Hindawi Publishing Corporation
VLSI Design
Volume 2013, Article ID 913018, 14 pages
http://dx.doi.org/10.1155/2013/913018

Research Article

LDPC Decoder with an Adaptive Wordwidth Datapath for Energy and BER Co-Optimization

Tinoosh Mohsenin,[1] Houshmand Shirani-mehr,[2] and Bevan M. Baas[2]

[1] *CSEE Department, University of Maryland, Baltimore County, MD 21250, USA*
[2] *ECE Department, University of California, Davis, MD 95616, USA*

Correspondence should be addressed to Tinoosh Mohsenin; tinoosh@umbc.edu

Received 10 September 2012; Accepted 25 December 2012

Academic Editor: Sungjoo Yoo

An energy efficient low-density parity-check (LDPC) decoder using an adaptive wordwidth datapath is presented. The decoder switches between a *Normal Mode* and a reduced wordwidth *Low Power Mode*. Signal toggling is reduced as variable node processing inputs change in fewer bits. The duration of time that the decoder stays in a given mode is optimized for power and BER requirements and the received SNR. The paper explores different *Low Power Mode* algorithms to reduce the wordwidth and their implementations. Analysis of the BER performance and power consumption from fixed-point numerical and post-layout power simulations, respectively, is presented for a full parallel 10GBASE-T LDPC decoder in 65 nm CMOS. A 5.10 mm^2 low power decoder implementation achieves 85.7 Gbps while operating at 185 MHz and dissipates 16.4 pJ/bit at 1.3 V with early termination. At 0.6 V the decoder throughput is 9.3 Gbps (greater than 6.4 Gbps required for 10GBASE-T) while dissipating an average power of 31 mW. This is 4.6× lower than the state of the art reported power with an SNR loss of 0.35 dB at BER = 10^{-7}.

1. Introduction

Communication systems are becoming a standard requirement of every computing platform from wireless sensors, mobile telephony, netbooks, and server class computers. Local and cellular wireless communication throughputs are expected to increase to hundreds of Mbps and even beyond 1 Gbps [1–3]. With this increased growth for bandwidth comes larger systems integration complexity and higher energy consumption per packet. Low power design is therefore a major design criterion alongside the standards' throughput requirement as both will determine the quality of service and cost.

Additionally, mobile computing will take on a new dimension as a portal into *Software as a Service* (i.e., cloud computing) where low performance computers can tap into the power of a distant high-performance computer cluster [4, 5]. So far the emerging 10GBASE-T standard has not been adopted as quickly as predicted into the data center infrastructures because of their power consumption [6]. The power consumption of the 10GBASE-T PHY layer (more specifically the receiver, whose implementation is left open by the 802.3 an standard [7]) has become difficult to reduce [8].

LDPC code was first developed in 1962 [9] as an error correction technique that allowed communication over noisy channels possibly near the Shannon limit. With advancements in VLSI, LDPC codes have recently received a lot of attention because of their superior error correction performance and have been adopted by many recent standards such as digital video broadcasting via satellite (DVB-S2) [10], the WiMAX standard (802.16e) [11], the G.hn/G.9960 standard for wired home networking [12], and the 10GBASE-T standard for 10 Gigabit Ethernet (802.3 an) [7].

LDPC decoder architectures can be categorized into two domains: *full parallel* and *partial parallel*. Full parallel is a direct implementation of the LDPC decoding algorithm with every computational unit and interconnection between them realized in hardware. Partial parallel decoders use pipelining, large memory resources, and shared computational blocks to deal with the inherent communication complexity and massive bandwidth. Since the amount of operations achievable per cycle is larger with a full parallel processor, their

energy efficiencies are theoretically the best [13]. For example, an LDPC decoder implementing the 10GBASE-T standard requires 24,576 operations per iteration (this is the total number of check node update and variable node update computation in message-passing algorithm [14]). A full parallel decoder can take one cycle to perform one iteration, while a partial parallel decoder takes multiple cycles (e.g., in a design, each iteration takes 12 cycles [15]). Compared to partial parallel decoders, full parallel decoders can achieve the same throughput performance while operating at a lower clock frequency, that is, runing at lower minimum supply voltages and thus reducing energy. However, for complex codes, full parallel decoders deviate strongly from this ideal due to their large interconnect complexity and low clock rate [14]. Given equivalent 10GBASE-T compliant LDPC codes, throughput requirements, and 65 nm CMOS technology, a full parallel LDPC decoder achieves a 2.6 TOPS per Watt efficiency compared to a partial parallel LDPC decoder at 1.4 TOPS per Watt [14, 15]. Thus practical full parallel decoders show less than 2 × performance-power efficiency compared to the 12 × promised in the ideal scenario.

To improve their efficiency, previous research has focused on reducing routing congestion and wire delay of the full parallel decoder implementations through bit-serial communication [13], wire partitioning [16], and algorithm modification [17]. A full parallel design using the *Split-Row* algorithm modification resulted in an implemented architecture that achieved 14 TOPS per Watt, that is, 10× the efficiency of a partial parallel decoder [14].

This paper proposes an adaptive wordwidth algorithm that takes advantage of data input patterns during the LDPC decoding process. We show that the method is valid for both MinSum and Split-Threshold, and, for demonstration, we implement the proposed method for Split-16 Threshold decoder. Switching activity reduction through adaptive arithmetic datapath wordwidth reduction has been explored in low power designs based on data spatial correlation [18]. To our knowledge this has not been explored in LDPC decoding yet. The paper presents an architecture which switches between *Normal Mode* and *Low Power Mode* operation with a final post-layout implementation. It also optimizes energy efficiency by minimizing unnecessary bit toggling while maximizing bit error rate (BER) performance.

The paper is organized as follows: Section 2 gives an overview of LDPC decoding, the *Split-Row Threshold* algorithm, and common power reduction techniques; Section 3 introduces the adaptive wordwidth power reduction method with analysis for three different methods along with their bit error performance results; Section 4 gives details of their architecture; Section 5 presents the results of the post-layout implementations of three full parallel 10GBASE-T LDPC decoders that implement the low power adaptive algorithm.

2. Background

2.1. LDPC Codes and MinSum Normalized Decoding. The LDPC decoding algorithm works by performing an iterative computation known as *message passing*. Each iteration consists of variable node and check node computations. Common iterative decoding algorithms are Sum-Product Algorithm (SPA) [19] and MinSum algorithms [20]. 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 simplifies the SPA check node update equation, which replaces the computation of a nonlinear equation with a min( ) function. The MinSum check node update equation is given as

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

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 *Sfactor* is included to improve error performance, and so this variant of MinSum is called "MinSum Normalized" [21].

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; row weight W_r and column weight W_c, which define the $1'$s per rows and columns, respectively. In this work, we examine cases where H is *regular,* and thus W_r and W_c are constants. For clearer explanations, in this paper we will use a (6,32) (2048,1723) RS-LDPC code adopted by the 10GBASE-T standard [22]. This code is described by a 384×2048 H matrix with $W_r = 32$ 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. There are $M \times W_r = 12,288$ variable nodes and $N \times W_c = 12,288$ check node computations, for a total of 24,576 computations per iteration. Each variable node sends the result (i.e., its message) to its connected check nodes, and vice versa. A single cycle per iteration full parallel architecture requires 24,576 message transfers (message-passing) per cycle. Given that each message can be as large as four to six bits, the bisection bandwidth of the communication links between the check to variable node processors, the memory to check node, and variable node to memory, are from 98 to 147 Kbit per cycle each. These links not only cause problems in interconnect latencies, but also add capacitance due to wires and repeaters, which increases the circuit power [13].

2.2. Split-Row Threshold Decoding. The proposed *Split-Row Threshold* [23] algorithm significantly reduces the interconnect complexity and circuit area by partitioning the links

needed in the message-passing algorithm, which localizes message-passing. A minimal amount of information is transferred amongst partitions to ensure computational accuracy while reducing global communication. This is most effective in reducing wire congestion and back-end engineering time for full parallel architectures with large codes (e.g., from 2 Kbits to 64 Kbits) or high check node degrees.

The *Split-Row Threshold* algorithm gains back the loss in error performance by adding an additional form of information based on a comparison with a threshold value (T). Based on this comparison, a "threshold enable" bit (*Threshold_en*) is sent between each partition [14]. The check node update equation is modified as follows:

$$\alpha_{ij:Spi} = \text{Sfactor} \times \underbrace{\prod_{j' \in V(i) \setminus j} \text{sign}\left(\beta_{ij'}\right)}_{\text{Sign Calculation}} \times \underbrace{\text{Min}_{Spi}}_{\text{Magnitude Calculation}}, \tag{3}$$

where

$$\text{Min}_{Spi} = \begin{cases} \min\limits_{j' \in V_{Spi}(i) \setminus j} \left(\left|\beta_{ij'}\right|\right), & \text{if } \min\limits_{j' \in V_{Spi}(i) \setminus j} \left(\left|\beta_{ij'}\right|\right) \leq T \quad \text{(a)} \\ \min\limits_{j' \in V_{Spi}(i) \setminus j} \left(\left|\beta_{ij'}\right|\right), & \text{if } \min\limits_{j' \in V_{Spi}(i) \setminus j} \left(\left|\beta_{ij'}\right|\right) > T \\ & \text{and } \mathit{Threshold\_en} == 0 \quad \text{(b)} \\ T, & \text{if } \min\limits_{j' \in V_{Spi}(i) \setminus j} \left(\left|\beta_{ij'}\right|\right) > T \\ & \text{and } \mathit{Threshold\_en} == 1, \quad \text{(c)} \end{cases} \tag{4}$$

where V_{Spi} represents the $V(i)$ variable nodes only contained in decoder partition *Spi* on row i (each partition has N/Spn variable nodes). With the threshold comparison based information, error performance loss is improved from a 0.07 to 0.22 dB reduction (depending on the level of partitioning) from MinSum Normalized performance.

This paper discusses power improvements of a Split-16 Threshold decoder architecture (i.e., there are $Spn = 16$ partitions) using the proposed adaptive wordwidth technique. Since the row weight of the 10GBASE-T code is 32, each partition contains check nodes that have $W_r/Spn = 32/16 = 2$ inputs. The optimum values for T and Sfactor depend on the code rate, size, and the level of partitioning. For example, for (6,32) (2048,1723) LDPC code using Split-16 Threshold, $T = \text{Sfactor} = 0.25$ results in the best BER performance with 0.3 dB SNR loss from MinSum Normalized.

2.3. Power Reduction Methods

2.3.1. Early Termination. An efficient technique to reduce the energy dissipation is through controlling the number of decoding iterations that a block requires for a successful decoding convergence. The common method is to verify if the computed codeword satisfies all parity-check constraints at the end of each iteration. Once convergence has been verified, the decoding process is terminated. Several methods are proposed to efficiently implement this *early termination* [13, 24, 25]. LDPC codes, especially high rate codes, converge early at high SNR [26]. Therefore, by detecting early decoder convergence, throughput and energy can potentially improve significantly while maintaining the same error performance.

2.3.2. Voltage Scaling. In order to save power and energy one effective technique is to employ voltage scaling in the decoder such that the application throughput requirement is met. For the Split-16 Threshold decoder, the minimum voltage to meet the 6.4 Gbps 10GBASE-T compliant throughput is 0.7 V in 65 nm CMOS [14]. For most cases, near-threshold operation is not advisable in nanometer technologies due to increased susceptibility to variations and soft errors [27], and so any further energy savings using voltage scaling will reduce the functional integrity of the decoder's circuits.

2.3.3. Switching Activity and Wordwidth Reduction. Because decoders exhibit large switching activity due to their largely computational nature, we can decrease power by lowering the effective capacitance, C_{eff}. For full parallel architectures, this was done through the Split-Row implementations which reduced overall hardware complexity and thus eliminated interconnect repeaters and wire capacitance. The datapath wordwidth of the decoder directly determines the required memory capacity, routing complexity, decoder area, and critical path delays. Moreover, it affects the amount of switching activity on wires and logic gates, thus affecting the power dissipation.

For partial parallel architectures, wordwidth reduction using nonuniform quantization has been used to reduce the amount of information needed in check node processing and memory storage requirements (thus also reducing the SRAM capacitance as well) [28]. However, conversion steps are needed to do variable node computation in the original wider wordwidth. In [15], additional postprocessing is required to improve the error correction performance which also improves the error floor. Implementing nonuniform quantization to full parallel architectures may result in more costs than benefits. Conversion steps across all communication links add hardware between every check and variable node. Since memory is not a large part of such architectures, this method does not save on memory area. In this work, rather than statically fixing the wordwidth at run time we will introduce a low cost adaptive wordwidth datapath technique to reduce switching activity for a full parallel decoder.

3. Adaptive Wordwidth Decoder Algorithm

A simplified block diagram of a single cycle LDPC decoder is shown in Figure 1. With the Split-Row Threshold architecture, the check node processor logic generally has lower C_{eff} than that of the variable node processor due to its reduced hardware [14]. The figure shows some of the variable node details such as the adder tree. For the (6,32) (2048,1723) 10GBASE-T code, variable node processors add seven inputs: six inputs from the messages passed by the check node processors (α) as well as the original received data from the channel (λ). Since the wordwidth growth is required to maintain correct summation and given that the 10GBASE-T code length is large ($N = 2048$), the amount of power dissipated by 2048 variable node processors in a full parallel decoder is significant.

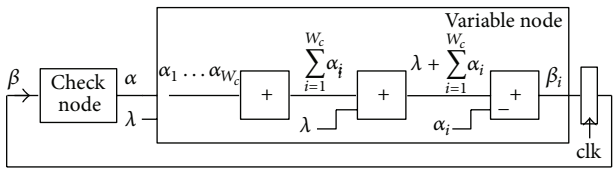

Figure 1: Single cycle LDPC decoding with variable node processor (with W_c input α messages) in partial detail.

Our proposed algorithm adapts the wordwidth datapath of variable node processing based on its data input patterns (α values). The algorithm switches between two modes: *Low Power Mode* and *Normal Mode*. In *Normal Mode* a full wordwidth computation is done, while *Low Power Mode* performs a reduced wordwidth computation. We first show α values are largely concentrated in $[-\text{Sfactor} \times T, +\text{Sfactor} \times T]$ interval then present the algorithm.

3.1. Theoretical Investigations. Let the variable node messages $\beta_1, \beta_2, \ldots, \beta_{Wr}$ be the inputs to a check node C_i. Since variable node messages are initialized with channel information (assuming α messages in (1) are initially zero), for BPSK modulation and an AWGN channel, their distribution at the first iteration is Gaussian.

For iterations >1, the variable node messages in MinSum Normalized are approximated to the Gaussian distributions [29]. Similarly, in Split-Row Threshold, variable node messages can be fitted with the sum of two Gaussian distributions, and a very good agreement (R-square = 0.99) was achieved for the fit. Therefore, the distribution at iteration l can be described as

$$P_V(x_l) = \frac{1}{\sqrt{2\pi\sigma_l^2}} \left(e^{-(x_l-\mu_l)^2/2\sigma_l^2} + e^{-(x_l+\mu_l)^2/2\sigma_l^2} \right), \quad (5)$$

where σ_l^2 and μ_l are the variance and the mean of the distribution. For this distribution, the probability that a variable node message β has a magnitude less than a given value D is

$$P_{|\beta|<D} = \frac{1}{\sigma_l\sqrt{2\pi}} \int_{-D}^{+D} e^{-(x-\mu_l)^2/2\sigma_l^2} dx. \quad (6)$$

Thus assuming $\beta_1, \beta_2, \ldots, \beta_{Wr}$ are i.i.d., the probability that at least one input of the check node C_i has a magnitude less than D is

$$P\left(\exists \beta_i \in \{\beta_1, \beta_2, \ldots, \beta_{Wr}\} \mid |\beta_i| < D\right) = \left(1 - P_{|\beta|<D}\right)^{W_r}. \quad (7)$$

In MinSum Normalized and Split-Row Threshold, for each check node if there exists one input, β, whose magnitude is less than D, then applying (2) and (3) the other $W_r - 1$ outputs of the check node (α messages) have absolute values less than $D \times$ Sfactor after being normalized with Sfactor. Thus if the probability from (7) is high enough for a particular D, we should expect a large concentration of $\alpha \in [-D \times \text{Sfactor}, +D \times \text{Sfactor}]$. Simulation results show, for the (2048,1723) 10GBASE-T code using MinSum Normalized, when $D = 0.5$, the probability from (7) at SNR = 4.4 dB is 99%, 92%, and 65% for iterations 1 through 3. Also they show 99%, 90%, and 62% of α values which are within $\pm D \times$ Sfactor $= \pm 0.25$ (Sfactor = 0.5 results in a near optimum BER performance).

Table 1: The percentage that $\alpha \in$ Threshold Region and $\alpha = \pm T \times$ Sfactor condition in 1000 sets of input data for two SNR values. For SNR = 4.4 dB, most blocks converge at iterations >4.

Iteration	$\alpha \in$ Threshold Region		$\alpha = \pm T \times$ Sfactor	
	3.4 dB	4.4 dB	3.4 dB	4.4 dB
1	95%	90%	90%	86%
2	93%	74%	88%	72%
3	91%	48%	87%	47%
4	89%	—	85%	—
5	87%	—	83%	—
6	86%	—	82%	—
7	85%	—	81%	—

In Split-Row Threshold, D is set to threshold T. For 10GBASE-T code in Split-16 Threshold, the probability value of $P(\exists \beta_i \in \{\beta_1, \beta_2, \ldots, \beta_{Wr}\} \mid |\beta_i| < T)$ is 99%–67% for SNR ranges 3.4–4.2 dB and iterations 1 through 4. If there exists an input in a partition whose absolute value is smaller than T, then the *Threshold_en* signal is asserted high and is globally sent to other partitions. Therefore, the check nodes in other partitions set their minimum (Min_{Spi} from (3)) to T, if their local minimum was larger than T. Due to this key characteristic and applying (3), a large number of check node messages (α) are $\pm T \times$ Sfactor.

Table 1 shows the percentage of $\alpha \in [-\text{Sfactor} \times T, +\text{Sfactor} \times T]$ and $\alpha = \pm T \times$ Sfactor for a large number of decoding iterations at SNR = 3.4 and 4.2 dB. We call $[-\text{Sfactor} \times T, +\text{Sfactor} \times T]$ interval as *Threshold Region*. The table shows that for SNR = 3.4 dB and through iterations 7, 95% down to 85% of all α values are in the *Threshold Region* of which 90%–81% are $\pm T \times$ Sfactor. For a high SNR value of 4.2 dB and through iterations 3,90% down to 48%α values are in the region, with 86%–47% being $\pm T \times$Sfactor. This is shown in Figure 2. Most blocks converge beyond four iterations at SNR $\geq$ 4.4 dB.

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

3.2. Power Reduction Algorithm. Given that variable node input wordwidths can be reduced without losing significant information at low SNR values and also at low iteration counts in high SNRs, we propose a Low Power Mode operation for the decoder which significantly reduces the switching activity of the variable node processors in the following.

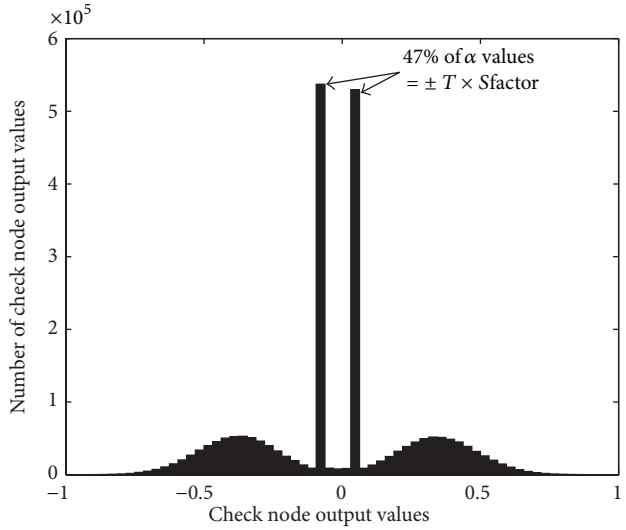

FIGURE 2: Check node output (α) distribution at iteration; three iterations for (2048,1723) LDPC code using Split-16 Threshold decoder at SNR = 4.4 dB, where T = *S*factor = 0.25.

After check node processing and when the current iteration count (*Iteration*) is less than a preset Low Power Mode iteration max count (*Low Power Iteration*), we chop or saturate α such that it is within the Threshold Region. Three methods are explored which have different BER performance, convergence behavior, and hardware complexity. All three methods try to remap all α into the Threshold Region. In Method 1, we saturate α values outside the Threshold Region into $[-T \times S\text{factor}, +T \times S\text{factor}]$. In Method 2, we set all α magnitudes to $T \times S$factor, because the majority of them are concentrated at $T \times S$factor value. In Method 3, we only keep the minimum number of LSB bits that can represent the values within the Threshold Region (in other words, the α MSBs are chopped). These methods are described in Algorithm 1.

A qualitative perspective shows that Method 1 has the best error performance since it preserves any α already within the *Threshold Region* and also maps α values regularly. Method 2 offers a simple hardware solution at the cost of losing some information for $\alpha \in (-T \times S\text{factor}, +T \times S\text{factor})$, but it has a high reduction in bit toggling (to be explained in Section 4). For Method 3, its benefit comes from the compromise between the hardware cost of Method 1 and a better error correction performance than Method 2 (even though the α values are irregularly mapped).

By reducing the information range of α into the *Threshold Region*, the required datapath wordwidth is reduced, and thus variable node computation can be done with less switching activity in *Low Power Mode*. The challenges come from implementing a low overhead flexible datapath as well as deciding when to switch out of *Low Power Mode* such that the final convergence does not take much more iterations than running completely in *Normal Mode*. Algorithm 2 describes the complete *Split-Row Threshold Low Power* decoding process.

For our 10GBASE-T decoder implementation, the decoding message wordwidth is chosen to be 6 bits in *Normal Mode*. During *Low Power Mode*, for Methods 1 and 3, the 6-bit input additions in variable node are reduced into 3-bit input additions, while in Method 2, it is reduced to 1-bit input additions (see Section 4). In order to simplify hardware and further reduce the toggling, the variable node final subtractions (see (1)) can be bypassed during *Low Power Mode* without causing a significant distortion of β messages. This is shown in Figure 3 which compares the β distributions for 10GBASE-T code using Split-Row Threshold and modified version with *Low Power Mode* using Method 1 at iteration 4 at SNR = 3.8 dB. As shown in the figure, the distributions are closely matched.

Figure 4 illustrates the BER performance of the 2048-bit 10GBASE-T code using Split-Row Threshold for only *Normal Mode* operation (*Low_Power_Iteration* = 0) and adaptive low power operation using Methods 1, 2, and 3 when *Low_Power_Iteration* is 3, 5, and 6. The figure also shows that Methods 1, 2, and 3 have nearly the same bit error performance. They also perform very closely to *All Normal Mode*, with a 0.06–0.1 dB decrease at BER = 10^{-7} when *Low_Power_Iteration* = 3. With *Low_Power_Iteration* = 6, this SNR gap increases to 0.15–0.2 dB.

4. Architecture Design

The single pipeline block diagram for the proposed full parallel Split-Row Threshold decoder with *Spn* partitions is shown in Figure 5. In each partition, there are M check processors (each takes W_r/N inputs) and N/Spn variable processors. The Sign and *Threshold_en* passing signals are the only wires passing (serially) between the partitions which are generated in the check node processors in parallel. The *Lowpower_flag* global signal is sent to every block and sets the operation mode to either *Normal Mode* or *Low Power Mode* (see Algorithm 2).

4.1. Check Node Processor. The check node processor implementation for partition *Spi* is shown in Figure 6 and consists of two parts, which are described in the next two minor sections.

4.1.1. Split-Row Threshold. The magnitude update of α is shown along the upper part of the figure while the global sign is determined by the XOR logic along the lower part. In Split-Row Threshold 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 equations (2) and (3).

In both MinSum Normalized and Split-Row Threshold decoding, the first minimum Min 1 and the second minimum Min 2 are found alongside the signal *Index* Min 1, which indicates whether Min 1 or Min 2 is chosen for a particular α. These are found through using multiple stages of comparators. 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

for $Spi = 1, 2, \ldots, Spn$ **do**
  for $i = 0, 1, \ldots, M-1$ **do**
    for all $j' \in V_{Spi}(i) \setminus j$ **do**
      if $Lowpower\_flag = 0$ **then**

$$\mathrm{Min}_{Spi} = \begin{cases} \begin{cases} \mathrm{Min}\,1_i, & \text{if } j \neq \operatorname{argmin}(\mathrm{Min}\,1_i) \\ \mathrm{Min}\,2_i, & \text{if } j = \operatorname{argmin}(\mathrm{Min}\,1_i) \end{cases}; & \text{if } \mathrm{Min}\,1_i < T \text{ and } \mathrm{Min}\,2_i < T \quad 8(a) \\ \begin{cases} \mathrm{Min}\,1_i, & \text{if } j \neq \operatorname{argmin}(\mathrm{Min}\,1_i) \\ T, & \text{if } j = \operatorname{argmin}(\mathrm{Min}\,1_i) \end{cases}; & \text{if } \mathrm{Min}\,1_i < T \text{ and } \mathrm{Min}\,2_i > T \text{ and } Threshold\_en = 1 \quad 8(b) \\ T, & \text{if } \mathrm{Min}\,1_i > T \text{ and } \mathrm{Min}\,2_i > T \text{ and } Threshold\_en = 1 \quad 8(c) \\ \begin{cases} \mathrm{Min}\,1_i, & \text{if } j \neq \operatorname{argmin}(\mathrm{Min}\,1_i) \\ \mathrm{Min}\,2_i, & \text{if } j = \operatorname{argmin}(\mathrm{Min}\,1_i) \end{cases}; & \text{if } \mathrm{Min}\,1_i > T \text{ and } \mathrm{Min}\,2_i > T \text{ and } Threshold\_en = 0 \quad 8(d) \end{cases}$$

$$\alpha_{ij:Spi} = \mathrm{Sfactor} \times \textstyle\prod_{j'} \mathrm{sign}(\beta_{ij'}) \times \mathrm{Min}_{Spi} \quad 3(a)$$

      else

$$\mathrm{Min}'_{Spi} = \begin{cases} \text{Method 1:} \begin{cases} \mathrm{Min}\,1_i, & \text{if } j \neq \operatorname{argmin}(\mathrm{Min}\,1_i) \\ \mathrm{Min}\,2_i, & \text{if } j = \operatorname{argmin}(\mathrm{Min}\,1_i) \end{cases}; & \text{if } \mathrm{Min}\,1_i < T \text{ and } \mathrm{Min}\,2_i < T \quad 9(a) \\ T; & \text{if } \mathrm{Min}\,1_i > T \text{ and } \mathrm{Min}\,2_i > T \\ \text{Method 2:} \quad T & 9(b) \\ \text{Method 3:} \quad \text{Equations } \textbf{do } (a), (b), (c), \textbf{ or } (d) \textbf{ then } (\mathrm{Min}'_{Spi} \mod T) & 9(c) \end{cases}$$

$$\alpha_{ij:Spi} = \mathrm{Sfactor} \times \textstyle\prod_{j'} \mathrm{sign}(\beta_{ij'}) \times \mathrm{Min}'_{Spi} \quad 3(b)$$

      end if
    end for
  end for
end for

Algorithm 1: Split-Row Low Power Threshold Algorithm—Check Node Processing.

Required: λ, that is, channel information
  $Iteration = 1$
  while $H \cdot \widehat{V}^T \neq 0$ **do**
    $Lowpower\_flag = (Iteration \leq Low\_Power\_Iteration)$
    for $j = 0, 1, \ldots, N-1$ **do**
      if $Lowpower\_flag = 0$ **then**
        $i' \in C(j) \setminus i$
      else
        $i' \in C(j)$
      end if
      for all i' **do**

$$\beta_{ij} = \lambda_j + \textstyle\sum_{i'} \alpha_{i'j} \quad (1)$$

      end for
    end for
    do Algorithm 1
    $Iteration = Iteration + 1$
  end while

Algorithm 2: Split-Row Low Power Threshold Algorithm—Variable Node Processing.

comparisons between Min 1 and *Threshold*, and Min 2 and *Threshold*, which are used to generate the final α values. The local *Threshold_en* signal that is generated by comparing *Threshold* and Min 1 is ORed 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. The next stage is *S*factor multiplication according to (3).

4.1.2. Low Power Mode Implementation. This step (shown as the *Mode Adjust*) block in Figure 6 includes a multiplexer which selects the appropriate message magnitude ($|\alpha|$ or $|\alpha_{\mathrm{adjust}}|$) based on the status of the *Lowpower_flag* global signal. In order to shutoff the toggling of unused bits in *Low Power Mode*, they are kept zero (their initial value). For a k-bit wordwidth implementation, we assume the *Threshold*

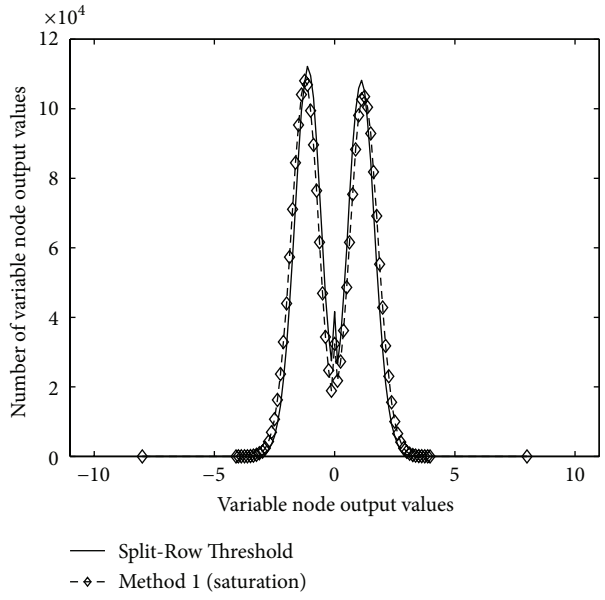

FIGURE 3: Variable node output (β) distributions for *Split-Row Threshold* and Method 1 at iteration 4 with SNR = 4.2 dB.

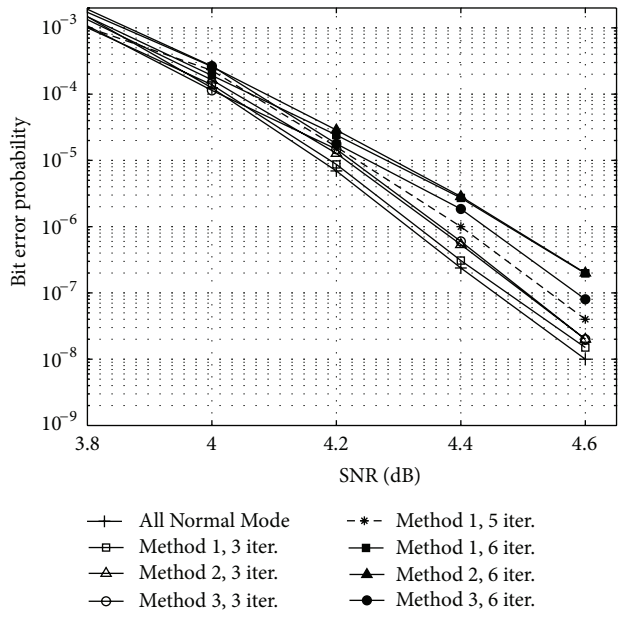

FIGURE 4: Bit error performance of the 2048-bit 10GBASE-T code using Split-Row Threshold (only *Normal Mode*, that is, *Low_Power_Iteration=0*) and Split-Row Low Power Threshold with Methods 1, 2, and 3 when *Low_Power_Iteration* is 3, 5, and 6.

Region $[-T \times \text{Sfactor}, +T \times \text{Sfactor}]$ can be implemented with d bits. Therefore, in $b_{k-1}b_{k-2}b_{k-3}\cdots b_{d-1}b_{d-2}\cdots b_1b_0$ signed format, $b_{k-2}b_{k-3}\cdots b_{d-1} = b_{d-1}b_{d-1}\cdots b_{d-1}$ in *Low Power Mode*. However, to eliminate the extra logic to perform the sign extensions in the variable node processor, $b_{k-2}b_{k-3}\cdots b_{d-1}$ are set to zero and are swapped with LSB bits (i.e., α_{adjust} becomes $b_{k-1}b_{d-2}\cdots b_1b_00\cdots00$).

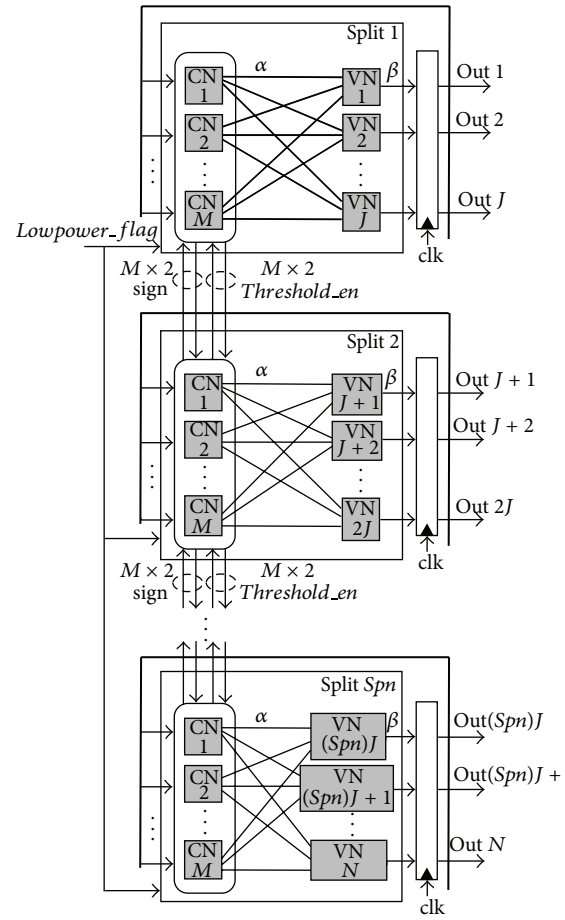

FIGURE 5: Block diagram of the proposed full parallel Split-Row Threshold adaptive wordwidth decoder with *Spn* partitions.

In Method 1 (α saturation to $[-T \times \text{Sfactor}, +T \times \text{Sfactor}]$), α is adjusted based on (Equation (9(a)) in Algorithm 1). This can be easily implemented using the *Sat_Control* signal that is generated in *Threshold Logic* and determines whether $|\alpha| > T \times \text{Sfactor}$. Overall, α bit toggling is reduced to at most d bits.

In Method 2, which implements (Equation (9(b)) in Algorithm 1), all α outputs are set to $\pm T \times \text{Sfactor}$. Therefore, $|\alpha_{\text{adjust}}|$ always becomes *Sat_Value*, regardless of its input magnitude. Thus in addition to reducing the gate count in Method 1, Method 2 reduces the α bit toggling to $\pm$*Sat_Value*.

In Method 3, which implements (Equation (9(c)) in Algorithm 1), only the first $d-1$ LSB bits are kept along with the sign bit, and bit toggling is reduced to d bits.

4.2. Variable Node Processor. The block diagram of the variable node processor is shown in Figure 7, which implements (1) in Algorithm 2. The key benefit of *Low Power Mode* operation is in the variable node processor, where all addition datapath wordwidths are reduced by at least $k - d$ bits (depending on the "Method" of implementation), which results in reduction of switching activity for the majority of the variable node processor. This wordwidth reduction is applied to all N variable processors (for 10GBASE-T code

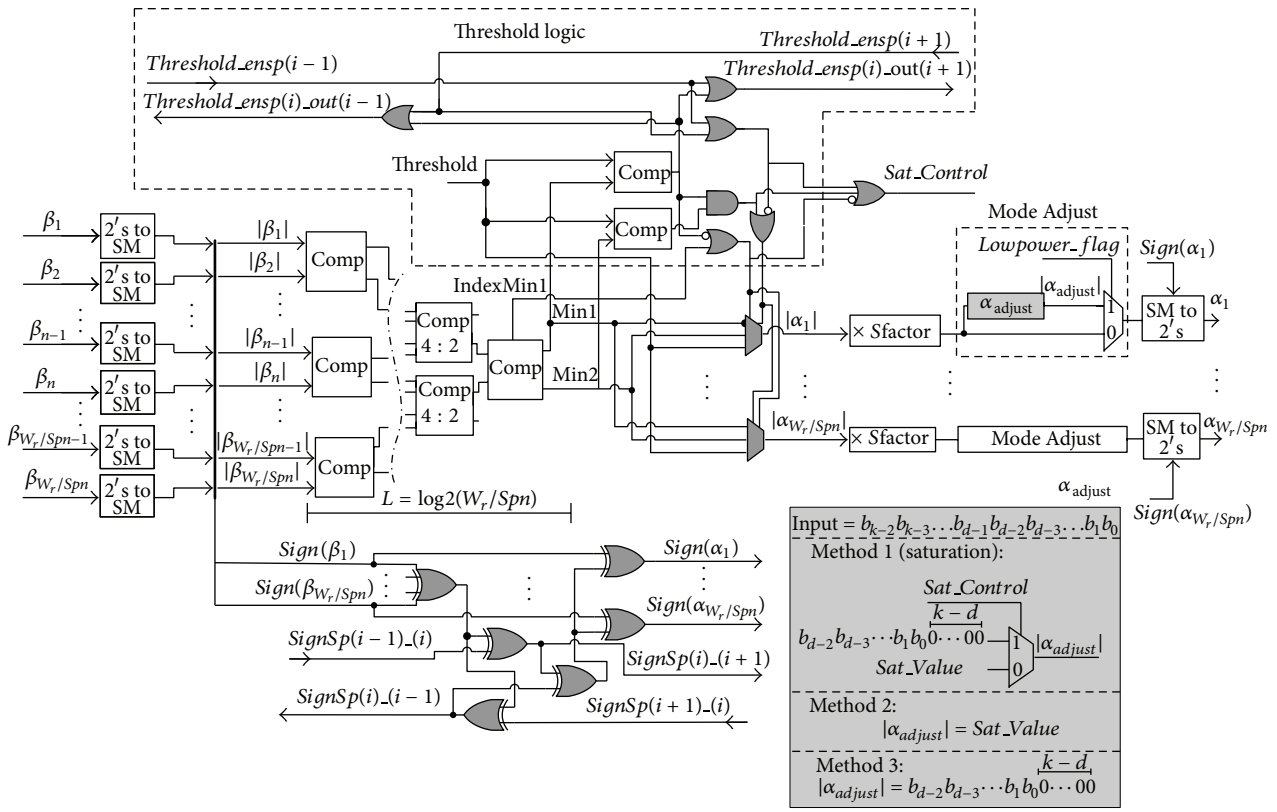

Figure 6: Check node processor design for the proposed adaptive wordwidth Split-Row Threshold decoder. The adaptive wordwidth logic is shown in *α Adjust* block (shaded box).

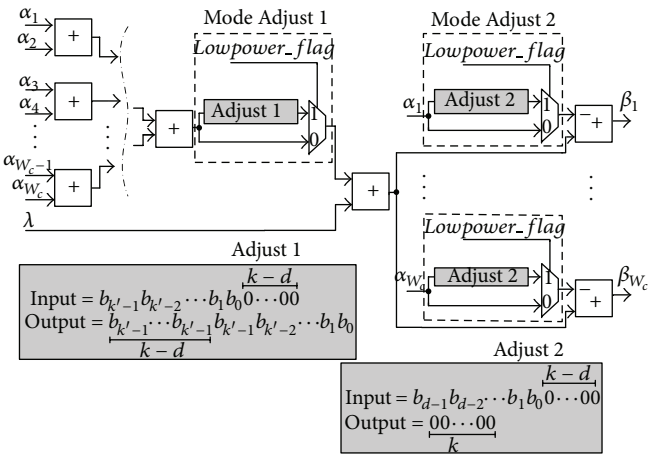

Figure 7: Variable node processor design for the proposed adaptive wordwidth decoder.

$N = 2048$) in the decoder. Two adjustments (conversion steps) are performed to make the variable node processor operate correctly in both *Normal Mode* and *Low Power Mode*. *Mode Adjust 1* is made before adding the sum of W_c variable node inputs (α) to the channel information, λ, which shifts the addition result bits back to their original LSB positions (Recall that α bits were shifted from k–d positions to the left at the end of check node processing). *Mode Adjust 2* is made in the subtraction stage, where α bits are kept zero (their initial value) in order to bypass the subtraction in *Low Power Mode*.

5. Design of CMOS Decoders

To further investigate the impact of the proposed decoder on the hardware, we have implemented three full parallel

decoders using Methods 1, 2, and 3 for the (6,32) (2048,1723) 10GBASE-T LDPC code in 65 nm 7-metal layer CMOS.

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

(1) Choosing the number of partitioning (*Spn*), Threshold (T), and *S*factor values: it is shown that the routing congestion, circuit delay, area, and power dissipation reduce as the number of partitions increases with a modest error performance loss [14]. The Threshold (T) and *S*factor which directly affect the error performance are found through empirical simulations. For the 10GBASE-T decoder design, *Spn* is set to 16, and the closest fixed-point values for T and *S*factor which attain a near optimum floating-point performance are both 0.25.

(2) Number of supported wordwidths: as discussed in Section 3, when using Split-Row Threshold, check node messages (α) are largely concentrated at $\pm T \times$ *S*factor at low iteration counts and low SNR values, (e.g., more than 80% for 10GBASE-T). Therefore, it naturally makes sense to define two regions, where one region represents α values in $\pm T \times$ *S*factor which we call Threshold Region or Low Power Mode region and the other which represents the majority of α values and we call *Normal Mode*. As long as there is no significant region in the distribution of α values, increasing the number of regions (more wordwidth representation selection) is not efficient due to the large hardware overhead and error performance loss of introducing another mode into all check and variable node processors. For example, if we want to add one more region it requires an additional global signal to choose between regions. It also adds additional comparators to select the region (mode) that α can fit in and requires us to increase the size of the muxes to choose between the outputs.

(3) Normal Mode wordwidth selection: this is the major datapath width of the decoder and is chosen to optimize the error performance with minimum hardware. The BER performance simulations for the (2048,1723) 10GBASE-T LDPC code using Split-Row Threshold indicate that the minimum wordwidth for fixed-point implementation which attains the near floating point error correction performance is 6 bit (0.03 dB gap at BER = 10^{-7}). Therefore, $k = 6$ for our implementation.

(4) *Low Power Mode* wordwidth selection: this is the subset of Normal Mode wordwidth where the Threshold Region ($\pm T \times$ *S*factor) values can be represented. For the 10GBASE-T code, the Threshold Region is within $\pm T \times$ *S*factor = ± 0.0625. Therefore, its values in 6-bit (1.5 format) quantization are -0.0625, -0.03125, 0, $+0.03125$, and $+0.0625$. These values can be represented with a 3-bit subset. Figure 8 shows the check node output (α) distribution using Split-Row Threshold decoder for (2048,1723) LDPC code which is binned into discrete values set by 6-bit (1.5 format) quantization. The 3-bit subset can cover all values within the Threshold Region. Representation with less bits, such as a 2-bit subset that is shown in the figure, will miss some values of the Threshold Region. Also there is no benefit if we use a 4-bit subset because the additional values represented by the 4-bit subset are not within the Threshold Region. Therefore, $d = 3$ for our implementation.

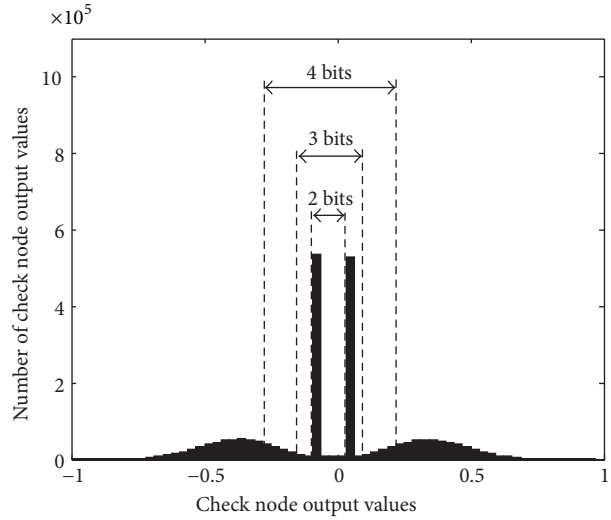

Figure 8: Check node output (α) distribution using Split-Row Threshold decoder for (2048,1723) LDPC code, which are binned into discrete values set by a 6-bit (1.5 format) quantization. The 3-bit subset can cover all values within the *Threshold Region*. Data are for SNR = 4.4 dB and *iteration* = 3, where T = *S*factor = 0.25.

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

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

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

Table 2: Comparison of hardware increase in check node processor and variable processor with synthesis area for the three low power "Methods". (For the Split-Row Threshold design none of these methods are applied).

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

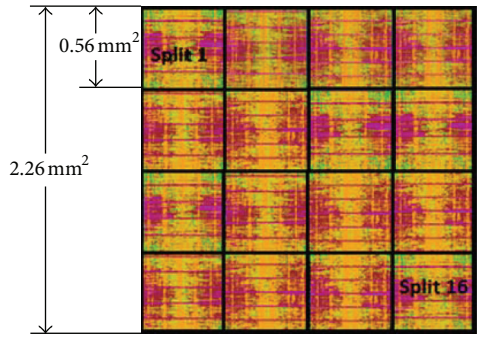

Figure 9: Post-layout view of the proposed 10GBASE-T adaptive wordwidth decoder with Method 1.

fed back into SoC Encounter to compute a simulation-based power analysis. This analysis is performed for 100 test vectors for each SNR.

The chip layout of Methods 1 is shown in Figure 9. A summary of the post-layout results for the low power proposed Method 1, 2, and 3 decoders, when *Low_Power_Iteration* = 6, is given in Table 3. For comparison a Method 1 decoder only running in *Normal Mode* is included in the table.

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

The table also summarizes the power results for the case that decoders in three methods are kept in *Low Power Mode* for 6 iterations and *Normal Mode* for 9 iterations out of a total $I_{max} = 15$ iterations. Energy data are reported for 15 decoding iterations without early termination at SNR = 3.6 dB. Under these conditions, Method 2 has the smallest energy dissipation per bit, 46 pJ/bit, which is 20% lower than running only with *Normal Mode*. Overall, the average power among the three methods is 1172–1215 mW, which is 181–224 mW lower than when running on only *Normal Mode*.

Figure 10 shows the power breakdown for Method 2 in *Normal Mode* only, *Low Power Mode* only, and adaptive mode (*Low_Power_Iteration* = 6 out of 15 total iterations). Shown are the power contributions from variable node processors, check node processors, and the clock tree (including registers). By itself, *Low Power Mode* results in 41% reductions when compared to *Normal Mode* only. For an adaptive mode where *Low_Power_Iteration* = 6 iterations out of a total 15 iterations, this results in a net improvement of 22% in average power. Therefore, it is important to realize the tradeoff between the amount of *Low Power Mode Iterations* versus the number of convergence iterations (i.e., average iterations from early termination).

Energy gains are dependent on the *Low_Power_Iteration* since the desired BER performance (depends on I_{max} as discussed later) and the convergence behavior (early termination and average iterations) of the proposed decoders also depend on the *Low_Power_Iteration*. The longer the *Low Power Mode* is enabled, the longer it will take to converge, and as a result the energy becomes dependent on both a tradeoff of the set *Low_Power_Iteration* and the final convergence iteration count. Figure 11 shows the energy consumption for Methods 1, 2, and 3 when the the *Low Power Mode* is enabled for three and six iterations over a range of SNR values: 2.2–4.6 dB. Notice that for *Low_Power_Iteration* = 6 the energy starts to become worse for SNR $\geq$ 4.0 because of longer average convergence times (i.e., larger average iterations).

5.5. SNR Adaptive Design. In Split-Row Threshold, a larger maximum number of iterations, I_{max}, can improve bit error performance. This is shown by running on *Normal Mode* only while using $I_{max} = 25$. In this case, BER performance of the proposed decoder is only 0.2 dB away from MinSum Normalized at BER = 10^{-9} (a significant BER improvement is not observed for $I_{max} > 25$). Although higher maximum iteration count has almost no effect on the average iterations at high SNRs, it increases the average iterations at low SNRs [15] (more of the channel information is corrupted beyond the ability for LDPC to correct), which results in higher energy dissipation. Given the fact that running in *Low Power Mode* at low SNRs results in larger energy savings it is more beneficial to use a larger *Low_Power_Iteration* with lower I_{max}. Conversely, we can use only *Normal Mode* with a higher

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

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

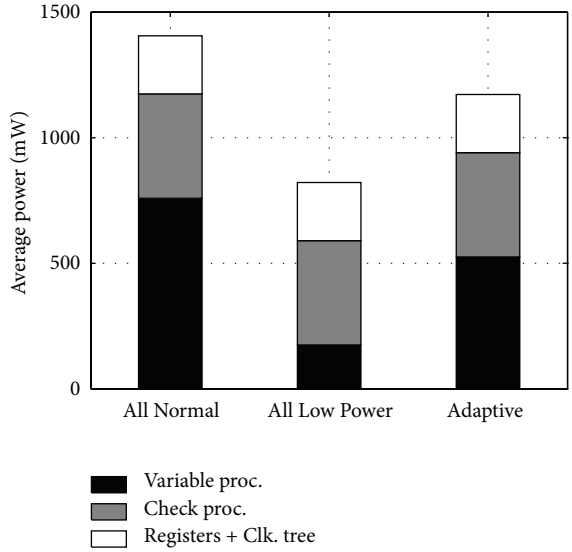

Figure 10: Power breakdown for Method 2: *Normal Mode* only, *Low Power Mode Mode* only, and adaptive mode (6 iterations with *Low Power Mode* and 9 iterations with *Normal Mode*).

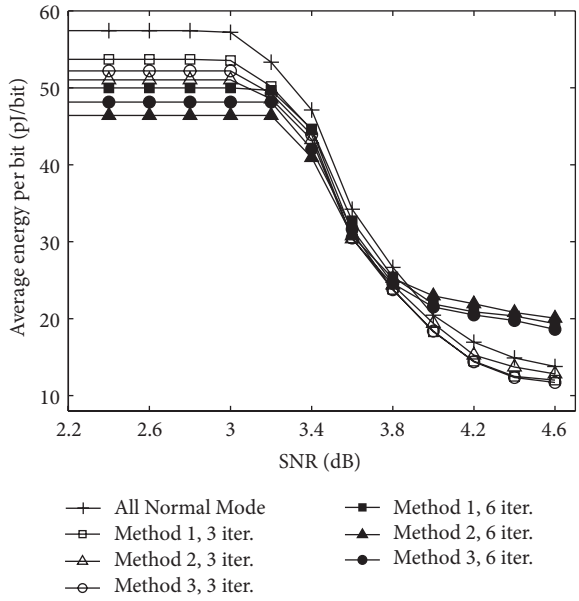

Figure 11: Energy per bit versus SNR for different low power decoder designs and different *Low_Power_Iteration*, compared with a design only running in *Normal Mode*.

maximum iteration count to get the BER required at high SNR with lesser energy penalties as compared to operating the decoder with a large *Low_Power_Iteration*.

These scenarios are illustrated in Figure 12 where the bit error performance versus energy per bit dissipation of the proposed decoder with Method 2 is shown under two conditions.

(1) Adaptive mode operation with Method 2, *Low_Power_Iteration* = 6, and $I_{max} = 15$.

(2) The decoder runs in only *Normal Mode*, and $I_{max} = 25$.

Given the worst case $I_{max} = 25$ and a 10GBASE-T LDPC decoder throughput of 6.4 Gbps, both designs are set to 0.87 V and compared with early termination enabled. As shown in the figure, when BER > 10^{-4} (implying a low SNR) the energy dissipation of Method 2 decoder is about 20%–50% lower than that of the decoder in *Normal Mode* at the same BER. However, when the BER < 10^{-6} (SNR > 4.0 dB), the decoder at *Normal Mode* attains greater than an order of magnitude improvement in BER at nearly the same energy per bit dissipation.

Therefore, using an efficient SNR detector circuit, we can switch between different modes at SNR = 4.0 dB. Similar to [33], the proposed SNR detector compares the number of unsatisfied checks with a checksum threshold at the end of the first iteration and estimates the SNR range. For the 2048-bit 10GBASE-T code, it was found that a checksum threshold of 91 after the first iteration can estimate if the SNR is larger or smaller than 4.0 dB with a probability of being 89% true. By using this detection scheme the *Low Power Mode* iteration count and I_{max} can be adjusted. The SNR detector circuit requires only one additional comparator in the early termination circuit.

5.6. Comparison with Others. The post-layout simulation results of the proposed wordwidth adaptive decoder using Method 2 are compared with recently implemented decoders [15, 30–32] for 2048-bit LDPC codes and are summarized in Table 4. The 10GBASE-T code is implemented in [15, 30, 31]. Results for two supply voltages are reported for a Method 2 decoder: 1.3 and 0.7 V. (Note that, at 0.7 V, for $I_{max} = 15$, the

Table 4: A comparison of the proposed adaptive decoder using the wordwidth adaptive Method 2 decoder with recently published LDPC decoder implementations.

	Liu and Shi [30]	Ueng et al. [31]	Mansour and Shanbhag [32]	Mohsenin et al. [14]	Zhang et al. [15]		This work	
Technology	90 nm, 8M	90 nm	180 nm	65 nm	65 nm, 7 M		65 nm, 7 M	
Implementation	P&R	P&R	Measured	P&R	Measured		P&R	
Architecture	partial parallel	partial parallel	partial parallel	full parallel	partial parallel		full parallel	
Decoding Alg.	SMP	Shuffled MPD	TDMP	Split-Threshold	TPMP		Adaptive wordwidth[1]	
Code Length	2048	1536–3968	2048	2048	2048		2048	
Edges	12288	7680–23808	—	12288	12288		12288	
Code Rate	0.84	0.79–0.93	0.5	0.84	0.84		0.84	
Bits per message	5	—	—	5	4		6	
Logic utilization	50%	—	50%	95%	80%		96%	
Chip area (mm^2)	14.5	4.41	14.3	4.55	5.35		5.10	
Max. iterations (I_{max})	16	8, 4	16	11	8		15	
Supply voltage (V)	—	1.0	1.8	1.3	1.2	0.7	1.3	0.7
Clock speed (MHz)	207	303	125	195	700	100	185	40
Maximum Latency (ns)	—	—	—	56.4	137	960	81	375
Throughput @ I_{max} (Gbps)	5.3	4.85, 9.7	0.400	36.3	14.9[2]	2.1[2]	25.26	5.4
Throughput w/early term. (Gbps)	—	4.85, 9.7	6.4	92.8	47.7	6.67	85.7	18.5
Throughput per area ($Gbps/mm^2$)	0.36	11, 22	—	19.1	8.9	1.2	16.8	3.6
Power (mW)	—	855	787	1359	2800	144	1172	73
Energy/bit w/early term. (pJ/bit)	—	176, 88	—	15	58.7	21.5	**13.6**	3.9

[1]This work is also a variant of Split-Threshold method. [2]Throughput is computed based on the maximum latency reported. [3]Power numbers are for *Low_Power_Iteration* = 6, Method 2.

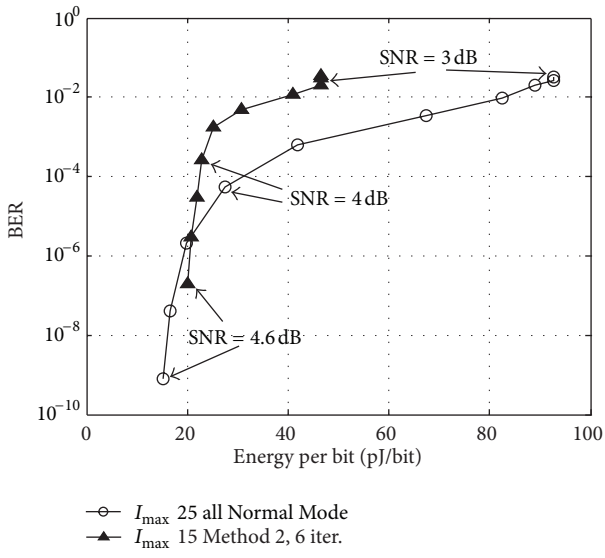

Figure 12: Bit error rate versus energy per bit dissipation of two decoders for different adaptive decoder settings to meet the 10GBASE-T standard throughput (dependent on the worst case I_{max} and maximum frequency at 0.87 V).

10GBASE-T required throughput is met.) The supply voltage can be lowered to 0.6 V based on a previously fabricated chip measurements [34]. At this voltage, the decoder throughput is 9.3 Gbps (greater than 6.4 Gbps required for 10GBASE-T) while dissipating an average power of 31 mW.

The sliced message passing (SMP) scheme in [30] is proposed for Sum-Product algorithm, divides the check node processing into equal size blocks, and performs the check node computation sequentially. The post-layout simulations for a 10GBASE-T partial parallel decoder are shown in the table. The multirate decoder in [31] supports RS-LDPC codes with different code lengths (1536–3968 bits) through the use of reconfigurable permutators. The post-layout simulation results of a 10GBASE-T decoder are reported in 90 nm CMOS in the table. The partial parallel 2048-bit decoder chip is fabricated in 180 nm CMOS. The decoder which supports turbo-decoding massage passing (TDMP) algorithm supports multiple code rates between 8/16 and 14/16. The partial parallel decoder chip [15] is fabricated in 65 nm and consists of a two-step decoder: MinSum and a postprocessing scheme which lowers the error floor down to BER = 10^{-14}. Compared to a previous reduced wordwidth 5-bit implementation of

original Split-Row Threshold decoder [14], the proposed 6-bit decoder attains 10% improvement in energy dissipation with 15 decoding iterations. Compared to the sliced message passing decoder [30], the proposed wordwidth adaptive decoder is about 3 × smaller and has 6.8 × higher throughput with 0.2 dB coding gain reduction. Compared to the two-step decoder chip [15], the proposed decoder has 1.7 × higher throughput and dissipates 3.57 times less energy, with the same area at a cost of 0.35 dB coding gain reduction.

6. Conclusion

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

References

[1] F. Clermidy, C. Bernard, R. Lemaire et al., "A 477mW NoC-based digital baseband for MIMO 4G SDR," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC '10)*, pp. 278–279, February 2010.

[2] T. Limberg, M. Winter, M. Bimberg et al., "A fully programmable 40 GOPS SDR single chip baseband for LTE/WiMAX terminals," in *Proceedings of the 34th European Solid-State Circuits Conference (ESSCIRC '08)*, pp. 466–469, September 2008.

[3] T. Mohsenin and B. Baas, "Trends and challenges in LDPC hardware decoders," in *Proceedings of the 43rd Asilomar Conference on Signals, Systems and Computers*, pp. 1273–1277, November 2009.

[4] A. Weiss, "Computing in the clouds," *NetWorker*, vol. 11, no. 4, pp. 16–25, 2007.

[5] M. Armbrust, A. Fox, R. Griffth et al., "Above the clouds: a berkeley view of cloud computing," Tech. Rep., EECS Department, University of California, Berkeley, Calif, USA, 2009.

[6] R. Wilson, "10GBase-T: Is it really coming this time?" http://www.edn.com/article/459408-10GBase, 2009.

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

[8] S. Pope, "Look for power tradeoffs in 10GBASE-T ethernet," http://www.eetimes.com/design/power-managementdesign/4005683/Look-for-power-tradeos-in-10GBASE-T-Ethernet, 2008.

[9] R. G. Gallager, "Low-density parity check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

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

[11] IEEE 802. 16e. air interface for fixed and mobile broadband wireless access systems. ieee p802. 16e/d12 draft, 2005.

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

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

[14] T. Mohsenin, D. N. Truong, and B. M. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders," *IEEE Transactions on Circuits and Systems I*, vol. 57, no. 5, pp. 1048–1061, 2010.

[15] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolić, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, 2010.

[16] N. Onizawa, T. Hanyu, and V. C. Gaudet, "Design of high-throughput fully parallel LDPC decoders based on wire partitioning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 3, pp. 482–489, 2010.

[17] T. Mohsenin and B. Baas, "A split-decoding message passing algorithm for low density parity check decoders," *Journal of Signal Processing Systems*, vol. 61, pp. 329–323, 2010.

[18] T. Xanthopoulos and A. P. Chandrakasan, "A low-power dct core using adaptive bitwidth and arithmetic activity exploiting signal correlations and quantization," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 5, pp. 740–750, 2000.

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

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

[21] J. Chen and M. P. C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 406–414, 2002.

[22] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on reed-solomon codes with two information symbols," *IEEE Communications Letters*, vol. 7, no. 7, pp. 317–319, 2003.

[23] T. Mohsenin, D. Truong, and B. Baas, "An improved split-row threshold decoding algorithm for LDPC codes," in *Proceedings of the IEEE International Conference on Communications (ICC '09)*, June 2009.

[24] Y. Sun and J. R. Cavallaro, "A low-power 1-Gbps reconfigurable LDPC decoder design for multiple 4G wireless standards," in *Proceedings of the IEEE International SOC Conference (SOCC '08)*, pp. 367–370, September 2008.

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

[26] E. Yeo and B. Nikolić, "A 1.1-Gb/s 4092-bit low-density parity-check decoder," in *Proceedings of the 1st IEEE Asian Solid-State Circuits Conference (ASSCC '05)*, pp. 237–240, November 2005.

[27] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[28] D. Oh and K. K. Parhi, "Nonuniformly quantized min-sum decoder architecture for low-density parity-check codes," in *Proceedings of the 18th ACM Great Lakes Symposium on (VLSI '08)*, pp. 451–456, ACM, New York, NY, USA, March 2008.

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

[30] L. Liu and C. J. R. Shi, "Sliced message passing: high throughput overlapped decoding of high-rate low-density parity-check codes," *IEEE Transactions on Circuits and Systems I*, vol. 55, no. 11, pp. 3697–3710, 2008.

[31] Y. L. Ueng, C. J. Yang, K. C. Wang, and C. J. Chen, "A multimode shuffled iterative decoder architecture for high-rate RS-LDPC codes," *IEEE Transactions on Circuits and Systems I*, vol. 57, no. 10, pp. 2790–2803, 2010.

[32] M. M. Mansour and N. R. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 684–697, 2006.

[33] W. Weihuang, C. Gwan, and K. Gunnam, "Low-power VLSI design of LDPC decoder using DVFS for AWGN channels," in *Proceedings of the 22nd International Conference on VLSI Design*, pp. 51–56, January 2009.

[34] D. N. Truong, W. H. Cheng, T. Mohsenin et al., "A 167-processor computational platform in 65 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, 2009.