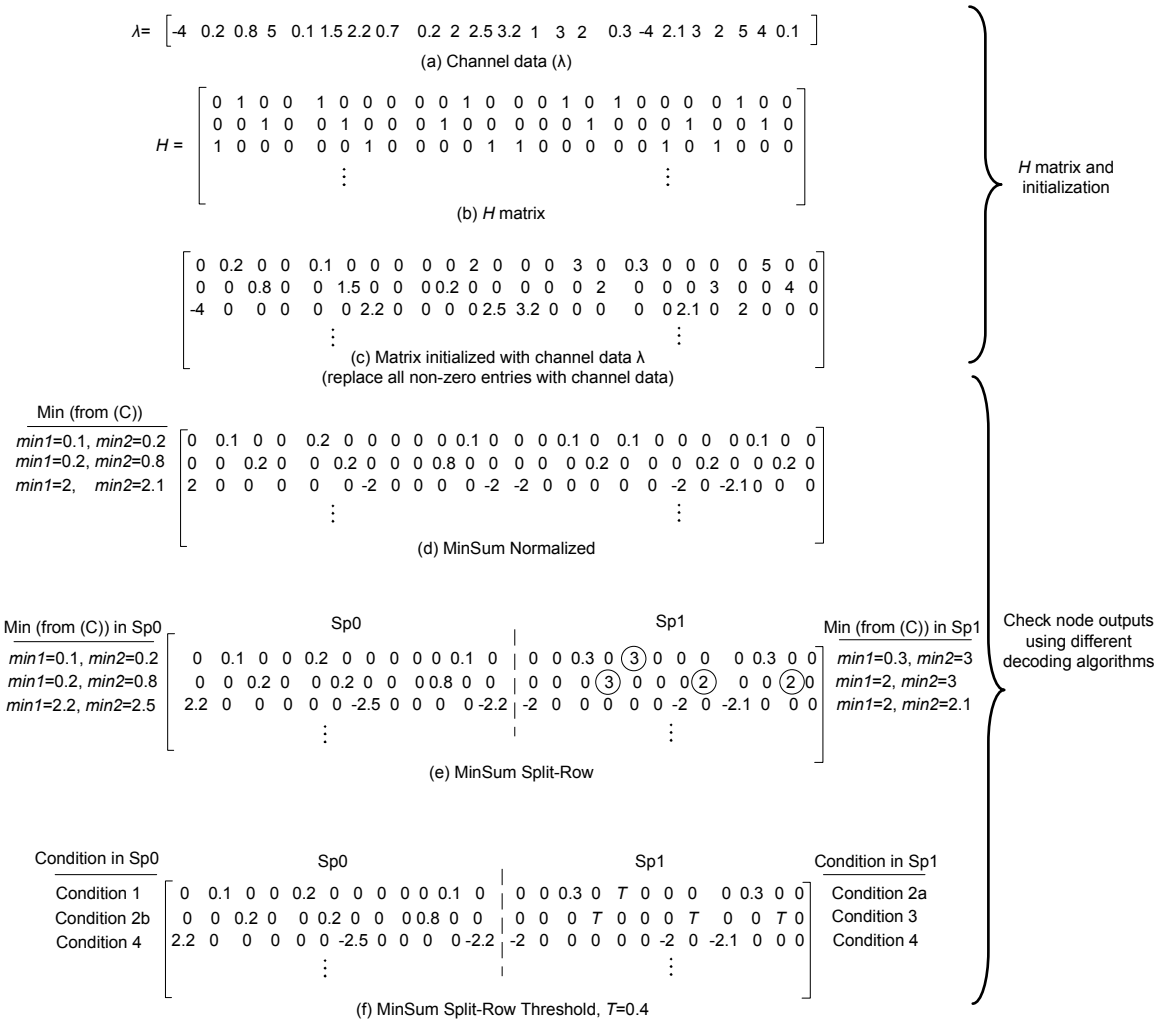


Figure 4.2: Channel data (λ), H matrix, the initialization matrix and the check node output (α) values after the first iteration using MinSum Normalized, MinSum Split-Row and Split-Row Threshold algorithm. The Split-Row entries in (e) with the largest deviation from MinSum Normalized are circled and are largely corrected with Split-Row Threshold method in (f). Correction factors are set to be one here.

is not restricted to either full-parallel, full-serial, or partial-parallel architectures. But the major benefit from "splitting" is its ability to reduce routing congestion, which primarily only affects the global interconnects caused by message-passing. Full-serial architectures will gain nothing, while full-parallel architectures will improve greatly; partial-parallel architectures will find their level of improvement based on the amount of "parallelism" versus "serialism" contained in their designs.

The major cost of partitioning comes from the incomplete message-passing that specifically affects check node updates (to be discussed in Sec. 4.2.1). The major drawback of Split-Row is that it suffers from a 0.4–0.7 dB error performance loss proportional to Spn compared to MinSum and SPA [43] decoders. Because each Split-Row partition has no information about the minimum value of the other partition, the minimum value in one partition could be much larger than the global minimum. Then the check node generated α values in the partition with the error are overestimated. This leads to a possible incorrect estimation of the bits during its variable node update.

Figure 4.2 shows (a) the channel data (λ) and (b) the first three rows of the parity check matrix for an LDPC code with $W_r = 6$, and $N = 24$. Figure 4.2 (c) shows the initialization step where all non-zero entries are initialized with channel data. Also, Fig. 4.2 shows the check node outputs using (d) MinSum, (e) MinSum Split-Row and (f) Split-Row Threshold based on the initialization step. To find $Min1$ and $Min2$ for each case, reader should look at Fig. 4.2(c). For example, in Fig. 4.2(c) in the first row, the entries are: 0.2, 0.1, 2, 3, 0.3 and 5. Therefore, $Min1$=0.1 and $Min2$=0.2 in Fig. 4.2(d).

In Split-Row, entries with the largest deviations from MinSum Normalized are circled. For example, in the second row of the Split-Row matrix output in Fig. 4.2(e), the local minimum ($Min1$) in Sp1 is "2", which is 10 times larger than the local minimum in Sp0, "0.2", which is also the global minimum of the entire row. This results in an overestimation of α values for the bits on the right side of the second row, possibly causing an incorrect decision for these three bits. In the first row, although $Min1$ in Sp1 which is "0.3" is close to the $Min1$ in Sp0 ("0.1"), $Min2$ in Sp1 ("3") deviates significantly from $Min2$ in Sp0 ("0.2") and this results in a large error in the bit on the right side.

4.2.2 Split-Row Threshold Algorithm

The Split-Row Threshold algorithm significantly improves the error performance without reducing the effectiveness of Split-Row while adding negligible hardware to the check node processor and one additional wire between blocks [53]. Like Split-Row, the check node processing step is partitioned into multiple semi-autonomous (Spn) partitions. Each partition computes local $Min1$ and $Min2$ simultaneously and sends the sign-bit with a single wire to the neighboring blocks serially. However, in the Split-Row Threshold algorithm, both $Min1$ and $Min2$ are additionally compared with a predefined threshold (T), and a single bit *threshold enable* ($Threshold\_en\_out$) global signal is sent to indicate the presence of a potential global minimum to other partitions. The $\alpha_{ij:Spk}$ values in Split-Row Threshold are computed as follows:

$$\alpha_{ij:Spk} = Sfactor \times \underbrace{\prod_{j' \in V(i) \setminus j} \text{sign}(\beta_{ij'})}_{\text{Sign Calculation}} \times \underbrace{Min_{Spk}}_{\text{Magnitude Calculation}} \tag{4.4}$$

where Min_{Spk} is given in Algorithm 1. As shown, four conditions will occur: **Condition 1** occurs when both $Min1$ and $Min2$ are less than threshold T, thus they are used to calculate α messages according to Eq. 4.1. Additionally, $Threshold\_ensp(\text{k})\_out(\text{k} \pm 1)$, which represents the general threshold enable signal of a partition Spk with two neighbors, asserted high, indicating that the least minimum ($Min1$) in this partition is smaller than T. **Condition 2**, as represented by *lines* **7 to 13**, occurs when only $Min1$ is less than T. As with **Condition 1**, $Threshold\_ensp(\text{k})\_out(\text{k}\pm1) = 1$. If at least one $Threshold\_ensp(\text{k}\pm1)$ signal from the nearest neighboring partitions is high, indicating that local minimum in the other partition is less than T, then we use $Min1$ and T to update the α messages, while using Eq. 4.2 (**Condition 2a**). Otherwise, we use Eq. 4.1 (**Condition 2b**). **Condition 3**, as represented by *lines* **14 to 16**, occurs when the local $Min1$ is larger than T and at least one $Threshold\_ensp(\text{k}\pm1)$ signal from the nearest neighboring partitions is high; thus, we only use T to compute all α messages for the partition using Eq. 4.3. **Condition 4**, as represented by *lines* **17 to 19** occurs when the local $Min1$ is larger than T and if the $Threshold\_ensp(\text{k} \pm 1)$ signals are all low; thus, we again use Eq. 4.1. The variable node

Algorithm 1 Split-Row Threshold Algorithm

Require: $j' \in V_{Sp\mathrm{k}}(i) \backslash j$

Require: $Min1_i$ and $Min2_i$ as given in Eqs. (2.8) and (2.9)

Require: T threshold value

// Finds $\min(|\beta_{ij'}|)$ and $Threshold\_ensp(\mathrm{k})\_out(\mathrm{k} \pm 1)$

// for the i-th partition of a Spn-decoder ($Sp\mathrm{k}$).

if $Min1_i \leq T$ **and** $Min2_i \leq T$ **then**

$Threshold\_ensp(\mathrm{k})\_out(\mathrm{k} \pm 1) = 1$

$$Min_{Spk} = \begin{cases} Min1_i \ , & \text{if } j \neq \operatorname{argmin}(Min1_i) \\ Min2_i \ , & \text{if } j = \operatorname{argmin}(Min1_i) \end{cases} \tag{4.1}$$

else if $Min1_i \leq T$ **and** $Min2_i > T$ **then**

$Threshold\_ensp(\mathrm{k})\_out(\mathrm{k} \pm 1) = 1$

if $Threshold\_ensp(\mathrm{k}+1) == 1$ **or** $Threshold\_ensp(\mathrm{k}-1) == 1$ **then**

$$Min_{Spk} = \begin{cases} Min1_i \ , & \text{if } j \neq \operatorname{argmin}(Min1_i) \\ T \ , & \text{if } j = \operatorname{argmin}(Min1_i) \end{cases} \tag{4.2}$$

else

do Equation (4.1)

end if

else if $Min1_i > T$ **and** ($Threshold\_ensp(\mathrm{k}+1) == 1$ **or** $Threshold\_ensp(\mathrm{k}-1) == 1$) **then**

$Threshold\_ensp(\mathrm{k})\_out(\mathrm{k} \pm 1) = 0$

$$Min_{Spk} == T \tag{4.3}$$

else

$Threshold\_ensp(\mathrm{k})\_out(\mathrm{k} \pm 1) = 0$

do Equation (4.1)

end if

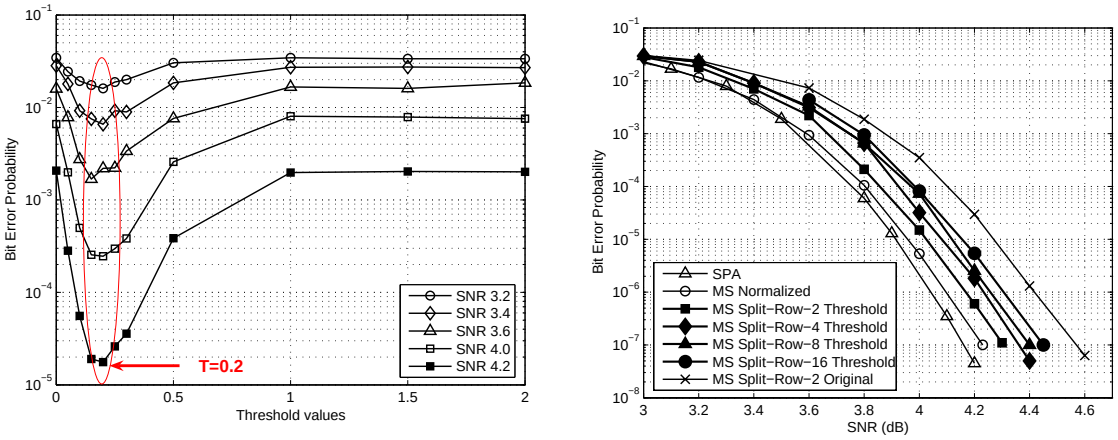
operation in Split-Row Threshold is identical to the MinSum Normalized and Split-Row algorithms.

The updated check node messages (α) after one iteration using Split-Row Threshold ($T = 0.40$) are shown in Fig. 4.2(f). Condition 1 and Condition 4 lead to the original Split-Row decoding, while Condition 2 and Condition 3 largely correct the errors by the original Split-Row algorithm. Note that this is a simplistic representation of the improvement that Split-Row Threshold decoding will have over Split-Row decoding method for larger parity check matrices.

4.2.3 Bit Error Simulation Results

The error performance depends strongly on the choice of threshold values. If the threshold T is chosen to be very large, most local *Min1* and *Min2* values will be smaller than T which results in only Condition 1 being met and the algorithm behaves like the original MinSum Split-Row. On the other hand if the threshold value is very small, most local minimums will be larger than T and only Condition 4 is met which is again the original MinSum Split-Row algorithm.

The optimum value for T is obtained by empirical simulations. Although the Threshold algorithm itself is independent of the modulation scheme and channel model, in this work we use BPSK modulation and an additive white Gaussian noise (AWGN) channel for all simulations. Simulations were run until 80 error blocks were recorded. Blocks were processed until the decoder converged early or the maximum of 11 decoding iterations was reached. To further illustrate the impact of the threshold value on error performance, Fig. 4.3(a) plots the error performance of a (6,32) (2048,1723) RS-based LDPC code [19], adopted for 10GBASE-T [4] versus threshold values for different SNRs, using Split-Row Threshold with $Spn = 2$. As shown in the figure, there are two limits for the threshold value which lead the algorithm to converge to the error performance of the original Split-Row. Also shown is the optimum value for the threshold ($T = 0.2$) for which the algorithm performs best. BER simulation results for a (16,16) (175,255) EG-LDPC and (4,16) (1536,1155) QC-LDPC codes show that the average optimal threshold T with two



(a) The BER performance versus threshold value (T) with different SNR values for Split-2 Threshold. The optimal region is circled with an average value of $T = 0.2$.

(b) Error performance results

Figure 4.3: The impact of choosing threshold value (T) on the error performance and BER comparisons for a (6,32)(2048,1723) LDPC code using Sum Product algorithm (SPA), MinSum Normalized, MinSum Split-Row(original) and Split-Row Threshold with different levels of partitioning, and with optimal threshold and correction factor values

partitions is 0.22 and 0.23, respectively.

The threshold value can be dynamically varied or made static. Example implementations include: T dynamically changed while processing, T statically configured at runtime, or T hard-wired directly into the hardware. In general, decoders will have higher throughput and higher energy efficiency the less T is allowed to vary. Fortunately, as can be seen in Fig. 4.3(a), the optimal BER performance is near the same threshold value for a wide range of SNR values, which means that dynamically varying T produces little benefit. Efficient implementations can use a static optimal value for T found through BER simulations.

One benefit of Split-Row Threshold decoding is that partitioning of the check node processing can be arbitrary so long as there are two variable nodes per partition and the error performance loss is less than 0.3 dB. For example, Fig. 4.3(b) shows the error performance results for a (6,32) (2048,1723) LDPC code for (from left to right) SPA, MinSum Normalized, MinSum Split-Row Threshold with different levels of splitting (Spn) and with optimal threshold values and, lastly Split-Row (original). As the figure shows, Split-2 Threshold is

	Split-2	Split-4	Split-8	Split-16
Spn	2	4	8	16
Average optimal T	0.20	0.23	0.24	0.24

Table 4.1: Average optimal Threshold value (T) for the Split-Row Threshold decoder with different levels of partitioning, for a (6,32) (2048,1723) LDPC code.

about 0.13 dB and 0.07 dB away from SPA and MinSum Normalized, respectively. The SNR loss between multiple Split-Row Threshold decoders is less than 0.05 dB and total loss from Split-16 Threshold to Split-2 Threshold is 0.15 dB at BER = 10^{-7}. Also shown in the plot is the Split-2 original algorithm which is still 0.12 dB away from Split-16 Threshold algorithm. Table 4.1 summarizes the average optimal threshold values (T) for Split-Row Threshold, and shows small changes with different partitioning levels.

4.3 Split-Row Threshold Decoding Architecture

4.3.1 Check Node Processor

The logical implementation of a check node processor in partition Spk using Split-Row Threshold decoding is shown in Fig. 4.4. The magnitude update of α is shown along the upper part of the figure while the global sign is determined with the XOR logic along the lower part. In Split-Row Threshold decoding, the sign bit calculated from partition Spk is passed to the $Sp(\mathrm{k}-1)$ and $Sp(\mathrm{k}+1)$ neighboring partitions to correctly calculate the global sign bit according to the check node processing equation Eq. 2.6 and Eq. 4.4.

In both MinSum Normalized and Split-Row Threshold decoding, the first minimum $Min1$ and the second minimum $Min2$ are found alongside the signal $IndexMin1$, which indicates whether $Min1$ or $Min2$ is chosen for a particular α. We use multiple stages of comparators to find $Min1$ and $Min2$. The first stage (the left most) is composed of simple comparators which sort the two inputs and generate min and max outputs. The second stage and afterwards consist of 4 to 2 comparators and a block is shown in the right corner of Fig. 4.4. One benefit of the Split-Row Threshold algorithm is that the number of β inputs

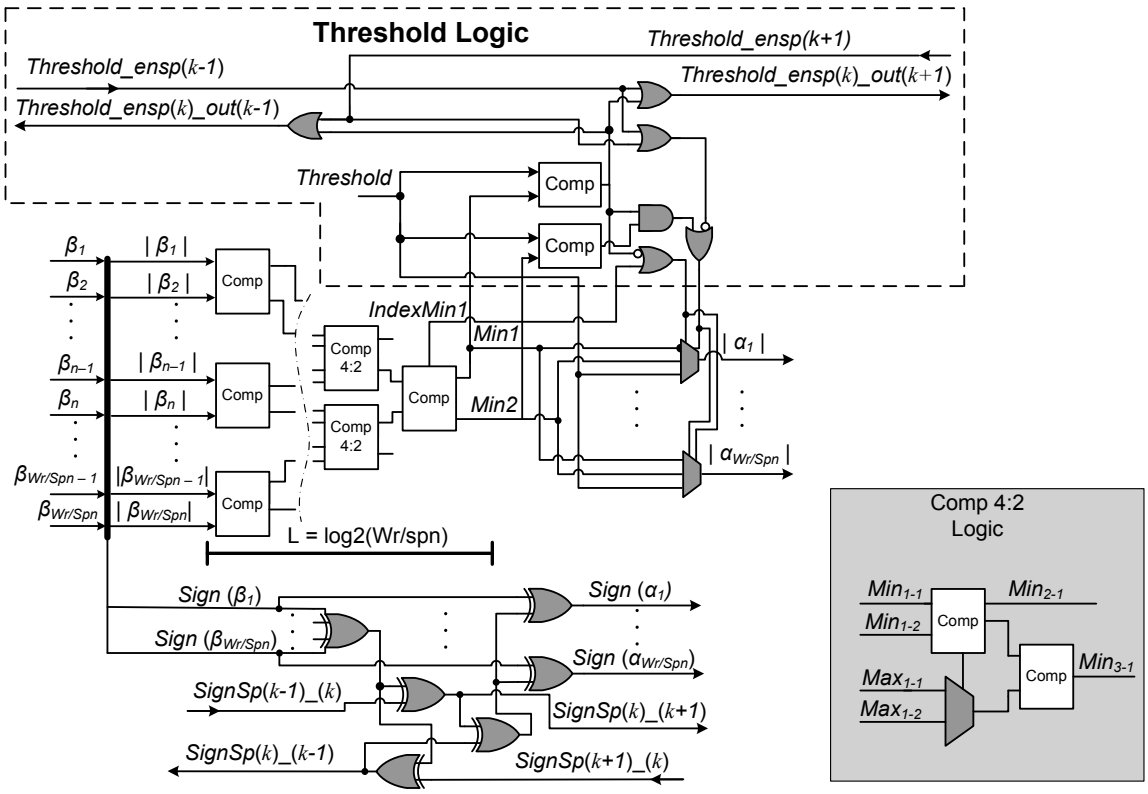


Figure 4.4: The block diagram of magnitude and sign update in check node processor of partition Sp(k) in Split-Row Threshold decoder. The *Threshold Logic* is shown within the dashed line. The 4:2 comparator block is shown in the right.

(variable node outputs) to each check node is reduced by a factor of $1/Spn$, which lowers the circuit complexity of each check node processor as the number of comparator stages is reduced to $L = \log_2(W_r/Spn)$.

The threshold logic implementation is shown within the dashed line which consists of two comparators and a few logic gates. The *Threshold Logic* contains two additional comparisons between $Min1$ and $Threshold$, and $Min2$ and $Threshold$, which are used to generate the final α values. The local $Threshold\_en$ signal that is generated by comparing $Threshold$ and $Min1$ is OR'ed with one of the incoming $Threshold\_en$ signals from $Sp(\mathrm{k}-1)$ and $Sp(\mathrm{k}+1)$ neighboring partitions and is then sent to their opposite neighbors.

4.3.2 Variable Node Processor

The variable node equations remain unchanged between MinSum Normalized and Split-Row Threshold algorithms, and thus the variable node processors are identical in all cases. Fig. 4.5 shows the variable node processor architecture for both MinSum and Split-Row Threshold decoders, which computes β messages according to Eq. 2.3 and contains multi-stage adders. Its complexity highly depends on the numbers of inputs (column weight W_c) and the input wordwidths. As shown in the figure, this implementation uses a 5-bit datapath.

4.3.3 Full-parallel Decoder Implementation

The block diagram of a full-parallel implementation of Split-Row Threshold decoding with Spn partitions, highlighting the $Sign$ and $Threshold\_en$ passing signals, is shown in Fig. 4.6. These are the only wires passing between the partitions. In each partition, local minimums are generated and compared with T simultaneously. If the local minimum is smaller than T then the $Threshold\_en$ signal is asserted high. The magnitude of the check node outputs are computed using local minimums and the $Threshold\_en$ signal from neighboring partitions. If the local partition's minimums are larger than T, and at least one of the $Threshold\_en$ signals is high, then T is used to update its check node outputs. Otherwise, local minimums are used to update check node outputs. Figure 4.7 (a) shows the

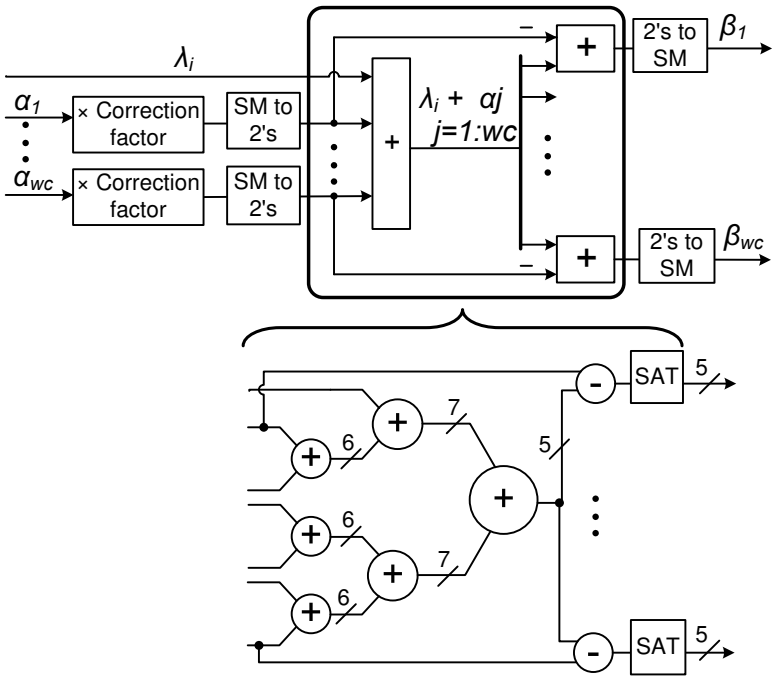


Figure 4.5: The block diagram of variable node update architecture for MinSum Normalized and Split-Row Threshold decoders.

pipeline diagram of one partition in the decoder. The timing diagram of a check node and a variable node update is shown in Fig. 4.7 (b). The check and variable node messages are updated one after the other in one cycle after receiving the $Sign$ and $Threshold\_en$ signals from its neighboring partitions. In full-parallel implementations all check and variable processor outputs are updated in parallel, and as shown in the timing diagram in Fig. 4.7 (b), it takes one cycle to update all messages for one iteration. Therefore the throughput for the proposed full-parallel decoder with code length N is:

$$Throughput = \frac{N * f}{Iterations} \tag{4.5}$$

where f is the maximum speed of the decoder and is based on the delay of one iterative check node and variable node processing.

4.4 Design of Five CMOS Decoders

To further investigate the impact on the hardware implementation due to partitioning, we have implemented five full-parallel decoders using MinSum Normalized and Split-

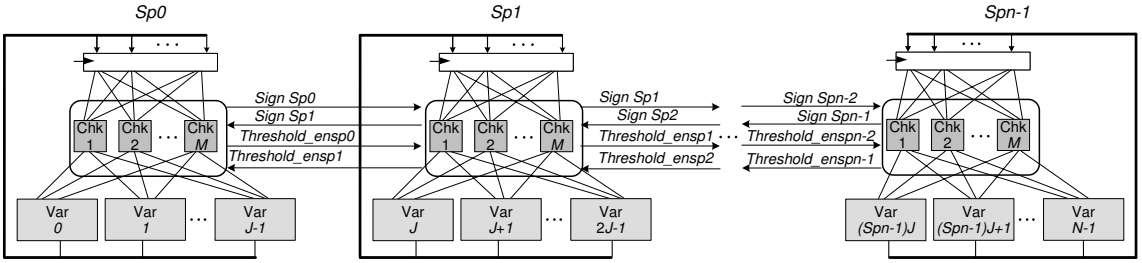


Figure 4.6: Top level block diagram of a full-parallel decoder corresponding to a $M \times N$ parity check matrix, using Split-Row Threshold with Spn partitions. The inter-partition $Sign$ and $Threshold\_en$ signals are highlighted. $J = N/Spn$, where N is the code length.

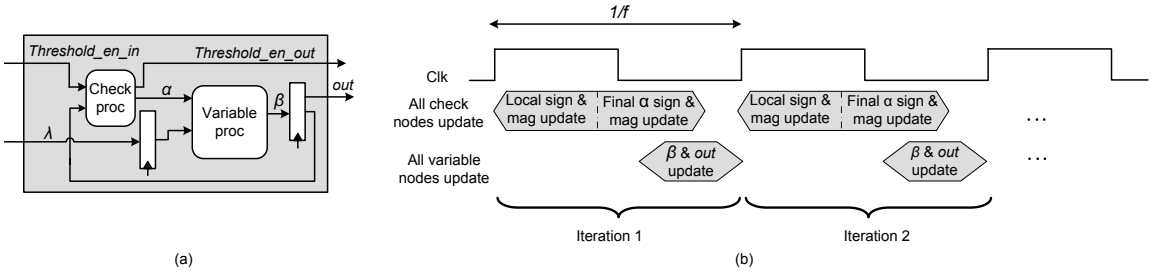


Figure 4.7: (a) The pipeline and (b) the timing diagram for one partition of Split-Row Threshold decoder. In each partition, the check and variable node messages are updated in one cycle after receiving the $Sign$ and $Threshold\_en$ signals from the nearest neighboring partitions.

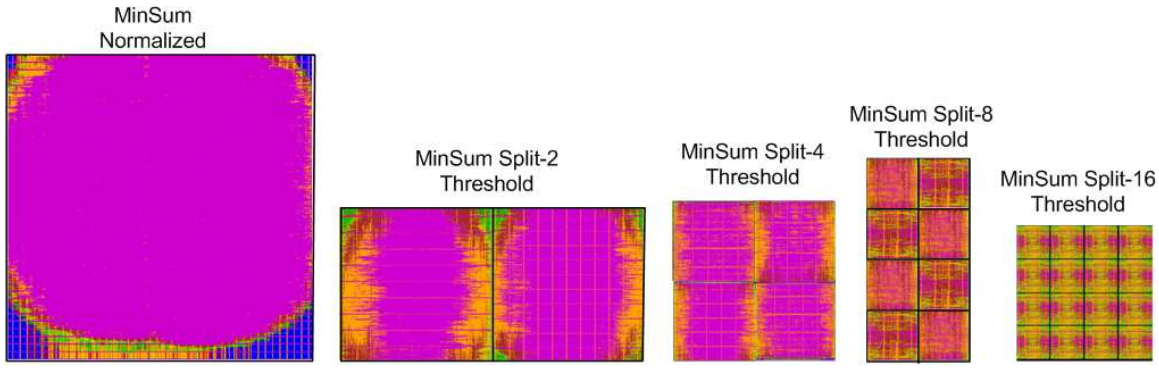


Figure 4.8: Layout of MinSum Normalized and Split-Row Threshold decoder implementations shown approximately to scale for the same code and design flow

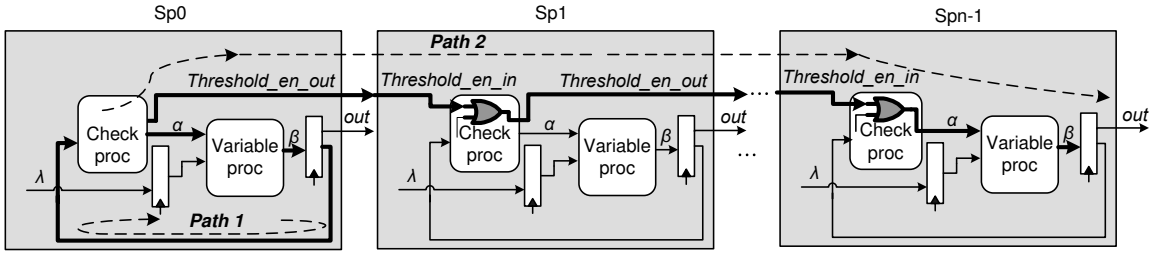


Figure 4.9: The check to variable processor critical path *Path1*, and the inter-partition $Threshold\_en$ critical path *Path2* for the Split-Row Threshold decoding method with Spn partitions.

Row Threshold methods with multiple partitioning for the (6,32) (2048,1723) 10GBASE-T LDPC code in 65 nm, 7-metal layer CMOS. All circuit-related performance results are measured under "typical" process and temperature conditions.

The parity check matrix of the 10GBASE-T code has 384 rows, 2048 columns, row weight 32 ($W_r = 32$), column weight 6 ($W_c = 6$) and information length 1723. The full-parallel MinSum Normalized decoder has 384 check and 2048 variable processors corresponding to the parity check matrix dimensions M and N, respectively. The split architectures reduce the number of interconnects by reducing the number of columns per sub-block by a factor of $1/Spn$. For example, in each Split-16 sub-block there are 384 check processors (though simplified), but only 128 (2048/16) variable processors. The area and speed advantage of a Split-Row Threshold decoder is significantly higher than in a MinSum Normalized implementation due to the benefits of smaller and relatively lower complexity partitions, each of which communicate with short and structured sign and $Threshold\_en$-passing wires. In this implementation, we use a 5-bit fixed-point datapath, which results in about 0.1 dB error performance loss for MinSum Normalized and MinSum Split-Row Threshold decoders, when compared to the floating point implementation. Increasing the fixed-point word width improves the error performance at the cost of a larger number of global wires and larger circuit area.

4.4.1 Design Flow and Implementation

We use a standard-cell-based automatic place and route flow to implement all decoders. The decoders were developed using Verilog to describe the architecture and hard-

ware, synthesized with Synopsys Design Compiler, and placed and routed using Cadence SOC Encounter. Each block is independently implemented and connected to the neighboring blocks with *Sign* and *Threshold_en* wires. We pass these signals across each block serially. One of the key benefits of the Split-Row Threshold decoder is that it reduces the time and effort for a full-parallel decoder implementation of large LDPC codes using automatic CAD tools. Since Split-Row Threshold reduces the check node processor complexity and the interconnection between check nodes and variable nodes per block, then each block becomes more compact whose internal wires are all relatively short. The blocks are interconnected by a small number of sign wires. This results in denser, faster and more energy efficient circuits. Split-Row also has the potential to minimize cross talk and IR drops due to reduced wire lengths, reduced routing congestion, more compact standard cell placement, and lower overall area.

Figure 4.8 shows the post-route GDSII layout implementations drawn roughly to scale for the five decoders using MinSum Normalized and Split-Row Threshold algorithms with multiple levels of partitioning. In addition to the significant differences in circuit area for complete decoders, the even more dramatic difference in individual "place and route blocks" is also apparent.

4.4.2 Delay Analysis

In MinSum Normalized decoder, the critical path is the path along a partition's local logic and wire consisting of the longest path through the check node and variable node processors. However, in Split-Row Threshold decoder, since the *Threshold Logic* (see Fig. 4.4) is also dependent on the neighboring $Threshold\_en$ signals from partitions $Sp(\mathrm{k}-1)$ and $Sp(\mathrm{k}+1)$, one possible critical path is a $Threshold\_en$ signal that finally propagates to partition $Sp\mathrm{k}$ and changes the mux select bits influencing check node output α. To illustrate both critical paths, Fig. 4.9 highlights two possible worst case paths for Split-Row Threshold decoders in bold. Path1 shows the original delay path through the check and variable processors. While Path2 shows the propagation of the $Threshold\_en$ signal starting from the leftmost partition's ($Sp0$) check node processor, through $Spn-2$ middle blocks,

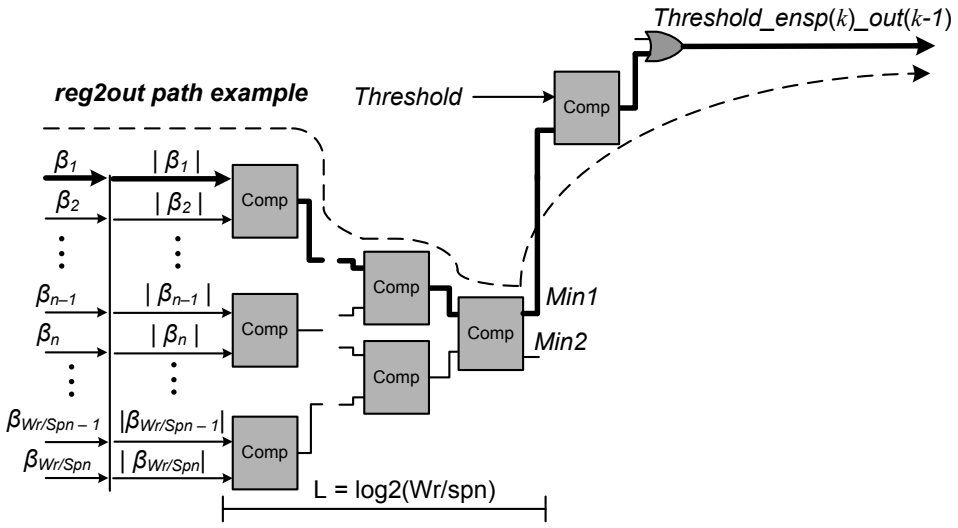


Figure 4.10: The components in the *reg2out* delay path for *Threshold_en* propagation signal in Split-Row Threshold decoding.

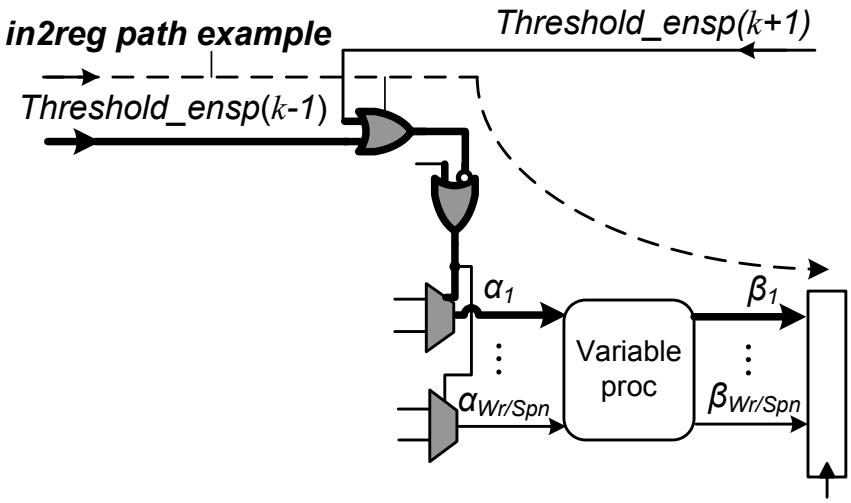


Figure 4.11: The components in the *in2reg* delay path for *Threshold_en* propagation signal in Split-Row Threshold decoding.

and finally to the variable node processor of the rightmost partition (Sp_{n-1}). In general, $Threshold\_en$ propagation path consists of three delay blocks:

1. *Origin Block (reg2out delay):* This is the path where $Threshold\_en$ signal is generated in a block, and is shown in Fig. 4.10. As shown in the figure, the path consists of comparators to generate $Min1$ and $Min2$, in addition to a comparison with $Threshold$ (T) and an OR gate to generate the $Thrshold\_en\_out$ signal going to the next partition.

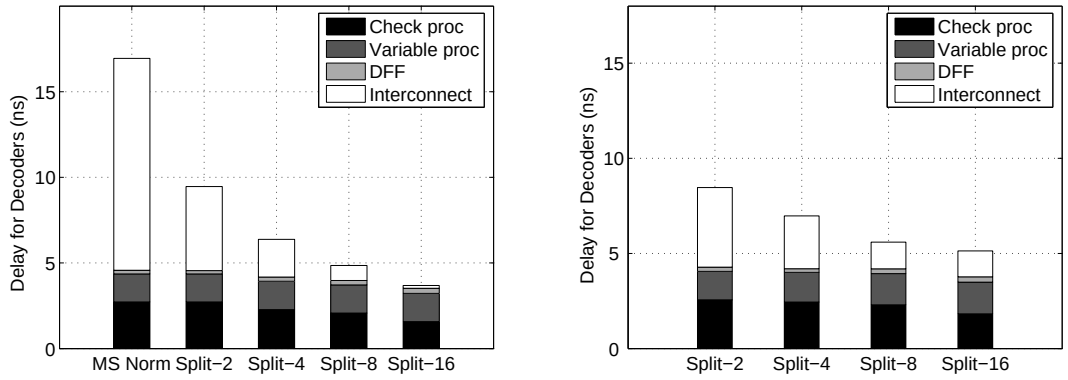
2. *Middle Blocks (in2out delay):* This path consists of middle blocks where $Threshold\_en$ signal is passing through. Assuming local $Min1$ and $Min2$ in all blocks are generated simultaneously, the delay in a middle block is one OR gate which generates the $Threshold\_en\_out$ signal.

3. *Destination Block (in2reg delay):* This is the path that a block updates the final check node output (α) and is using the $Threshold\_en$ signal from neighboring partitions. The path is shown in Fig. 4.9 which goes through the variable processor and ends at the register.

Table 4.2 summarizes the *reg2out*, *in2out* and *in2reg* delay values for four Split-Row Threshold decoders. As shown in the table, the *in2out* delay remains almost unchanged due to the fact that it is one OR gate delay. For Split-2, there is no middle block and therefore *in2out* delay is not available. Total inter-partition $Threshold\_en$ delay (T_{Th_en}) for an Spn-way Split-Row Threshold decoder is the summation of all three delay categories:

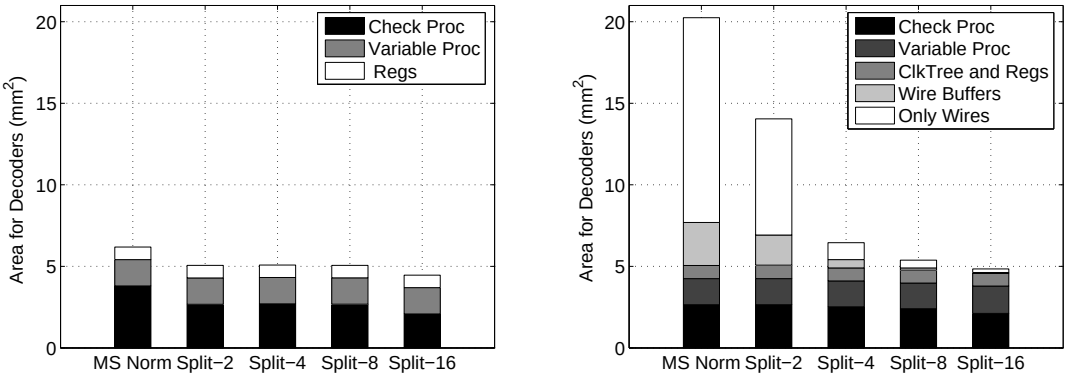
$$T_{Th\_en} = T_{reg2out} + (Spn - 2) \cdot T_{in2out} + T_{in2reg} \tag{4.6}$$

Just as with check to variable delays, $Threshold\_en$ delays also are subject to the effects of interconnect delays. Delays in the *in2reg* and *reg2out* paths both decrease with increased splittings due to the lessening wiring complexity. Note that because of the decrease in comparator stages with each increase in splitting in the check node processor, the *reg2out* delay sees a significant reduction while the worst case serial $Threshold\_en$ signal path's *in2out* increases its contribution by $Spn - 2$ as shown in Eq. 4.6.



(a) Path1: Check to Variable processor delay (b) Path2: *Threshold_en* propagation delay

Figure 4.12: Post-route delay breakdown of major components in the critical paths of five decoders using MinSum Normalized and Split-Row Threshold methods.



(a) Post synthesis (b) Post layout

Figure 4.13: Area breakdown for five decoders using MinSum Normalized and Split-Row Threshold methods. The interconnect and wire buffers are added after layout which take a large portion of MinSum Normalized and Split-2 Threshold decoders.

	Split-2	Split-4	Split-8	Split-16
Spn	2	4	8	16
$T_{reg2out}$ (ns)	3.667	2.784	1.911	1.328
T_{in2reg} (ns)	4.788	3.964	3.189	2.729
T_{in2out} (ns)	-	0.113	0.082	0.077

Table 4.2: *Threshold_en* delay path components for the Split-Row Threshold decoders.

The maximum speed of an Spn-way Threshold decoder ($T_{Spn\_threshold}$) is determined by the maximum between $Threshold\_en$ delay (T_{Th_en}) and the check to variable processor delay ($T_{C\_to\_V}$) paths:

$$T_{Spn\_threshold} = max\{T_{Th\_en}, T_{C\_to\_V}\} \tag{4.7}$$

Although the $Sign$ bit is also passed serially since its final value is updated with a single XOR logic in each block, its delay is smaller than the $Threshold\_en$ propagation delay and therefore is not considered.

The bar plots in Fig. 4.12 show the post-route delay breakdown of (a) Path1 (Check to Variable processor) and (b) Path2 ($Threhsold\_en$ propagation) in the decoders and are partitioned into interconnect and logic (check, variable processors and registers). The timing results are obtained using extracted delay/parasitic annotation files. As shown in the figures, for MinSum Normalized and Split-2 Threshold, Path1 is the dominant critical path but for $Spn > 2$, Path2 ($Threshold\_en$ propagate path) begins to dominate due to the overwhelming contribution of the $(Spn - 2)$ term in Eq. 4.6.

The figures show that while the variable processor delay remains constant (because all decoders use the same variable node architecture) the check node processor delay improves with the increase of splitting. For MinSum Normalized and Split-2 Threshold, the interconnect delay is largely dominant. This is caused by the long global wires between large number of processors. The interconnect path in these decoders is composed primarily of a long series of buffers and wire segments. Some buffers have long RC delays due to large fanouts of their outputs. For the MinSum Normalized and Split-2 decoders, the summation of interconnect delays caused by buffers and wires (intrinsic gate delay and RC delay) in Path1 are 12.4 ns and 5.1 ns, which are 73% and 50% of their total delay, respectively.

4.4.3 Area Analysis

Figure 4.13 shows the decoder area after (a) synthesis and (b) layout. The area of decoder after synthesis remains almost the same. However, for the MinSum Normalized and Split-2 decoders the layout area deviates significantly from the synthesized area. The reason is because of the inherent interdependence between many number of check and

variable nodes for large row weight LDPC codes, the number of timing critical wires that the automatic place and route tool must constrain becomes an exponentially challenging problem. Typically, the layout algorithm will try to spread standard cells apart to route the gates. This results in a lower logic (i.e. silicon/transistor) utilization and a larger overall area. As an additional illustration, Fig. 4.13 shows the area breakdown of the basic contributors of synthesis and layout for the decoders. As shown in the postlayout figure, more than 62% and 49% of the MinSum Normalized and Split-2 decoder area is without standard cells and is required for wiring.

Also, another indication of circuit area is the wire length in the decoder chips where there exist a limited number of metal layers (7 metal layers). In MinSum Normalized and Split-2 decoders, average wire lengths are 93 μm and 71 μm which are 4.4 and 3.4 times longer than Split-16.

4.4.4 Power and Energy Analysis

The energy consumed by a decoder is directly proportional to capacitance, and by setting all decoder designs to the same voltage, then their given capacitances will indicate energy efficiency. Because our routing congestion model (Fig. 4.1(a)) can follow capacitance versus the number of partitions, then for a given Spn, normalized capacitance is $C_{Spn}/C_{Spn=1} = 1/\sqrt{Spn}$. For $Spn \to \infty$ energy efficiency will be limited to the algorithm and architecture—not the routing congestion in layout.

The average power for Split-16 Threshold is 0.70 times that of MinSum. Interestingly Split-16's operating frequency is 3.3 times that of MinSum, so if we simplify by assuming equal activity for both designs, then effective lumped capacitance is decreasing at a rate faster than the increased performance in terms of RC delay (again, for simplicity, we assume core logic gates have also been unchanged)—see Fig. 4.14.

Additional savings to average power and energy (along with increased throughput) can be achieved through early termination. This technique checks the decoded bits every cycle and will terminate the decoding process when convergence is detected. The cost of the early termination circuit is the use of the already existing XOR signals (the sign(β) calculation), which gives "1" and "0". Parity is then checked through an OR gate tree with

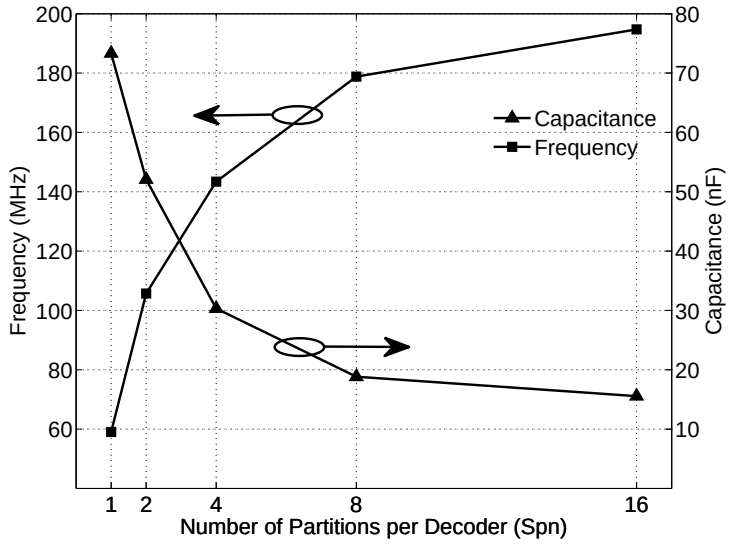


Figure 4.14: Capacitance and maximum clock frequency versus the number of partitioning Spn.

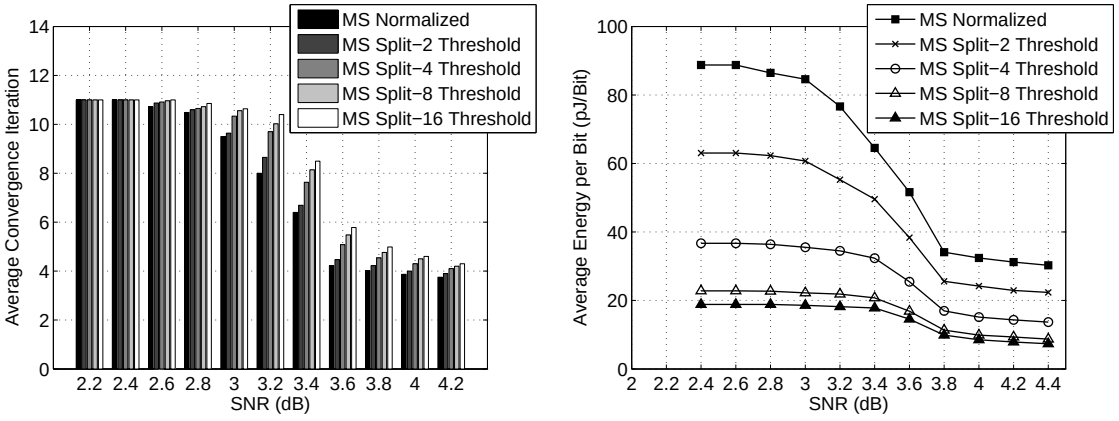
these XOR signals as inputs [18]. Post-layout results show that the early termination block for a (2048,1723) code occupies only approximately 0.1 mm^2.

Figure 4.15(a) shows the average convergence iterations for MinSum Normalized and Split-Row Threshold decoders for a range of SNR values with $I_{max} = 11$. At low SNR values ($SNR \leq 2.8$ dB) most decoders cannot converge within 11 iterations. For SNR values between 3.0 to 3.8 dB, the average convergence iteration of MinSum Normalized is about 30% to 8% less than the Split-Row Threshold decoders. For large SNRs ($SNR \geq 3.8$ dB) the difference between number of iterations for decoders ranges from 18% to 1%, indicating that all decoders can converge almost within the same average number of iterations. Figure 4.15(b) shows the average energy dissipation per bit of the five decoders. The MinSum Normalized decoder dissipates 3.4 to 4.7 times higher energy per bit compared to the Split-16 decoder.

4.4.5 Summary and Further Comparisons

Table 4.3 summarizes the post layout implementation results for the decoders.

The Split-16 decoder's final logic utilization is 97% which is 2.6 times higher than



(a) Average convergence iteration

(b) Average energy dissipation

Figure 4.15: Average convergence iteration and energy dissipation versus a large number of SNR values for five decoders using MinSum Normalized and Split-Row Threshold methods.

MinSum Normalized. The average wire length in each sub-block of Split-16 is 21 μm, which is 4.4 times shorter than in MinSum Normalized. It occupies 4.84 mm^2, runs at 195 MHz, delivers 36.3 Gbps throughput, and dissipates 37 pJ/bit with 11 iterations.

Compared to MinSum Normalized, Split-16 is 4.1 times smaller, has a clock rate and throughput 3.3 times higher, is 4.8 times more energy efficient, and has an error performance degradation of only 0.23 dB with 11 iterations. At BER = 10^{-7}, the average number of iterations of Split-16 is 1.15 times larger than MinSum Normalized and it has a coding loss of 0.23 dB compared to MinSum Normalized. At this BER point, its average decoding throughput is 92.8 Gbps which is 2.9 times higher and dissipates 15 pJ/bit which is 4 times lower than the MinSum Normalized decoder.

At a supply voltage of 0.7 V, the Split-16 decoder runs at 35 MHz and achieves the minimum 6.5 Gbps throughput required by the 10GBASE-T standard [4] (which requires 6.4 Gbps). Power dissipation is 62 mW at this operating point. These results are obtained by interpolating operating points using measured data from a recently fabricated chip on the exact same process [71].

	Normalized MinSum	Split-2	Split-4	Split-8	Split-16
CMOS fabrication process			65 nm CMOS, 1.3 V		
Initial area utilization	25%	35%	75%	88%	**94%**
Final area utilization	38%	51%	85%	92%	**97%**
Wire length per sub-block (μm)	93	71	44	28	**21**
Core area (mm^2)	20	14.05	6.45	5.38	**4.84**
Clock frequency (MHz)	59	106	143	179	**195**
Average Power (mW)	1941	2463	1950	1511	**1359**
Leakage power (mW)	20.3	18.1	8.9	6.4	6.5
Throughput @$Imax$ (Gbps)	11.0	19.7	26.7	33.3	**36.3**
Energy per bit @$Imax$ (pJ/bit)	177	125	73	45	**37**
SNR@ BER = 10^{-7} (dB)†	4.32	4.40	4.46	4.50	4.55
Iterations @BER = 10^{-7} ($Iavg$)	3.75	3.9	4.1	4.2	4.3
Throughput @$Iavg$ (Gbps)	32.2	55.6	71.4	87.2	**92.8**
Energy per bit @$Iavg$ (pJ/bit)	60	44	27	17	**15**

Table 4.3: Comparison of full-parallel decoders in 65 *nm*, 1.3 V CMOS, for a (6,32) (2048,1723) code implemented using MinSum Normalized and Split-Row Threshold with different levels of splitting. Maximum number of iterations is $Imax$ =11.
† The BER and SNR values are for 5-bit fixed-point implementations.

	Liu [41]	Zhang [84]		**This work**	
Decoding algorithm	Sum Product	Two step MinSum		Split-16 Threshold	
CMOS fabrication process	90 nm, 8M	65 nm, 7M		65 nm, 7M	
Level of parallelism	partial-parallel	partial-parallel		full-parallel	
Bits per message	5	4		5	
Logic utilization	50%	80%		97%	
Total chip area (mm^2)	14.5	5.35		4.84	
Maximum iterations ($Imax$)	16	8		11	
SNR @ BER = 10^{-7} (dB)	4.35	4.25		4.55	
Supply voltage (V)	-	1.2	0.7	1.3	0.7
Clock speed (MHz)	207	700	100	195	35
Latency (ns)	-	137	960	56.4	314
Throughput @ $Imax$ (Gbps)	5.3	14.9*	2.1*	**36.3**	**6.5**
Throughput with ET† (Gbps)	-	47.7	6.67	**92.8**	**16.6**
Throughput per area (Gbps/mm^2)	0.36	8.9	1.2	**19.1**	**3.4**
Power (mW)	-	2800	144	1359	62
Energy per bit with ET (pJ/bit)	-	58.7	21.5	**15**	**3.7**

Table 4.4: Comparison of the Split-16 Threshold decoder with published LDPC decoder implementations for the 10GBASE-T code.
† ET stands for "early termination". * Throughput is computed based on the maximum latency reported in the paper.

4.4.6 Comparison with Other Implementations

The Split-16 Threshold MinSum decoder post-layout simulation results are compared with recently implemented decoders [41], [84] for the 10GBASE-T code and are summarized in Table 5.4. Results for two supply voltages are reported for the Split-16 decoder: a nominal 1.3 V and 0.7 V, which is the minimum voltage that can achieve the 6.5 Gbps throughput required by the 10GBASE-T standard.

The partial parallel decoder chip [84] fabricated in 65 nm and consists of a two-step decoder: MinSum and a post-processing scheme which lowers the error floor down to BER = 10^{-14}. The sliced message passing (SMP) scheme in [41] is proposed for Sum Product algorithm and divides the check node processing into equal size blocks and perform the check node computation sequentially. The post-layout simulations for a partial-parallel decoder are shown in the table.

Compared to the two-step decoder chip [84], the Split-16 decoder is 1.1 times smaller, has 1.9 times higher throughput and dissipates 3.9 times less energy, at a cost of 0.3 dB coding gain reduction. Compared to the sliced message passing decoder [41], Split-16 is about 3 times smaller and has 6.8 times higher throughput with 0.2 dB coding gain reduction.

4.5 Summary

In this chapter, we gave a complete, detailed and unified presentation of Split-Row Threshold which utilizes a threshold enable signal to compensate for the loss of min() inter-partition information in Split-Row. It provides at least 0.3 dB error performance improvement over Split-Row with $Spn = 2$. Details of the algorithm with a step-by-step matrix example along with BER simulations are given.

The architecture and layout of five full-parallel LDPC decoders for 10GBASE-T using MinSum Normalized and MinSum-Split-Row Threshold methods in 65 nm CMOS are presented.

Post-layout results show that when compared with MinSum Normalized, Split-16 Threshold has 2.6× higher logic utilization, is 4.1× smaller, has a clock rate and throughput

3.3× higher, is 4.8× more energy efficient, and has an error performance degradation of 0.23 dB with 11 iterations. At 1.3 V, it can attain up to 92.8 Gbps and at 0.7 V it can meet the necessary 6.4 Gbps throughput for 10GBASE-T while dissipating 62 mW. In comparisons to other published LDPC chips, Split-16 can be up to 3× smaller with 6.8× more throughput and 4.2× lower energy consumption.

Chapter 5

Adaptive Wordwidth Decoder

In order to achieve additional power savings this chapter introduces an adaptive wordwidth algorithm that takes advantage of data input patterns during the LDPC decoding process, along with an architecture and final post-layout implementation to increase energy efficiency by minimizing unnecessary bit toggling in Split-Row Threshold while maximizing BER performance. We first overview power reduction methods and present some preliminary investigation for the proposed algorithm.

5.1 Power Reduction Methods

5.1.1 Early Termination

For a basic message-passing algorithm, simulation can be used to determine a predefined set of iterations for a range of expected SNRs. However, a more efficient method is to determine if the decoder has converged to a valid code word by estimating the variable node outputs to a binary code word ($\hat{V}$), and checking if all parity check constraints, $H \cdot \hat{V}^T = 0$, are satisfied at the end of each iteration. This is also referred to as *syndrome checking*. Once convergence has been verified, the decoder can terminate early. Several methods are proposed to efficiently implement this *early termination* [18, 67, 66].

LDPC codes, especially high rate codes, converge early at high SNR [77]. Because the majority of frames require only a few decoding iterations to converge, by detecting early decoder convergence, throughput and energy can potentially improve significantly

while maintaining the same error performance. When satisfied, *early termination* occurs and the message is considered error-free since parity among the rows of H has been met. Energy is then reduced because less iterations are needed to decode a block (i.e., lower average iteration).

5.1.2 Voltage Scaling

In order to save power and energy the most effective technique is voltage scaling. In general static voltage scaling is employed and set by the application requirement. Results show that efficiency is improved by 3 to 4× [55]. So far, *dynamic voltage scaling* (DVS) has been proposed at the decoder level [6, 75, 74], where the majority of the decoder is in a single adaptive voltage domain.

For the Split-Row Threshold 10GBASE-T compliant decoder, even with the worst case number of iterations required to decode one block of bits, the minimum voltage is 0.7 V in 65 nm CMOS [55]. For most cases, near-threshold operation is not advisable in nanometer technologies due to increased susceptibility to variations and soft errors [20], and so any further energy savings using voltage scaling will reduce the functional integrity of the decoder's circuits. Another difficulty is that most DC-DC converters take microseconds to settle to a different voltage (20 mV per μsec [75]), which is unacceptable for $>$ 1 Gbps communication standards (e.g. a 10GBASE-T decoder will process over 3000 blocks per μsec). An alternative is to use voltage dithering to create virtual voltages from quantized voltage levels [30], but settling time and effectiveness of such policies are highly dependent on the control algorithm, application, and environment [72].

5.1.3 Switching Activity Reduction

Because decoders exhibit a lot of switching activity due to their largely computational nature, we can decrease power by lowering the effective capacitance, C_{eff}. For full parallel architectures, this was done through the Split-Row implementations which reduced overall hardware complexity, and thus eliminated interconnect repeaters and wire capacitance. For partial parallel architectures, wordwidth reduction using non-uniform quantization has been used to reduce the amount of information needed in check node processing

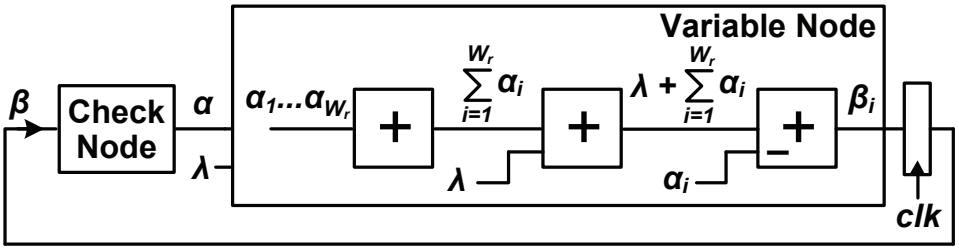


Figure 5.1: Single cycle message passing datapath with variable node processor ($W_c = 6$) in partial detail.

and memory storage requirements (thus also reducing the SRAM capacitance as well) [58]. However, conversion steps are needed to do variable node computation in the original wider wordwidth.

Implementing non-uniform quantization to full parallel architectures may result in more cost than benefits. Conversion steps across all communication links adds hardware between every check and variable node. Since memory is not a large part of such architectures, this method does not save on memory area. In contrast, communication links make up the majority of the area, and routing reduction methods such as bit-serial [18], where messages are transmitted serially or Split-Row techniques result in further reduction in routing congestion than non-uniform quantization methods. Moreover, applying both Split-Row algorithm and non-uniform quantization techniques is likely to worsen BER performance. As an alternative, rather than statically fixing the wordwidth at run time we will introduce an adaptive wordwidth datapath power reduction algorithm to reduce switching activity for a full parallel decoder.

5.2 Adaptive wordwidth Decoder Algorithm

The datapath wordwidth of the decoder directly determines the required memory capacity, routing complexity, decoder area, and datapath critical path delays. Moreover, it effects the amount of switching activity on wires and logic gates, and thus affecting the power dissipation.

A simplified block diagram of a single cycle datapath is shown in Fig. 5.1. With

the Split-Row Threshold architecture, the check node processor logic generally has lower message activity than that of the variable node processor due to its reduced hardware. The figure shows some of the variable node details such as the summations and a final subtraction. For the (6,32) (2048,1723) 10GBASE-T code, variable node processors take seven inputs: six inputs from the messages passed by the check node processors as well as the original received data from the channel (λ). The number of inputs results in a large datapath due to the wordwidth growth in additions and the fact that there are 2048 variable node processors.

Therefore, our proposed algorithm adapts the wordwith datapath of variable node processing which has a larger effect in message activity reduction. The algorithm switches between two modes: a *Low Power Mode* and *Normal Mode*. In *Normal Mode* a full 6 bit computation is done while *Low Power Mode* performs a reduced wordwidth computation in the variable node processing.

5.2.1 Preliminary Investigations

Let the variable node messages β_1, β_2, ..., β_{Wr} be the inputs to a check node C_i. Since variable node massages are initialized with channel information (assuming α messages in Eq. 2.3 are initially zero), for BPSK modulation and an AWGN channel their distribution at the first iteration is:

$$P_V(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \left(e^{\frac{-(x-1)^2}{2\sigma^2}} + e^{\frac{-(x+1)^2}{2\sigma^2}} \right) \tag{5.1}$$

where σ^2 is the variance of the channel noise.

For iterations > 1, the variable node messages in MinSum Normalized are approximated with Gaussian distributions [10]. Similarly, in Split-Row Threshold, variable node messages can be fitted with the sum of two Gaussian distributions, and a very good agreement (R-square = 0.99) was achieved for the fit. Therefore, the β distribution at iteration l can be described as:

$$P_V(x_l) = \frac{1}{\sqrt{2\pi\sigma_l^2}} \left(e^{\frac{-(x_l-\mu_l)^2}{2\sigma_l^2}} + e^{\frac{-(x_l+\mu_l)^2}{2\sigma_l^2}} \right) \tag{5.2}$$

For this distribution, the probability that a variable node message β has a magnitude less than a given value D is:

$$P_{|\beta|<D} = \frac{1}{\sigma_l\sqrt{2\pi}} \int_{-D}^{+D} e^{\frac{-(x-\mu_l)^2}{2\sigma_l^2}} dx \tag{5.3}$$

Thus assuming β_1, β_2, ..., β_{Wr} are i.i.d., the probability that at least one input of the check node C_i has a magnitude less than D is:

$$P(\exists\ \beta_i \in \{\beta_1, \beta_2, ..., \beta_{Wr}\} \mid |\beta_i| < D) = \left(1 - P_{|\beta|<D}\right)^{W_r} \tag{5.4}$$

In MinSum Normalized and Split-Row Threshold, for each check node if there exists one input, β, whose magnitude is less than D, then applying Eqs. 2.6 and 4.4, the other $W_r - 1$ outputs of the check node (α messages) have absolute values less than $D \times Sfactor$ after being normalized with $Sfactor$. Thus if the probability from Eq. 5.4 is high for a particular D, we should expect a large concentration of $\alpha \in [-D \times Sfactor, +D \times Sfactor]$, which we call *Threshold Region*. This is implied in Fig. 5.2, which plots the distribution of α for different iterations of MinSum Normalized at SNR = 4.4 dB, with $D = S_{MS} = 0.5$. In this SNR, the probability from Eq. 5.4 is 90%–80% for iteration 1 through 3 for MinSum Normalized, and as it is shown in the figure, the majority of α values at those iterations are within the region $\equiv [-0.25, +0.25]$,

The probability from Eq. 5.4 in Split-Row Threshold is plotted in Fig. 5.3 for SNR = 3.4 dB through 4.4 dB and iterations 1,2 and 3, where $D = T = 0.25$. As shown in the plot, the probability value of $P(\exists\ \beta_i \in \{\beta_1, \beta_2, ..., \beta_{Wr}\}$ is 90%–75% for SNR ranges between 3.4–4.1 dB. For SNR > 4.2 dB the probability is 75%–60% for iteration 3.

As expected, Table 5.1 shows the percentage of $\alpha \in [-Sfactor \times T, +Sfactor \times T]$ in Split-Row Threshold decoding for a large number of decoding iteration at SNR = 3.6 and 4.4 dB. The table also shows that for SNR = 3.6 dB and through iteration 7, 93% down to 80% of all α values are in the *Threshold Region*. For a high SNR value of 4.4 dB and through iteration 3, 90% down to 66% of α values are in the region. Most blocks converge beyond five iterations at SNR $\geq$ 4.4 dB.

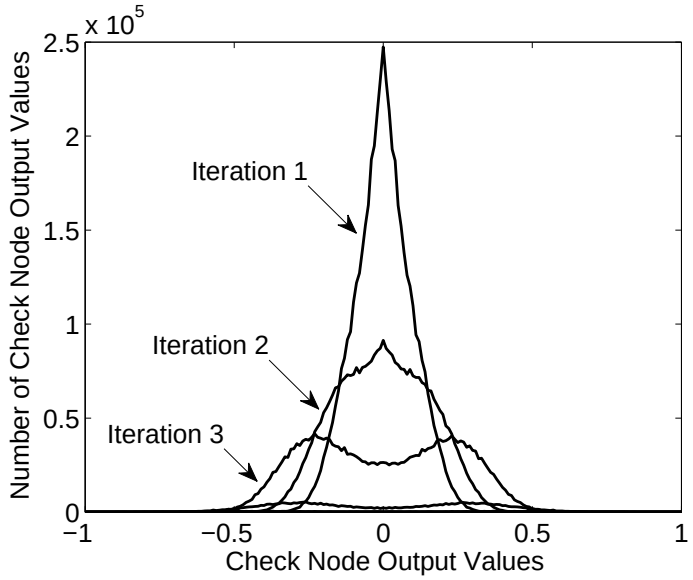


Figure 5.2: An overlay of check node output (α) distributions using MinSum Normalized over many iterations, at $SNR = 4.4$ dB, where $Sfactor = 0.5$.

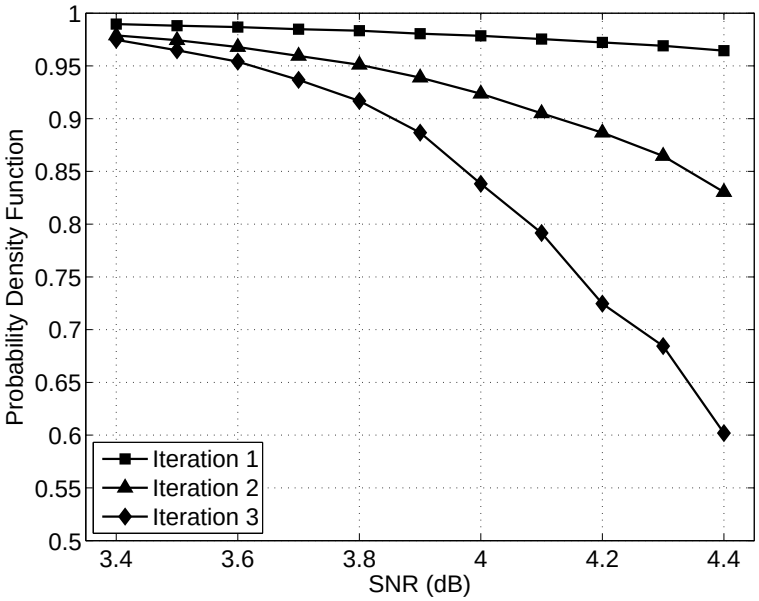
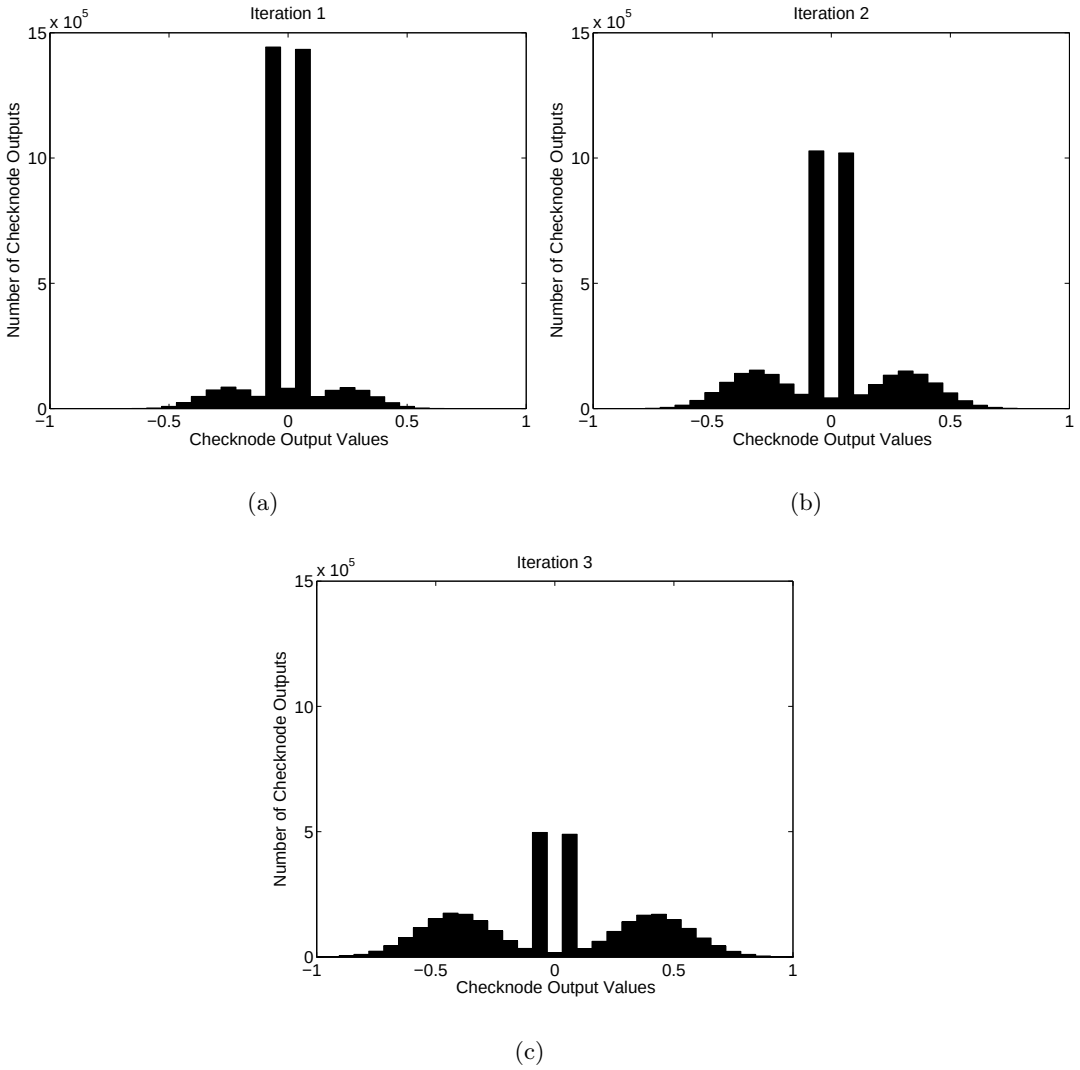


Figure 5.3: The pdf of variable node outputs to be less than a predefined threshold ($D = T = 0.25$), for a large range of SNR values at iterations 1 through 3. Data are obtained for (6,32) (2048,1723) 10GBASE-T code using Split-Row Threshold decoding for 1000 blocks, and applying Eq. 5.4.



(a) (b)

(c)

Figure 5.4: Check node output (α) distribution in the first three iteration of Split-Row Threshold decoder for (2048,1723) LDPC code at SNR = 4.4 dB, where $T = S = 0.25$.

For Split-Row Threshold, if there exists an input in a partition whose absolute value is smaller than T, then the $Threshod\_en$ signal is asserted high and is globally sent to other partitions. Therefore, the check nodes in other partitions set their minimum (Min_{Spi} from Eq. 4.4) to T, if their local minimum was larger than T. Due to this key characteristic and applying Eq. 4.4, check node messages (α) are largely concentrated at $\pm T \times Sfactor$. This is shown in Fig. 5.4 which plots the α distributions in Split-Row Threshold at SNR = 4.4 dB for the first three iterations.

At low iteration counts and low SNR values, since most α messages lie within the *Threshold Region* the inputs to the variable node processors can be represented by less bits. This implies that variable node additions can be done in smaller wordwidths, allowing us to adaptively change the wordwidth of the variable node processor depending on SNR and iteration count in order to reduce the final energy per bit without losing significant error correction performance.

5.2.2 Power Reduction Algorithm

Given that variable node input wordwidths can be reduced without losing significant information at low SNR values and also at low iteration counts in high SNRs, we propose a *Low Power Mode* operation for the decoder which significantly reduces the switching activity of the variable node processors in the following: After check node processing, we chop or saturate α such that it is within the *Threshold Region* for when the current iteration count, $Iteration$, is less than a set *Low Power Mode* iteration max count

Iteration	$SNR = 3.6\ dB$	$SNR = 4.4\ dB$
1	93%	90%
2	90%	78%
3	87%	66%
4	86%	61%
5	85%	55%
6	83%	-
7	80%	-

Table 5.1: The percentage that $|\alpha| \leq (Sfactor \times T)$ condition is met in 1000 sets of input data for two SNR values: 3.6 dB and 4.4 dB. For SNR=4.4 dB, most block converge at iterations > 5

($Low\_Power\_Iteration$). Three methods are explored which have different BER performance, convergence behavior and hardware complexity. All three methods try to remap all α into the *Threshold Region*. In Method 1, we saturate α values outside the *Threshold Region* into $[-T \times Sfactor, +T \times Sfactor]$. In Method 2, we set all α magnitudes to $T \times Sfactor$, because the majority of them are concentrated at $T \times Sfactor$ value. In Method 3, we only keep the minimum number of LSB bits that can represent the values within the *Threshold Region* (in other words, the α MSBs are chopped). These methods are described in Algorithm 2.

A qualitative perspective shows that Method 1 has the best error performance since it preserves any α already within the *Threshold Region* and also maps α values regularly. Method 2 offers a simple hardware solution at the cost of losing some information for $\alpha \in (-T \times Sfactor, +T \times Sfactor)$, but it has a high reduction in bit toggling (to be explained in Sec. 5.3). For Method 3, its benefit comes from the compromise between the hardware cost of Method 1 and a better error correction performance than Method 2 (even though the α values are irregularly mapped).

By reducing the information range of α into the *Threshold Region*, the required datapath wordwidth is reduced and thus variable node computation can be done with less switching activity in *Low Power Mode*. The challenges come from implementing a low overhead flexible datapath as well as deciding when to switch out of *Low Power Mode* such that the final convergence does not take much more iterations than running completely in *Normal Mode*. Algorithm 3 describes the complete *Split-Row Threshold Low Power* decoding process.

For our 10GBASE-T decoder implementation, the decoding message wordwidth is chosen to be 6 bits in *Normal Mode*. During *Low Power Mode*, for Methods 1 and 3, the 6-bit input additions in variable node are reduced into 3-bit input additions, while in Method 2, it is reduced to 1-bit input additions (see Sec. 5.3). In order to simplify hardware and further reduce the toggling, the variable node final subtractions (see Eq. 2.3) can be bypassed during *Low Power Mode* without causing a significant distortion of β messages. This is

Algorithm 2 Split-Row Low Power Threshold Algorithm — Check Node Processing

for $Spk = 1, 2, \ldots, Spn$ **do**

for $i = 0, 1, \ldots, M-1$ **do**

for all $j' \in V_{Spk}(i) \backslash j$ **do**

if $Lowpower\_flag = 0$ **then**

$$Min_{Spk} = \begin{cases} \begin{cases} Min1_i\ , & \textbf{if } j \neq \text{argmin}(Min1_i) \\ Min2_i\ , & \textbf{if } j = \text{argmin}(Min1_i) \end{cases}; & \textbf{if} \quad Min1_i < T \quad \textbf{and} \quad Min2_i < T & (5.5a) \\ \begin{cases} Min1_i\ , & \textbf{if } j \neq \text{argmin}(Min1_i) \\ T\ , & \textbf{if } j = \text{argmin}(Min1_i) \end{cases}; & \textbf{if} \quad Min1_i < T \quad \textbf{and} \quad Min2_i > T \quad \textbf{and} \quad Threshold\_en = 1 & (5.5b) \\ T\ ; & & \textbf{if} \quad Min1_i > T \quad \textbf{and} \quad Min2_i > T \quad \textbf{and} \quad Threshold\_en = 1 & (5.5c) \\ \begin{cases} Min1_i\ , & \textbf{if } j \neq \text{argmin}(Min1_i) \\ Min2_i\ , & \textbf{if } j = \text{argmin}(Min1_i) \end{cases}; & \textbf{if} \quad Min1_i > T \quad \textbf{and} \quad Min2_i > T \quad \textbf{and} \quad Threshold\_en = 0 & (5.5d) \end{cases}$$

$$\alpha_{ij:Spk} = Sfactor \times \prod_{j'} \text{sign}(\beta_{ij'}) \times Min_{Spk} \qquad (4.4a)$$

else

$$Min'_{Spk} = \begin{cases} \text{Method 1:} \quad \begin{cases} \begin{cases} Min1_i\ , & \textbf{if } j \neq \text{argmin}(Min1_i) \\ Min2_i\ , & \textbf{if } j = \text{argmin}(Min1_i) \end{cases}; & \textbf{if } Min1_i < T \quad \textbf{and } Min2_i < T \\ T\ ; & & \textbf{if } Min1_i > T \quad \textbf{and } Min2_i > T \end{cases} & (5.6a) \\ \text{Method 2:} \quad T & & (5.6b) \\ \text{Method 3:} \quad \textbf{do} \quad \text{Eqs. 5.5a, b, c, or d} \quad \textbf{then} \quad \left(Min_{Spk} \mod T\right) & & (5.6c) \end{cases}$$

$$\alpha_{ij:Spk} = Sfactor \times \prod_{j'} \text{sign}(\beta_{ij'}) \times Min'_{Spk} \qquad (4.4b)$$

end if

end for

end for

end for

Algorithm 3 Split-Row Low Power Threshold Algorithm

Require: λ, i.e. channel information

$Iteration = 1$

while $H \cdot \hat{V}^T \neq 0$ **do**

$Lowpower\_flag = (Iteration \leq Low\_Power\_Iteration)$

for $j = 0, 1, \ldots, N-1$ **do**

if Lowpower_flag $= 0$ **then**

$i' \in C(j) \backslash i$

else

$i' \in C(j)$

end if

for all i' **do**

$$\beta_{ij} = \lambda_j + \sum_{i'} \alpha_{i'j} \qquad (2.3)$$

end for

end for

do Algorithm 2

$Iteration = Iteration + 1$

end while

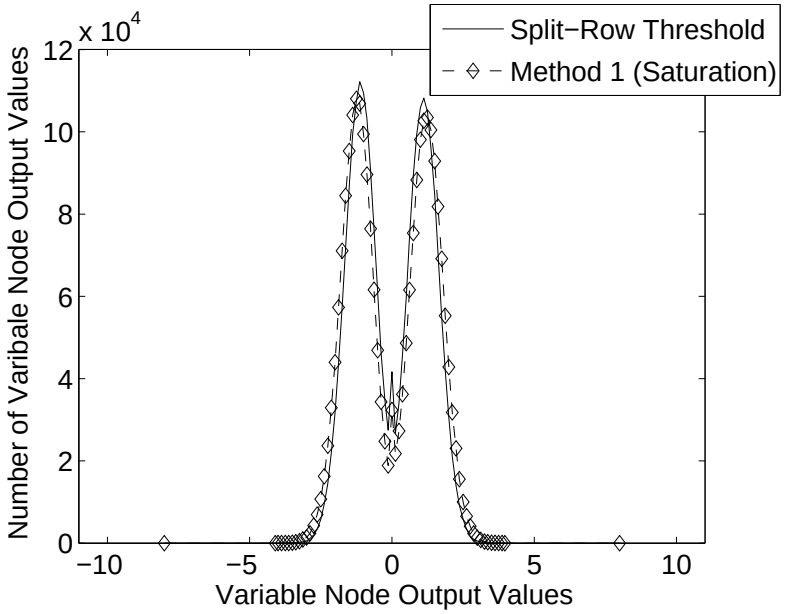


Figure 5.5: Variable node output (β) distributions for Split-Row Threshold and Method 1 at iteration 4 with $SNR = 4.2$ dB.

.

shown in Fig. 5.5 which compares the β distributions for 10GBASE-T code using Split-Row Threshold and modified version with *Low Power Mode* using Method 1 at iteration 4 at SNR = 3.8 dB. As shown in the figure, the distributions are closely matched.

Figure 5.6 illustrates the BER performance of the 2048-bit 10GBASE-T code using Split-Row Threshold for only *Normal Mode* operation ($Low\_Power\_Iteration = 0$), and adaptive low power operation using Method 1, 2, and 3 when $Low\_Power\_Iteration$ varies between 3 to 6. The figure also shows that Method 1, 2, and 3 have nearly the same bit error performance. They also perform very closely to Split-Row Threshold in only *Normal Mode*, with a 0.06–0.1 dB decrease at $BER = 10^{-7}$ when $Low\_Power\_Iteration = 3$. With $Low\_Power\_Iteration = 6$, this SNR gap increases to 0.15–0.2 dB.

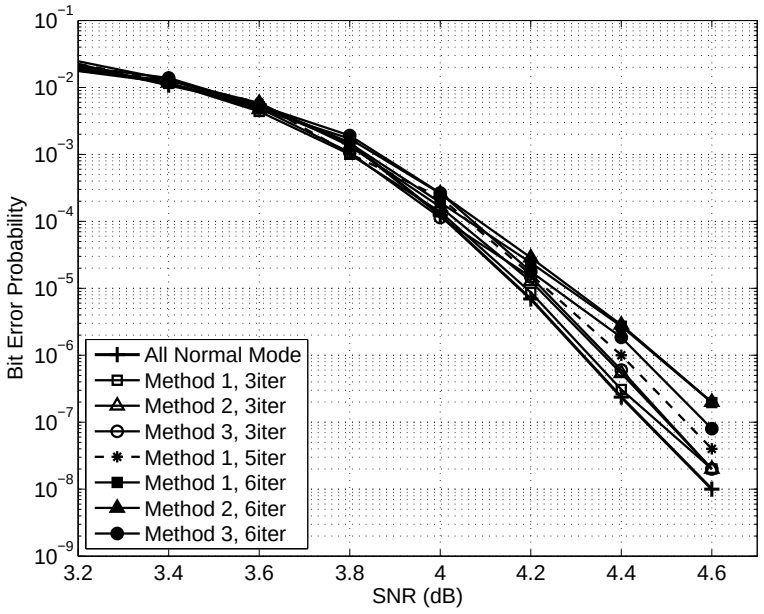


Figure 5.6: Bit error performance of the 2048 bit 10GBASE-T code using Split-Row Threshold (only *Normal Mode*, i.e. $Low\_Power\_Iteration = 0$), and Split-Row Low Power Threshold with Method 1, 2, and 3 when $Low\_Power\_Iteration$ varies from 3 to 6.

5.3 Architecture Design

The single pipeline block diagram for the proposed full parallel Split-Row Threshold decoder for 10GBASE-T code with 16 partitions is shown in Fig. 5.7. In each partition, there are 384 check processors (each takes 2, i.e. $W_r/N = 32/16$, β inputs), and only 128 ($N/Spn = 2048/16$) variable processors. The $Sign$ and $Threshold\_en$ passing signals are the only wires passing (serially) between the partitions which are generated in the check node processors in parallel. The $Lowpower\_flag$ global signal is sent to every block and sets the operation mode to either *Normal Mode* or *Low Power Mode* (see Algorithm 3).

5.3.1 Check Node Processor

The check node processor implementation is shown in Fig. 5.8 and consists of two parts:

1. Split-Row Threshold Implementation

2. *Low Power Mode* Implementation

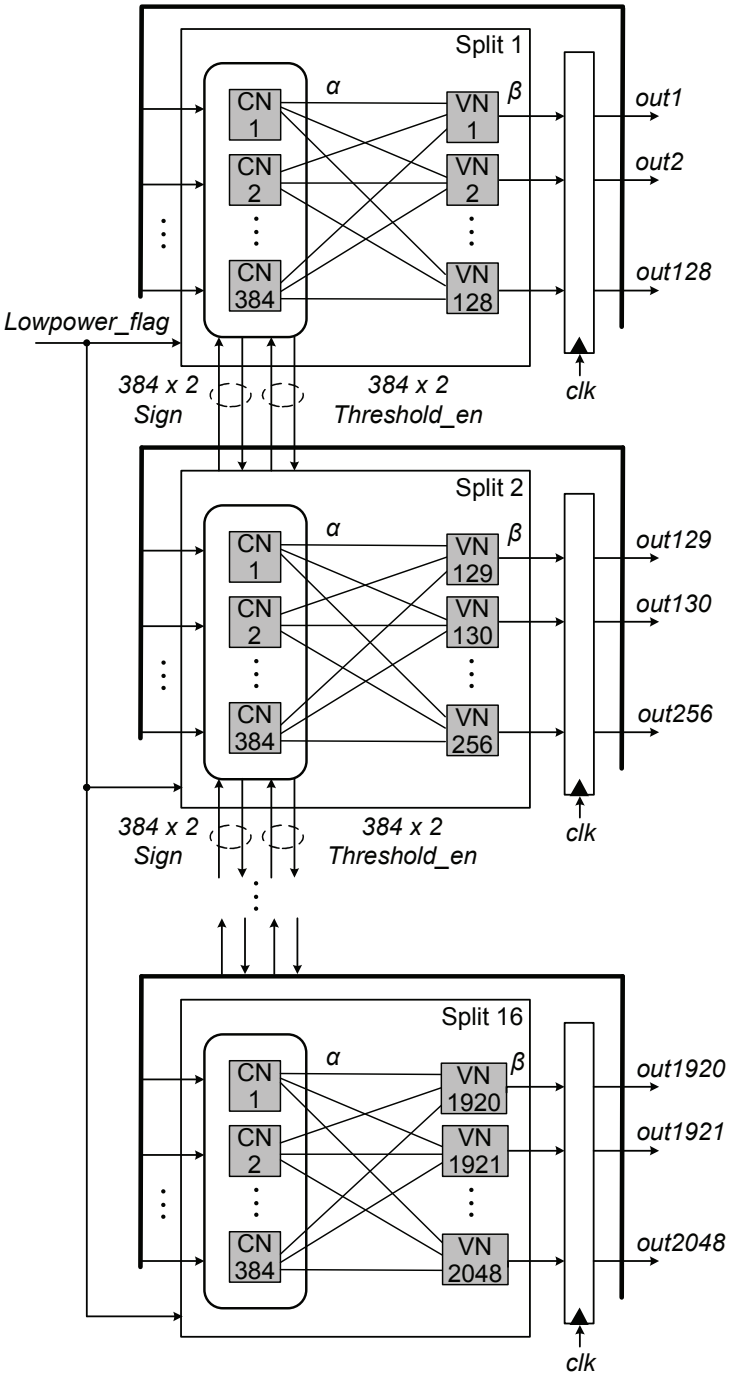


Figure 5.7: Architecture diagram of the full parallel Split-Row Low Power Threshold 10GBASE-T decoder.

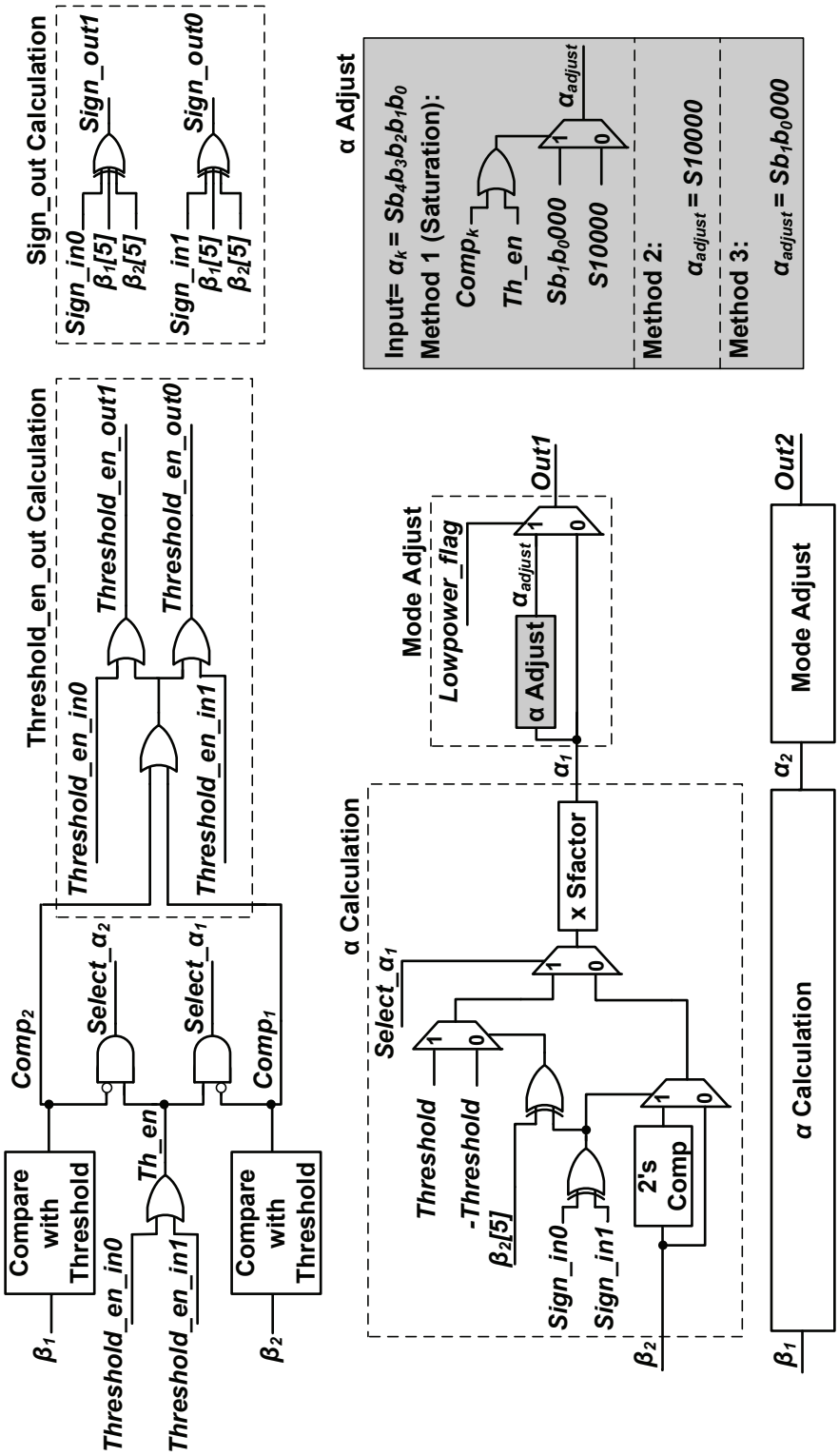


Figure 5.8: Check node processor design for Split-Row Low Power Threshold decoder. In α *Adjust* block (shaded box), α_k ($k = 1$ or 2) is shown as a 6 bit binary $Sb_4b_3b_2b_1b_0$ and α_{adjust} is computed according to low power Methods 1, 2, and 3.

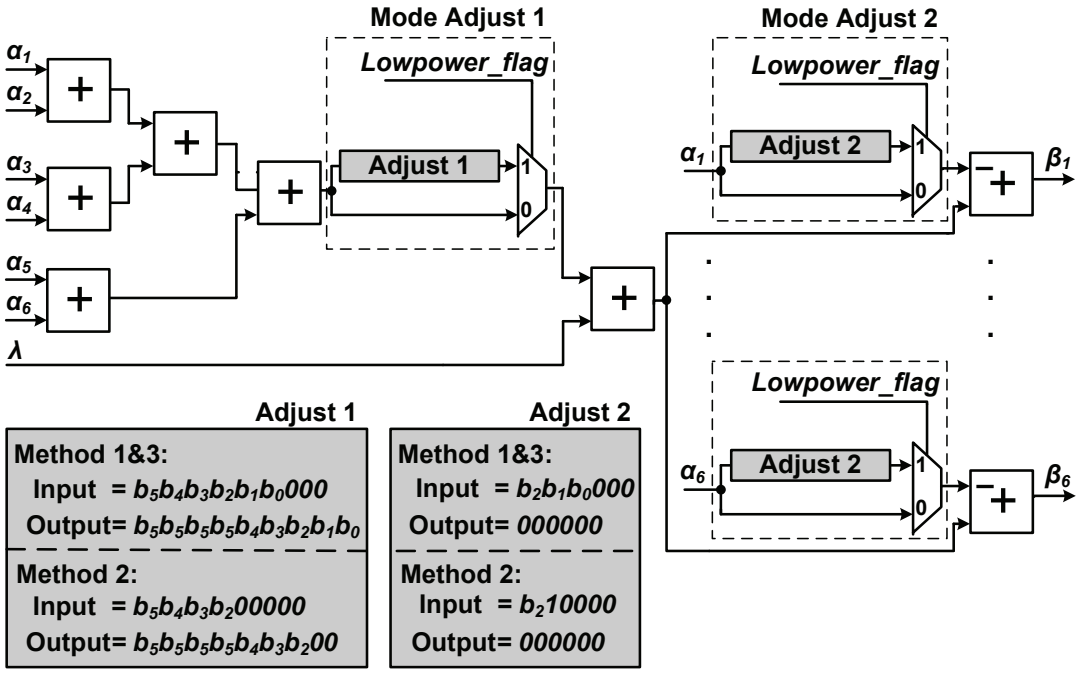


Figure 5.9: Variable node processor design for Split-Row Low Power Threshold decoder.

These are described in the next two minor sections.

Optimized Split-Row Threshold

The first step includes three nearly-parallel building blocks: *Threshold_en_out Calculation*, *Sign_out Calculation* and α *Calculation*. In each partition, the β inputs are compared with the threshold, T. If a β is smaller than T the $Comp$ signal is asserted high. Then in the $Threshold\_en\_out$ *Calculation* block, $Comp$ signals along with $Threshold\_en\_in$ from one neighboring partition generate the $Threshold\_en\_out$ signal which goes to the other neighboring partition. Meanwhile in the *Sign_out Calculation* block, the sign bits of two β inputs in each partition are XORed with $Sign\_in$ bit from one neighboring partition to generate the $Sign\_out$ signal which is sent to the other neighboring partition.

For a two-input check node processor, the magnitude computation of the Split-Row Threshold algorithm in the α *Calculation* block is optimized as follows. Instead of finding $Min1$ and $Min2$ of the inputs first and then comparing them with T, the inputs

are directly compared with T and the generated α output is based on the other β input or T using $Select\_\alpha$ signals (the minimum of two numbers over a subset, excluding one of them is the other one). The final sign bit of each output is generated by XORing the $Sign\_in$ bits from two near neighboring partitions and the sign bit of the other β input. This logic simplification results in a 17% reduction of gate area compared to a previous implementation [55]. The next stage is $Sfactor$ multiplication. As mentioned before, $Sfactor$ is chosen to be 0.25 which can be implemented with a simple shift.

Low Power Mode Implementation

This step (shown as the *Mode Adjust* block in Fig. 5.8 includes a multiplexer which selects the appropriate message (α or α_{adjust}) based on the status of the $Lowpower\_flag$ global signal. For the three different low power method implementations, which are shown in the shaded box on the right, several logic optimizations are made to minimize the hardware. In order to shutoff the toggling of unused bits in *Low Power Mode*, they are kept zero (their initial value). For our 6 bit wordwidth implementation the *Threshold Region* ($[-0.0625, +0.0625]$, for $T = Sfactor = 0.25$) can be implemented with three bits. Therefore, in $Sb_4b_3b_2b_1b_0$ format (S is the sign bit) [1], $b_4b_3b_2$ are zero in *Low Power Mode*. However, to eliminate the extra logic to perform the sign extensions in the variable node processor, where the 3 bit additions in *Low Power Mode* are handled by 6 bit additions, the three zero MSB bits are swapped with LSB bits (i.e. α_{adjust} becomes Sb_1b_0000 instead of $S000b_1b_0$. This results in approximately 5% gate count reduction per variable node processor.

In Method 1 (α saturation to $[-0.625, +0.625]$), α is adjusted based on Eq. 5.6a. This can be easily implemented using the $Comp$ and $Threshold\_en$ signals, which determine whether $|\alpha| > T \times Sfactor = 0.625$. (Note that the saturated values of +0.0625 is *000010* and -0.0625 is *111110* in 2's complement binary format.) If α is outside *Threshold Region*, α_{adjust} (after bit swapping) always becomes *S10000*. This is one of the key advantages of choosing $T = Sfactor = 0.25$ in our implementation. If α is inside *Threshold Region*, α_{adjust} becomes Sb_1b_0000. Overall, α bit toggling is reduced to at most three bits.

[1] 1.5 fixed-point format.

	Check Processor		Variable Processor		
Design -	Mode Adjust	Synth. Area (μm^2)	Mode Adjust 1	Mode Adjust 2	Synth. Area (μm^2)
Original	–	3644	–	–	1200
Method 1	8 MUX + 6 AND + 4 OR	4193	8 MUX	18 AND	1270
Method 2	8 AND + 2 OR	3835	5 MUX + 2 AND	12 AND	1258
Method 3	4 MUX + 6 AND	4068	8 MUX	18 AND	1270

Table 5.2: Comparison of hardware increase in check processor and variable processor with synthesis area for the three low power "Methods". (For Original none of these methods are applied.)

In Method 2, which implements Eq. 5.6b, all α outputs are set to $\pm T \times Sfactor = \pm 0.0625$. Therefore, α_{adjust} always becomes S10000, regardless of its input magnitude. Thus in addition to reducing the gate count in Method 1, Method 2 reduces the α bit toggling to only one bit.

In Method 3, which implements Eq. 5.6c, only the first two LSB bits are kept along with the sign bit. The two LSB bits are then shifted to the b_4b_3 positions and α_{adjust} becomes Sb_1b_0000, and bit toggling is reduced to three bits.

5.3.2 Variable Node Processor

The block diagram of variable node processor is shown in Fig. 5.9, which implements Eq. 2.3 in Algorithm 3. The key benefit of *Low Power Mode* operation is in the variable node processor, where all addition datapath wordwidths are reduced by 3 to 5 bits (depending on the "Method" of implementation), which results in 35%–47% reduction in switching activity for the majority of the variable node processor. This wordwidth reduction is applied to all 2048 variable processors in the decoder.

Two adjustments (conversion steps) are performed to make the variable node processor operate correctly in both *Normal Mode* and *Low Power Mode*. *Mode Adjust 1* is made before adding the sum of six α values to the channel information, λ, which shifts the addition result bits back to their original LSB positions. (Recall that α bits were shifted 3 positions to the left at the end of check node processing—applies to all low power "Methods". *Mode Adjust 2* is made in the subtraction stage, where α bits are kept zero (their

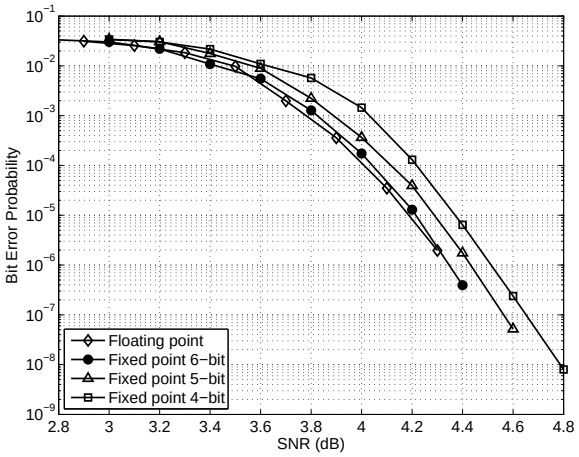


Figure 5.10: Bit error performance of (6,32) (2048,1723) 10GBASE-T LDPC code using Split-Row Threshold decoding in floating point and fixed point with different wordwidth quantizations.

initial value) in order to bypass the subtraction in *Low Power Mode*.

5.4 Design of CMOS Decoders

To further investigate the impact of the proposed decoder on the hardware, we have implemented three full parallel decoders using Method 1, 2, 3 for the (6,32) (2048,1723) 10GBASE-T LDPC code in 65 nm, 7-metal layer CMOS.

5.4.1 Design Steps

In order to design the proposed decoder using Split-Row Threshold with an adaptive wordwidth, these key steps are required:

1. Choosing the number of partitioning (*Spn*), Threshold (T), and *Sfactor* values: It is shown that the routing congestion, circuit delay, area and power dissipation reduces as the number of partitions increases with a modest error performance loss [55]. The Threshold (T) and *Sfactor* which directly effect the error performance are found through empirical simulations. For the 10GBASE-T decoder design, *Spn* is set to

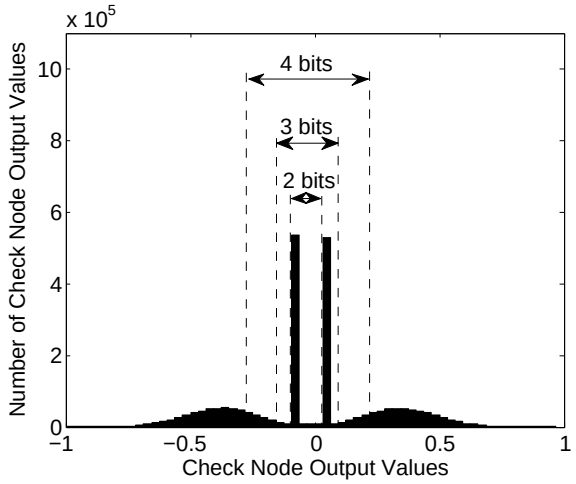


Figure 5.11: Check node output (α) distribution using Split-Row Threshold decoder for (2048,1723) LDPC code, which are binned into discrete values set by a 6-bit (1.5 format) quantization. The 3-bit subset can cover all values within the *Threshold Region*. Data are for SNR = 4.4 dB and $iteration = 3$, where $T = Sfactor = 0.25$.

16 and the closest fixed-point values for T and *Sfactor* which attain a near optimum floating-point performance are both 0.25.

2. Number of supported wordwidths: As discussed in Section 5.2, when using Split-Row Threshold, check node messages (α) are largely concentrated at $\pm T \times Sfactor$ at low iteration counts and low SNR values, (e.g. more than 80% for 10GBASE-T). Therefore, it naturally makes sense to define two regions, where one region represents α values in $\pm T \times Sfactor$ which we call *Threshold Region* or *Low Power Mode* region and the other which represents the majority of α values and we call *Low Power Mode*. As long as there is no significant region in the distribution of α values, increasing the number of regions (more wordwidth representation selection) is not efficient due to the large hardware overhead and error performance loss of introducing another mode into all check and variable node processors. For example, if we want to add one more region it requires an additional global signal to choose between regions. It also adds additional comparators to select the region (mode) that alpha can fit in and requires us to increase the size of the muxes to choose between the outputs.

3. *Normal Mode* wordwidth selection: This is the major datapath width of the decoder

and is chosen to optimize the error performance with minimum hardware. Figure 5.10 shows the bit error performance of (2048,1723) 10GBASE-T LDPC code using Split-Row Threshold for floating point and fixed point with 6-bit, 5-bit and 4-bit quantization. As shown in the plot, the minimum wordwidth which attains the optimal bit error performance is 6 bit, therefore $k = 6$ for our implementation.

4. *Low Power Mode* wordwidth selection: This is the subset of *Normal Mode* wordwidth where the Threshold Region ($\pm T \times Sfactor$) values can be represented. For the 10GBASE-T code, the *Threshold Region* is within $\pm T \times Sfactor = \pm 0.0625$. Therefore, its values in 6-bit (1.5 format) quantization are: -0.0625, -0.03125, 0, $+0.03125$ and $+0.0625$. These values can be represented with a 3-bit subset. Figure 5.11 shows the check node output (α) distribution using Split-Row Threshold decoder for (2048,1723) LDPC code which are binned into discrete values set by 6-bit (1.5 format) quantization. The 3-bit subset can cover all values within the Threshold Region. Representation with less bits, such as a 2-bit subset that is shown in the figure, will miss some values of the *Threshold Region*. Also there is no benefit if we use a 4-bit subset because the additional values represented by the 4-bit subset are not within the *Threshold Region*. Therefore, $d = 3$ for our implementation.

5.4.2 Synthesis Results

The amount of hardware overhead to implement these three low power "Methods" is shown in Table 5.2. Among them, Method 2 has the least hardware increase, which is a 5% increase in check node processor and variable node processor area compared to Split-Row Threshold (which has none of the methods applied). Method 1 has the largest hardware overhead due to the added Muxes and gates for saturation implementation with a 15% increase in check node processor area and a 6% increase in variable node processor area compared to the original design.

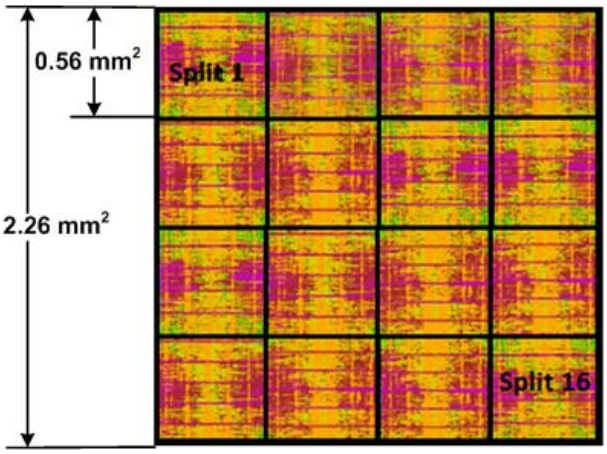


Figure 5.12: Post layout view of the proposed low power decoder with Method 2.

5.4.3 Back-end Implementations

Method 1, 2 and 3 decoders are implemented using STMicroelectronics LP 65 nm CMOS technology with a nominal supply voltage of 1.2 V (max. at 1.3 V). We use a standard-cell RTL to GDSII flow using synthesis and automatic place and route to implement all decoders. The decoders were developed using Verilog to describe the architecture and hardware, synthesized with Synopsys Design Compiler, and placed and routed using Cadence SOC Encounter. Each block is independently implemented and connected to the neighboring blocks with $Sign$ and $Threshold\_en$ wires.

To generate reliable power numbers, SoC Encounter is used to extract RC delays using the final place and route information and timing information from the standard cell libraries. The delays are exported into a "standard delay format" (SDF) file. This file is then used to annotate the post-layout Verilog gate netlist for simulation in Cadence NC-Verilog. This generates a timing-accurate "value change dump" (VCD) file that records the signal switching for each net as simulated using a testbench. The VCD file is then fed back into SoC Encounter to compute a simulation-based power analysis. This analysis is performed for 100 test vectors for each SNR.

The chip layout of Method 2 is shown in Fig. 5.12. A summary of the post-layout results for the low power proposed Method 1, 2, and 3 decoders, when $Low\_Power\_Iteration =$

	Normal Mode	Method 1	Method 2	Method 3
Final area utilization	95%	95%	96%	96%
Core area (mm^2)	5.27	5.27	5.10	5.20
Maximum clock frequency (MHz)	178	178	185	182
Average Power @Worst case freq (mW)	1396	1215	1172	1196
Throughput @$Imax$ (Gbps)	24.3	24.3	25.25	24.8
Energy per bit @$Imax$ (pJ/bit)	57	50	46	48

Table 5.3: Comparison of three proposed full-parallel decoders with the proposed low power Methods 1, 2, and 3 implemented in 65 *nm*, 1.3 V CMOS, for a (6,32) (2048,1723) LDPC code. Maximum number of iterations is $Imax = 15$. Power numbers are for $Low\_Power\_Iteration = 6$. Normal Mode: Method 1 with $Low\_Power\_Iteration = 0$

6, are summarized in Table 5.3. For comparison a Method 1 decoder only running in *Normal Mode* is included in the table.

5.4.4 Results and Analysis

Due to the nature of Split-Row Threshold algorithm, which significantly reduces wire interconnect complexity, all three full parallel decoders achieve a very high logic utilization, 95%–96%. In this case synthesis results have a good correlation with the layout increases. For instance, as shown in Table 5.3, the decoders in Methods 1, 2, and 3 occupy 5.10–5.27 mm^2. Method 2, which has the minimum number of added gates (see Table 5.2), has the smallest area among the three. Conversely, Method 1 has the most, and Method 2 is in between the other two. Also, results show that the critical path in general is about equal (implementations are optimized for area with circuit delay a less priority). Method 1 has a 2%–3% greater critical path delay than the other decoders due to the increased path delays through the additional MUXes and AND/OR gates.

The table also summarizes the power results for the case that decoders in three methods are kept in *Low Power Mode* for 6 iterations and *Normal Mode* for 9 iterations out of a total $Imax = 15$ iterations. Energy data are reported for 15 decoding iterations without early termination at SNR = 3.6 dB. Under these conditions, Method 2 has the smallest energy dissipation per bit, 46 pJ/bit which is 20% lower than running only with *Normal Mode*. Overall, the average power among the three methods is 1172–1215 mW,

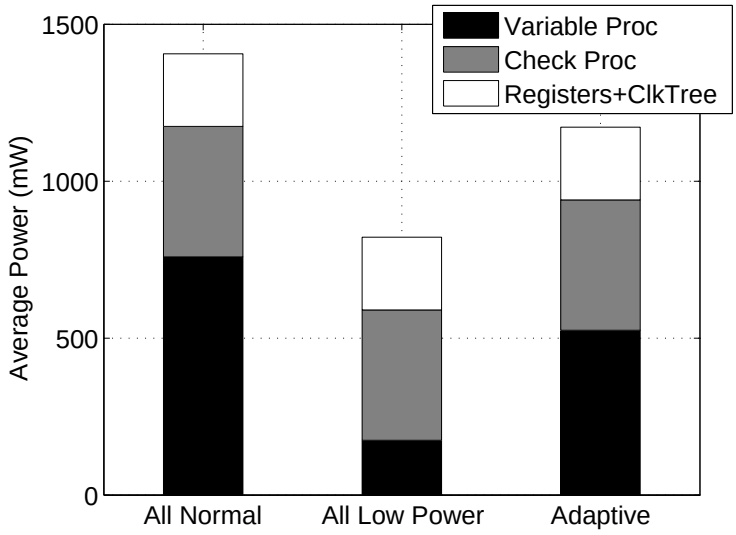


Figure 5.13: Power breakdown for Method 2: *Normal Mode* only, *Low Power Mode Mode* only, and adaptive mode (6 iterations with *Low Power Mode* and 9 iterations with *Normal Mode*).

which is 181–224 mW lower than when running on only *Normal Mode*.

Figure 5.13 shows the power breakdown for Method 2 in *Normal Mode* only, *Low Power Mode* only, and adaptive mode ($Low\_Power\_Iteration = 6$ out of 15 total iterations). Shown are the power contributions from variable node processors, check node processors, and the clk tree (including registers). By itself, *Low Power Mode* results in 41% reductions when compared to *Normal Mode* only. For an adaptive mode where $Low\_Power\_Iteration = 6$ iterations out of a total 15 iterations, this results in a net improvement of 22% in average power. Therefore, it is important to realize the tradeoff between the amount of *Low Power Mode Iterations* versus the number of convergence iterations (i.e. average iterations from early termination).

$Low\_Power\_Iteration$ effects on energy gains since the desired BER performance (depends on $Imax$ as discussed later) and the convergence behavior (early termination and average iterations) of the proposed decoders also depend on the $Low\_Power\_Iteration$. The longer *Low Power Mode* is enabled, the longer it will take to converge, and as a result the energy becomes dependent on both a tradeoff of the set $Low\_Power\_Iteration$ and the final

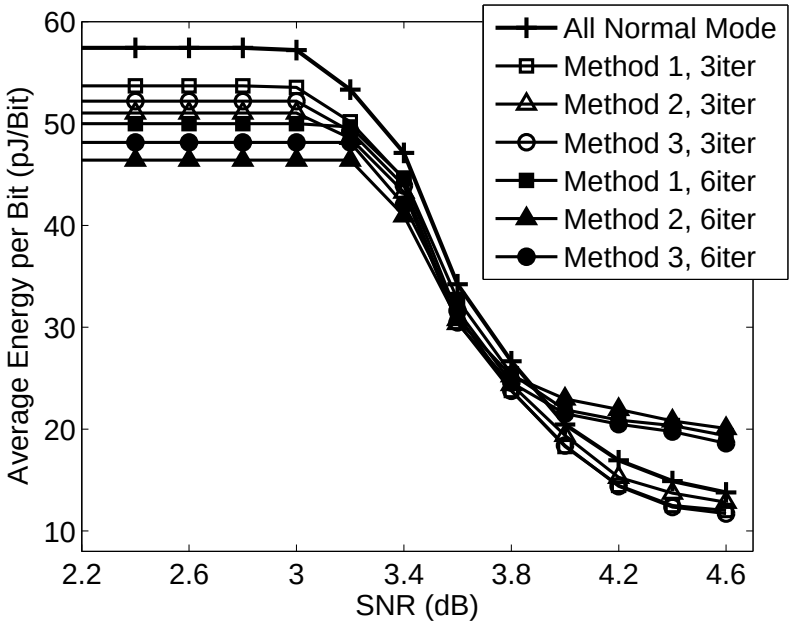


Figure 5.14: Energy per bit versus SNR for different low power decoder designs and different *Low_Power_Iteration*, compared with a design only running in *Normal Mode*.

convergence iteration count. Figure 5.14 shows the energy consumption for Methods 1, 2, and 3 when the the *Low Power Mode* is enabled for three and six iterations over a range of SNR values: 2.2–4.6 dB. Notice that for $Low\_Power\_Iteration = 6$ the energy starts to become worse for SNR $\geq$ 4.0 because of longer average convergence times (i.e. larger average iterations).

5.4.5 SNR Adaptive Design

In Split-Row Threshold, a larger maximum number of iterations, $Imax$, can improve bit error performance. This is shown by running on *Normal Mode* only while using $Imax = 25$. In this case, BER performance of the proposed decoder is only 0.2 dB away from MinSum Normalized at BER $= 10^{-9}$ (Not a significant BER improvement is observed for $Imax > 25$). Although higher maximum iteration count has almost no effect on the average iterations at high SNRs, it increases the average iterations at low SNRs [82] (more of the channel information is corrupted beyond the ability for LDPC to correct), which results in higher energy dissipation. Given the fact that running in *Low Power Mode* at low SNRs

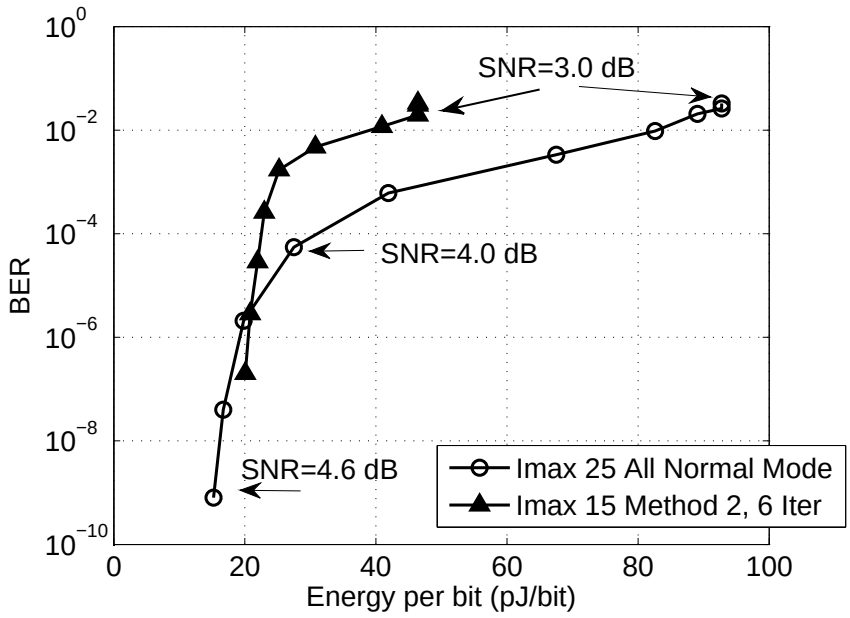


Figure 5.15: Bit error rate versus energy per bit dissipation of two decoders for different adaptive decoder settings to meet the 10GBASE-T standard throughput (dependent on the worst case $Imax$ and maximum frequency at 0.87 V).

results in larger energy savings it is more beneficial to use a larger $Low\_Power\_Iteration$ with lower $Imax$. Conversely, we can use only *Normal Mode* with a higher maximum iteration count to get the BER required at high SNR with lesser energy penalties as compared to operating the decoder with a large $Low\_Power\_Iteration$.

These scenarios are illustrated in Fig. 5.15 where the bit error performance versus energy per bit dissipation of the proposed decoder with Method 2 is shown under two conditions:

1. Adaptive mode operation with Method 2, $Low\_Power\_Iteration = 6$, and $Imax = 15$.

2. The decoder runs in only *Normal Mode*, and $Imax = 25$.

Given the worst case $Imax = 25$ and a 10GBASE-T LDPC decoder throughput of 6.4 Gbps, both designs are set to 0.87 V and compared with early termination enabled. As shown in the figure, when BER $> 10^{-4}$ (implying a low SNR) the energy dissipation of Method 2 decoder is about 20%–50% lower than that of the decoder in *Normal Mode* at the same

BER. However, when the BER $< 10^{-6}$ (SNR $>$ 4.0 dB), the decoder at *Normal Mode* attains greater than an order of magnitude improvement in BER at nearly the same energy per bit dissipation.

Therefore, using an efficient SNR detector circuit, we can switch between different modes at SNR = 4.0 dB. Similar to [75], the proposed SNR detector compares the number of unsatisfied checks with a checksum threshold at the end of the first iteration, and estimates the SNR range. For the 2048 bit 10GBASE-T code, it was found that a checksum threshold of 91 after the first iteration can estimate if the SNR is larger or smaller than 4.0 dB with a probability of being 89% true. By using this detection scheme the *Low Power Mode* iteration count and $Imax$ can be adjusted. The SNR detector circuit requires only one additional comparator in the early termination circuit.

5.4.6 Comparison with Others

The post-layout simulation results of the proposed wordwidth adaptive decoder using Method 2 are compared with recently implemented decoders [41, 73, 47, 82] for 2048 bit LDPC codes and are summarized in Table 5.4. The 10GBASE-T code is implemented in [73, 41, 82]. Results for two supply voltages are reported for a Method 2 decoder: 1.3 and 0.7 V. (Note that at 0.7 V, for $Imax = 15$, the 10GBASE-T required throughput is met.) The supply voltage can be lowered to 0.6 V based on a previously fabricated chip measurements [71]. At this voltage, the decoder throughput is 9.3 Gbps (greater than 6.4 Gbps required for 10GBASE-T) while dissipating an average power of 31 mW.

The sliced message passing (SMP) scheme in [41] is proposed for Sum Product algorithm and divides the check node processing into equal size blocks and perform the check node computation sequentially. The post-layout simulations for a 10GBASE-T partial parallel decoder are shown in the table. The multi-rate decoder in [73], supports RS-LDPC codes with different code lengths (1536–3968 bits) through the use of reconfigurable permutators. The post layout simulation results of a 10GBASE-T decoder are reported in 90 nm CMOS in the table. The partial parallel 2048-bit decoder chip is fabricated in 180 nm CMOS. The decoder which supports turbo-decoding massage passing (TDMP) algorithm

	Liu [41]	Ueng [73]	Mansour [47]	Zhang [82]		**This work**	
Technology	90 nm, 8M	90 nm	180 nm	65 nm, 7M		65 nm, 7M	
Architecture	partial parallel	partial parallel	partial parallel	partial parallel		full parallel	
Scheme	SMP	Shuffled MPD	TDMP	TPMP		Spit-Threshold	
Code Length	2048	1536-3968	2048	2048		2048	
Edges	12288	7680-23808	-	12288		12288	
Code Rate	0.84	0.79-0.93	0.5	0.84		0.84	
Bits per message	5	-	-	4		6	
Logic utilization	50%	-	50%	80%		96%	
Chip area (mm^2)	14.5	4.41	14.3	5.35		5.10	
Max. iterations ($Imax$)	16	8, 4	16	8		15	
SNR @ BER = 10^{-7} (dB)	4.35	4.2, 4.57	4.1 @ 10^{-5}	4.25		4.45	
Supply voltage (V)	-	1.0	1.8	1.2	0.7	1.3	0.7
Clock speed (MHz)	207	303	125	700	100	185	40
Maximum Latency (ns)	-	-	-	137	960	81	375
Throughput @ $Imax$ (Gbps)	5.3	4.85, 9.7	0.400	14.9*	2.1*	25.26	5.4
Throughput w/ ET. (Gbps)	-	4.85, 9.7	6.4	47.7	6.67	**85.7**	**18.5**
Throughput per Area ($Gbps/mm^2$)	0.36	11, 22	-	8.9	1.2	**16.8**	**3.6**
Power (mW)	-	855	787	2800	144	1406	89
Energy/bit w/ ET. (pJ/bit)	-	176, 88	-	58.7	21.5	**16.4**	**4.8**

Table 5.4: A comparison of the proposed adaptive decoder using the wordwidth adaptive Method 2 decoder with recently published LDPC decoder implementations. *Throughput is computed based on the maximum latency reported.

supports multiple code rates between 8/16 and 14/16. The partial parallel decoder chip [82] is fabricated in 65 nm and consists of a two-step decoder: MinSum and a post-processing scheme which lowers the error floor down to BER = 10^{-14}.

Compared to the sliced message passing decoder [41], the proposed wordwidth adaptive decoder is about 3× smaller and has 6.8× higher throughput with 0.2 dB coding gain reduction. Compared to the two-step decoder chip [82], the proposed decoder has 1.7× higher throughput and dissipates 3.57 times less energy, with the same area at a cost of 0.2 dB coding gain reduction.

5.5 Summary

As high throughput LDPC decoders are becoming more ubiquitous for upcoming communication standards, energy efficient low power decoder algorithms and architectures are a design priority. We have presented a low cost adaptive wordwidth LDPC decoder algorithm and architecture based on the input patterns during the decoding process. Depending on the SNR and decoding iteration, different low power settings were determined to find the best tradeoff between bit error performance and energy consumption. Of the three low power wordwidth adaptive methods explored one implementation had a post-layout decoder area of 5.10 mm^2, while attaining a 85.7 Gbps throughput with early termination while dissipating 16.4 pJ/bit at 1.3 V. Compared to another 10GBASE-T design with similar areas in 65 nm and operating at 0.7 V, we achieved nearly 2× improvement in throughput, and thus meeting the 6.4 Gbps required by the standard. Energy efficiency was over 3.5× better with only 0.2 dB loss in coding gain. This loss compares favorably with the non-uniform quantization bit reduction technique.

Chapter 6

Conclusion and Future Directions

6.1 Conclusion

Message-passing LDPC decoding algorithms and architectures are introduced which significantly reduce processor logical complexity and local and global interconnections. The methods are specially well suited for long-length regular block-structured codes with high check node degrees.

The proposed Split-Row and Multi-Split architectures break check node processors into multiple blocks whose internal wires are all relatively short. These blocks are interconnected by a small number of nearly zero-length "sign" bit wires. This results in denser, faster and more energy efficient circuits. Compared to MinSum Normalized and SPA decoding, the error performance loss of the methods is about 0.35–0.65 dB depending on LDPC code and the level of partitioning.

Split-Row Threshold which utilizes a "threshold enable" signal to compensate for the loss of min() inter-partition information in Split-Row provides at least 0.3 dB error performance improvement over Split-Row. The optimal threshold (T) and correction factor ($Sfactor$) values are found empirically at the point where the bit error rate is minimum for a large range of SNR values. With Split-Row Threshold higher levels of partitioning are possible with SNR loss of 0.05-0.3 dB, when compared to MinSum Normalized. Theoretical and post-layout implementation analysis show that Split-Row Threshold can reduce routing congestion by a factor of $1/\sqrt{Spn}$, where Spn is the number of partitions.

Five full-parallel LDPC decoders compatible with the 10GBASE-T standard are implemented using MinSum Normalized and Split-Row Threshold algorithms with different levels of partitioning. All decoders are built using a standard cell design flow and include all steps through the generation of GDS II layout in 65 nm CMOS. A decoder with 16-way partitioning occupies 4.84 mm^2 with a final post-layout area utilization of 97%, and operates at 195 MHz at 1.3 V with an average throughput of 92.8 Gbps with early-termination enabled. It achieves improvements in area, throughput and energy efficiency of 4.1x, 3.3x, and 4.8x respectively, compared to a MinSum Normalized implementation, with an SNR loss of 0.25 dB at BER = 10^{-7}.

For additional power saving, an adaptive wordwidth decoding algorithm is proposed which switches between a 6-bit *Normal Mode* and a reduced 3-bit *Low Power Mode*. The method reduces the unnecessary bit toggling while maximizing the bit error performance in Split-Row Threshold decoding with a modest hardware increase. The length of time that the decoder stays in a given mode is based on power and BER requirements and received SNR. Different *Low Power Mode* algorithm implementations are explored. In one implementation, signal toggling of variable node processing inputs is reduced to a single bit. Further power savings are achieved with voltage and frequency scaling. At 0.6 V, a 5.10 mm^2 low power decoder attains throughput of 9.3 Gbps (greater than 6.4 Gbps required for 10GBASE-T standard) and dissipates an average power of 31 mW.

6.2 Future Work

LDPC codes are appearing in an increasing number of applications, which have strict power and throughput constraints than the current generation and require very good error performance. On the other hand, the benefits of straightforward CMOS scaling has been slowed down as the supply voltage, capacitance and global wire delay will hardly decrease in future deep-submicron technology. Thus, it is critical to have a technique which provides regularity and reduces design dependencies on low-level optimizations in order to achieve the high throughput and high energy efficiency requirements of future applications. The Split-Row Threshold technique presents an algorithmic and architectural solution that

can be compatible with both future LDPC codes and submicron CMOS technology and can be extended for a wide range of applications. The additional requirements for some of these applications are reconfigurability for different code size/rates, support for irregularly structured codes and low error floor.

The high throughput and energy efficiency of the proposed method can be applied in the next generation of wireless communications systems. These systems usually require reconfigurability, variable high data rate and high energy efficiency. One of the key kernels of such systems is LDPC code which will be widely used in emerging wireless standards. One example is wireless high definition video transmission (WirelessHD) in the 60 GHz frequency band, which achieves a raw airlink data rate of 2.2 Gbps and decoding throughput of 1.6 Gbps [48]. Another example is the next generation of WiMAX for 4G (IEEE 802.16m) [1]. The standard requires irregular LDPC codes and must support code size/rate reconfigurability. Although reconfigurability of decoder was not addressed in this work, Split-Row Threshold architecture can simplify the switch-network interconnect overhead that is often used in the current reconfigurable LDPC decoders. This work used regular block-structured and quasi-cyclic codes which makes the number of connections per split equal. The irregular partitioning results in unequal routing congestion reduction in subblocks of Split-Row Threshold.

LDPC codes have received a lot of attention in hard disks with magnetic recording channels. Recent advances in magnetic recording are aimed at densities up to 2 Terabits per square [33]. To achieve such a high density with high system reliability, powerful coding schemes with efficient hardware implementations are required. Although there is no standard for magnetic recording hard disks, they demand high code rate, low error floor, and high decoding throughput [87, 23, 24, 35]. For example, magnetic recording Read channels use LDPC code lengths of 4608 to 36,864 bits and require a throughput beyond 5 Gbps [33, 68]. In order to make Split-Row Threshold useable in hard disks, the investigation of error floor in Split-Row Threshold is essential.

The low power proposed design can be extended to wireless medical technology in biomedical implanted devices [86] where long battery life is critical (10 μW to 10 mW [8]). Wireless medical technologies have created opportunities for new methods of preventive

care using biomedical implanted and body-worn devices. The design of the technologies that will enable these applications will demand a unique set of requirements focused on low cost, low area, ultra-low power and very reliable designs. Although biomedical devices do not require long distances (less than 10 m) and high throughput communication, they are highly constrained by limited energy resources. This is especially true for implanted devices where battery replacement may not be feasible. Using error correction improves the error performance and thus helps lower the transmit power to achieve a certain SNR. The key challenge is that both encoder and decoder circuit power dissipation must be ultra low to meet the implant transmit power requirements [29]. While an uncoded system can be used for uplink distances less than 0.5 m, a short length LDPC code using low power Split-Row Threshold can be efficient at distances of 4 m through 10 m usage.

Bibliography

[1] Amendment text proposal on rate compatible LDPC-convolutional codes. `http://www.ieee802.org/16/tgm/IEEEC802.16m-09/0339`.

[2] G.hn/G.9960. next generation standard for wired home network. http://www.itu.int/ITU-T.

[3] IEEE 802.16e. air interface for fixed and mobile broadband wireless access systems. ieee p802.16e/d12 draft, oct 2005.

[4] IEEE P802.3an, 10GBASE-T task force. `http://www.ieee802.org/3/an`.

[5] T.T.S.I. digital video broadcasting (DVB) second generation framing structure for broadband satellite applications. `http://www.dvb.org.`

[6] E. Amador, R. Knopp, V. Rezard, and R. Pacalet. Dynamic power management on LDPC decoders. In *ISVLSI 2010, IEEE Computer Society Annual Symposium on VLSI*, July 2010.

[7] A. Blanksby and C. J. Howland. A 690-mW 1-Gb/s 1024-b, rate 1/2 low-density parity-check code decoder. *JSSC*, 37(3):404–412, March 2002.

[8] A. Chandrakasan et al. Ultralow-power electronics for biomedical applications. *Annual Review of Biomedical Engineering*, pages 247–274, April 2008.

[9] M. Chao, J. Wen, et al. A triple-mode LDPC decoder design for IEEE 802.11n system. In *ISCAS*, pages 2445–2448, May 2009.

[10] J. Chen, A. Dholakia, E. Eleftheriou, and M. Fossorier. Reduced-complexity decoding of LDPC codes. *IEEE Transactions on Communications*, 53:1288–1299, August 2005.

[11] J. Chen and M. Fossorier. Near optimum universal belief propagation based decoding of low-density parity check codes. *IEEE Transactions on Communications*, 50:406–414, March 2002.

[12] L. Chen, J. Xu, I. Djurdjevic, and S. Lin. Near-shannon-limit quasi- cyclic low-density parity-check codes. *IEEE Transactions on Communications*, 52:1038–1042, 2004.

[13] X. Chen, J. Kang, S. Lin, and V. Akella. Memory system optimization for fpga-based implementation of quasi-cyclic ldpc codes decoders. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, PP(99):1 –1, 2010.

[14] Y. Chen and K. Parhi. Overlapped message passing for quasi-cyclic low-density parity check codes. *IEEE Transactions on Circuits and Systems I*, 51:1106–1113, 2004.

[15] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, and N. Wehn. A 477 mW NoC-based digital baseband for MIMO 4G SDR. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 278 –279, February 2010.

[16] Y. Dai, N. Chen, and Z. Yan. Memory efficient decoder architectures for quasi-cyclic LDPC codes. *IEEE Transactions on Circuits and Systems I*, 55:2898 – 2911, October 2008.

[17] A. Darabiha, A.C. Carusone, and F.R. Kschischang. A 3.3-Gbps bit-serial block-interlaced Min-Sum LDPC decoder in 0.13-um CMOS. In *IEEE Custom Integrated Circuits Conference*, pages 459–462, 2007.

[18] A. Darabiha, A.C. Carusone, and F.R. Kschischang. Power reduction techniques for LDPC decoders. *JSSC*, 43:1835–1845, August 2008.

[19] 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*, 7:317–319, 2003.

[20] R.G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253 –266, February 2010.

[21] M. Fossorier. Quasi-cyclic low-density parity-check codes from circulant permutation matrices. *IEEE Transaction Information Theory*, 50:1788–1793, August 2004.

[22] M. Fossorier et al. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on Communications*, 47:673–680, May 1999.

[23] R. Galbraith and T. Oenning. Iterative detection read channel technology in hard disk drives. *Hitachi white paper*, Nov 2008.

[24] R.L. Galbraith, T. Oenning, M. Ross, B. Wilson, I. Djurdjevic, and Jihoon Park. Architecture and implementation of a first-generation iterative detection read channel. *Magnetics, IEEE Transactions on*, 46(3):837 –843, March 2010.

[25] R. G. Gallager. Low-density parity check codes. *IRE Transaction Info.Theory*, IT-8:21–28, January 1962.

[26] K. K. Gunnam et al. Decoding of quasi-cyclic LDPC codes using an on-the-fly computation. In *40th Asilomar Conference on Signals, Systems and Computers*, pages 1192–1199, October 2006.

[27] Yang H. and W. Ryan. Low-floor decoders for ldpc codes. *Communications, IEEE Transactions on*, 57(6):1663 –1673, June 2009.

[28] D. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. In *Sips*, pages 107–112, October 2004.

[29] K. Iniewski. *VLSI Circuits for Biomedical Applications*. Artech-House, 685 Canton Street, Norwood, MA, USA, first edition, 2008.

[30] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISLPED '98: Proceedings of the 1998 international symposium on Low power electronics and design*, pages 197–202, New York, NY, USA, 1998. ACM.

[31] ITRS. International technology roadmap for semiconductors, 2007 update, interconnect section. Online. http://www.itrs.net/reports.html.

[32] J. Kang et al. A two-stage iterative decoding of LDPC codes for lowering error floors. In *Globecom*, pages 1–4, 2008.

[33] K.K.Gunnam et al. Next generation iterative LDPC solutions for magnetic recording storage. In *ACSSC*, pages 1148–1152, October 2008.

[34] 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*, 47(7):2711–2736, 2001.

[35] E.M. Kurtas, A.V. Kuznetsov, and I. Djurdjevic. System perspectives for the application of structured ldpc codes to data storage devices. *Magnetics, IEEE Transactions on*, 42(2):200 – 207, February 2006.

[36] L. Lan, L. Zeng, Y.Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar. Construction of quasi-cyclic ldpc codes for awgn and binary erasure channels: A finite field approach. *Information Theory, IEEE Transactions on*, 53(7):2429 –2458, July 2007.

[37] Z. Li, L. Chen, L. Zeng, S. Lin, and W.H. Fong. Efficient encoding of quasi-cyclic low-density parity-check codes. *Communications, IEEE Transactions on*, 54(1):71 – 81, January 2006.

[38] T. Limberg, M. Winter, M. Bimberg, R. Klemm, E. Matus, M.B.S. Tavares, G. Fettweis, H. Ahlendorf, and P. Robelly. A fully programmable 40 gops sdr single chip baseband for lte/wimax terminals. In *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*, pages 466 –469, September 2008.

[39] S. Lin and D. J. Castello Jr. *Error Control Coding.* Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2004.

[40] C. Liu, S. Yen, et al. An LDPC decoder chip based on self-routing network for IEEE 802.16e applications. *JSSC*, 43:684–694, March 2008.

[41] L. Liu and C.-J. R. Shi. Sliced message passing: High throughput overlapped decoding of high-rate low density parity-check codes. *IEEE Tran. on Circuits and Systems I*, 55:3697 – 3710, December 2008.

[42] R. Lynch et al. The search for a practical iterative detector for magnetic recording. *IEEE Transactions on Magnetics*, 40(1):213–218, January 2004.

[43] D. J. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45:399–431, March 1999.

[44] D.J.C. MacKay. *Information Theory Inference and Learning Algorithms.* Cambridge University Press, Cambridge, UK, 3rd edition, 2003.

[45] D.J.C. MacKay and R.M. Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 33(6):457–458, March 1997.

[46] M. Mansour and N. Shanbhag. Turbo decoder architectures for low-density parity-check codes. In *Globecom*, pages 1383–1388, November 2002.

[47] M. Mansour and N.R. Shanbhag. A 640-Mb/s 2048-bit programmable LDPC decoder chip. *JSSC*, 41:684–698, March 2006.

[48] F. Mlinarsky. Wireless HD video: Raising the throughput bar. *Wireless Net Designline*, Feb 2008.

[49] T. Mohsenin and B. Baas. Split-Row: A reduced complexity, high throughput LDPC decoder architecture. In *ICCD*, pages 13–16, October 2006.

[50] T. Mohsenin and B. Baas. High-throughput LDPC decoders using a multiple Split-Row method. In *ICASSP*, volume 2, pages 13–16, 2007.

[51] T. Mohsenin and B. Baas. Trends and challenges in ldpc hardware decoders. In *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, pages 1273–1277, November. 2009.

[52] T. Mohsenin and B. Baas. A split-decoding message passing algorithm for low density parity check decoders. *Journal of Signal Processing Systems*, February 2010. 10.1007/s11265-010-0456-y.

[53] T. Mohsenin, D. Truong, and B. Baas. An improved Split-Row Threshold decoding algorithm for LDPC codes. In *ICC*, 2009.

[54] T. Mohsenin, D. Truong, and B. Baas. Multi-Split-Row Threshold decoding implementations for LDPC codes. In *ISCAS*, May 2009.

[55] T. Mohsenin, D. Truong, and B. Baas. A low-complexity message-passing algorithm for reduced routing congestion in ldpc decoders. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 57(5):1048–1061, May. 2010.

[56] T. Mohsenin, P. Urard, and B. Baas. A thresholding algorithm for improved Split-Row decoding of LDPC codes. In *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, 2008.

[57] E. Morifuji et al. Power optimization for SRAM and its scaling. *IEEE Transactions on Electron Devices*, 54(4):715–722, April 2007.

[58] D. Oh and K. K. Parhi. Nonuniformly quantized min-sum decoder architecture for low-density parity-check codes. In *GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 451–456, New York, NY, USA, 2008. ACM.

[59] N. Onizawa, T. Hanyu, and V.C. Gaudet. Design of high-throughput fully parallel ldpc decoders based on wire partitioning. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(3):482 –489, March. 2010.

[60] Steven Pope. Look for power tradeoffs in 10GBASE-T ethernet. Online, January 2008. http://www.eetimes.com/design/power-management-design/4005683/Look-for-power-tradeoffs-in-10GBASE-T-Ethernet.

[61] J. M. Rabaey, A. Chandrakasan, and B. Nikolić. *Digital Integrated Circuits - A Design Perspective*. Prentice-Hall, New Jersey, USA, 2nd edition, 2003.

[62] T. Richardson. Error floors of LDPC codes. In *Allerton*, October 2003.

[63] T. Richardson and R. Urbanke. The capacity of low-density parity check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47:599–618, February 2001.

[64] P. Saxena, N. Menezes, P. Cocchini, and D.A. Kirkpatric. Repeater scaling and its impact on cad. *Computer-Aided Design of Integrated Circuits and SYstems, IEEE Transactions on*, 23(4):451–463, April 2004.

[65] P. Saxena, R. S. Shelar, and S. S. Sapatnekar. *Routing Congestion in VLSI Circuits*. Springer Science, NYC, NY, USA, 1st edition, 2007.

[66] X. Shih, C. Zhan, et al. An 8.29 mm^2 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 CMOS process. *JSSC*, 43:672–683, March 2008.

[67] Yang Sun and J.R. Cavallaro. A low-power 1-gbps reconfigurable ldpc decoder design for multiple 4g wireless standards. In *SOC Conference, 2008 IEEE International*, pages 367 –370, September 2008.

[68] W. Tan. Design of inner ldpc codes for magnetic recording channels. *IEEE Transactions on Magnetics*, 44:217–222, January 2008.

[69] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27:533–547, 1981.

[70] R. M. Tanner et al. LDPC block and convolutional codes based on circulant matrices. *TIT*, 50:2966–2984, December 2004.

[71] D. Truong et al. A 167-processor computational platform in 65 nm CMOS. *IEEE Journal of Solid-State Circuits (JSSC)*, 44(4):1130–1144, April 2009.

[72] D. N. Truong and B. M. Baas. Circuit modeling for practical many-core architecture design exploration. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 627–628, June 2010.

[73] Y.-L. Ueng, C.-J. Yang, K.-C. Wang, and C.-J. Chen. A multimode shuffled iterative decoder architecture for high-rate rs-ldpc codes. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, PP(99):1 –1, 2010.

[74] W. Wang and G. Choi. Speculative energy scheduling for ldpc decoding. In *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, pages 79 –84, March 2007.

[75] W. Wang, G. Choi, and K.K. Gunnam. Low-power VLSI design of ldpc decoder using DVFS for AWGN channels. In *VLSI Design, 2009 22nd International Conference on*, pages 51 –56, january 2009.

[76] R. Wilson. 10GBase-T: Is it really coming this time? Online, August 2009. http://www.edn.com/article/459408-10GBase_T_Is_it_really_coming_this_time_.php.

[77] E. Yeo and B. Nikolic. A 1.1-gb/s 4092-bit low-density parity-check decoder. In *Asian Solid-State Circuits Conference, 2005*, pages 237 –240, November. 2005.

[78] Y.Sun and J.R.Cavallaro. A low-power 1-Gbps reconfigurable LDPC decoder design for multiple 4G wireless standards. In *SOC Conference*, pages 367–370, 2008.

[79] H. Zhang, J. Zhu, et al. Layered approx-regular LDPC code construction and encoder/decoder design. *IEEE Transactions on Circuits and Systems I*, 55:572 – 585, March 2008.

[80] J. Zhang and M. P. C. Fossorier. A modified weighted bit-flipping decoding of low-density parity-check codes. *IEEE Communications Letters*, 8:165–167, March 2004.

[81] K. Zhang, X. Huang, and Z. Wang. High-throughput layered decoder implementation for quasi-cyclic ldpc codes. *Selected Areas in Communications, IEEE Journal on*, 27(6):985 –994, August 2009.

[82] Z. Zhang, V. Anantharam, M.J. Wainwright, and B. Nikolic. An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors. *Solid-State Circuits, IEEE Journal of*, 45(4):843 –855, April 2010.

[83] Z. Zhang, L. Dolecek, et al. Lowering LDPC error floors by postprocessing. In *Globecom*, pages 1–6, 2008.

[84] Z. Zhang et al. A 47 Gb/s LDPC decoder with improved low error rate performance. In *Symposium on VLSI Circuits*, pages 22–23, 2009.

[85] Z. Zhang, A. Venkat, et al. Quantization effects in low-density parity-check decoders. In *ICC*, pages 6231–6237, 2007.

[86] B. Zhen et al. IEEE body area networks and medical implant communications. In *ICST*, March 2008.

[87] H. Zhong et al. Area-efficient min-sum decoder design for high-rate quasi-cyclic low-density parity-check codes in magnetic recording. *IEEE Transactions on Magnetics*, 43:4117–4122, December 2007.