Hindawi Publishing Corporation
ISRN Sensor Networks
Volume 2013, Article ID 650740, 12 pages
http://dx.doi.org/10.1155/2013/650740

Research Article

Low Power Decoding of LDPC Codes

Mohamed Ismail, Imran Ahmed, and Justin Coon

Telecommunications Research Laboratory (TRL), Toshiba Research Europe Ltd., 32 Queen Square, Bristol BS1 4ND, UK

Correspondence should be addressed to Mohamed Ismail; mohamed.ismail@toshiba-trel.com

Received 27 November 2012; Accepted 19 December 2012

Academic Editors: M. Ekström and Y. Yu

Wireless sensor networks are used in many diverse application scenarios that require the network designer to trade off different factors. Two such factors of importance in many wireless sensor networks are communication reliability and battery life. This paper describes an efficient, low complexity, high throughput channel decoder suited to decoding low-density parity-check (LDPC) codes. LDPC codes have demonstrated excellent error-correcting ability such that a number of recent wireless standards have opted for their inclusion. Hardware realisation of practical LDPC decoders is a challenging area especially when power efficient solutions are needed. Implementation details are given for an LDPC decoding algorithm, termed adaptive threshold bit flipping (ATBF), designed for low complexity and low power operation. The ATBF decoder was implemented in 90 nm CMOS at 0.9 V using a standard cell design flow and was shown to operate at 250 MHz achieving a throughput of 252 Gb/s/iteration. The decoder area was 0.72 mm^2 with a power consumption of 33.14 mW and a very small energy/decoded bit figure of 1.3 pJ.

1. Introduction

Wireless sensor networks (WSN) have been deployed in many applications, from oceanography, environmental monitoring, understanding wildlife behaviour to intrusion detection, and building structural analysis, see [1]. WSNs are composed of a number of cheap, small, resource-limited nodes with some form of sensor and wireless communication capability. As such, careful design of the overall network is necessary to ensure sufficient robustness of the network to node failures, deleterious communications, and sensor data correlation. The radio system can be a major source of power drain compared to other systems within the sensor node, as stated in [2]. Of the many functions performed within a radio system, channel decoding is one of the most computationally intensive and, thus, resource demanding.

Low-density parity-check (LDPC) channel codes were described by Gallager [3] in 1963 and subsequently rediscovered by Mackay [4]. LDPC codes are capacity approaching codes, coming within 0.0045 dB of the Shannon capacity limit [5], whilst being attractive from an implementation viewpoint due to the availability of iterative decoding algorithms.

In addition to describing error-correcting codes based on low-density parity-check matrices, Gallager presented a number of practical decoding algorithms. A probabilistic soft decision message passing algorithm referred to as the sum-product (SP) algorithm capable of achieving optimum performance was one class of decoding algorithm. Although SP techniques have been shown to perform exceptionally well [5], they do suffer from implementation complexity. In [6] for a fully parallel SP decoder, three major implementation problems are highlighted as follows: (1) routing bottleneck caused by large wire-dominated circuit delays due to mirroring of the LDPC matrix connectivity, (2) significant memory storage for messages, (3) large number of processing elements due to large code lengths. Thus, partially parallel solutions based on structured codes have been proposed [7, 8] that address some of these problems but generally have lower throughput than a fully parallel implementation. An alternative decoding algorithm, falling into the class of *bit flipping* (BF) methods was designed to be less complex than SP decoding. One such method, termed adaptive threshold bit flipping (ATBF) was described in [9], designed for low power, high throughput operation.

The work in this paper reports on the implementation of the ATBF algorithm deriving figures for throughput, power consumption and silicon area. This paper is organised as follows. Section 2 presents a review of simplified hard

and soft decision decoding algorithms. Section 3 provides details of the adaptive threshold bit flipping algorithm and a method for reducing power consumption. Section 4 describes a criteria for early stopping of decoding of the ATBF algorithm. Simulation results for the ATBF and early stopping ATBF (ES-ATBF) algorithms are compared to published bit flipping methods in Section 5. Section 6 describes the VLSI architecture of the ATBF decoder, highlighting the issue of routing and providing figures for power consumption, area, and timing analysis. Section 7 concludes this paper.

2. Background

2.1. Simplified Hard Decision Decoding. Simplified decoding methods are broadly categorised into hard and soft decision algorithms. The majority of bit flipping techniques fall into the category of soft decision decoding methods and form the basis of a novel decoding algorithm and subsequent improvements as presented in this paper.

Let $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ be the soft-valued received vector of the transmitted codeword $\mathbf{c}$ and $\mathbf{x}$ be the binary hard decision of $\mathbf{y}$. Let $N(i) = \{j : h_{ij} = 1\}$ for $0 < j \leq N$ be the index of nonzero elements in row i and $M(j) = \{i : h_{ij} = 1\}$ for $0 < i \leq M$ be the index of nonzero elements in column j of the parity-check matrix. Furthermore, let $\mathbf{e}$ be the error vector associated with the received hard decision codeword $\mathbf{x}$ such that $\mathbf{x} = \mathbf{c} + \mathbf{e}$. Let the row and column weights of the parity-check matrix be w_r and w_c, respectively. The syndrome, $\mathbf{s}$, of the received bits can be written as

$$\mathbf{s} = \mathbf{x}\mathbf{H}^T. \tag{1}$$

Since $\mathbf{x} = \mathbf{c} + \mathbf{e}$ and $\mathbf{c}\mathbf{H}^T = \mathbf{0}$, (1) can be rewritten as

$$\mathbf{s} = \mathbf{e}\mathbf{H}^T. \tag{2}$$

2.1.1. Gallager Bit Flipping. The number of unsatisfied syndrome checks on a given bit give one measure of the reliability associated with the bit. By performing the syndrome check in (1), forming a parity-check failure count and flipping the bits involved in some number of parity-check failures above a preset threshold, the basic steps of the hard decision Gallager bit flipping algorithm are described. The complete bit flipping algorithm described by Gallager [3] can be summarised as shown in Algorithm 1.

For a BSC channel, Gallager [3] derived the optimum threshold value, t, to be used in the bit flipping algorithm.

2.2. Simplified Soft Decision Decoding. It is intuitive to understand how the use of additional reliability information derived from soft decisions of the received data could be used to improve decoding performance of the bit flipping technique. The precise way in which this soft information is utilised in the decoding process has led to a number of different bit flipping algorithms being developed, some of which are described below.

2.2.1. Weighted Bit Flipping. In addition to describing a geometric method of constructing LDPC codes, the authors of [10] describe a method for incorporating reliability information into the one-step majority-logic decoding and Gallager's bit flipping algorithms. Let an LDPC code be described by a parity-check matrix H consisting of M rows, $[\mathbf{h}_1, \ldots, \mathbf{h}_M]$ and N columns. For $1 \leq l \leq N$ and $1 \leq j \leq M$, let

$$\omega_{j,l} = \min\left\{|y_i| : 1 \leq i \leq N, h_{j,i} = 1\right\}, \tag{3}$$

$$E_l = \sum_{s_j^{(l)} \in S_l} \left(2s_j^{(l)} - 1\right)\omega_{j,l}, \tag{4}$$

where S_l is the set of parity-checks on bit position l and y_j is the received value at bit position j. The term in brackets in (4) is a bipolar, $[+1, -1]$, representation of the syndrome value. Thus, E_l represents a weighting of a check sum by the least reliable received value participating in the check sum resulting in E_l having the range $[-w_c\omega_{j,l} : +w_c\omega_{j,l}]$. The weighted bit flipping algorithm (WBF) can be stated as shown in Algorithm 2.

Clearly, weighted bit flipping requires handling of soft values, specifically real addition operations which makes it an inherently more complex algorithm than Gallager bit flipping. For a WBF implementation, the number of real addition operations per iteration is given as $2K_{\mathrm{BF}}E$, where K_{BF} is an implementation dependent constant typically less than three [11] and E is the total number of edges in the parity-check matrix. Additionally, the WBF algorithm requires storage of the $M \times \omega_{j,l}$ values. However, the additional decoding complexity does provide a coding gain over the hard decision Gallager bit flipping algorithm.

2.2.2. Modified Weighted Bit Flipping. In the weighted bit flipping formulation given in (4), the $\omega_{j,l}$ term weights the check sum by the least reliable received value participating in the check sum. In [12] the authors interpret (4) as a message about the hard decision x_l based on the check sum values which is iteratively updated. Their observation and subsequent modification is that (4) gives an indication of the extent to which x_l should change its value based on the information provided by the check values whilst ignoring to what extent x_l should maintain its value based on the information provided by $|y_l|$. The modified weighted bit flipping (MWBF) algorithm changes (4) to include the y_l term as follows:

$$E_l = \sum_{s_j^{(l)} \in S_l} \left(2s_j^{(l)} - 1\right)\omega_{j,l} - |y_l|. \tag{5}$$

Clearly, as stated in [12], the weight of $|y_l|$ will vary with different values of signal-to-noise ratio and with different row weights E_l will also vary. Thus, a weighting factor is added to the $|y_l|$ term in (5) to give

$$E_l' = \sum_{s_j^{(l)} \in S_l} \left(2s_j^{(l)} - 1\right)\omega_{j,l} - \alpha|y_l|, \tag{6}$$

where α is a positive real value. In [12], it is observed that the optimal value of α increases with the column weight, regardless of the increase in code length. Compared to the WBF algorithm, the MWBF algorithm shows no appreciable

(1) **For** $j = 1 : N$,
$x_j = \text{sign}(y_j)$
(2) **Let** $s_j = \sum_{i \in N(j)} x_i$ over $\mathbb{F}^2$ for all $j \in \{1, \ldots, M\}$.
(3) **If** $\sum_j s_j = 0$ for all $j \in \{1, \ldots, M\}$,
output $\mathbf{x}$ and stop.
(4) **Form** a count, f_j, of the number of failed parity checks for all $j \in \{1, \ldots, M\}$.
(5) **If** the count f_j for any bit j exceeds a threshold t,
change the sign of the bit j.
(6) **If** the maximum number of iterations is reached
output $\mathbf{x}$ and stop.
otherwise
go to step 2.

Algorithm 1: Gallager bit flipping.

(1) **For** $j = 1 : N$,
$x_j = \text{sign}(y_j)$
(2) **Let** $s_j = \sum_{i \in N(j)} x_i$ over $\mathbb{F}^2$ for all $j \in \{1, \ldots, M\}$.
(3) **If** $\sum_j s_j = 0$ for all $j \in \{1, \ldots, M\}$,
output $\mathbf{x}$ and stop.
(4) **Compute** E_l in (4) for $l \in \{1, \ldots, N\}$.
(5) **Find** the bit position k for which $E_k = \max\{E_l : 0 < l \leq N\}$,
change the sign of the bit at position k.
(6) **If** the maximum number of iterations is reached
output $\mathbf{x}$ and stop.
otherwise
go to step 2.

Algorithm 2: Weighted bit flipping.

difference in the average number of decoding iterations for codes of short length. However, for longer length codes, the MWBF requires fewer iterations than the WBF as such codes tend to converge slower [12].

2.2.3. Gradient Descent Bit Flipping. Recently, Wadayama [13] formulated a new way of calculating the decision metric for flipping a given bit based on the *gradient descent* formulation. The gradient descent bit flipping (GDBF) algorithms outperform the WBF and MWBF algorithms in error-correcting ability and more significantly in the number of average iterations needed for successful decoding.

If the elements of the hard decision vector $\mathbf{x}$ are represented in bipolar form, $[+1, -1]$, the maximum likelihood decoder can be represented as

$$\begin{aligned} \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \sum_i (y_i - x_i)^2 \\ &= \arg\min_{\mathbf{x}} \sum_i \left(y_i^2 - 2x_i y_i + 1\right) \\ &= \arg\max_{\mathbf{x}} \sum_i x_i y_i. \end{aligned} \tag{7}$$

That is, the ML solution is that codeword which gives the largest correlation to the received vector $\mathbf{y}$. This is the basis used in [13] for deriving the correlation decoding rule, where the following objective function is defined:

$$f(\mathbf{x}) = \sum_{i=1}^{N} x_i y_i + \sum_{i=1}^{M} \prod_{j \in N(i)} x_j. \tag{8}$$

The correlation term in the objective function should be maximised as per the ML formulation. The second term is equivalent to the modulo 2 summation of the binary representation of $\mathbf{x}$ for the bits in the codeword, that is, it is the bipolar syndrome of $\mathbf{x}$. When the parity-check involving $\mathbf{x}$ is satisfied this term will have a maximum value equal to the degree of the check node. The gradient descent inversion function is based on the partial derivative of $f(\mathbf{x})$ with respect to the variable x_i and can be stated as

$$\Delta_k^{(\text{GD})}(\mathbf{x}) = x_k y_k + \sum_{i \in M(k)} \prod_{j \in N(i)} x_j. \tag{9}$$

Based on (9) the single-step GDBF algorithm can be stated as shown in Algorithm 3. The single-step GDBF

(1) **For** $j = 1 : N$
  $x_j = \text{sign}(y_j)$
(2) **If** $\prod_{j \in N(i)} x_j = +1$
  for all $i \in \{1, \ldots, M\}$, output $\mathbf{x}$ and stop
(3) **Flip** bit x_l where, $l = \arg\min_{k \in \{1,\ldots,n\}} \Delta_k^{(\text{GD})}(x)$
(4) **If** the maximum number of iterations is reached
  output $\mathbf{x}$ and stop
 otherwise
  go to step 2

Algorithm 3: Single-step gradient descent bit flipping.

algorithm flips only a single bit in each iteration which results in a large number of iterations before the objective function reaches a local maximum. To address this problem, a second algorithm is proposed in [13] termed the multibit flipping gradient descent (MGDBF) algorithm which modifies the condition in step 3 such that all bits with $\Delta_k^{(\text{GD})} < \theta$ are flipped, where θ is the *inversion* threshold. This approach has the advantage of achieving faster convergence due to the larger step size arising from flipping more than a single bit. However, as noted in [13], when close to a local maximum, a large step size is not suitable and thus the algorithm has to drop down to single-bit flipping mode. This switch in operating modes is facilitated by evaluating the behaviour of an *objective* function, shown in (10), and comparing to a threshold as

$$f(x) = \sum_{k=1}^{n} x_k y_k + \sum_{i \in M(k)} \prod_{j \in N(i)} x_j. \tag{10}$$

The multistep GDBF algorithm achieves better error-correcting performance than both the WBF or MWBF algorithms whilst using fewer iterations at a cost of additional complexity.

3. Adaptive Threshold Bit Flipping (ATBF)

All the soft-decision bit flipping algorithms described in the previous section have a key weakness when a very high speed or low power hardware implementation is required. They all require, in one form or another, locating a minimum (or maximum) value over the whole block length. Which in hardware requires $\log_2(N)$ sequential comparator stages and, since N is large for LDPC codes this forms a major bottleneck to practical high speed decoding and low power operation. Thus, the motivation for developing a novel bit flipping algorithm came from this observation as described in [9].

The gradient descent bit flipping algorithms seem to give the best error-correcting performance with the fewest decoder iterations compared to the other bit flipping algorithms in the literature. In addition, in deriving the gradient descent formulation, an elegant framework was established allowing the other major bit flipping algorithms to be mathematically represented. Hence, using the single-step GDBF algorithm as a starting point, the adaptive threshold bit flipping (ATBF) algorithm was derived as described next.

Let λ_k, $k \in \{1, \ldots, N\}$ be a negative threshold value associated with each received bit and $\theta \in [0, 1]$ a constant scaling factor used to modify λ_k. The adaptive threshold bit flipping algorithm can then be stated as shown in Algorithm 4, where λ_0 is the initial threshold value.

The ATBF algorithm replaces step 3 of the single-step GDBF algorithm by flipping a bit when the inversion function associated with the bit has a value below some threshold. If the inversion function value associated with a bit, $\Delta_k^{(\text{GD})}$, is not below the threshold, λ_k, the threshold is scaled using a constant factor, θ. A beneficial consequence of thresholding on a per bit level is that multiple bits will be flipped to start with progressing to fewer flips as most checks are satisfied. Thus, ATBF intrinsically moves from multiple bit flipping to single bit flipping as necessary. Using the proposed modification removes the need to locate a minimum value over the entire block length, instead requiring only localised operations.

3.1. Reduced Power Decoding. With each iteration of the ATBF algorithm, step 4 will result in either a change of sign of x_k or the lowering of $|\lambda_k|$. As $\lambda_k \to 0$ the probability of a sign change will become smaller leading to scaling of λ_k being the dominant operation. Thus, detecting the point at which a particular bit processor enters into this state and terminating further computation should lead to saving in power consumption.

To determine when a bit processor is unlikely to change the sign of a bit, a flipping threshold, ϕ, is introduced against which a given λ_k is compared. If $\lambda_k \geq \phi$ the bit processor is placed in a quiescent state and no further operations are undertaken. Determining the optimal flipping threshold value was deemed too complex as it is a function of the signal-to-noise ratio and the LDPC code characteristics therefore a empirical approach was adopted. With the maximum number of permitted iterations set to a high value, 100, the variation of λ_k was averaged over at least 1000 blocks in an additive white Gaussian noise (AWGN) channel over a range of SNR values. The LDPC code used was the regular-(3,6), half-rate (504,1008) code taken from [14] referred to as PEGReg504x1008. Figure 1 shows how the mean of each of the 1008 threshold values changes as a function of SNR,

(1) **Initialise** $\lambda_k = \lambda_0, k \in \{1, \ldots, N\}$
(2) **For** $j = 1 : N$
    $x_j = \text{sign}(y)$
(3) **If** $\prod_{j \in N(i)} x_j = +1$
    output **x** and stop
(4) **For** $k = 1 : N$
    If $\Delta_k^{(\text{GD})} < \lambda_k$
        flip bit x_k
    otherwise
        $\lambda_k = \theta \lambda_k$
(5) **If** the maximum number of iterations is reached
    output **x** and stop
  otherwise
    go to step 3

Algorithm 4: Adaptive threshold bit flipping (ATBF).

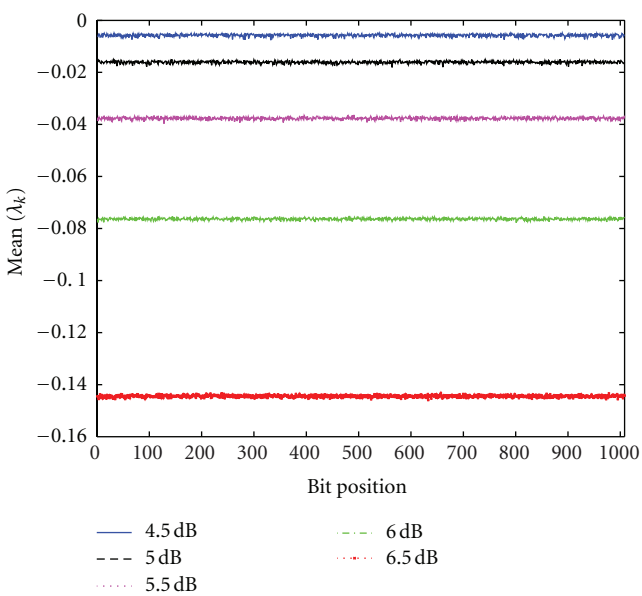

Figure 1: Mean λ value as a function of SNR, $\theta = 0.25$, $\lambda_0 = -10$.

averaged over at least 1000 blocks. Let $\overline{\lambda}_k = (1/S) \sum_{j=1}^{S} \lambda_{j,k}$, $k \in \{1, \ldots, N\}$ and S is the number of blocks simulated. For a given signal-to-noise ratio, we define $\overline{\lambda}_{\text{SNR}} = \sum_{k=1}^{N} \overline{\lambda}_k$. Using the $\overline{\lambda}_{\text{SNR}}$ values, a fourth-order least squares polynomial fit is applied to derive a relationship between the mean threshold value reached in decoding and the SNR value. A flipping threshold, ϕ_{SNR}, based on the polynomial shown in (11) is defined as the threshold value for which a bit processor is put in a quiescent state, where a is the SNR in dB as

$$\phi_{\text{SNR}} = -0.005a^4 + 0.089a^3 - 0.666a^2 + 2.258a - 2.897. \tag{11}$$

For a practical implementation, the scaling factor applied to the λ values is made a power of two, $\theta = 2^d$, where $\{d \in \mathbb{Z}^-\}$. This permits the threshold values to be scaled in hardware by a simple shift operation. In such an implementation, the flipping threshold, ϕ, can be derived from ϕ_{SNR} by setting ϕ to the nearest value to ϕ_{SNR} achievable by a θ scaling as follows:

$$z = \left\lceil -\log_2 \left(\frac{\phi_{\text{SNR}}}{\lambda_0} \right) \right\rceil, \tag{12a}$$

$$z' = -(z + \text{mod}(z, d)), \tag{12b}$$

$$\phi = \lambda_0 2^{z'}. \tag{12c}$$

Figure 2 shows the average number of values with $\lambda_k = \phi$ as they vary with each iteration for different SNR values. We estimate the power saving possible using the method described by taking the point in Figure 2 at which the mean number of $\lambda_k = \phi$ values becomes nonzero for a given SNR. All previous iterations prior to this point will use the full 1008 bit processors and all iterations thereafter will use, at most, 1008 minus the mean number of bit processors indicated by the curve at this point. The upper limit on the number of iterations at a given SNR point is taken from Figure 5, which gives the average number of decoding iterations as a function of SNR for different decoding algorithms. Figure 3 shows the estimated percentage power saving as a function of SNR (dB) when using the power reduction method described. At 3 dB we observe a power saving of 76% over the standard ATBF algorithm which reduces as the SNR increases. It is interesting to observe the power saving going to zero beyond 5 dB, this suggests it may be possible to reduce the maximum allowable iterations below the observed values.

4. Early Stopping of Decoding

In decoding of LDPC codes, it is well known the majority of bits are decoded in the first few iterations leaving a small number of bits to drive on the decoding process without successful decoding. Thus, it is desirable to detect such undecodable blocks and terminate the decoding process to save time and energy. In [15], a good overview of various criterion which can be used for early stopping of turbo decoders is presented. Some of the methods have also been

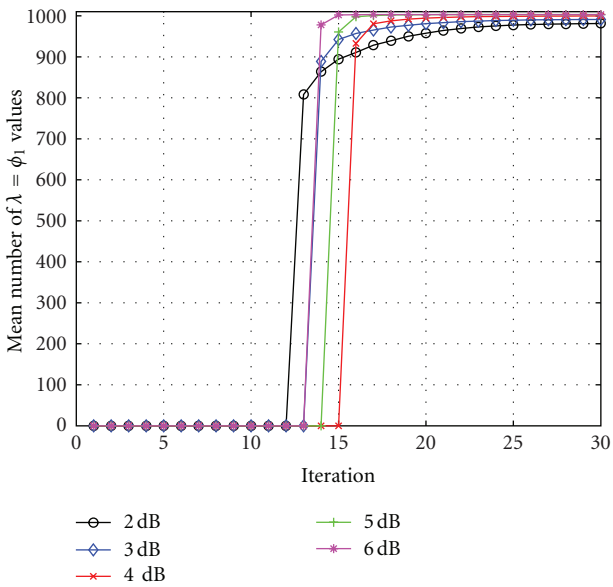

Figure 2: Mean number of $\lambda = \phi$ values as a function of iterations at different SNR values, $\theta = 0.25$, $\lambda_0 = -10$.

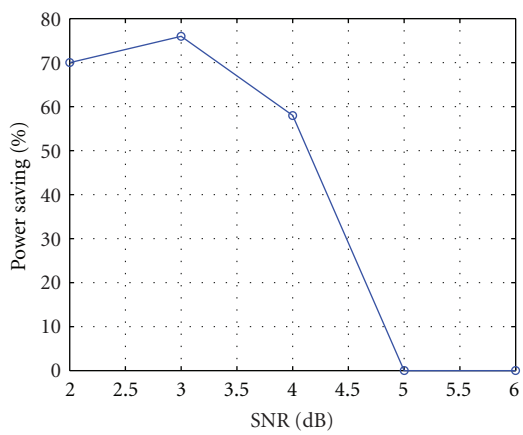

Figure 3: Estimated saving in power as a function of SNR for PEGReg504x1008 code using the power reduction method.

used in studies into early stopping of LDPC decoders. Much of the existing literature addresses the problem of early stopping when using the belief propagation decoding algorithm. Such methods involve taking some measure of the message reliability [16, 17], at the check or variable node, or a combination of the two [18].

The previous section introduced the concept of a flipping threshold used to determine when a node processor is placed in a quiescent state. A further observation from Figure 2 should allow for early termination of decoding without significant impact on the error-correcting ability of the ATBF algorithm. The near step change seen in Figure 2 indicating no threshold values reaching the flipping threshold prior to 13 iterations followed by a majority of threshold values reaching the flipping threshold suggests it should be possible to terminate decoding at this point. This is the case because Figure 1 gives the flipping threshold value which is essentially the steady-state value.

To keep latency to a minimum, we want to avoid use of operations requiring computation over the block length, such as a summation or minimum operation. Thus, based on Figure 2, we define a simple, bit-local metric for determining when to terminate the decoding process. When any $\lambda_k \geq \phi$ the decoding process is terminated, this is a straightforward edge detection of the transition from zero values with $\lambda_k \geq \phi$ to many values for which $\lambda_k \geq \phi$. The ATBF algorithm is modified to incorporate the early stopping criterion as shown in Algorithm 5.

5. Simulation Results

The PEGReg504x1008 code from [14] was used to encode a random binary sequence, BPSK modulated and passed through an additive white Gaussian noise (AWGN) channel. The bit error rate (BER) and average number of decoding iterations for the ATBF and ES-ATBF algorithms are compared to the WBF, MWBF, single-step GDBF, and multistep GDBF algorithms in Figures 4 and 5, respectively.

Both the ATBF and ES-ATBF algorithms BER performance is very close to the GDBF algorithms being within 0.25 of the single-step and 0.5 dB of the multistep algorithm. The advantage of correctly selecting the stopping point is evident from the ES-ATBF algorithm showing almost no loss in performance over the ATBF algorithm. The ES-ATBF algorithm uses, at most, eleven iterations, compared to the other algorithms which are permitted up to 100 iterations, a reduction of 89%.

6. ATBF Decoder Architecture

In this design exercise, a fully parallel implementation was opted for to give a worst-case figure for silicon area and power consumption whilst giving a best-case figure for throughput. A fully parallel implementation directly maps each row and each column of the parity-check matrix H to a different processing unit, whilst all the processing units execute in parallel. In this implementation, the PEGReg504x1008 parity-check matrix has M ($M = 504$) check nodes and N ($N = 1008$) variable nodes connected by edges wherever $H(i, j) = 1$. There are $(M \times w_r) + (N \times w_c)$ total connections, where w_r and w_c are the weight of check nodes and variable nodes, respectively. The PEGReg504x1008 code is a regular graph with $w_r = 6$ and $w_c = 3$; therefore, a fully parallel implementation has a set of $(3024 + 3024) \times F_i$ global wires and their associated wire length compensating circuits (i.e., routing buffers). Here F_i is the number of bits used to represent the data. The next section provides an analysis of the routing congestion.

6.1. Design Flow and Routing Congestion Analysis. In the ATBF algorithm, the sign information is the only global signal propagating between the processing elements; therefore $F_i = 1$ and the wiring load of the circuit is 6048 global interconnects. The interconnect complexity of the

(1) **Initialise** $\lambda_k = \lambda_0, k \in \{1, \ldots, N\}$
(2) **For** $j = 1 : n$
    $x_j = \text{sign}(y)$
(3) **If** $\prod_{j \in N(i)} x_j = +1$
    output **x** and stop
(4) **For** $k = 1 : n$
    If $\Delta_k^{(\text{GD})} < \lambda_k$
        flip bit x_k
    otherwise
        $\lambda_k = \theta \lambda_k$
(5) **If** $\exists k \in \{1, \ldots, N\} : \lambda_k \geq \phi$
  Or the maximum number of iterations is reached
    output **x** and stop
  otherwise
    go to step 3

Algorithm 5: Early stopping adaptive threshold bit flipping (ES-ATBF).

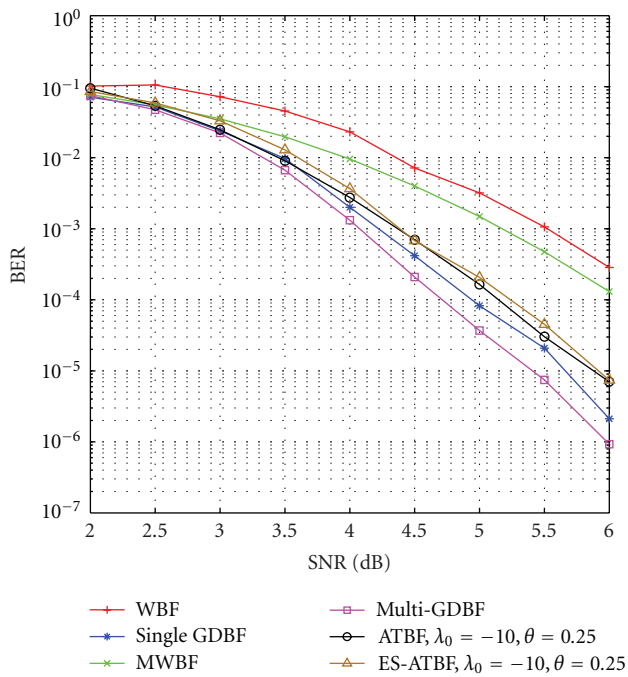

Figure 4: Bit error rate of bit flipping algorithms using PEGReg504x1008 code in an AWGN channel, maximum iterations limited to 100.

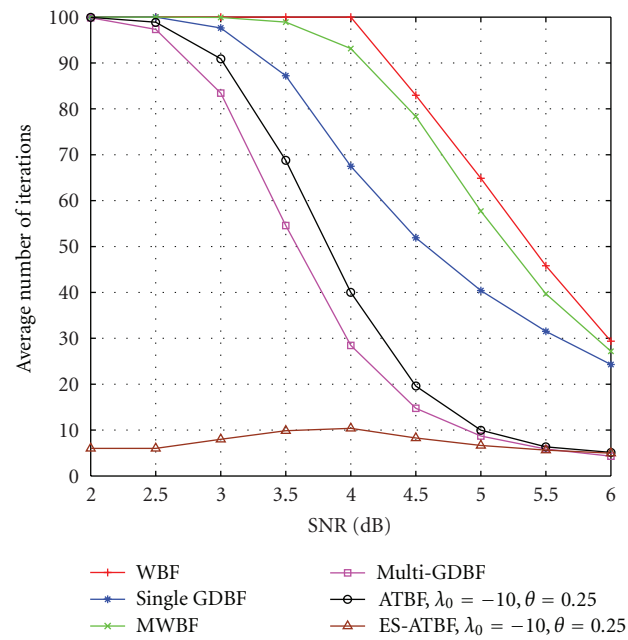

Figure 5: Average number of iterations of bit flipping algorithms using PEGReg504x1008 code in an AWGN channel, maximum iterations limited to 100.

ATBF algorithm is F_i times (typically 4–6) less complex than the equivalent SP variants where 4–6 bits of soft information is used for message passing. In our analysis, the Synopsys IC-Compiler computer-aided design (CAD) tool was used for placement, clock tree synthesis, and routing (global, track, and detailed routing). We used a standard-cell-based automatic place and route flow with a square floor plan. The decoder was developed using Verilog to describe the architecture and synthesised with Synopsys Design Compiler. The total wire length and compiler route CPU time were used as a pessimistic measure of estimating routing congestion. Initially with five metal layers used for the power grid, the row utilisation was set to a high value (87%) resulting in significant routing congestion. Keeping the metal layer configuration the same routing closure was achieved by relaxing the row utilisation with 100% routing closure achieved using 63.75% row utilisation.

In conclusion, the measured results have shown that the ATBF algorithm is routable in a square floor plan using 90 nm standard cell process using five metal layers and a single polysilicon layer. Figure 6 shows the completed design with a core size 796.36 × 793.80 microns and chip size of 854.36 × 853.80 microns.

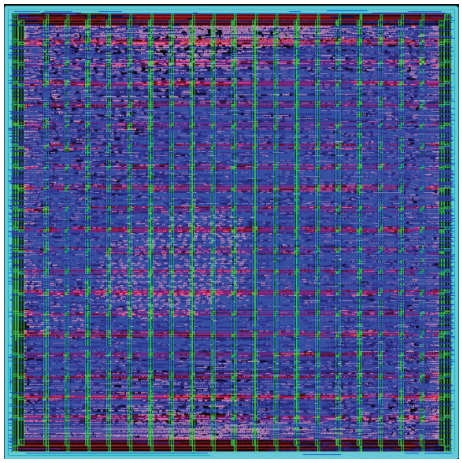

Figure 6: Fully parallel ATBF decoder implementation.

6.2. Fully Parallel ATBF Decoder Implementation. The block diagram of a fully parallel implementation of the ATBF decoder is shown in Figure 7. Two processing elements, variable nodes and check nodes, are labelled VNP and CNP along with the wiring connections between the VNP and CNP blocks labelled as the routing network. Input and output edges of the check node processor are labelled by the connectivity defined in the bipartite graph corresponding to the parity-check matrix. The 4-bit soft LLR values received from the channel are passed to the VNP blocks and these channel values are maintained between the iterations by the register feedback mechanism. The final outputs are taken from the VNP modules once the decoding is complete or the permissible number of iterations has been reached.

The timing diagram of a check node and a variable node update is shown in Figure 8. The check node and variable node messages are updated one after the other in one clock cycle. All check and variable nodes outputs are updated in parallel thus taking one cycle to update all messages for one iteration. For a clock frequency of 250 MHz, the coded throughput for the fully parallel ATBF decoder with a code length of 1008 bits is 252 Gb/sec/iteration.

6.3. Variable Node Processing. Figure 9 shows the variable node processor architecture which computes the hard decision vector, $\mathbf{x}$ as shown in Algorithm 4. This vector is routed though the routing network to the CNP blocks. Since routing between the two node types is of a single bit value, the size of the routing network is smaller than would be used in an implementation of the sum-product algorithm.

The variable node processing block consists of a correlator, inversion function, threshold adaptation, sign-magnitude to 2's complement converter, and a flip-detection circuit. At the start of decoding the received signal, $\mathbf{y}$, is supplied to the VNPs by a register-feedback arrangement as shown. The received soft values are in 4-bit signed-magnitude (SM) format. Let $[sn : mn]$ denote the nth 4-bit soft bit value supplied to the nth VNP, where sn is the inverted 1-bit hard decision value and mn is the magnitude or the reliability of this hard decision value. The SM representation makes the correlation calculation in (4) simpler to implement. As shown, the correlator circuit is made up from a 1-bit multiplexer and inverter. The VNP also performs the threshold adaptation of step 4 in Algorithm 4, which would appear to require a real value multiplier. However, by fixing the scaling factor θ to a power of two the adaptation can simply be implemented as a shift operation of the value λ. The result of threshold adaptation obtained from the comparison (most significant bit of the subtractor in the threshold adaptation circuit) is also fed to the flip detection circuit. If a bit flip is detected the threshold value is unchanged otherwise the threshold is altered for the next iteration by a right shift.

6.4. Check Node Processing Block. The function performed by the check node processor is the second term in (9), a parity-check of the received bits as defined by the parity-check matrix. The CNP takes hard decision values and calculates the resulting sign to form the bipolar syndrome by forming an XOR from the corresponding hard decision values. The number of input values to each check node processor is determined by the row weight in the parity-check matrix, which for the PEGReg504x1008 code was six. The global sign signal determined from this XOR logic of each CNP block is routed by using the interleaved address given by the edge numbering of the equivalent bipartite graph.

6.5. Fixed-Point Analysis. A fixed-point software implementation of the ATBF decoder was written to model the ASIC bit width precision values shown in Figure 9. Figure 10 shows the BER results in an AWGN channel compared to the floating point result with $\lambda_k = 10$, $\theta = 0.25$ and the maximum iterations limited to 100. Migrating from floating to fixed point representation results in a small (<0.25 dB) loss in BER performance. This shows the ATBF algorithm to be robust to fixed-point arithmetic.

6.6. Area Analysis. The design was implemented using UMC 90 nm standard cell CMOS libraries. In a fully parallel approach, the wire length in the decoder design is an important indication of the area; especially where there exists a limited number of metal layers (five metal layers). The total area in different stages of the design flow is shown in Table 1. During the clock tree synthesis stage 83 clock buffers were inserted with the subsequent increase in area shown in the second column of Table 1. Routing was completed with a standard cell utilisation of 63.75% with the final chip area of the placed and routed design being 0.729 mm^2.

6.7. Power Analysis. Power measurements were averaged over 10000 blocks assuming a signal-to-noise ratio of 5.5 dB. Each received block consisted of 1008 4-bit sign-magnitude soft values. The circuit was toggled at a clock frequency of 250 MHz and all frames were subject to ten iterations. The global supply voltage was 0.9 V and kept the same for all cells.

Table 2 shows the breakdown of power usage for the clock tree and Table 3 the breakdown of power usage for the decoder. The clock tree uses approximately 10% of the total decoder power usage.

Table 1: ATBF decoder area breakdown.

	After synthesis stage (mm^2)	After clock tree stage (mm^2)	After routing stage (mm^2)
Combinational area	0.237	0.270	0.402
Noncombinational area	0.109	0.109	
Core area			0.63
Chip area			0.729
Average interconnect length			4 mm

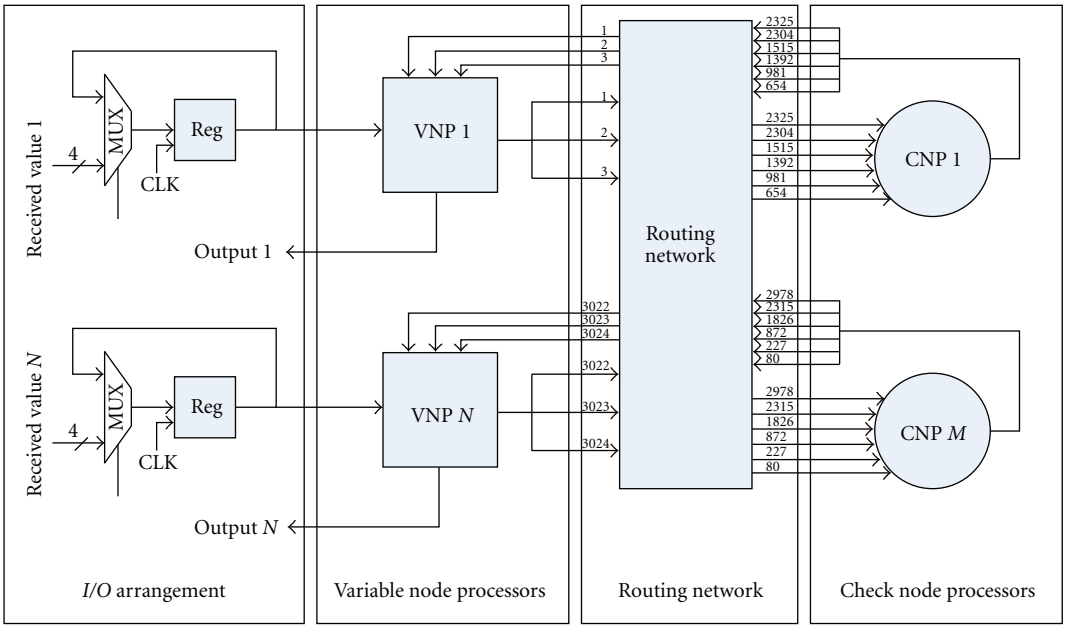

Figure 7: Fully parallel ATBF decoder implementation.

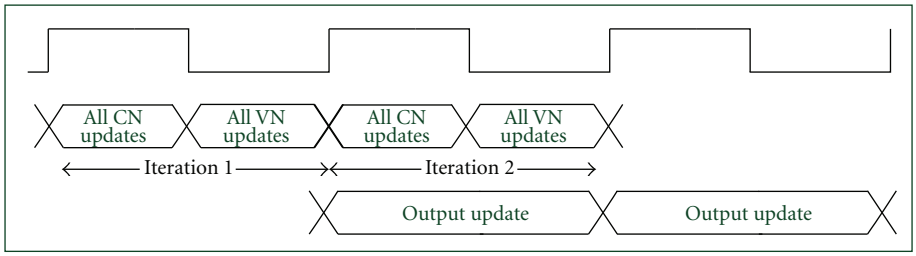

Figure 8: Timing diagram for variable and check node updates.

Table 2: Clock tree power usage breakdown.

Cell internal power	0.14 mW
Net switching power	2.89 mW
Total dynamic power	3.03 mW
Cell leakage power	7.8 μW

Table 3: Decoder power usage breakdown.

Cell internal power	21.96 mW
Net switching power	11.19 mW
Total dynamic power	33.14 mW
Cell leakage power	1.63 mW

The design results for area, power and throughput are presented in Table 4. Table 5 compares the implementation results to another bit flipping implementation described in [19] and various SP implementations. From Table 4, the overall area occupied by the design is 0.729 mm^2 operating at a clock frequency of 250 MHz whilst consuming 33.14 mW of power. Comparing this to the soft-bit-flipping (SBF) implementation of [19] in Table 5 shows the ATBF implementation to have a higher throughput as compared to the SBF algorithm operating at the peak rate which assumes all received bits were error free. Similarly, the area figure for the ATBF decoder is approximately 60% of the

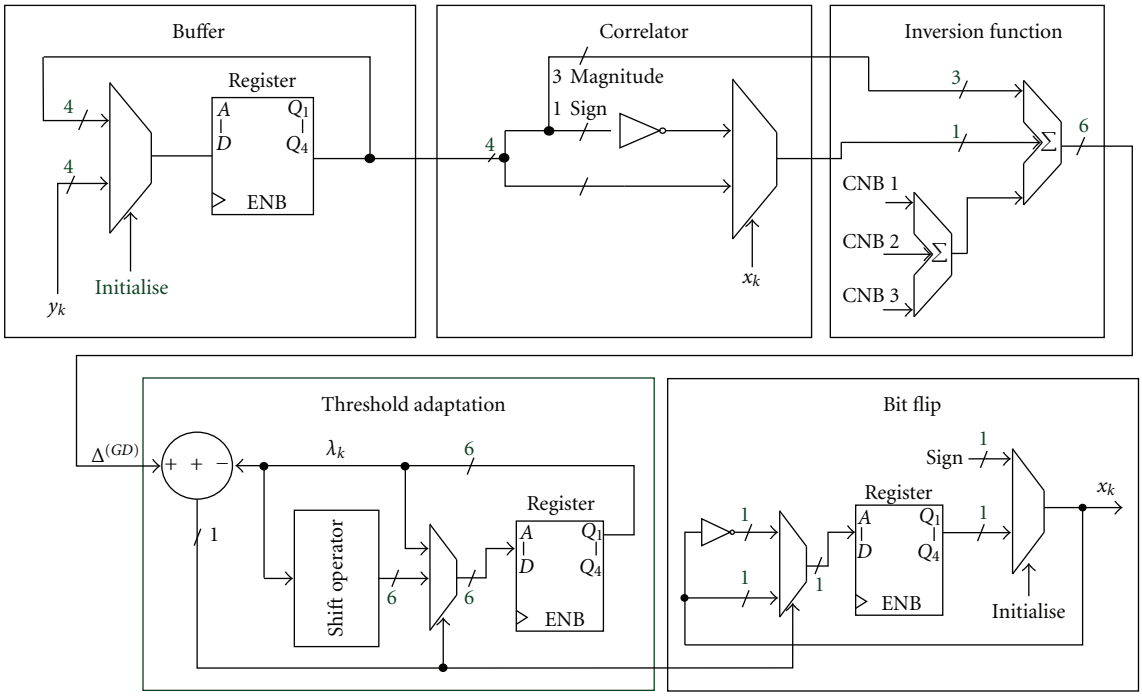

Figure 9: Variable node processing block.

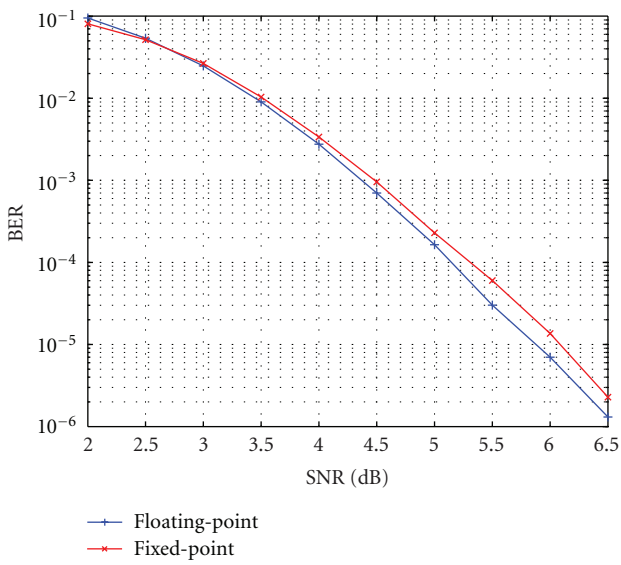

Figure 10: Fixed-point versus floating-point BER comparison of ATBF algorithm using PEGReg504x1008 code in an AWGN channel.

SBF implementation when scaled to a 90 nm fabrication process. The largest difference between the ATBF and SBF implementations is seen in the power consumption figures, the comparatively simple computations of the ATBF algorithm together with lower row and column weights leads to less power consuming and fewer operations resulting in significantly lower power consumption. The *split-row* design in [20] gives an exceptionally high throughput at the expense of high power consumption and large silicon area.

Table 4: ATBF decoder implementation results.

Decoding algorithm	ATBF
CMOS fabrication process	90 nm, 5 M
Supply voltage	0.9 V
Clock speed	250 MHz
Parallelism	Fully parallel
Message bits	4
Logic utilisation	63.75%
Total chip area	0.729 mm^2
Throughput (at 10 iterations)	25.2 Gb/s
Throughput per area (Gb/s/mm^2)	34.6
Power consumption	33.14 mW
Energy/bit	1.3 pJ/bit

7. Conclusion

LDPC codes have shown near-capacity error-correcting performance whilst efficient hardware implementations have led to adoption of LDPC error coding in high throughput wired and wireless standards. For LDPC codes to be used in low power wireless sensor devices and networks, power efficiency and silicon area are important factors. This paper presented results from the implementation in hardware of a low power decoding algorithm for LDPC codes. An efficient architecture suitable for implementation in CMOS was described and throughput, power and area results from a design implementation exercise presented. Implementation in 90 nm CMOS showed the ATBF algorithm to operate at 250 MHz achieving a throughput of 252 Gb/s/iteration whilst consuming 33.14 mW of power (for 10 decoding operations)

Table 5: Comparison of ATBF decoder implementation with published results.

	ATBF Decoder	[19]	[21]	[22]	[23]	[24]	[20]
Process (nm)	90	180	90	65	160	180	65
Algorithm	ATBF	SBF	Block-interlaced	Layered-SP	SP	Layered-SP	SP
Code size (m, n)	(504,1008)	(813,1057)	(480,660)	(900,1200)	(512,1024)	(1024,2048)	(384,2048)
Clock frequency (MHz)	250	345	142	528	64	125	195
Total area (mm^2)	0.729	7.4	3.1	0.21	52.5	14.3	4.84
Total power (mW)	33.14	1450	—	—	690	787	1359
Average throughput (Gb/s)	25.2	17.37 (max) 0.25 (min)	5.86	0.53	1.0	0.64	92.8
Average iterations	10	1 (max) 40 (min)	16	8	64	10	11
Energy/bit (pJ)	1.3	83.5 (max)	—	—	10.8	123.0	15

and occupying an area of 0.72 mm^2. The energy per decoded bit was 1.3 pJ, one of the lowest figures in the published literature.

Acknowledgments

The authors would like to thank the directors of the Telecommunications Research Laboratory (TRL), Toshiba Research Europe Ltd and Lee Keep from Synopsys.

References

[1] J. Beutel, K. Rmer, M. Ringwald, and M. Woehrle, "Deployment techniques for sensor networks," in *Sensor Networks*, G. Ferrari, Ed., Signals and Communication Technology, pp. 219–248, Springer, Berlin, Germany, 2009.

[2] A. Bachir, M. Dohler, T. Watteyne, and K. K. Leung, "MAC essentials for wireless sensor networks," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 2, pp. 222–248, 2010.

[3] R. G. Gallager, *Low Density Parity Check Codes,*, Monograph, M.I.T. Press, 1963.

[4] D. J. C. MacKay, "Good codes based on very sparse matrices," in *Proceedings of the 5th IMA Conference on Cryptography and Coding*, pp. 100–111, Springer, London, UK, 1995.

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

[6] M. Mansour and N. Shanbhag, "A 1.6 Gbit/sec 2048-bit programmable and code-rate tunable LDPC decoder chip," in *Proceedings of the 3rd International Symposium on Turbo Codes & Related Topics (ISTC '03)*, pp. 137–140, Brest, France, September 2003.

[7] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 11, no. 6, pp. 976–996, 2003.

[8] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS '04)*, pp. 107–112, October 2004.

[9] M. Ismail, I. Ahmed, J. Coon, S. Armour, T. Kocak, and J. McGeehan, "Low latency low power bit flipping algorithms for LDPC decoding," in *Proceedings of the IEEE 21st International Personal Indoor and Mobile Radio Communications (PIMRC '10)*, pp. 278–282, 2010.

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

[11] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low density parity check codes based on finite geometries: a rediscovery," in *Proceedings of the IEEE International Symposium on Information Theory*, p. 200, ita, June 2000.

[12] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," *IEEE Communications Letters*, vol. 8, no. 3, pp. 165–167, 2004.

[13] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," in *Proceedings of the International Symposium on Information Theory and its Applications (ISITA '08)*, pp. 1–6, December 2008.

[14] D. J. C. Mackay, "Encyclopedia of sparse graph codes," http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[15] Z. Wang, Y. Zhang, and K. K. Parhi, "Study of early stopping criteria for turbo decoding and their applications in WCDMA systems," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '06)*, pp. III1016–III1019, May 2006.

[16] J. Li, X. Hu You, and J. Li, "Early stopping for LDPC decoding: convergence of mean magnitude (CMM)," *IEEE Communications Letters*, vol. 10, no. 9, pp. 667–669, 2006.

[17] F. Kienle and N. Wehn, "Low complexity stopping criterion for LDPC code decoders," in *Proceedings of the IEEE 61st Vehicular Technology Conference (VTC '05)*, vol. 1, pp. 606–6609, May 2005.

[18] Z. Cui, L. Chen, and Z. Wang, "An efficient early stopping scheme for LDPC decoding," in *Proceedings of the 13th NASA Symposium on VLSI Design*, Moscow, Idaho, USA, June 2007.

[19] 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.

[20] T. Mohsenin, D. Truong, and B. 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. 1048–11061, 2010.

[21] A. Darabiha, A. Carusone, and F. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *IEEE Transactions on Circuits and Systems II*, vol. 55, no. 1, pp. 74–778, 2008.

[22] T. Brack, M. Alles, T. Lehnigk-Emden et al., "A survey on LDPC codes and decoders for OFDM-based UWB systems," in *Proceedings of the IEEE 65th Vehicular Technology Conference (VTC '07)*, pp. 1549–1553, 2007.

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

[24] M. Mansour and N. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, 2006.