J Sign Process Syst (2010) 61:329–345
DOI 10.1007/s11265-010-0456-y

A Split-Decoding Message Passing Algorithm for Low Density Parity Check Decoders

Tinoosh Mohsenin · Bevan M. Baas

Received: 30 November 2009 / Revised: 25 January 2010 / Accepted: 25 January 2010 / Published online: 26 February 2010

Abstract A Split decoding algorithm is proposed which divides each row of the parity check matrix into two or multiple nearly-independent simplified partitions. The proposed method significantly reduces the wire interconnect and decoder complexity and therefore results in fast, small, and high energy efficiency circuits. Three full-parallel decoder chips for a (2048, 1723) LDPC code compliant with the 10GBASE-T standard using MinSum normalized, MinSum Split-2, and MinSum Split-4 methods are designed in 65 nm, seven metal layer CMOS. The Split-4 decoder occupies 6.1 mm^2, operates at 146 MHz, delivers 19.9 Gbps throughput, with 15 decoding iterations. At 0.79 V, it operates at 47 MHz, delivers 6.4 Gbps and dissipates 226 mW. Compared to MinSum normalized, the Split-4 decoder chip is 3.3 times smaller, has a clock rate and throughput 2.5 times higher, is 2.5 times more energy efficient, and has an error performance degradation of 0.55 dB with 15 iterations.

Keywords Low density parity check · LDPC · Iterative decoder · Split-Row · CMOS · 65 nm · 10GBASE-T · VLSI

1 Introduction

Low density parity check (LDPC) codes first introduced by Gallager [1] have recently received significant attention due to their error correction performance near the Shannon limit and their inherently parallelizable decoder architectures. Many recent communication standards such as 10 Gigabit Ethernet (10GBASE-T) [2], digital video broadcasting (DVB-S2) [3], and WiMAX (IEEE 802.16e) [4] have adopted LDPC codes. Implementing high throughput and energy efficient LDPC decoders remains a challenge largely due to the high interconnect complexity and high memory bandwidth requirements of existing decoding algorithms stemming from the irregular and global communication inherent in the codes.

This paper overviews Split-Row and the more general Multi-Split, two reduced complexity decoding methods which partition each row of the parity check matrix into two or multiple nearly-independent simplified partitions. These two methods reduce the wire interconnect complexity between row and column processors, and simplify row processors leading to an overall smaller, faster, and more energy efficient decoder. Full-parallel decoders, which are not efficient to build due to their high routing congesting and large circuit area, take the greatest benefit of Split decoding method. In this paper, we present the first complete overview of the Split decoding algorithm, architecture and VLSI implementation.

The paper is organized as follows: Section 2 reviews LDPC codes and the message passing algorithm. In Section 3, LDPC decoder architectures are explained. Sections 4 and 5 introduce Split-Row and Multi-Split decoding methods, respectively, for regular permutation-based LDPC codes. The error performance comparisons for different codes with the multiple splitting method are shown in Section 6. The mapping architecture of the multiple splitting method is presented in Section 7. In Section 8 the results of three

T. Mohsenin (✉) · B. M. Baas
Department of Electrical and Computer Engineering, University of California, Davis 95616 Davis CA, USA
e-mail: tmohsenin@ucdavis.edu

full-parallel decoders implemented with the proposed and standard decoding techniques are presented and compared.

2 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 [5]. 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. 1. The rank of the matrix is 5, therefore the information length K is 7. In the graph, there are two sets of nodes in the Tanner graph: 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.

$$H = \begin{array}{cccccccccccc|c}
V1 & V2 & V3 & V4 & V5 & V6 & V7 & V8 & V9 & V10 & V11 & V12 & \\
\mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & C1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & C2 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & C3 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & C4 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & C5 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & C6
\end{array}$$

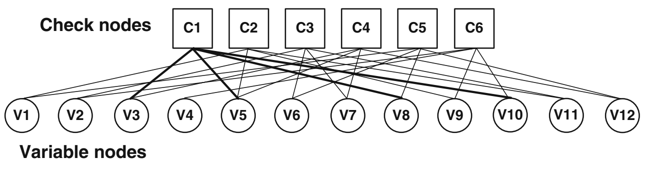


Figure 1 Parity check matrix (*upper*) and Tanner graph (*lower*) representation of a 12 column (N), six row (M), column weight 2 (W_c), row weight 4 (W_r), LDPC code with information length 7 (K).

Total number of edges (connections) between variable node and check nodes is $N \times W_c$ or $M \times W_r$.

LDPC codes are commonly decoded by an iterative message passing algorithm which consists of two sequential operations: row processing or check node update and column processing or variable node update. In row processing, all check nodes receive messages from neighboring variable nodes, perform parity check operations and send the results back to neighboring variable nodes. The variable nodes update soft information associated with the decoded bits using information from check nodes, then send the updates back to the check nodes, and this process continues iteratively.

Sum-Product (SPA) [6] and MinSum (MS) [7] are widely-used decoding algorithms which we refer to as *standard decoders* in this paper. The following subsections describe these two algorithms in detail.

2.1 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. 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,

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

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

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

$$\alpha_{ijSPA} = \prod_{j' \in V(i)\backslash j} sign(\beta_{ij'}) \times \phi\left(\sum_{j' \in V(i)\backslash j} \phi(|\beta_{ij'}|)\right) \quad (2)$$

where the non-linear function $\phi(x) = -\log\left(\tanh\frac{|x|}{2}\right)$. The first product term in Eq. 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.

$$\beta_{ij} = \lambda_j + \sum_{i' \in C(j)\backslash i} \alpha_{i'j} \quad (3)$$

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

$$z_j = \lambda_j + \sum_{i' \in C(j)} \alpha_{i'j} \quad (4)$$

From the updated vector, an estimated code vector $\hat{X} = \{\hat{x}_1, \hat{x}_2, ..., \hat{x}_N\}$ is calculated by:

$$\hat{x}_i = \begin{cases} 1, & \text{if } z_i \leq 0 \\ 0, & \text{if } z_i > 0 \end{cases} \quad (5)$$

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. 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.

A block diagram of a standard serial decoder is shown in Fig. 2 where row and column processors share a memory to store α and β messages. For simplicity the initialization and syndrome check stages are not shown.

2.2 MinSum Algorithm (MS)

The magnitude part of check node update in SPA decoding can be simplified by approximating the magnitude computation in the row processing step (Eq. 2), with a minimum function. This algorithm is called

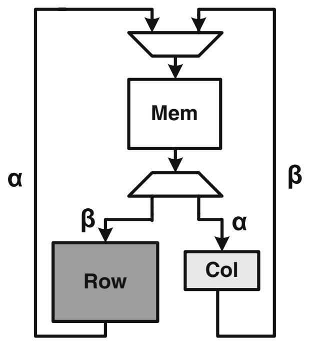


Figure 2 Standard two-phase decoder block diagram.

MinSum (MS) [7, 8], and the row processing output is calculated by:

$$\alpha_{ijMinSum} = \prod_{j' \in V(i)\backslash j} sign(\beta_{ij'}) \times \min_{j' \in V(i)\backslash j} (|\beta_{ij'}|) \quad (6)$$

All other steps are the same as in SPA decoding. The error performance loss of MS decoding can be improved by normalizing the row processor outputs (α) in Eq. 6 with a correction factor $S \leq 1$ [9, 10], resulting in the MinSum normalized algorithm.

$$\alpha_{ij} = S \times \prod_{j' \in V(i)\backslash j} sign(\beta_{ij'}) \times \min_{j' \in V(i)\backslash j} (|\beta_{ij'}|) \quad (7)$$

3 LDPC Decoding Architectures

The message passing algorithm is inherently parallel because row processing operations are fully independent with respect to each other, and the same is true for column processing operations.

We define a metric called the parallelism fraction (*pfraction*) which indicates the extent to which a hardware implementation makes use of the available parallelism as,

$$pfraction = \frac{numprocs}{M + N} \quad (8)$$

where *numprocs* is the sum of the number of row and column processors in a particular decoder. Based on the value of *pfraction,* LDPC decoders may be classified into the three styles described in the following sections.

3.1 Serial Decoders

Serial decoders process one word at a time by using one row and one column processor. Although they have minimal hardware requirements, they also have a large decoding latency and low throughput. A 4,096-bit serial LDPC convolutional decoder [11] is implemented on an Altera Stratix FPGA with $pfraction = \left(\frac{3}{4{,}096+2{,}048}\right) = 0.00049$. The decoder utilizes only 4K logic elements and 776 Kbit memory, runs at 180 MHz, and delivers 9 Mbps throughput.

3.2 Partial-Parallel Decoders

Partial-parallel decoders [12–18] contain multiple processing units and shared memories. A major challenge is efficiently handling simultaneous memory accesses into the shared memories. Following are details of ten partial-parallel decoders containing 3–2,112 processors with *pfraction* from 0.001–0.87.

Two 2,048-bit partial-parallel decoders compliant with 10GBASE-T standard are designed with a high parallelism: The first one is a 47 Gbps decoder chip designed with 2,048 column processors and 64 row processors. It has a $pfraction = \left(\frac{2{,}048+64}{2{,}048+384}\right) = 0.87$ and occupies 5.35 mm^2 in 65 nm CMOS. The second decoder is designed using a reduced routing complexity decoding method, called Sliced-Message Passing. It utilizes 512 column processor, 65 row processors, has a $pfraction = \left(\frac{512+384}{2{,}048+384}\right) = 0.37$, occupies 14.5 mm^2 and delivers 5.3 Gbps in 90 nm.

A multi-rate 2,048-bit programmable partial-parallel decoder chip [19] has a $pfraction = \left(\frac{64}{2{,}048+1{,}024}\right) = 0.02$, utilizes about 50 Kbit memory, occupies 14.3 mm^2 and delivers 640 Mbps in 0.18 μm technology. An FPGA implementation of a 8,176-bit decoder [20] has a $pfraction = \left(\frac{36}{8{,}176+1{,}024}\right) = 0.004$ and achieves source decoding of 172 Mbps. A 1,536-bit memory-bank based decoder [13] utilizes about 540 Kbit memory and has $pfraction = \left(\frac{3}{1{,}536+768}\right) = 0.001$. A Virtex-II FPGA implementation of the decoder runs at 125 MHz and delivers 98.7 Mbps. A 64,800-bit DVB-S2 decoder chip in 65 nm CMOS utilizes 180 processors and 3.1 Mb of memory, attains a throughput of 135 Mbps [21], occupies 6.07 mm^2, handles 21 different codes, and thus its *pfraction* ranges from 0.01–0.001. A 600-bit LDPC-COFDM chip [22] employs 50 row processors and 150 column processors, has $pfraction = \left(\frac{200}{600+450}\right) = 0.19$, delivers 480 Mbps, and occupies 21.45 mm^2 in 0.18 μm CMOS. A 6,912-bit decoder [23] implemented on a Virtex-4 FPGA utilizes 64 processors with 46 Block-RAMs, runs at 181 MHz, has 3.86–4.31 Gbps throughput, and has *pfraction* from 0.005–0.007. A 32-processor 32-memory decoder [24] supports both IEEE 802.11n and IEEE 802.16e codes, occupies 3.88 mm^2, delivers 31.2–64.4 Mbps in 0.13 μm technology, and has *pfraction* of 0.003–0.1. A multi-rate, multi-length decoder [25] has 18 processors, *pfraction* 0.005–0.01, and runs at 100 MHz and delivers 60 Mbps on a Virtex-II FPGA.

3.3 Full-Parallel Decoders

Full-parallel decoders directly map each node in the Tanner graph to a different processing unit and thus $pfraction = 1$. They provide the highest throughputs and require no memory to store intermediate messages. The greatest challenges in their implementation are large circuit area and routing congestion which are caused by the large number of processing units and the very large number of wires between them.

A 1,024-bit full-parallel decoder chip [26] occupies 52.5 mm^2, runs at 64 MHz and delivers 1 Gbps throughput in 0.16 μm technology. Two full-parallel decoders designed for 1,536-bit and 2,048-bit LDPC codes [27, 28] occupy 16.8 and 43.9 mm^2 and deliver 5.4 and 7.1 Gbps, respectively in 0.18 μm technology. A 660-bit decoder chip [29] occupies 9 mm^2, runs at 300 MHz and obtains 3.3 Gbps throughput in 0.13 μm technology. A full-parallel decoder designed for a family of codes with different rates and code lengths up to 1,024 bits, attains 2.4 Gbps decoding throughput [30].

Previous studies for reducing wire interconnect complexity are based on reformulating the message passing algorithm. The SPA decoder can be reformulated so that instead of sending different α values, each check node sends only the summation value in Eq. 2 to its variable nodes. Then the α messages are recovered by post-processing in the variable nodes. This results in a 26% reduction in total global wire length [31]. More reformulation was performed so that both check nodes and variable nodes send the summation values in Eqs. 2 and 3, respectively to each other [32]. MinSum was reformulated so that the check node sends only the minimum values to its variable nodes which results in 90% outgoing wire reduction from check nodes [33]. These architectures require more processing in row and column processors with additional storage units to recover α and β messages and therefore unfortunately result in larger decoder areas.

4 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, row processing is partitioned into two blocks, where the row 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

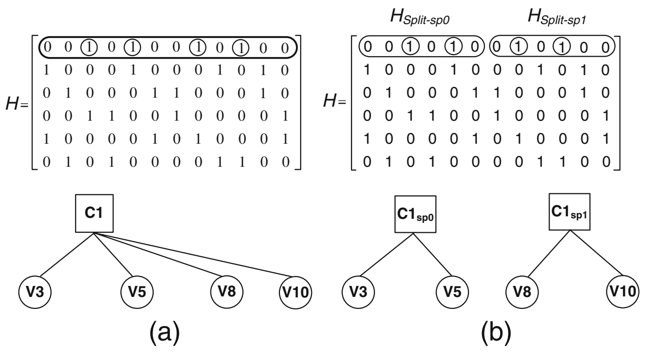


Figure 3 The parity check matrix example highlighting the first row processing step using **a** standard decoding (SPA or MinSum) and **b** Split-Row decoding. The check node C_1 and its connected variable nodes are shown for each method.

decoding where row processing requires the passing of all row processor input data across the entire row of the parity check matrix. As an illustration, Fig. 3a shows a parity check matrix highlighting the processing of the first row 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. 3b where the row processing is divided into two parallel row 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 row 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. 4.

This architecture has two major benefits: 1) it decreases the number of inputs and outputs per row processor, resulting in many fewer wires between row and column processors, and 2) it makes each row 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 row processing outputs, and that the error can be largely compensated with a correction factor.

4.1 SPA Split

From a mathematical point of view, all steps are similar to the SPA decoder except the row processing step. In each half of the Split-Row decoder's row 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 row 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. 3b 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 using half of the messages yields:

$$\alpha_{ijSPASplit} = \prod_{j' \in V(i)\backslash j} \text{sign}(\beta_{ij'}) \times \phi\left(\sum_{j' \in V_{Split}(i)\backslash j} \phi(|\beta_{ij'}|)\right) \tag{9}$$

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:

$$\sum_{j' \in V_{Split}(i)\backslash j} \phi(|\beta_{ij'}|) \leq \sum_{j' \in V(i)\backslash j} \phi(|\beta_{ij'}|) \tag{10}$$

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

$$\phi\left(\sum_{j' \in V_{Split}(i)\backslash j} \phi(|\beta_{ij'}|)\right) \geq \phi\left(\sum_{j' \in V(i)\backslash j} \phi(|\beta_{ij'}|)\right) \tag{11}$$

And we obtain:

$$|\alpha_{ijSPASplit}| \geq |\alpha_{ijSPA}| \tag{12}$$

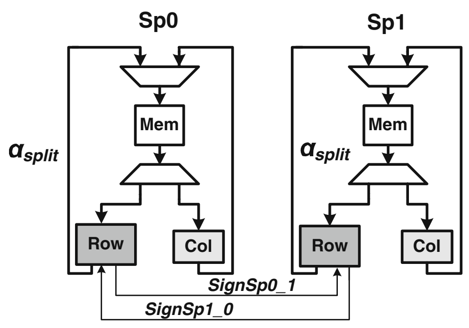


Figure 4 Block diagram of the proposed Split-Row decoder.

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

$$\alpha_{ijSPASplit} = S \times \prod_{j' \in V(i)\backslash j} \text{sign}(\beta_{ij'}) \times \phi\left(\sum_{j' \in V_{Split}(i)\backslash j} \phi(|\beta_{ij'}|)\right) \tag{13}$$

4.2 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.

$$\alpha_{ijMinSumSplit} = \prod_{j' \in V(i)\backslash j} \text{sign}(\beta_{ij'}) \times \min_{j' \in V_{split}(i)\backslash j} (|\beta_{ij'}|) \tag{14}$$

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:

$$|\alpha_{ijMinSumSplit}| \geq |\alpha_{ijMinSum}|. \tag{15}$$

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

$$\alpha_{ijMinSumSplit} = S \times \prod_{j' \in V(i)\backslash j} \text{sign}(\beta_{ij'}) \times \min_{j' \in V_{split}(i)\backslash j} (|\beta_{ij'}|) \tag{16}$$

The correction factor S is relatively easily obtained empirically through simulations and examples are given in Section 6.

5 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 row processing operation is shown in Fig. 5. We denote each partition of the parity check H which is divided into Spn partitions column wise by $H_{Split\text{-}Spk}$, $k = 0, ..., n$. As shown in the figure, in row 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

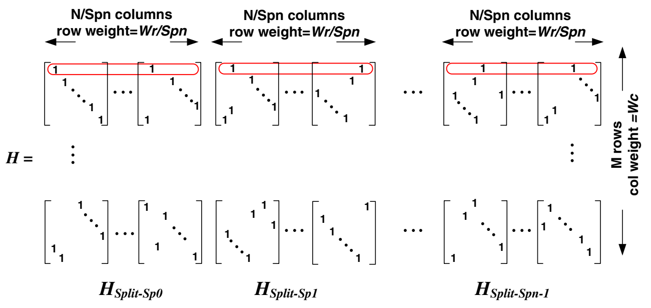


Figure 5 The parity check matrix of a (W_c, W_r) (N, K) permutation-based LDPC code highlighting the first row processing operation with *Spn*-way splitting (Multi-Split) method.

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 row 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 S less than one. Modifying Eq. 2 using the messages in each partition yields:

$$\alpha_{ijSPASplit\text{-}Spk} = S \times \prod_{j' \in V(i)\backslash j} \text{sign}(\beta_{ij'}) \times \phi\left(\sum_{j' \in V_{Split\text{-}Spk(i)}\backslash j} \phi(|\beta_{ij'}|)\right) \tag{17}$$

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

Similarly, in MinSum Multi-Split, row processor outputs are normalized with a correction factor S:

$$\alpha_{ijMinSumSplit\text{-}Spk} = S \times \prod_{j' \in V(i)\backslash j} \text{sign}(\beta_{ij'}) \times \min_{j' \in V_{Split\text{-}Spk}(i)\backslash j} (|\beta_{ij'}|) \tag{18}$$

Figure 6 shows how the sign bits are calculated according to Eq. 17 or Eq. 18. 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 half shows the sign logic inside each row 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.

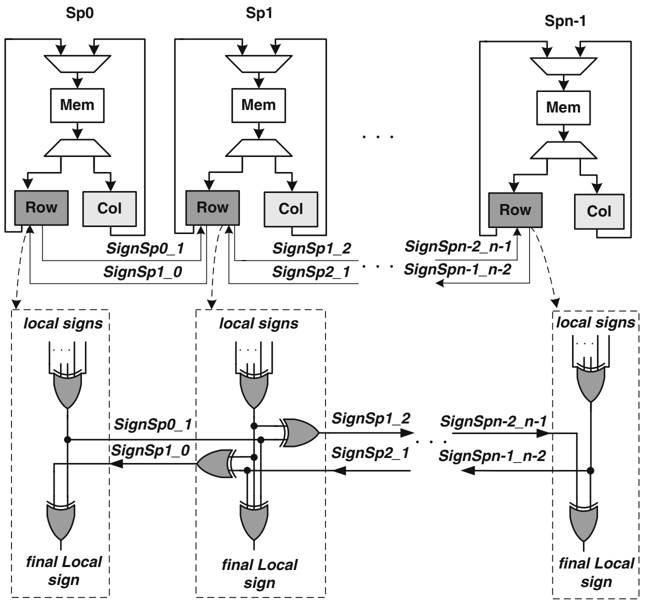


Figure 6 Multi-Split decoder with *Spn*-way splitting method, highlighting inter-partition *sign* wires and the simplified logic for implementation of the *sign* bit in each row processor.

6 Correction Factor and Error Performance Simulation Results

6.1 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 [34]. 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 row 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. 10 decreases in each partition and since $\phi(x)$ is a decreasing function, the summation on the left side of Eq. 11 becomes larger which results in larger magnitude row 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 row processor output magnitude gets larger as the number of partitions increases and a smaller correction factor is required to normalize the row processing outputs in each partition.

Achieving an absolute minimum error performance would require a different correction factor for each row processor output—but this is impractical because it would require knowledge of unavailable information such as row 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 7 plots the error performance of a (6,32) (2048, 1723) RS-based LDPC code [35] 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 for

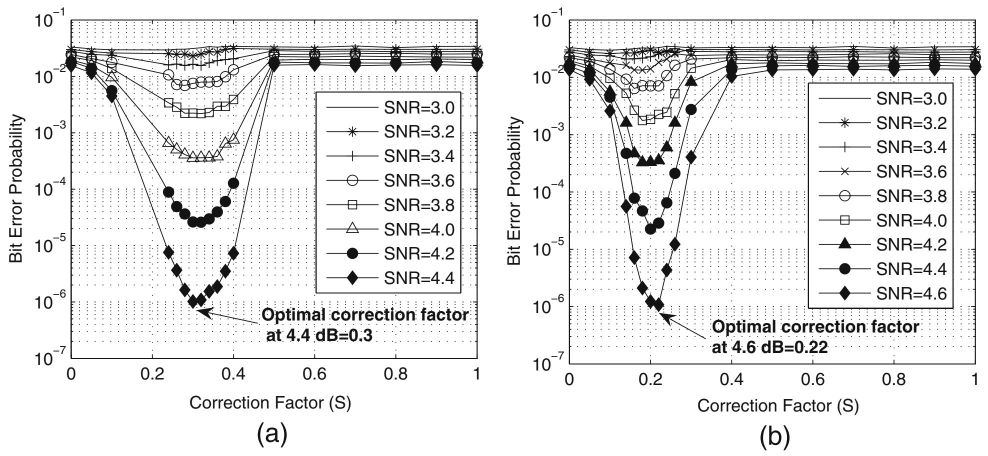


Figure 7 Determination of correction factor for a (6,32) (2048, 1723) RS-based LDPC code using **a** MinSum Split-2 and **b** MinSum Split-4 decoders. The optimal correction factor variations with the SNR values are very small with the average value of 0.3 for Split-2 and 0.19 of Split-4.

Table 1 Average optimal correction factor S for different constructed regular codes.

(N, K)	(W_c, W_r)	Average optimal correction factor S				
		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

+ indicates that the row weight of the code is not evenly divisible by that level of splitting, – indicates that the row weight for that level of splitting is very small and error performance loss is therefore significant (≥0.7 dB).

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. 7.

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 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.

6.2 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 8 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

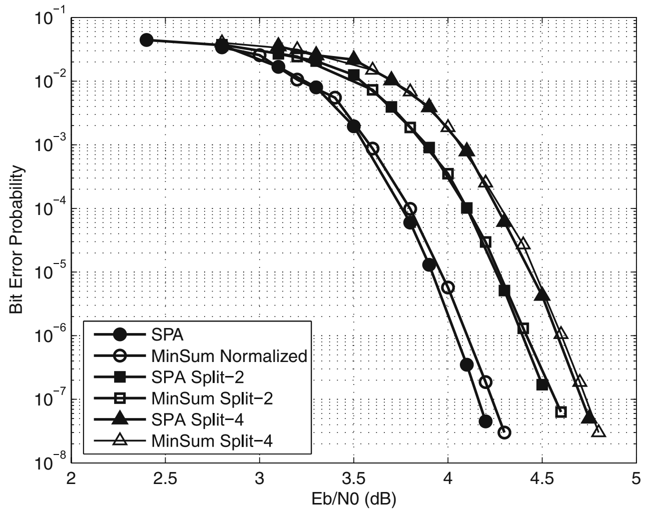


Figure 8 BER performance of the (6, 32) (2048, 1723) code using the Multi-Split method in SPA and MinSum decoders with optimal correction factors.

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 9 shows the error performance of the (4,32) (8176, 7156) Euclidean geometry-based QC-LDPC code [36] using standard and Split-2 decoding in SPA and MinSum algorithms. Also shown are reduced complexity hard decision algorithms: Bit-Flipping (BF) [1], weighted Bit-Flipping (WBF) [37], and improved WBF [38]. Split-2 performs approximately 0.5 dB away

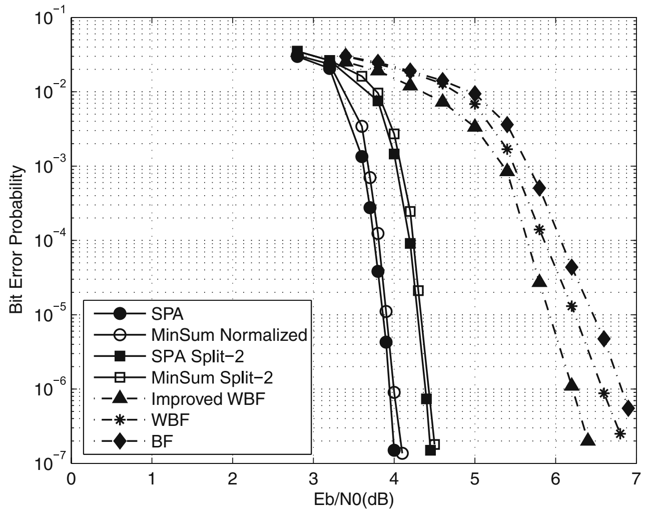


Figure 9 Error performance comparison with different decoding algorithms for a (4, 32) (8176, 7156) QC-LDPC code.

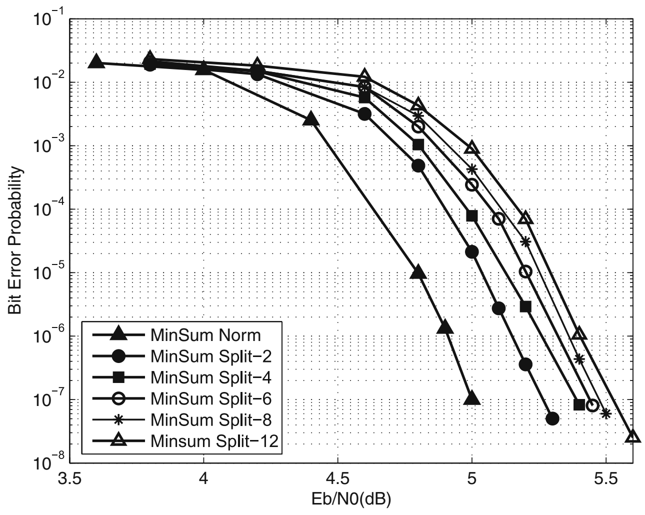


Figure 10 BER performance of a (6, 72) (5256, 4823) QC-LDPC code using various MinSum decoders with different levels of splitting and near-optimal correction factors.

from the SPA and MS decoders and attains around 1.7 dB gain 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. 10 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 $= 10^{-7}$.

7 Full-Parallel MinSum Multi-Split Decoders

Figure 11 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 row and column processing method. The

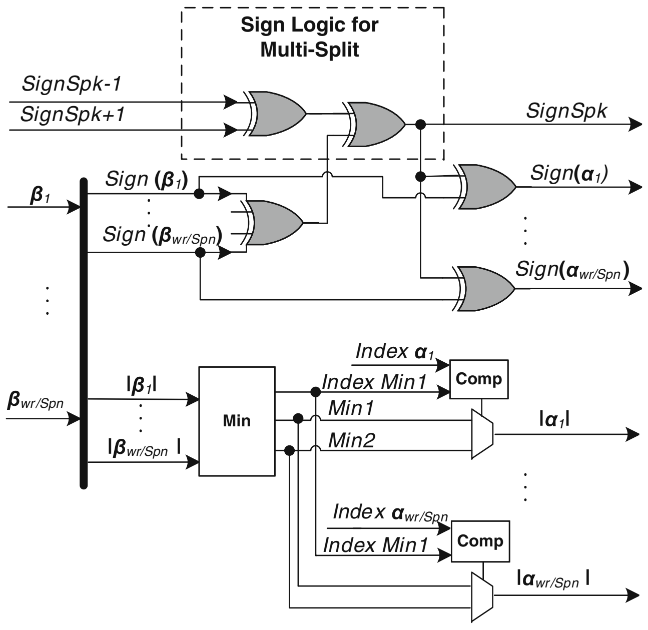
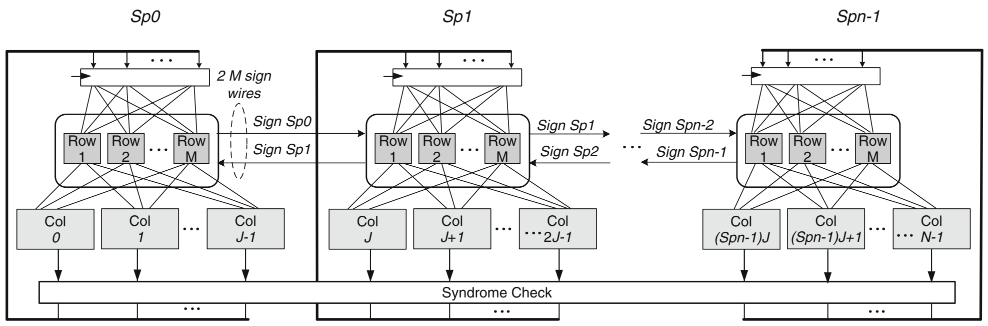


Figure 12 Row processor block diagram of MinSum Multi-Split for partition Spk, with sign logic on top and magnitude calculation of α at the bottom.

pipelined decoder is partitioned into Spn sub-blocks where the row processors are interconnected by $2M$ sign wires (each row processor sends its sign bit and receives a sign bit from the row processor in the neighboring partition). Each partition consists of M row processors and N/Spn column processors. The α messages from each row processor are routed to column processors according to the parity check matrix structure. Similarly, β messages are sent to corresponding row processors after syndrome check and after being synchronized by the global clock signal.

The row processor block diagram for the MinSum Multi-Split decoder in partition Spk is shown in Fig. 12. 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

Figure 11 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.



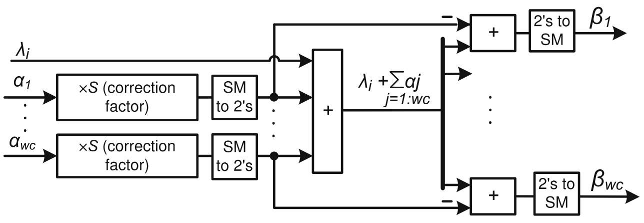


Figure 13 Column processing unit block diagram.

Table 2 Summary of the key parameters of the implemented (6,32) (2048, 1723) 10GBASE-T LDPC code.

Code length, No. of columns (N)	2,048
Information length (K)	1,723
Parity check equations, No. of rows (M)	384
Row weight (W_r)	32
Column weight (W_c)	6
Size of permutations	64

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*α_i), otherwise it selects *Min1* [39].

Figure 13 shows the block diagram of the column processor. Similar to MinSum normalized, the messages from row processors (check nodes) are multiplied by the correction factor S. 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 column 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 column processing equation Eq. 3. β messages are then converted to sign-magnitude for use in row processing in the next iteration.

8 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 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 2 summarizes the code parameters that specify all three decoder implementations.

The full-parallel decoder maps each variable node to one column processor and each check node to one row processor. Figure 14 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 2,048 column processors corresponding to the parity

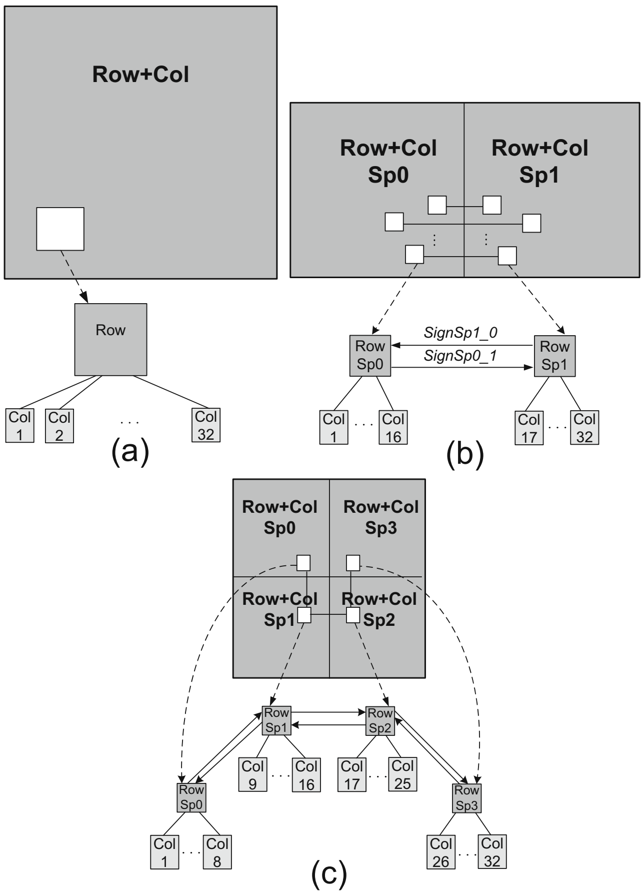


Figure 14 Mapping a full-parallel decoder with **a** MinSum normalized **b** Split-2 and **c** Split-4 decoding methods for the (6, 32) (2,048, 1,723) code.

check matrix dimensions M and N, respectively. As seen in Fig. 3 and further described in Section 7, 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 1,024 column processors. For Split-4 there are only 512 column 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.

8.1 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 15 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.

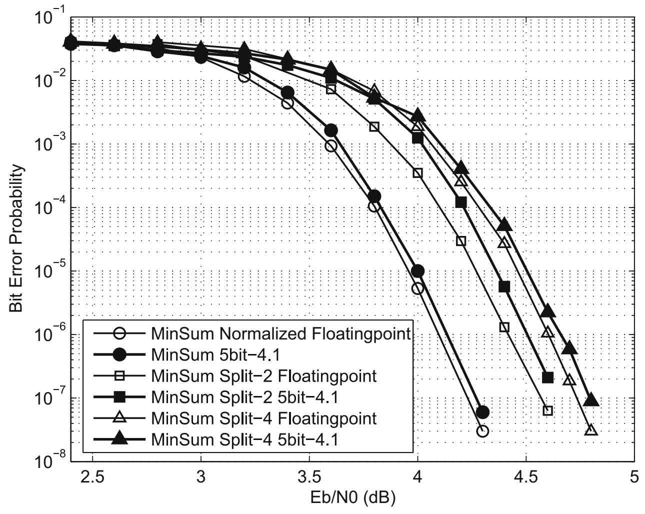


Figure 15 Final layout of **a** MinSum normalized, **b** MinSum Split-2 and **c** MinSum Split-4 decoder chips, shown approximately to scale.

Although there have been several studies on the quantization effects in LDPC decoders [10, 40], 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 column and row processors. For M row processors, the total number of word busses that pass α messages is $M \times W_r$, while N column 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 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 [26].

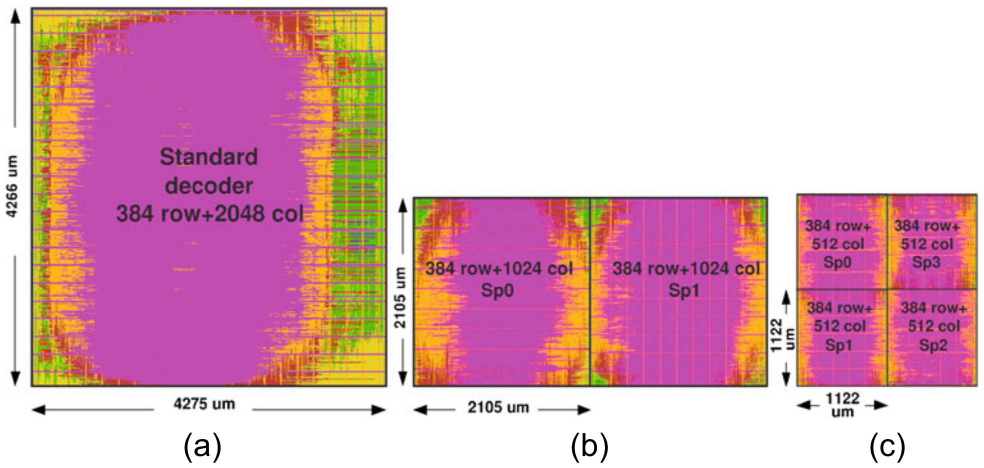
On the other hand, using wider fixed-point words improves the error performance. BER simulations show an approximate 0.07–0.09 dB improvement in all three decoders when using 6-bit words (4.2) instead of 5-bit words (4.1). To achieve this improved performance for MinSum normalized with one additional 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 column 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.

8.2 Area, Throughput and Power Comparison

Figure 16 shows the final chip layouts of the (a) MinSum normalized, (b) MinSum Split-2 and (c) MinSum Split-4 decoders, and Table 3 summarizes their post-layout results. In Split-2 and Split-4, although the number of row processors increases with higher splitting levels (they are replicated Spn times), Fig. 16 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.

Figure 16 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.



Since Split-Row reduces row processor area and eliminates significant communication between row and column 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 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 column processor interconnections force the place and route tool to spread standard cells apart to provide sufficient space for routing.

As an additional illustration, Table 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 six times smaller than that of MinSum. Thus, Split-4's speedup over MinSum normalized is due in part to its simplified row processing, but the major contributor is the significant reduction in column/row 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

Table 3 Comparison of the three full-parallel decoders implemented in 65 nm CMOS for a (6, 32) (2048, 1723) code.

	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 (μm)	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 row processors	13.2%	19.2%	41.3%
% area for column 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)	1,941	2,179	1,889
Throughput @$Imax = 15$ (Gbps)	8.1	15.0	**19.9**
Energy per bit @$Imax = 15$ (pJ/bit)	241	145	**95**
Average iterations @ BER = 3×10^{-5}, $Imax = 15$ ($Iavg$)	3.8	4.8	4.9
Throughput @$Iavg$ (Gbps)	31.8	46.9	**61.0**
Energy per bit @$Iavg$ (pJ/bit)	61	46	**31**

All area values are for final placed and routed layout. Maximum number of iterations $Imax = 15$.

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 two times when considering the cost of the wafer and die yield [41]. 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 [2]. 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 [42].

8.3 Wire Statistics

Figure 17 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.

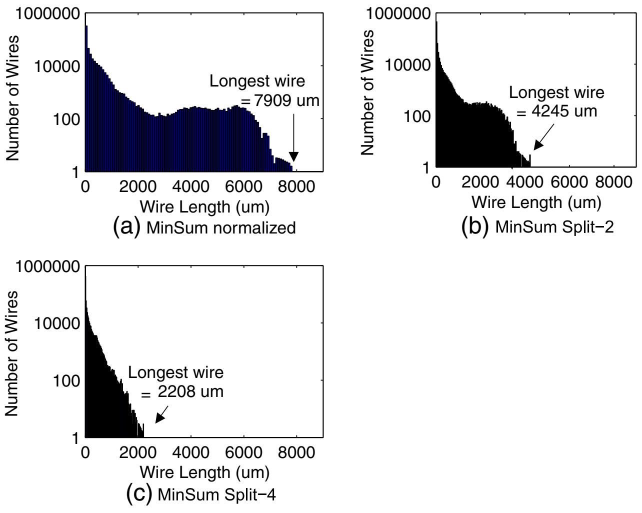


Figure 17 Wire length distribution for **a** MinSum normalized, **b** MinSum Split-2 and **c** MinSum Split-4 decoders.

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 energy efficient circuits.

8.4 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 [14, 43] which checks the decoded bits every cycle (see Fig. 11) 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 of the decoder.

Figure 18 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 dissipations 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 3.

Figure 19 shows the (a) average throughput and (b) average energy dissipation per bit for the same group of decoders shown in Fig. 18 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. 18). For throughput results, we assume that there is always a new frame to be decoded upon request.

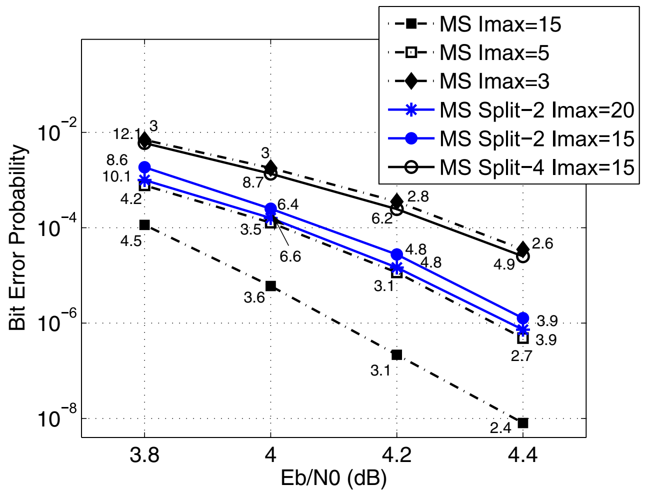


Figure 18 Error performance of the MinSum normalized, MinSum Split-2 and MinSum Split-4 decoders for (2,048, 1,723) code with various maximum number of iterations (*Imax*). The average number of decoding iterations is shown at every simulation point.

It is interesting to compare decoders at the same BER. From Fig. 18, 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.

8.5 Comparison with Other Chips

The post-layout simulation results of the described MinSum Split-4 decoder and two recently published full-parallel LDPC decoder chips [26, 29] are summarized in Table 4. Results for two supply voltages are reported for the Split-4 decoder: a nominal 1.3 V and 0.79 V, which is the minimum voltage that can achieve the 6.4 Gbps throughput required by the 10GBASE-T standard.

The (3.25, 6.5) (1024, 512) full-parallel decoder by Blanksby [26] (average row weight and column weight numbers are given) uses hierarchy-based and hand-placed design flow for routing and timing optimization. The Bit-Serial (4, 15) (660, 480) full-parallel decoder by Darabiha [29] uses a serial transfer of messages between the processing units to reduce routing congestion.

Although we are comparing only full-parallel decoders with each other, it is still challenging to fairly compare these decoders since they implement different LDPC codes (including code length, row weight, and column weight), different rates, and different CMOS technologies. Basic metrics such as throughput, energy, and circuit area are unfortunately complex functions of these parameters.

Table 4 gives the *No. of edges in LDPC code,* for each of the decoders. This value is a fraction of the number of global wires in a full-parallel decoder and it therefore gives a good first-order estimate of a

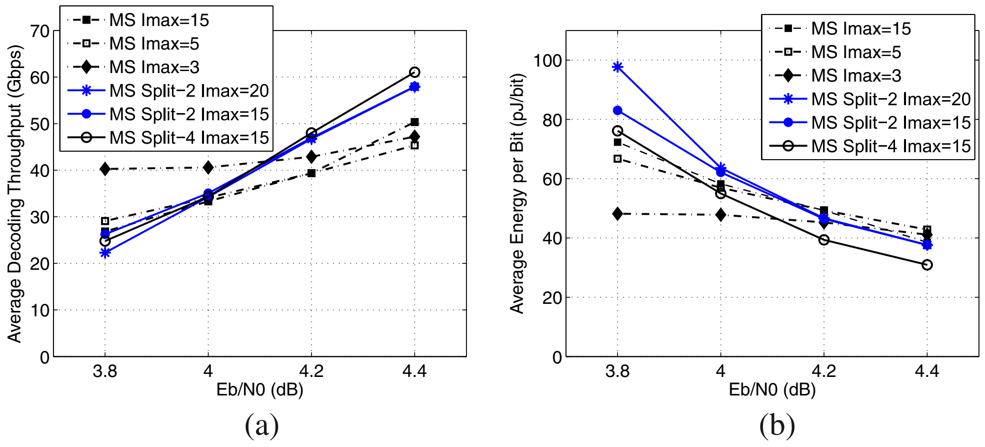


Figure 19 **a** Average decoding throughput and **b** average energy dissipation per bit in MinSum normalized, MinSum Split-2 and MinSum Split-4 decoders as a function of SNR and the average decoding iteration for different maximum numbers of iterations (*Imax*).

Table 4 Comparison of the Split-4 decoder with published full-parallel LDPC decoder Chips.

	Blanksby [26]	Darabiha [29]	**This work**	
Code length, no. of columns (N)	1024	660	2048	
Information length (K)	512	480	1723	
Parity check eqns., no. of rows (M)	512	176	384	
Bits per message (q)	4	4	5	
Row weight (W_r)	6.5	15	32	
Column weight (W_c)	3.25	4	6	
No. of edges in LDPC code, $MW_rq = NW_cq$	**13,312**	**10,560**	**61,440**	
Decoding algorithm	SPA	MinSum	Split-4 MinSum	
CMOS fabrication process (min. feature size)	160 nm	130 nm	65 nm	
Total chip area (mm^2)	52.5	9.0	6.4†	
No. of decoding iterations	64	15	15	
Supply voltage (V)	1.5	1.3	1.3	0.79
Clk speed (MHz)	64	300	146	47
Throughput (Gbps)	1.0	3.3	**19.9**	**6.4**
Throughput per area (Mbps/mm^2)	19	367	**3,109**	1,000
Power (mW)	690	1,408	1,889	226
Energy per bit (pJ/bit)	690	427	95	**35**

†The area of an IO pad ring has been added to the Split-4 decoder.

full-parallel decoder's complexity. Using this metric, the estimated code complexity of the Split-4 decoder's code is 4.6, and 5.8 times higher compared to the other decoders' codes.

A rough area comparison can be made by linearly normalizing the *Total chip area* of decoders to 61,440/*No. of edges in LDPC code* and normalizing quadratically with feature size, i.e., (65 nm/*Min. feature size*)^2. Scaling results in 40 mm^2 for the (1024, 512) full-parallel decoder [26] and 13.1 mm^2 for the (660, 480) full-parallel decoder [29]. It is important to note that scaling linearly with the *Total row/col processor input bits* factor favors simpler codes since decoder circuit area grows faster than this factor due to the limited routing resources in VLSI implementations. Nevertheless, the Split-4 decoder is 6.2 and two times smaller than the scaled (1024, 512) and (660, 480) full-parallel decoders respectively.

For energy comparisons, all decoders are scaled to 65 nm operating on a 1.3 V supply voltage. Scaling linearly with feature size and quadratically with supply voltage gives energy per bit of 210.5 pJ/bit for the (1024, 512) full-parallel decoder [26] and 213.5 pJ/bit for the (660, 480) full-parallel decoder [29]. The Split-4 decoder with its more complex code operates with an energy per bit that is 2.2 times lower and has 0.55 dB error performance degradation compared to the other two decoders.

In addition, with early termination enabled, Split-4 delivers 61 Gbps throughput and dissipates 31 pJ/bit at SNR=4.4 dB (see Table 3). When compared to the state-of-the art 47.7 Gbps, 58.7 pJ/bit partial parallel 10GBASE-T decoder [17], which is built in 65 nm and 1.2 V, Split-4 has 1.3 times higher throughput is 1.9 times more energy efficient, and is 1.1 times larger with error performance degradation of 0.60 dB.

9 Conclusion

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 (2,048, 1,723) code, depending on the level of splitting.

The proposed algorithm and architecture break row 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 2,048-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.

Acknowledgements The authors gratefully acknowledge support from ST Microelectronics, Intel, UC Micro, NSF Grant 0430090 and CAREER Award 0546907, SRC GRC Grant 1598 and CSR Grant 1659, Intellasys, Texas Instruments, IBM, SEM, and a UCD Faculty Research Grant; LDPC codes and assistance from Shu Lin and Lan Lan; and thank Zhengya Zhang, Dean Truong, Aaron Stillmaker, Lucas Stillmaker, Jean-Pierre Schoellkopf, Patrick Cogez, and Pascal Urard.

References

1. Gallager, R. G. (1962). Low-density parity check codes. *IRE Transactions on Information Theory, IT-8*, 21–28.
2. IEEE P802.3an, 10GBASE-T task force. http://www.ieee802.org/3/an.
3. T.T.S.I. digital video broadcasting (DVB) second generation framing structure for broadband satellite applications. http://www.dvb.org.
4. IEEE 802.16e (2005). Air interface for fixed and mobile broadband wireless access systems. IEEE p802.16e/d12 draft.
5. Tanner, R. M. (1981). A recursive approach to low complexity codes. *IEEE Transactions on Information Theory, 27*, 533–547.
6. MacKay, D. J. (1999). Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory, 45*, 399–431.
7. Fossorier, M., Mihaljevic, M., & Imai, H. (1999). Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on Communications, 47*, 673–680.
8. Hagenauer, J., Offer, E., & Papke, L. (1996). Iterative decoding of block and convolutional codes. *IEEE Transactions on Information Theory, 42*, 429–445.
9. Chen, J., & Fossorier, M. (2002). Near optimum universal belief propagation based decoding of low-density parity check codes. *IEEE Transactions on Communications, 50*, 406–414.
10. Chen, J., Dholakia, A., Eleftheriou, E., & Fossorier, M. (2005). Reduced-complexity decoding of LDPC codes. *IEEE Transactions on Communications, 53*, 1288–1299.
11. Bates, S., Chen, Z., et al. (2008). A low-cost serial decoder architecture for low-density parity-check convolutional codes. *IEEE Transactions on Circuits and Systems I, 55*, 1967–1976.
12. Yang, L., Liu, H., & Shi, R. (2006). Code construction and FPGA implementation of a low-error-floor multi-rate low-density parity-check decoder. *IEEE Transactions on Circuits and Systems I, 53*, 892.
13. Dai, Y., Chen, N., & Yan, Z. (2008). Memory efficient decoder architectures for quasi-cyclic LDPC codes. *IEEE Transactions on Circuits and Systems I, 55*, 2898–2911.
14. Shih, X., Zhan, C., Lin, C., & Wu, A. (2008). 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.
15. Liu, C. H., et al. (2008). An LDPC decoder chip based on self-routing network for IEEE 802.16e applications. *JSSC, 43*, 684–694.
16. Liu, L., & Shi, R. (2008). Sliced message passing: High throughput overlapped decoding of high-rate low density parity-check codes. *IEEE Transactions on Circuits and Systems I, 55*, 3697–3710.
17. Zhang, Z., Dolecek, L., et al. (2009). A 47 Gb/s LDPC decoder with improved low error rate performance. In *Symposium on VLSI circuits* (pp. 22–23).
18. Wang, Z., Li, L., et al. (2009). Efficient shuffle network architecture and application for WiMAX LDPC decoders. *IEEE Transactions on Circuits and Systems II: Express Briefs, 56*, 215–219.
19. Mansour, M., & Shanbhag, N. R. (2006). A 640-Mb/s 2048-bit programmable LDPC decoder chip. *JSSC, 41*, 684–698.
20. Wang, Z., & Cui, Z. (2007). Low-complexity high-speed decoder design for quasi-cyclic LDPC codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15*, 104–114.
21. Urard, P., Paumier, L., et al. (2008). A 360mW 105Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes enabling satellite-transmission portable devices. In *ISSCC* (pp. 310–311).
22. Liu, H., Lin, C., et al. (2005). A 480mb/s LDPC-COFDM-based UWB baseband transceiver. In *ISSCC* (Vol. 1, pp. 444–445).
23. Fewer, C., Flanagan, F., & Fagan, A. (2007). A versatile variable rate LDPC codec architecture. *IEEE Transactions on Circuits and Systems I, 54*, 2240–2251.
24. Masera, G., Quaglio, F., & Vacca, F. (2007). Implementation of a flexible LDPC decoder. *IEEE Transactions on Circuits and Systems I, 54*, 542–546.
25. Zhang, H., Zhu, J., Shi, H., & Wang, D. (2008). Layered approx-regular LDPC code construction and encoder/decoder design. *IEEE Transactions on Circuits and Systems I, 55*, 572–585.
26. Blanksby, A., & Howland, C. J. (2002). A 690-mW 1-Gb/s 1024-b, rate 1/2 low-density parity-check code decoder. *JSSC, 37*(3), 404–412.
27. Mohsenin, T., & Baas, B. (2006). Split-Row: A reduced complexity, high throughput LDPC decoder architecture. In *ICCD* (pp. 13–16).
28. Mohsenin, T., & Baas, B. (2007). High-throughput LDPC decoders using a multiple split-row method. In *ICASSP* (Vol. 2, pp. 13–16).
29. Darabiha, A., Carusone, A. C., & Kschischang, F. R. (2007). A 3.3-Gbps bit-serial block-interlaced Min-Sum LDPC decoder in 0.13-um CMOS. In *IEEE custom integrated circuits conference* (pp. 459–462).

30. Kim, E., Jayakumar, N., Bhagwat, P., & Khatri, S. P. (2006). A high-speed fully-programmable VLSI decoder for regular LDPC codes. In *International conference on acoustics, speech, and signal processing* (Vol. 3, pp. 972–975).
31. Darabiha, A., Carusone, A. C., & Kschischang, F. R. (2008). Block-interlaced LDPC decoders with reduced interconnect complexity. *IEEE Transactions on Circuits and Systems Part II: Express Briefs, 55*, 74–78.
32. Kang, S., & Park, I. (2006). Loosely coupled memory-based decoding architecture for low density parity check codes. *IEEE Transactions on Circuits and Systems I, 53*, 1045–1056.
33. Cui, Z., & Wang, Z. (2007). Efficient message passing architecture for high throughput LDPC decoder. In *ISCAS* (pp. 917–920).
34. Richardson, T., & Urbanke, R. (2001). The capacity of low-density parity check codes under message-passing decoding. *IEEE Transactions on Information Theory, 47*, 599–618.
35. Djurdjevic, I., Xu, J., Abdel-Ghaffar, K., & Lin, S. (2003). A class of low-density parity-check codes constructed based on Reed–Solomon codes with two information symbols. *IEEE Communications Letters, 7*, 317–319.
36. Chen, L., Xu, J., Djurdjevic, I., & Lin, S. (2004). Near-Shannon-limit quasi-cyclic low-density parity-check codes. *IEEE Transactions on Communications, 52*, 1038–1042.
37. Kou, Y., Lin, S., & Fossorier, M. P. C. (2001). Low-density parity-check codes based on finite geometries: A rediscovery and new results. *IEEE Transactions on Information Theory, 47*(7), 2711–2736.
38. Zhang, J., Fossorier, M. P. C. (2004). A modified weighted bit-flipping decoding of low-density parity-check codes. *IEEE Communications Letters, 8*, 165–167.
39. Gunnam, K. K., et al. (2006). Decoding of quasi-cyclic LDPC codes using an on-the-fly computation. In *40th asilomar conference on signals, systems and computers* (pp. 1192–1199).
40. Zhang, Z., Venkat, A., et al. (2007). Quantization effects in low-density parity-check decoders. In *ICC* (pp. 6231–6237).
41. Rabaey, J., Chandrakasan, A., & Nikolic, B. (2003). *Digital integrated circuits* (2nd ed.). Upper Saddle River: Prentice Hall.
42. Truong, D. N., Cheng, W. H., et al. (2009). A 167-processor computational platform in 65 nm CMOS. *IEEE Journal of Solid-State Circuits (JSSC), 44*(4), 1130–1144.
43. Darabiha, A., Carusone, A. C., & Kschischang, F. R. (2008). Power reduction techniques for LDPC decoders. *JSSC, 43*, 1835–1845.

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

She is the designer of the Split-Row, Multi-Split, 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 (AsAP) 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 (DSP), and error correction applications.

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 Mbps, 5 GHz) Wi-Fi wireless LAN solution.

In 2003 he joined the Department of Electrical and Computer Engineering at the University of California, Davis where he is now 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 (AsAP) chip, applications, and tools; a second generation 167-processor chip; low density parity check (LDPC) decoders; FFT processors; viterbi decoders; and H.264 video codecs. During the summer of 2006 he was a Visiting Professor in Intel's Circuit Research Lab.

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 AISES 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, the IEEE International Symposium on Asynchronous Circuits and Systems, and the Program Committee of the HotChips Symposium on High Performance Chips. He also serves as a member of the Technical Advisory Board of an early stage technology company.