A Low-Complexity Message-Passing Algorithm for Reduced Routing Congestion in LDPC Decoders

Tinoosh Mohsenin, Dean N. Truong, and Bevan M. Baas

***Abstract*—A low-complexity message-passing algorithm, called *Split-Row Threshold*, is used to implement low-density parity-check (LDPC) decoders with reduced layout routing congestion. Five LDPC decoders that are compatible with the 10GBASE-T standard are implemented using MinSum Normalized and MinSum Split-Row Threshold algorithms. All decoders are built using a standard cell design flow and include all steps through the generation of GDS II layout. An $Spn = 16$ decoder achieves improvements in area, throughput, and energy efficiency of 4.1 times, 3.3 times, and 4.8 times, respectively, compared to a MinSum Normalized implementation. Postlayout results show that a fully parallel $Spn = 16$ decoder in 65-nm CMOS operates at 195 MHz at 1.3 V with an average throughput of 92.8 Gbits/s with early termination enabled. Low-power operation at 0.7 V gives a worst case throughput of 6.5 Gbits/s—just above the 10GBASE-T requirement—and an estimated average power of 62 mW, resulting in 9.5 pJ/bit. At 0.7 V with early termination enabled, the throughput is 16.6 Gbits/s, and the energy is 3.7 pJ/bit, which is 5.8× lower than the previously reported lowest energy per bit. The decoder area is 4.84 mm^2 with a final postlayout area utilization of 97%.**

***Index Terms*—Full parallel, high throughput, low-density parity check (LDPC), low power, message passing, min sum, nanometer, 10GBASE-T, 65-nm CMOS, 802.3an.**

I. Introduction

STARTING in the 1990s, much work was done to enhance error-correction codes to where communication over noisy channels was possible near the Shannon limit. Defined by sparse random graphs and using probability-based message-passing algorithms, low-density parity-check (LDPC) codes [1] became popular for their error-correction and near-channel-capacity performances. At first, neglected since its discovery [2], advances in VLSI have given LDPC a recent revival [3]–[6]. LDPC has relatively low error floors, as well as better error performance with large code lengths, and as a result, they have been adopted as the forward error-correction method for many recent standards, such as digital video broadcasting via satellite (DVB-S2) [7], the WiMAX standard for microwave communications (802.16e) [8], the G.hn/G.9960 standard for wired home networking [9], and the 10GBASE-T standard for 10-Gbit Ethernet (802.3an) [10]. While there has been much research on LDPC decoders and their VLSI implementations, high-speed systems that 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. These delay and energy costs caused by wires are likely to increase in future fabrication technologies [11].

With these concerns in mind, the design of a future LDPC decoder will require high performance and low power with the following: 1) a large number of nodes that have a high degree of interconnectedness and/or 2) a large memory capacity with high memory bandwidths. These requirements are due, in part, to the message-passing algorithm used by the LDPC decoder. Traditionally, this was done with the sum–product algorithm (SPA) [12] or the MinSum algorithm [13]. Our previous work introduced two nonstandard LDPC decoding algorithms based on min sum, called "Split-Row" [14] and "multisplit" [15] algorithms, that were proven to increase throughput up to five times, and reduce wiring and area three times [16]. Split-row algorithm achieves this through the partitioning of MinSum algorithm's global $\min()$ operation into semiautonomous localized $\min()$ operations. As a result of the reduction in message passing, there is a 0.3- to 0.7-dB reduction in performance, depending on the level of partitioning.

The recently published "Split-Row Threshold" algorithms add a comparison to a threshold constant that is used to partially recover the lost $\min()$ information [17], [18]. A set of 1-bit global signals is needed with very few additional logic blocks; significant error performance recovery is achieved with only 0.07-dB loss from MinSum Normalized algorithm [18]. Greater levels of partitioning is now accessible at less error performance loss and will enable designs of fully parallel decoder architectures that have increased throughput and energy efficiency, and reduced area and power [19]. This paper is organized as follows: Section II reviews iterative message-passing algorithms for LDPC decoders and common decoder architectures, Section III studies the Split-Row algorithm and its ability to reduce routing congestion in layout, Section IV introduces a low-hardware-cost modification to Split-Row algorithm called Split-Row Threshold algorithm that improves error performance while maintaining the former's routing reduction benefits, Section V shows the detailed fully parallel Split-Row Threshold architecture, Section VI analyzes the postlayout results of fully parallel 10GBASE-T LDPC decoders that implement Split-Row Threshold, and Section VII concludes this paper.

II. Background

A. MinSum Decoding Algorithm

The iterative message-passing algorithm is the most widely used method for practical decoding [1], [20], and its basic flow is shown in Fig. 1. After receiving the corrupted information

Manuscript received October 07, 2009; revised December 25, 2009; accepted January 30, 2010. First published May 10, 2010; current version published May 21, 2010 . This work was supported in part by ST Microelectronics, by Intel Corporation, by UC Micro, by the National Science Foundation under CCF Grants 0430090 and 0903549 and CAREER Award 0546907, by Semiconductor Research Corporation under GRC Grants 1598 and 1971 and CSR Grant 1659, by Intellasys Corporation, by SEM, and by a University of California at Davis (UCD) Faculty Research grant. This paper was recommended by Associate Editor Y. Massoud.

The authors are with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616 USA (e-mail: tmohsenin@ucdavis.edu).

Digital Object Identifier 10.1109/TCSI.2010.2046957

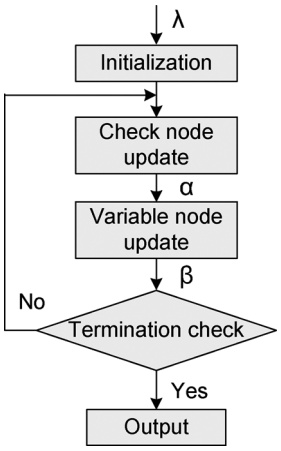

Fig. 1. Flow diagram of an iterative message-passing decoding algorithm.

from an additive white Gaussian noise (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 code word is detected, which is explained in Section II-D.

An LDPC code is defined by an $M \times N$ parity-check matrix H, which encapsulates important matrix parameters: The number of rows M is the number of check nodes, the number of columns (or *code length*) N is the number of variable nodes, and row weight W_r and column weight W_c define the 1's per row and column, respectively. For clearer explanations, in this paper, we examine cases where H is *regular* and, thus, W_r and W_c are constants. As an example, the 10GBASE-T code 384×2048 H matrix has a $W_r = 32$ and $W_c = 6$. There are $M = 384$ check nodes and $N = 2048$ variable nodes, and wherever $H(i, j) = 1$, there is an edge (interconnection) between check node C_i and variable node V_j. In other words, the LDPC matrix H is a matrix describing a graph, whose vertices are check and variable nodes and whose edges are the interconnections between the nodes of the LDPC code.

In practice, among the iterative message-passing algorithms, Sum Product algorithm (SPA) [12] and MinSum algorithm [13] have become the standard decoding methods. Both algorithms are defined by a check node update equation that generates α and a variable node update equation that generates β. The MinSum variable node update equation, which is identical to the SPA version, is given as

$$\beta_{ij} = \lambda_j + \sum_{i' \in C(j) \backslash i} \alpha_{i'j} \tag{1}$$

where each β_{ij} message is generated using the noisy channel information (of a single bit), λ_j, and the α messages from all check nodes $C(j)$ connected to variable node V_j, as defined by H (excluding C_i). MinSum algorithm simplifies the SPA check node update equation, which replaces the computation of an elementary equation by a $\min()$ function. The MinSum check node update equation is given as

$$\alpha_{ij} = S_{MS} \times \underbrace{\prod_{j' \in V(i) \backslash j} \operatorname{sign}(\beta_{ij'})}_{\text{Sign Calculation}} \times \underbrace{\min_{j' \in V(i) \backslash j} (|\beta_{ij'}|)}_{\text{Magnitude Calculation}} \tag{2}$$

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 scaling factor S_{MS} is included to improve error performance, and thus, this variant of MinSum algorithm is called "MinSum Normalized algorithm" [21], [22]. 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:

$$\min_{j' \in V(i) \backslash j} (|\beta_{ij'}|) = \begin{cases} Min1_i, & \text{if } j \neq \arg\min(Min1_i) \\ Min2_i, & \text{if } j = \arg\min(Min1_i) \end{cases} \tag{3}$$

where

$$Min1_i = \min_{j \in V(i)} (|\beta_{ij}|) \tag{4}$$

$$Min2_i = \min_{j'' \in V(i) \backslash \arg\min(Min1_i)} (|\beta_{ij''}|). \tag{5}$$

Moreover, the term $\prod \operatorname{sign}(\beta_{ij'})$ 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 (3).

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 six 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 $M \times W_r + N \times W_c = 384 \times 32 + 2048 \times 6 = 24\,576$ connections, where check nodes send, as an aggregate, 12 288 α messages to the variable nodes and the variable nodes, as an aggregate, send 12 288 β 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 $M \times W_r = 12\,288$ total check node update computations and $N \times W_c = 12\,288$ total variable node update computations, as well as pass these updated results for a total of $M \times W_r + N \times W_c = 24\,576$ unique messages.

B. Fully Parallel Decoders

Fully 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 [4], [5], [15], [19]. All M check nodes, N variable nodes, and their associated $M \times W_r + N \times W_c$ total connections are implemented. Thus, a fully parallel decoder will have $M + N$ check and variable node processors, and $M \times W_r + N \times W_c$ global interconnections. A fully parallel 10GBASE-T LDPC decoder will require 2432 processors which are interconnected

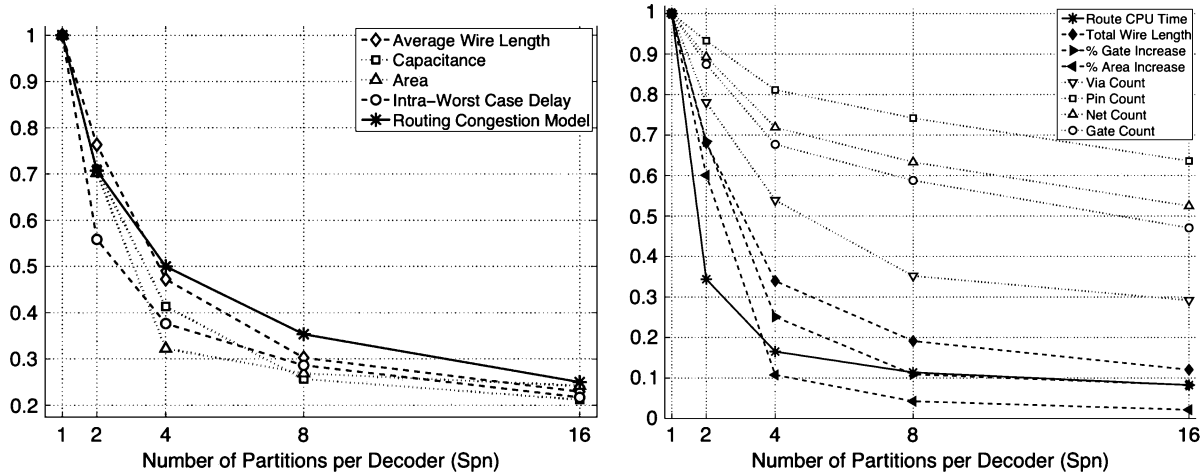

Fig. 2. Physical indicators of interconnection complexity over five Spn decoders ($Spn = 1, 2, 4, 8$, and 16) normalized to the the case where $Spn = 1$ (i.e., min sum). A 5-bit data path (1-bit sign and 4-bit magnitude) is used for all five decoder implementations. (a) Normalized ratios of per-partition 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 and area increase* (i.e., A_c versus A_{pl}), and *via/pin/net/gate count*, compared with routing CPU time.

by a set of $24\,576 \times b$ global wires and their associated wire buffers (i.e., repeaters), where b is the number of bits in the data path. Thus, 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 fully parallel decoders have larger area and capacitance, and lower operating frequencies than their partially parallel counterparts (to be discussed shortly in the next section), they typically only need a single cycle per message-passing iteration; thus, fully parallel decoders are inherently more energy efficient [5].

C. Fully Serial and Partially Parallel Decoders

In contrast to fully parallel decoders, fully 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 fully serial LDPC decoder will require a $24\,576 \times b$-bit memory. A 5-bit data path 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. However, this requires a multiport SRAM, which increases SRAM area significantly [23], and building large high-performance SRAMs in deep-submicrometer technologies results in sizable leakage currents [24].

Although much smaller than fully parallel decoders, fully serial decoders have much lower throughputs and larger latencies. Partially parallel designs [6], [25]–[28] 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 [29] or quasi-cyclic [30] codes are very well suited for partially parallel decoder implementations. The parity-check matrix of these codes consists of square submatrices, where each submatrix is either a zero matrix or a permuted identity. This structure makes the memory address generation for partially parallel decoders very efficient, and many communication standards, such as DVB-S2, 802.11n, and 802.16e, use this structure.

D. 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 whether the variable nodes have reached a valid solution to the syndrome check equation: $\beta_{V_j} \cdot H^{\mathrm{T}} = 0$[1], [20], where β_{V_j} represents the beta messages of each variable node V_j, i.e., the recovered bits of λ. When satisfied, *early termination* occurs, and the message is considered error free since parity among the rows of H has been met

III. Routing Congestion Reduction With Split-Row Algorithm

A. Split-Row Decoding Algorithm

The Split-Row decoding algorithm uses columnwise partitioning of the H matrix to reduce the interconnect complexity due to α message passing in the check node update step [14]–[16]. Block-structured and quasi-cyclic codes which have regular row weight (an equal number of ones per row) receive the greatest benefit from the Split-Row architecture because the routing congestion reduction of partitions is equal and, therefore, the total congestion is minimized. Then, if $Spn = 2$, the 10GBASE-T H matrix, which has a row weight of 32, is "split" into two submatrices, each with a dimension of $M \times (N/Spn) = 384 \times 1024$. For simplicity of presentation, we assume even partitioning such that W_r is divided equally into Spn partitions, where W_r/Spn is an integer, although there is no reason why the algorithm cannot be applied to decoders with H matrices of other forms. This results in two decoder partitions that each has reduced $M \times (W_r/Spn) = 6144$ total check node update computations because of their reduced row weight (W_r/Spn) and reduced $(N/Spn) \times W_c = 6144$ total variable node update computations because of their reduced number of variable nodes (N/Spn). The number of messages passed within each partition is now $M \times (W_r/Spn) + (N/Spn) \times W_c = 12\,288$, which is half that of a complete min sum.

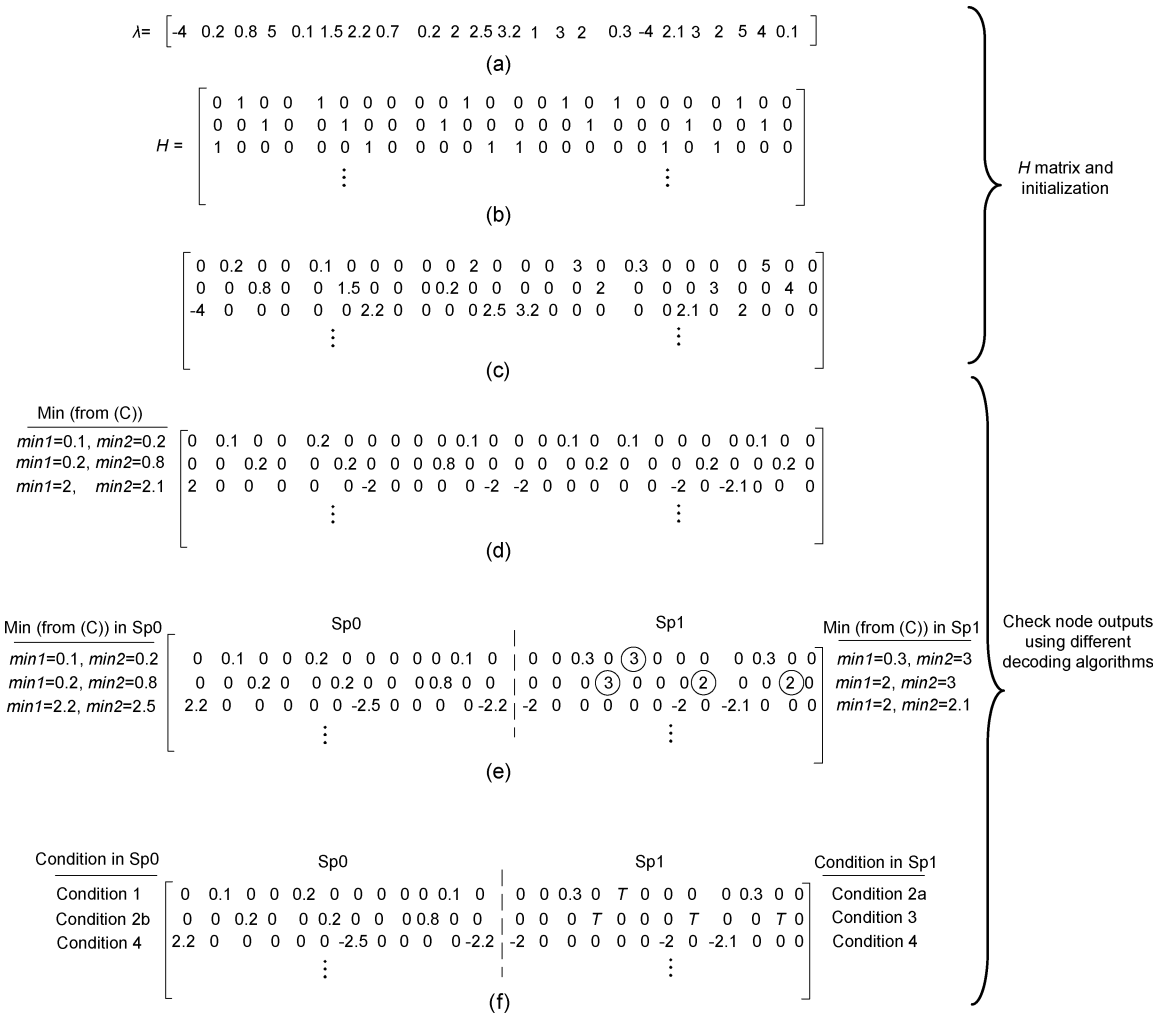

Fig. 3. Channel data (λ), H matrix, the initialization matrix, and the check node output (α) values after the first iteration using MinSum Normalized, MinSum Split-Row, and MinSum Split-Row Threshold algorithms. The Split-Row algorithm entries in (e) with the largest deviation from MinSum Normalized algorithm are circled and are largely corrected with the Split-Row Threshold method in (f). Correction factors are set to be one here.

Simply put, Split-Row algorithm modifies the check node update (2) of MinSum Normalized algorithm and results in the following:

$$\alpha_{ij:Spn} = S_{SR} \times \underbrace{\prod_{j' \in V(i) \backslash j} \operatorname{sign}(\beta_{ij'})}_{\text{Sign Calculation}} \times \underbrace{\min_{j' \in V_{Spn}(i) \backslash j} (|\beta_{ij'}|)}_{\text{Magnitude Calculation}} \tag{6}$$

where V_{Spn} represents the $V(i)$ variable nodes only contained in decoder partition Spn on row i (recall that each partition has N/Spn variable nodes). Notice that the sign information is complete and is the only global signal propagating between the "split" decoder partitions.

Because Split-Row algorithm is a general modification of the message-passing algorithm, it can be used with both SPA and MinSum algorithm [16]. In addition, Split-Row algorithm, like SPA or MinSum algorithm, is not restricted to either fully parallel, fully serial, or partially parallel architectures. However, the major benefit from "splitting" is its ability to reduce routing congestion, which primarily only affects the global interconnects caused by message passing. Fully serial architectures will gain nothing, while fully parallel architectures will improve greatly; partially 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 Section IV-A).

B. Routing Congestion Reduction

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. However, 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 fully parallel decoders, Split-Row algorithm provides significant increase in area utilization over other message-passing decoders that implement large row weight LDPC codes [5], [14], [15], [23]. 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, etc.). Routing congestion is defined as $g = T_d/T_s$, where T_d is the number of "tracks demanded" and T_s is the number of "tracks supplied" [31]. 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.

Fig. 2(a) shows the g_{Spn} with the per-partition capacitance, area, average wire length, and worst case (intrapartition) delay of five postlayout 10GBASE-T LDPC decoder implementations with $Spn = 1$, 2, 4, 8, and 16. (Note that $Spn = 1$ represents a decoder using MinSum Normalized algorithm.) Clearly, our routing congestion model ($g_{Spn} = 1/\sqrt{Spn}$) follows these metrics closely. For all five decoders, we use a 5-bit data path (1-bit sign and 4-bit magnitude). Since Split-Row algorithm eliminates the wires for check node magnitudes between partitions, its impact of routing congestion reduction is even more significant when implementing larger data-path widths. On the other hand, using a smaller data-path width results in significant error performance loss when compared to a floating-point implementation. Usually, a 4–6-bit data path results in near-floating-point error performance.

Cadence's SoC Encounter computer-aided design (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 the total wire length can be used as a pessimistic measure of routing congestion [31], then it should behave closely with the route CPU time. Fig. 2(b) shows 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 the total wire length, albeit at a different magnitude. Notice that gate, pin, and net counts do not increase rapidly until $Spn = 2$, which indicates that increases in wire buffering change more dramatically, starting at the $Spn = 2$ data point. Since the gate, pin, and net counts represent the connectivity needed by the design and are not decreasing as much as compared to the 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 postlayout (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 have shown that Split-Row algorithm can reduce routing congestion by a factor of $1/\sqrt{Spn}$. Actual implementation results of decoders at the postlayout step bore out this conclusion.

IV. Split-Row Threshold Decoding Method

A. Split-Row Error Performance

The major drawback of Split-Row algorithm is that it suffers from a 0.4–0.7 dB error performance loss that is proportional to Spn compared to MinSum algorithm and SPA [12] 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,

[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.

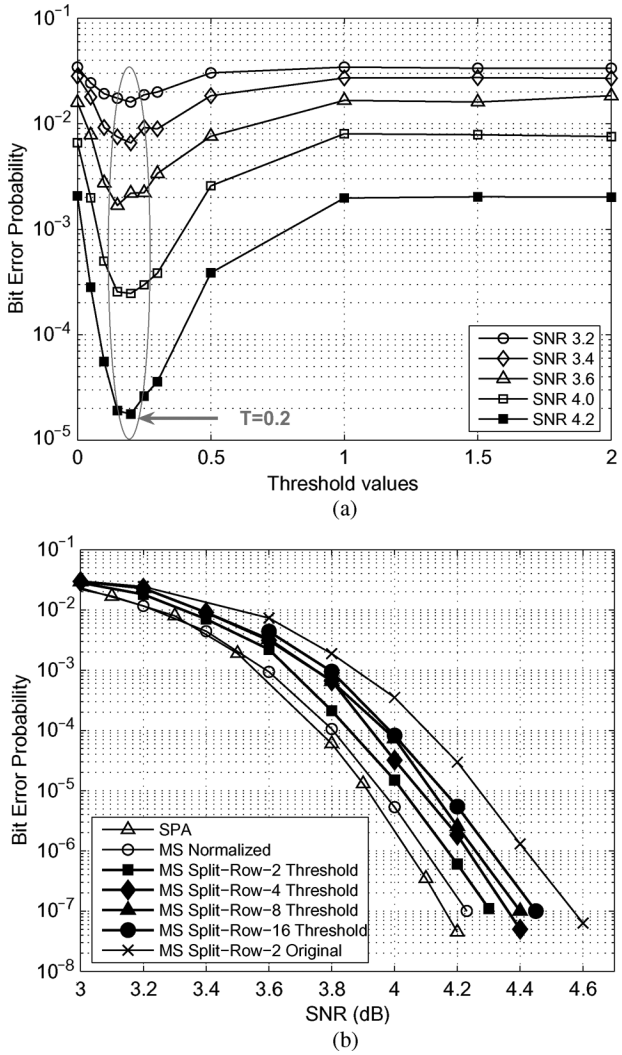

Fig. 4. Impact of choosing threshold value (T) on the error performance and BER comparisons for a (6, 32) (2048, 1723) LDPC code using SPA and MinSum Normalized, MinSum Split-Row (original), and MinSum Split-Row Threshold algorithms with different levels of partitioning and with optimal threshold and correction factor values. (a) 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.

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.

Fig. 3 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 $N = 24$. Fig. 3(c) shows the initialization step where all nonzero entries are initialized with channel data. Moreover, Fig. 3 shows the check node outputs using (d) MinSum, (e) MinSum Split-Row, and (f) MinSum Split-Row Threshold algorithms based on the initialization step. To find $Min1$ and $Min2$ for each case, reader should look at Fig. 3(c). For example, in Fig. 3(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. 3(d).

In Split-Row algorithm, entries with the largest deviations from MinSum Normalized algorithm are circled. For example, in the second row of the Split-Row matrix output in (e), the local minimum $(Min1)$ in Sp1 is "2," which is ten times larger than the local minimum in Sp0, i.e., "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"), the $Min2$ in Sp1 ("3") deviates significantly from the $Min2$ in Sp0 ("0.2"), and this results in a large error in the bit on the right side.

B. Split-Row Threshold Algorithm

The Split-Row Threshold algorithm significantly improves the error performance without reducing the effectiveness of Split-Row algorithm while adding negligible hardware to the check node processor and one additional wire between blocks [18]. Like Split-Row algorithm, the check node processing step is partitioned into multiple semiautonomous (Spn) partitions. Each partition computes the 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.

Algorithm 1 Split-Row Threshold Algorithm

Require: $j' \in V_{Spi}(i) \setminus j$

Require: $Min1_i$ and $Min2_i$ as given in (4) and (5)

Require: T threshold value

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

2: // for the ith partition of a Spn-decoder (Spi).

3:

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

5: $Threshold\_ensp(i)\_out(i \pm 1) = 1$

6:

$$\min(|\beta_{ij'}|) = \begin{cases} Min1_i, & \text{if } j \neq \arg\min(Min1_i) \\ Min2_i, & \text{if } j = \arg\min(Min1_i) \end{cases} \quad (7)$$

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

8: $Threshold\_ensp(i)\_out(i \pm 1) = 1$

9 **if** $Threshold\_ensp(i+1) == 1$ **or** $Threshold\_ensp(i-1) == 1$ **then**

10:

$$\min(|\beta_{ij'}|) = \begin{cases} Min1_i, & \text{if } j \neq \arg\min(Min1_i) \\ T, & \text{if } j = \arg\min(Min1_i) \end{cases} \quad (8)$$

11: **else**

12: **do** (7)

13: **end if**

14: **else if** $Min1_i > T$ **and** ($Threshold\_ensp(i+1) == 1$ **or** $Threshold\_ensp(i-1) == 1$) **then**

15: $Threshold\_ensp(i)\_out(i \pm 1) = 0$

16:

$$\min(|\beta_{ij'}|) = T \quad (9)$$

17: **else**

18 $Threshold\_ensp(i)\_out(i \pm 1) = 0$

19: **do** (7)

20: **end if**

The kernel of the Split-Row Threshold algorithm 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 (7). In addition, $Threshold\_ensp(i)\_out(i \pm 1)$, which represents the general threshold-enable signal of a partition Spi 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**, $Threshold\_ensp(i)\_out(i \pm 1) = 1$. If at least one $Threshold\_ensp(i \pm 1)$ signal from the nearest neighboring partitions is high, indicating that the local minimum in the other partition is less than T, then we use $Min1$ and T to update the α messages, while using (8) (**Condition 2a**). Otherwise, we use (7) (**Condition 2b**). **Condition 3**, as represented by *lines* **14 to 16**, occurs when the local $Min1$ is larger than T and at least one $Threshold\_ensp(i \pm 1)$ signal from the nearest neighboring partitions is high; thus, we only use T to compute all α messages for the partition using (9). **Condition 4**, as represented by *lines* **17 to 19** occurs when the local $Min1$ is larger than T and if the $Threshold\_ensp(i \pm 1)$ signals are all low; thus, we again use (7). The variable node operation in Split-Row Threshold algorithm is identical to the MinSum Normalized and Split-Row algorithms.

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

C. BER 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 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(a) plots the error performance of a (6,32) (2048,1723) RS-based LDPC code [32], adopted for 10GBASE-T [10] 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 ($T = 0.2$)

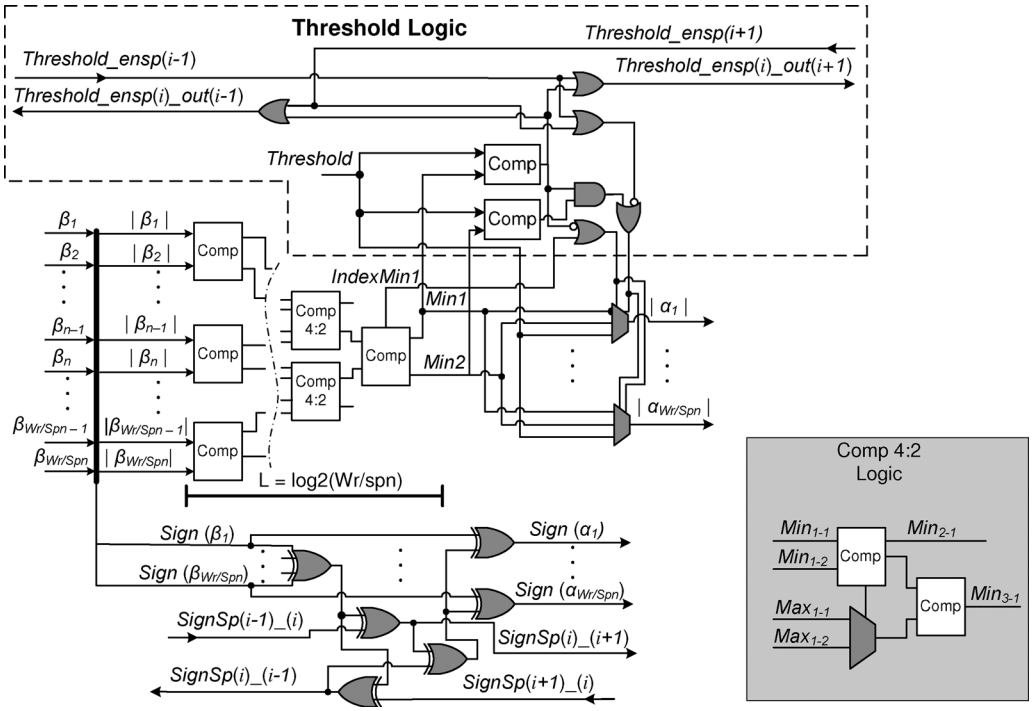

Fig. 5. Block diagram of magnitude and sign update in the check node processor of partition $Sp(i)$ in Split-Row Threshold decoder. The *Threshold Logic* is shown within the dashed line. The 4:2 comparator block is shown at the right.

TABLE I
AVERAGE OPTIMAL THRESHOLD VALUE (T) FOR THE SPLIT-ROW THRESHOLD DECODER WITH DIFFERENT LEVELS OF PARTITIONING, FOR A (6, 32) (2048, 1723) LDPC CODE

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

for which the algorithm performs best. Bit error rate (BER) simulation results for (16, 16) (175, 255) EG-LDPC and (4, 16) (1536, 1155) QC-LDPC codes show that the average optimal thresholds T with two partitions are 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, the less T is allowed to vary, decoders will have higher throughput, and higher energy efficiency. Fortunately, as can be seen in Fig. 4(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 as long as there are two variable nodes per partition and the error performance loss is less than 0.3 dB. For example, Fig. 4(b) shows the error performance results for a (6, 32) (2048, 1723) LDPC code for (from left to right) SPA, MinSum Normalized algorithm, MinSum Split-Row Threshold algorithm with different levels of splitting (Spn) and with optimal threshold values, and, lastly, Split-Row algorithm (original). As the figure shows, the MinSum Split-2 Thresholds are about 0.13 and 0.07 dB away from SPA and MinSum Normalized algorithm, respectively. The SNR loss between multiple Split-Row Threshold decoders is less than 0.05 dB, and the total loss from Split-16 Threshold

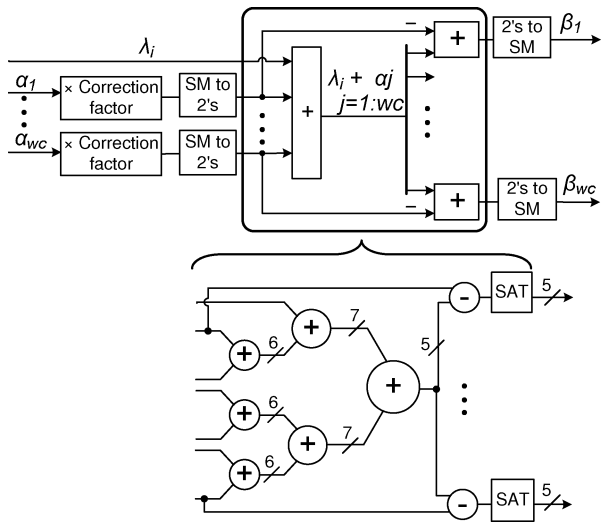

Fig. 6. Block diagram of variable node update architecture for MinSum Normalized and MinSum Split-Row Threshold decoders.

to Split-2 Threshold is 0.15 dB at $BER = 10^{-7}$. Moreover, shown in the plot is the Split-2 original algorithm, which is still 0.12 dB away from the Split-16 Threshold algorithm. Table I summarizes the average optimal threshold values (T) for Split-Row Threshold decoder and shows small changes with different partitioning levels.

V. SPLIT-ROW THRESHOLD DECODING ARCHITECTURE

A. Check Node Processor

The logical implementation of a check node processor in partition Spi using Split-Row Threshold decoding is shown in Fig. 5. 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

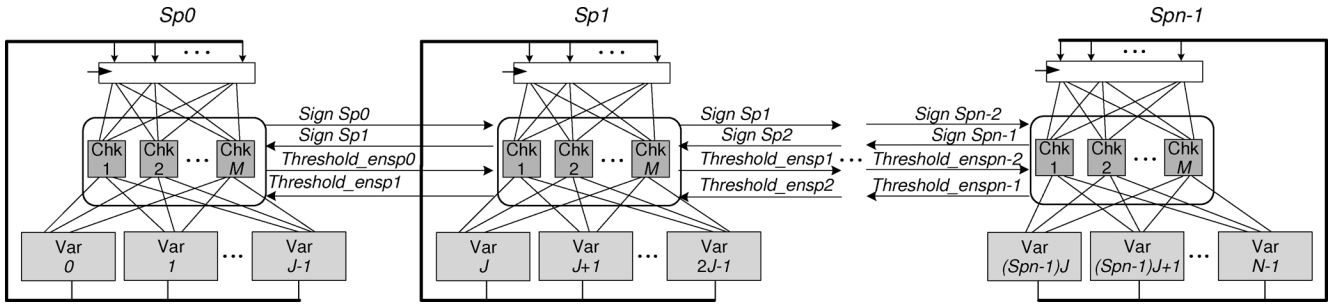

Fig. 7. Top-level block diagram of a fully parallel decoder corresponding to an $M \times N$ parity-check matrix, using Split-Row Threshold decoding with Spn partitions. The interpartition $Sign$ and $Threshold\_en$ signals are highlighted. $J = N/Spn$, where N is the code length.

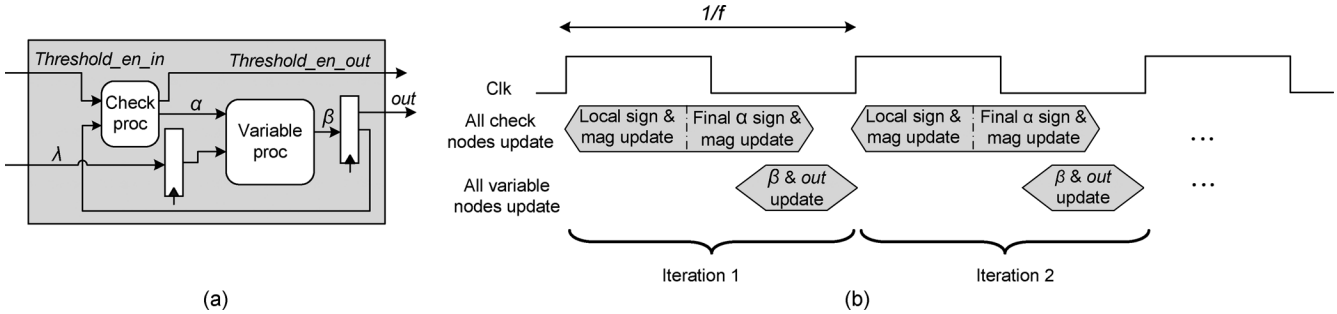

Fig. 8. (a) Pipeline and (b) timing diagram for one partition of Split-Row Threshold decoder. In each partition, the check and variable node messages are updated in one cycle after receiving the $Sign$ and $Threshold\_en$ signals from the nearest neighboring partitions.

decoding, the sign bit calculated from partition Spi is passed to the $Sp(i-1)$ and $Sp(i+1)$ neighboring partitions to correctly calculate the global sign bit according to the check node processing (2) and (6).

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 leftmost) is composed of simple comparators which sort the two inputs and generate min and max outputs. The second stage and afterwards consist of multiple 4 to 2 (4:2) comparators, and the details are shown in a block in the right corner of Fig. 5. One benefit of the Split-Row Threshold algorithm is that the number of β inputs (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 $Sp(i-1)$ and $Sp(i+1)$ neighboring partitions and is then sent to their opposite neighbors.

B. 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. 6 shows the variable node processor architecture for both MinSum and Split-Row Threshold decoders, which computes β messages according to (1) and contains multistage adders. Its complexity highly depends on the numbers of inputs (column weight W_c) and the input word widths. As shown in the figure, this implementation uses a 5-bit data path.

C. Fully Parallel Decoder Implementation

The block diagram of a fully parallel implementation of Split-Row Threshold decoding with Spn partitions, highlighting the $Sign$ and $Threshold\_en$ passing signals, is shown in Fig. 7. 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 is 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. Fig. 8(a) shows the pipeline diagram of one partition in the decoder. The timing diagram of a check node and a variable node update is shown in Fig. 8(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 fully parallel implementations, all check and variable processor outputs are updated in parallel, and as shown in the timing diagram in Fig. 8(b), it takes one cycle to update all messages for one iteration. Therefore, the throughput for the proposed fully parallel decoder with code length N is

$$Throughput = \frac{N * f}{Iterations} \tag{10}$$

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

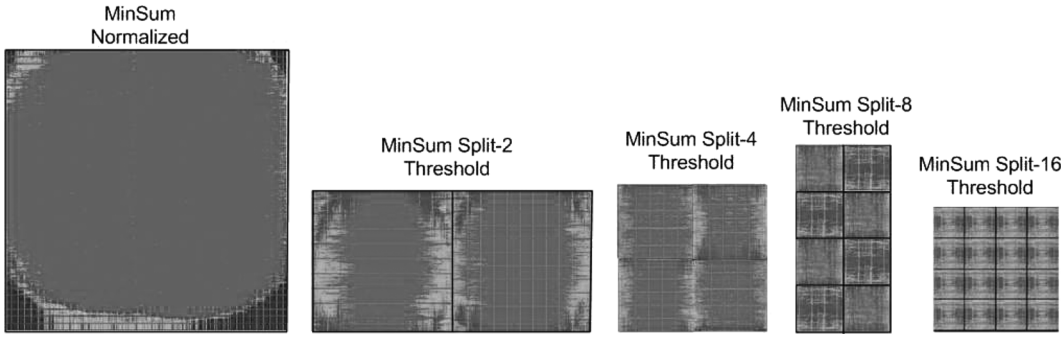

Fig. 9. Layout of MinSum Normalized and MinSum Split-Row Threshold decoder implementations shown approximately to scale for the same code and design flow.

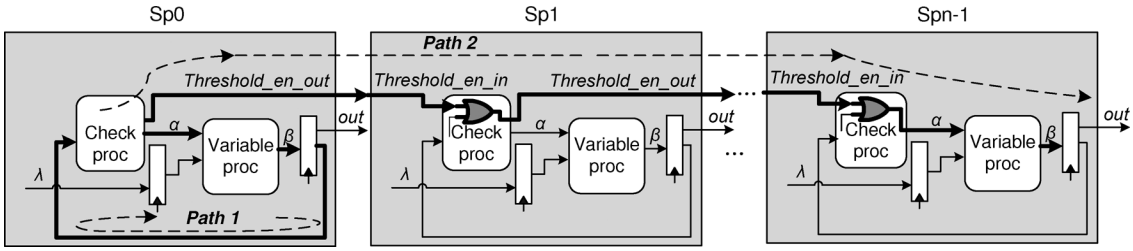

Fig. 10. Check to variable processor critical path *Path1* and the interpartition $Threshold\_en$ critical path *Path2* for the Split-Row Threshold decoding method with Spn partitions.

VI. Design of Five CMOS Decoders

To further investigate the impact on the hardware implementation due to partitioning, we have implemented five fully parallel decoders using MinSum Normalized and Split-Row Threshold methods with multiple partitionings for the (6, 32) (2048, 1723) 10GBASE-T LDPC code in 65-nm seven-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, a row weight of 32 ($W_r = 32$), a column weight of six ($W_c = 6$), and an information length of 1723. The fully 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 subblock by a factor of $1/Spn$. For example, in each Split-16 subblock, there are 384 check processors (although 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 communicates with short and structured sign and $Threshold\_en$ passing wires. In this implementation, we use a 5-bit fixed-point data path, 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.

A. 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 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. 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 fully parallel decoder implementation of large LDPC codes using automatic CAD tools. Since Split-Row Threshold decoder reduces the check node processor complexity and the interconnection between check 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.

Fig. 9 shows the postroute GDS II layout implementations drawn roughly to scale for the five decoders using MinSum Normalized and MinSum 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.

B. 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 and variable node processors. However, in Split-Row Threshold decoder, since the *Threshold Logic* (see Fig. 5) is also dependent on the neighboring $Threshold\_en$ signals from partitions $Sp(i-1)$ and $Sp(i+1)$, one possible critical path is a $Threshold\_en$ signal that finally propagates to partition Spi and changes the mux select bits influencing check node output α. To illustrate both critical paths, Fig. 10 shows 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

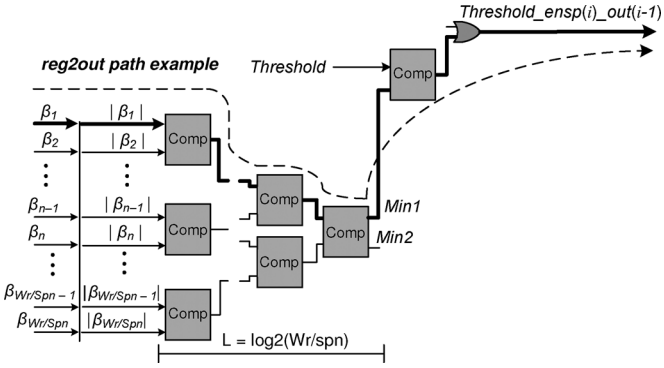

Fig. 11. Components in the $reg2out$ delay path for $Threshold\_en$ propagation signal in MinSum Split-Row Threshold decoding.

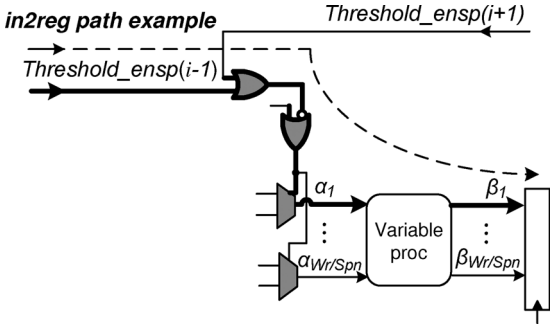

Fig. 12. Components in the $in2reg$ delay path for $Threshold\_en$ propagation signal in MinSum Split-Row Threshold decoding.

TABLE II
$Threshold\_en$ DELAY PATH COMPONENTS FOR THE SPLIT-ROW 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

the $Threshold\_en$ signal, starting from the leftmost partition's ($Sp0$) check node processor, through $Spn-2$ middle blocks, 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. 11. 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 $Threshold\_en\_out$ 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 that 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. 12, which goes through the variable processor and ends at the register.

Table II 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. The total interpartition $Threshold\_en$ delay (T_{Th_en}) for an Spn-way Split-Row Threshold decoder is the summation of all three delay categories

$$T_{\text{Th\_en}} = T_{\text{reg2out}} + (Spn-2)\cdot T_{\text{in2out}} + T_{\text{in2reg}}. \tag{11}$$

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 (11).

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

$$T_{\text{Spn\_threshold}} = \max\{T_{\text{Th\_en}}, T_{C\_\text{to}\_V}\}. \tag{12}$$

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. 13 show the postroute delay breakdown of (a) Path1 (check to variable processor) and (b) Path2 ($Threshold\_en$ propagation) in the decoders and are partitioned into interconnect and logic (check and 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 methods, Path1 is the dominant critical path, but for $Spn > 2$, Path2 ($Threshold\_en$ propagation path) begins to dominate due to the overwhelming contribution of the $(Spn-2)$ term in (11).

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 MinSum Split-2 Threshold methods, the interconnect delay is largely dominant. This is caused by the long global wires between large numbers 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 values of interconnect delays caused by buffers and wires (intrinsic gate delay and RC delay) in Path1 are 12.4 and 5.1 ns, which are 73% and 50% of their total delays, respectively.

C. Area Analysis

Fig. 14 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 numbers 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

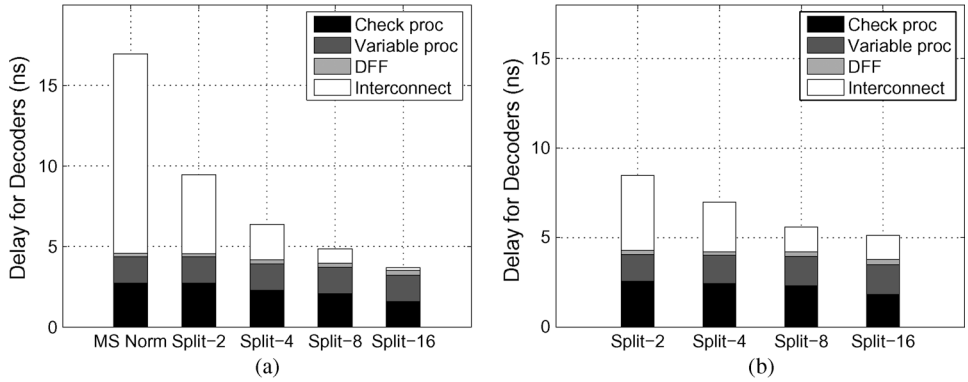

Fig. 13. Postroute delay breakdown of major components in the critical paths of five decoders using MinSum Normalized and MinSum Split-Row Threshold methods. (a) Path1: Check to variable processor delay. (b) Path2: $Threshold\_en$ propagation delay.

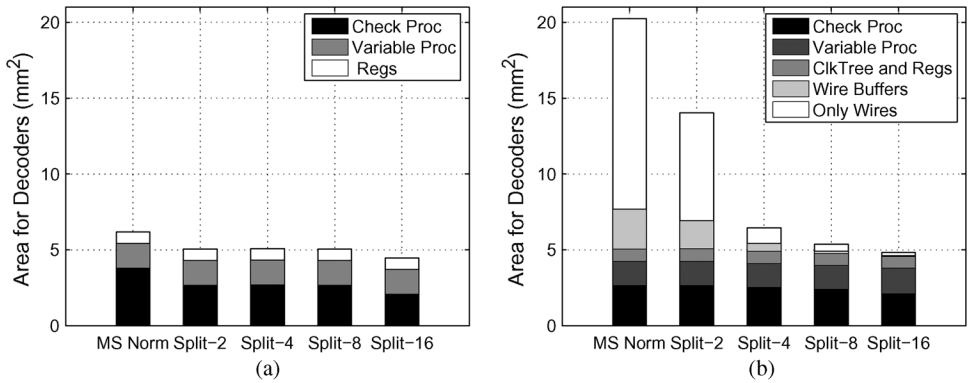

Fig. 14. Area breakdown for five decoders using MinSum Normalized and MinSum Split-Row Threshold methods. The interconnect and wire buffers are added after layout, which take a large portion of MinSum Normalized and MinSum Split-2 Threshold decoders. (a) Postsynthesis. (b) Postlayout.

the gates. This results in a lower logic (i.e., silicon/transistor) utilization and a larger overall area. As an additional illustration, Fig. 14 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 areas are without standard cells and are required for wiring.

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

D. 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. 2(a)] can follow capacitance versus the number of partitions, then, for a given Spn, the normalized capacitance is $C_{Spn}/C_{Spn=1} = 1/\sqrt{Spn}$. For $Spn \rightarrow \infty$, energy efficiency will be limited to the algorithm and architecture—not the routing congestion in layout.

The average power for Split-16 Threshold decoder is 0.70 times that of MinSum decoder. Interestingly, the operating frequency of Split-16 decoder is 3.3 times that of MinSum decoder; thus, if we simplify by assuming equal activities 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 that core logic gates have also been unchanged)—see Fig. 15.

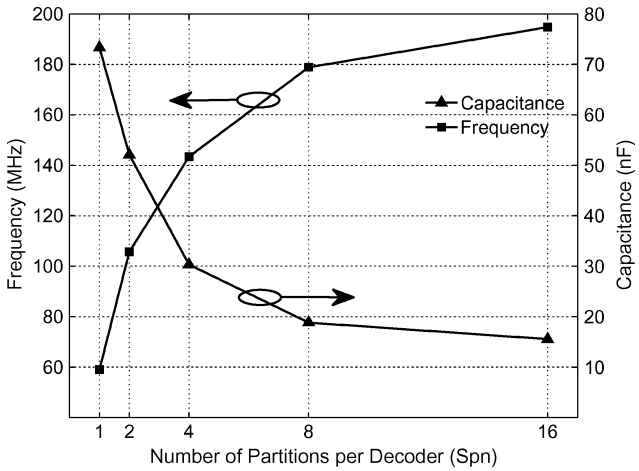

Fig. 15. Capacitance and maximum clock frequency versus the number of partitions Spn.

Additional savings to average power and energy (along with increased throughput) can be achieved through early termination, as mentioned in Section II-D. 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 these XOR signals as inputs [5]. Postlayout results show that the early termination block for a (2048, 1723) code occupies only approximately 0.1 mm^2.

Fig. 16(a) shows the average convergence iterations for MinSum Normalized and Split-Row Threshold decoders for

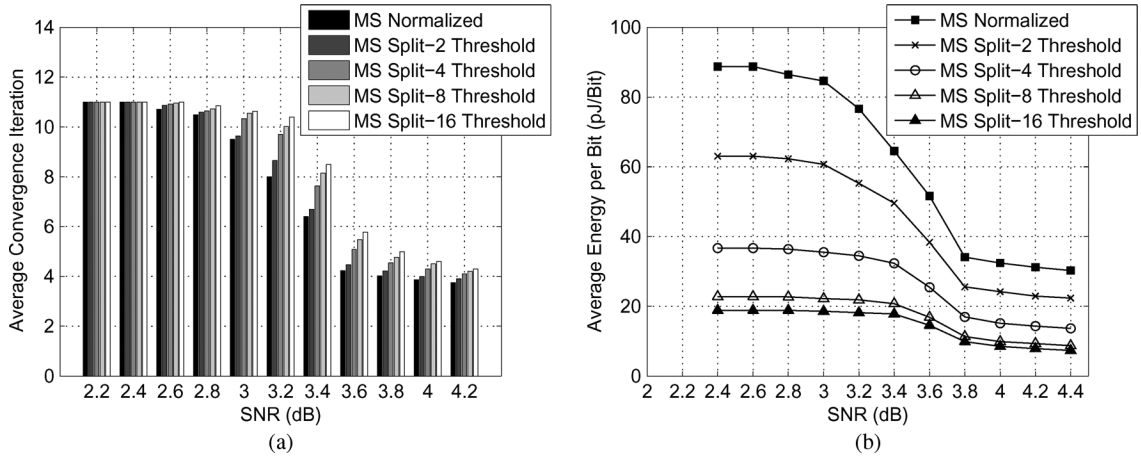

Fig. 16. Average convergence iteration and energy dissipation versus a large number of SNR values for five decoders using MinSum Normalized and MinSum Split-Row Threshold methods. (a) Average convergence iteration. (b) Average energy dissipation.

TABLE III
COMPARISON OF FULLY PARALLEL DECODERS IN 65-nm 1.3-V CMOS, FOR A (6, 32) (2048, 1723) CODE IMPLEMENTED USING MINSUM NORMALIZED AND MINSUM SPLIT-ROW THRESHOLD METHODS WITH DIFFERENT LEVELS OF SPLITTING. MAXIMUM NUMBER OF ITERATIONS IS $I_{\max} = 11$. †THE BER AND SNR VALUES ARE FOR 5-bit FIXED-POINT IMPLEMENTATIONS

	Normalized MinSum	Split-2 Threshold MinSum	Split-4 Threshold MinSum	Split-8 Threshold MinSum	Split-16 Threshold MinSum
CMOS fabrication process			65 nm CMOS, 1.3 V		
Initial area utilization (before clock tree synthesis)	25%	35%	75%	88%	**94%**
Final area utilization	38%	51%	85%	92%	**97%**
Avg. wire length per sub-block (μm)	93	71	44	28	**21**
Core area (mm^2)	20	14.05	6.45	5.38	**4.84**
Maximum clock frequency (MHz)	59	106	143	179	**195**
Average Power @Worst case freq (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
Average No. of 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**

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 and 3.8 dB, the average convergence iteration of MinSum Normalized decoder is about 30% to 8% less than the Split-Row Threshold decoder. For large SNRs ($SNR \geq 3.8$ dB), the difference between the numbers of iterations for decoders ranges from 18% to 1%, indicating that all decoders can converge almost within the same average number of iterations. Fig. 16(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.

E. Summary and Further Comparisons

Table III summarizes the postlayout implementation results for the decoders.

The Split-16 decoder's final logic utilization is 97%, which is 2.6 times higher than the MinSum Normalized decoder. The average wire length in each subblock of Split-16 decoder is 21 μm, which is 4.4 times shorter than in the MinSum Normalized decoder. It occupies 4.84 mm^2, runs at 195 MHz, delivers 36.3-Gbits/s throughput, and dissipates 37 pJ/bit with 11 iterations.

Compared to MinSum Normalized decoder, MinSum Split-16 decoder 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 decoder is 1.15 times larger than MinSum Normalized decoder, and it has a coding loss of 0.23 dB compared to MinSum Normalized decoder. At this BER point, its average decoding throughput is 92.8 Gbits/s, which is 2.9 times higher, and dissipates 15 pJ/bit, which is four 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-Gbits/s throughput required by the 10GBASE-T standard [10] (which requires 6.4 Gbits/s). Power dissipation is 62 mW at this operating point. These results are obtained by interpolating operating points using the measured data from a recently fabricated chip on the exact same process [33].

F. Comparison With Other Implementations

The MinSum Split-16 Threshold decoder postlayout simulation results are compared with recently implemented decoders

TABLE IV
COMPARISON OF THE SPLIT-16 THRESHOLD DECODER WITH THE PUBLISHED LDPC DECODER IMPLEMENTATIONS FOR THE 10GBASE-T CODE. ∗THROUGHPUT IS COMPUTED BASED ON THE MAXIMUM LATENCY REPORTED IN THIS PAPER

	Liu [28]	Zhang [6]		**This work**	
Decoding algorithm	Sum Product	MinSum with post-processing		Split-16 Threshold MinSum	
CMOS fabrication process (min. feature size)	90 nm, 8M	65 nm, 7M		65 nm, 7M	
Level of parallelism	partial-parallel	partial-parallel		fully-parallel	
Bits per message	5	4		5	
Logic utilization	50%	80%		97%	
Total chip area (mm^2)	14.5	5.35		4.84	
Maximum decoding 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 early termination (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 early termination (pJ/bit)	-	58.7	21.5	**15**	**3.7**

[6], [28] for the 10GBASE-T code and are summarized in Table IV. Results for two supply voltages are reported for the Split-16 decoder: Nominal 1.3 and 0.7 V, which is the minimum voltages that can achieve the 6.5 Gbits/s throughput required by the 10GBASE-T standard.

The partial parallel decoder chip [6] is fabricated in 65-nm CMOS and consists of a two-step decoder: MinSum algorithm and a postprocessing scheme which lowers the error floor down to $BER = 10^{-14}$. The Sliced Message Passing (SMP) scheme in [28] is proposed for Sum Product algorithm, divides the check node processing into equal-size blocks, and performs the check node computation sequentially. The postlayout simulations for a partial parallel decoder are shown in the table.

Compared to the two-step decoder chip [6], 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 SMP decoder [28], Split-16 decoder is about three times smaller and has 6.8 times higher throughput with 0.2-dB coding gain reduction.

VII. CONCLUSION

This paper has given a complete, detailed, and unified presentation of a low-complexity message-passing algorithm, called Split-Row Threshold, which utilizes a threshold-enable signal to compensate for the loss of $\min()$ interpartition information in Split-Row algorithm (original). It provides at least 0.3-dB error performance improvement over the Split-Row algorithm 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 fully parallel LDPC decoders for 10GBASE-T using MinSum Normalized and MinSum Split-Row Threshold methods in 65-nm CMOS have been presented.

Postlayout results show that, when compared with the MinSum Normalized decoder, Split-16 Threshold decoder has 2.6 times higher logic utilization, 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 0.23 dB with 11 iterations. At 1.3 V, it can attain up to 92.8 Gbits/s, and at 0.7 V, it can meet the necessary 6.4-Gbits/s throughput for 10GBASE-T while dissipating 62 mW. In comparison to other published LDPC chips, Split-16 decoder can be up to 3 times smaller with 6.8 times more throughput and 4.2 times lower energy consumption.

ACKNOWLEDGMENT

The authors would like to thank P. Urard for the key algorithmic contributions, Dr. S. Lin for the LDPC codes and assistance, and Dr. Z. Zhang.

REFERENCES

[1] D. J. C. MacKay, *Information Theory Inference and Learning Algorithms*, 3rd ed. Cambridge, U.K.: Cambridge Univ. Press, 2003.
[2] R. G. Gallager, "Low-density parity check codes," *IRE Trans. Inf. Theory*, vol. IT-8, no. 1, pp. 21–28, Jan. 1962.
[3] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.
[4] A. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate 1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.
[5] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Power reduction techniques for LDPC decoders," *IEEE J. Solid-State Circuits*, vol. 43, no. 8, pp. 1835–1845, Aug. 2008.
[6] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "A 47 Gb/s LDPC decoder with improved low error rate performance," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2009, pp. 286–287.
[7] T. T. S. I. Digital Video Broadcasting (DVB) Second Generation Framing Structure for Broadband Satellite Applications [Online]. Available: http://www.dvb.org
[8] *IEEE 802.16e. Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, IEEE p802.16e/d12 Draft, Oct. 2005.
[9] G.hn/G.9960. Next Generation Standard for Wired Home Network [Online]. Available: http://www.itu.int/ITU-T
[10] IEEE P802.3an, 10GBASE-T Task Force [Online]. Available: http://www.ieee802.org/3/an
[11] T. Mohsenin and B. Baas, "Trends and challenges in LDPC hardware decoders," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, Nov. 2009.
[12] D. J. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.
[13] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.
[14] T. Mohsenin and B. Baas, "Split-Row: A reduced complexity, high throughput LDPC decoder architecture," in *Proc. ICCD*, 2006, pp. 320–325.
[15] T. Mohsenin and B. Baas, "High-throughput LDPC decoders using a multiple split-row method," in *Proc. IEEE ICASSP*, Apr. 2007, vol. 2, pp. II-13–II-16.
[16] T. Mohsenin and B. Baas, "A split-decoding message passing algorithm for low density parity check codes," *J. Signal Process. Syst.*, 2010, to be published.

[17] T. Mohsenin, P. Urard, and B. Baas, "A thresholding algorithm for improved split-row decoding of LDPC codes," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, Nov. 2008, pp. 448–451.
[18] T. Mohsenin, D. Truong, and B. Baas, "An improved split-row threshold decoding algorithm for LDPC codes," in *Proc. IEEE ICC*, Jun. 2009, pp. 1–5.
[19] T. Mohsenin, D. Truong, and B. Baas, "Multi-split-row threshold decoding implementations for LDPC codes," in *Proc. ISCAS*, May 2009, pp. 2449–2452.
[20] S. Lin and D. J. Castello, Jr., *Error Control Coding*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2004.
[21] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Trans. Commun.*, vol. 50, no. 3, pp. 406–414, Mar. 2002.
[22] J. Chen, A. Dholakia, E. Eleftheriou, and M. Fossorier, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.
[23] R. Lynch, E. M. Kurtas, A. Kuznetsov, E. Yeo, and B. Nikolic, The search for a practical iterative detector for magnetic recording vol. 40, no. 1, pp. 213–218, Jan. 2004.
[24] E. Morifuji, D. Patil, M. Horowitz, and Y. Nishi, "Power optimization for SRAM and its scaling," *IEEE Trans. Electron Devices*, vol. 54, no. 4, pp. 715–722, Apr. 2007.
[25] M. Mansour and N. R. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, Mar. 2006.
[26] H. Zhang, J. Zhu, H. Shi, and D. Wang, "Layered approx-regular LDPC: Code construction and encoder/decoder design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 2, pp. 572–585, Mar. 2008.
[27] Y. Dai, N. Chen, and Z. Yan, "Memory efficient decoder architectures for quasi-cyclic LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 9, pp. 2898–2911, Oct. 2008.
[28] L. Liu and C.-J. R. Shi, "Sliced message passing: High throughput overlapped decoding of high-rate low density parity-check codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 11, pp. 3697–3710, Dec. 2008.
[29] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D.J. Costello, Jr., "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.
[30] M. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.
[31] P. Saxena, R. S. Shelar, and S. S. Sapatnekar, *Routing Congestion in VLSI Circuits*, 1st ed. New York: Springer-Verlag, 2007.
[32] I. Djurdjevic, Xu J., 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 Commun. Lett.*, vol. 7, no. 7, pp. 317–319, Jul. 2003.
[33] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, C. Watnik, A. T. Tran, Z. Xiao, E. W. Work, J. W. Webb, P. V. Mejia, and B. M. Baas, "A 167-processor computational platform in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.

Tinoosh Mohsenin received the B.S. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, and the M.S. degree in electrical and computer engineering from Rice University, Houston, TX. She is currently working toward the Ph.D. degree in electrical and computer engineering at the University of California, Davis.

She is the Designer of the Split-Row, multisplit, and Split-Row Threshold decoding algorithms and architectures for low-density parity-check (LDPC) codes. She was a Key Designer of the 167-processor Asynchronous Array of simple Processors chip. Her research interests include algorithms, architectures and VLSI design for high-performance and energy-efficient computation in the areas of networking and communications, digital signal processing, and error-correction applications.

Dean N. Truong received the B.S. degree in electrical and computer engineering from the University of California, Davis, in 2005, where he is currently working toward the Ph.D. degree.

He was a Key Designer of the second generation 167-processor 65-nm CMOS Asynchronous Array of simple Processors chip. His research interests include high-speed processor architectures, dynamic supply voltage and dynamic clock frequency algorithms and circuits, and VLSI design.

Bevan M. Baas received the B.S. degree in electronic engineering from California Polytechnic State University, San Luis Obispo, in 1987 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1990 and 1999, respectively.

From 1987 to 1989, he was with Hewlett-Packard, Cupertino, CA, where he participated in the development of the processor for a high-end minicomputer. In 1999, he joined Atheros Communications, Santa Clara, CA, as an early Employee and served as a Core Member of the team which developed the first IEEE 802.11a (54 Mbits/s; 5 GHz) Wi-Fi wireless LAN solution. In 2003, he joined the Department of Electrical and Computer Engineering, University of California, Davis, where he is currently an Associate Professor. He leads projects in architecture, hardware, software tools, and applications for VLSI computation with an emphasis on DSP workloads. Recent projects include the 36-processor Asynchronous Array of simple Processors chip, applications, and tools; a second-generation 167-processor chip; low-density parity-check decoders; fast Fourier transform processors; Viterbi decoders; and H.264 video codecs. He also serves as a Member of the Technical Advisory Board of an early stage technology company. During the summer of 2006, he was a Visiting Professor at the Circuit Research Laboratory, Intel Corporation.

Dr. Baas was a National Science Foundation fellow from 1990 to 1993 and a NASA Graduate Student Researcher Fellow from 1993 to 1996. He was a recipient of the National Science Foundation CAREER Award in 2006 and the Most Promising Engineer/Scientist Award by the American Indian Science and Engineering Society in 2006. Since 2007, he has been an Associate Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS. He has served as a member of the Technical Program Committees of the IEEE International Conference on Computer Design and the IEEE International Symposium on Asynchronous Circuits and Systems, and the Program Committee of the Hot Chips Symposium on High Performance Chips.