Low Latency Low Power Bit Flipping Algorithms For LDPC Decoding

Mohamed Ismail, Imran Ahmed and Justin Coon
Toshiba Research Europe Ltd.
Telecommunications Research Laboratory
Bristol, UK, BS1 4ND
Email: mohamed.ismail@toshiba-trel.com

Simon Armour, Taskin Kocak and Joseph McGeehan
University of Bristol
Dept. of Electrical and Electronic Eng.
Bristol, UK.

***Abstract*—Low Density Parity Check (LDPC) codes have been adopted in a number of wired and wireless communication standards due to their improved error correcting ability and relatively simple decoder structure. However, for very high throughput systems operating in the multi-Gb/s range conventional decoding methods based on message passing are limited, due largely to the sheer volume of messages being exchanged. Thus, simpler decoding methods have been proposed such as bit flipping permitting efficient and fast hardware implementation. This paper presents two new bit flipping algorithm designed to reduce latency and power consumption. For a small loss in bit error rate performance (0.5 dB) we show how the application of an early stopping criteria uses 89% fewer iterations compared to a similar published algorithm. We also present a method for reducing power consumption by placing processing elements into a quiescent state based on a bit-local metric. Using this technique we show a potential reduction in power consumption of 76%.**

I. Introduction

Low Density Parity Check (LDPC) codes were described by Gallager [1] in 1963 and subsequently rediscovered by Mackay *et al* [2]. As well as describing LDPC codes Gallager presented two practical decoding algorithms. A soft decision message passing algorithm variously referred to as Belief Propagation (BP), Sum-Product Algorithm (SPA) and Two Phase Message Passing (TPMP). The second algorithm, referred to as Algorithm B, falls into the class of *bit flipping* (BF) methods designed to be less complex than BP decoding. Although BP techniques have been shown to perform exceptionally well, coming within 0.0045 dB of the Shannon capacity limit [3], they do suffer from implementation problems. In [4] three major implementation problems for a fully parallel BP decoder are highlighted: 1) Routing bottleneck due to mirroring of the LDPC matrix connectivity in hardware, 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 [5] [6] that address these problems but generally have lower throughput than a fully parallel implementation.

At the expense of error correcting performance, bit flipping methods allow for significantly less complex decoding because they flip one or more bits at a time based on some objective function. A simple bit flipping technique can be described as follows. Let $\mathbf{y}$ be the soft-decision received vector and $\mathbf{x}$ be the binary hard-decision of the vector $\mathbf{y}$, then the *syndrome*, $\mathbf{s}$, is defined as

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

where, $\mathbf{H}$ is a $M \times N$ parity check matrix and $\mathbf{s} = (s_1, s_2, \ldots, s_M)$ represents the M individual parity checks. Bit flipping looks at the parity checks each received bit is involved in, if the total number of check failures the bit is involved in exceeds some threshold, the sign of the received bit is inverted to form an updated received vector. The next iteration of the algorithm will re-calculate the syndrome vector using the updated hard-decision vector. This process is repeated until all checks are error free or the number of iterations reaches some preset threshold. There are many variants to the basic bit flipping algorithm most notably the *Weighted Bit Flipping* (WBF) algorithm [7], which in formulating a count of failed checks adds a weighting factor based on the magnitude of the received sample. The *Modified* WBF (MWBF) algorithm [8] introduces an additional scaling factor, α, for the soft-decision values resulting in improved error correcting performance. Recently, Wadayama [9] formulated a new way of calculating the decision metric 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.

The majority of bit flipping algorithms, including ones described in [10] [11] [12], require locating a minimum value over the whole block length which for LDPC codes can be large. This *min* operation incurs significant delay when implemented in an Application Specific Integrated Circuit (ASIC) causing a limitation to maximum achievable throughput.

In this paper we present two variants of the GDBF algorithms designed to have lower latency and fewer iterations. The same methods may, in principle, be applied to other bit flipping algorithms for reducing latency, number of iterations and power consumption.

This paper is organised as follows, section II describes the GDBF algorithm, which forms the basis for our improvement and our proposed bit flipping algorithm. Section III describes a means for detecting when to disable bit processing elements to aid power conservation. Section IV proposes an early stopping criteria and applies it to our proposed algorithm. Section V evaluates the proposed algorithms comparing their error correcting performance and average number of decoding iterations to published algorithms. Section VI draws out conclusions from our work.

II. Bit Flipping Decoding Algorithms

A. Gradient Descent Bit Flipping

For a received vector of soft-decision values, $\mathbf{y} = (y_1, y_2, \dots, y_N)$, let $\mathbf{x} \in \{+1, -1\}^N$ be the hard decision of vector $\mathbf{y}$ such that $x_j = sign(y_j), j \in \{1, \dots, N\}$. If $N(i) = \{j : h_{ij} = 1\}$ is the index of non-zero elements in row i and $M(j) = \{i : h_{ij} = 1\}$ is an index of non-zero elements in column j of the parity check matrix, then the i'th *bipolar syndrome* of $\mathbf{x}$ is defined as $\prod_{j \in N(i)} x_j \in \{+1, -1\}$. In Gallager's Algorithm B, an *inversion* function is used to provide a measure of the confidence with which a bit should be flipped, formed by (2).

$$\Delta_k^{(G)}(\mathbf{x}) = \sum_{i \in M(k)} \prod_{j \in N(i)} x_j \tag{2}$$

Thus, the Gallager Algorithm B is a hard decision bit flipping algorithm which is straightforward to implement in hardware, requiring $5E - m$ operations (add, subtract and compare) per iteration, where E is the total number of non-zeros in the parity check matrix [13]. The WBF inversion function, which weights each check by the least reliable bit in the check, can be written as in (3).

$$\Delta_k^{(WBF)}(\mathbf{x}) = \sum_{i \in M(k)} min_{j \in N(i)} |y_j| \prod_{j \in N(i)} x_j \tag{3}$$

For a WBF implementation, being a soft-decision algorithm, the number of real addition operations per iteration is given as $2K_{BF}E$, where K_{BF} is an implementation dependent constant typically less than three [7]. The gradient descent inversion function [9] is defined as shown in (4).

$$\Delta_k^{(GD)}(\mathbf{x}) = x_k y_k + \sum_{i \in M(k)} \prod_{j \in N(i)} x_j \tag{4}$$

Equation (4) is a combination of the hard decision of the Gallager algorithm with an additional term representing the correlation between the current hard estimate of a bit, x_k and the initial soft value, y_k. Based on (4) the *single*-step GDBF algorithm can be stated as shown in Algorithm 1.

Two observations can be made from step 3 of Algorithm 1; the single-step GDBF algorithm flips only a single bit in an iteration and a global *min* operation over the block length N is necessary. For the large block lengths typical of LDPC codes an ASIC implementation of the *min* operation causes considerable delay in the critical path of the circuit. Secondly, by correcting only a single bit error per iteration the decoding process will take more iterations compared to a multi-bit flipping approach. In [9] a multi-bit flipping algorithm is given which modifies the condition in step 3 such that all bits with $\Delta_k^{(GD)} < \theta$, where θ is the *inversion* threshold, are flipped. 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 [9] 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 (5), and comparing to a threshold.

Algorithm 1 Single-step Gradient Descent Bit Flipping

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

$$f(x) = \sum_{k=1}^{n} x_k y_k + \sum_{i \in M(k)} \prod_{j \in N(i)} x_j \tag{5}$$

The multi-step GDBF algorithm achieves better error correcting performance than both the WBF or MWBF algorithms whilst using fewer iterations at a cost of additional complexity. Below we present a new localised bit flipping algorithm, termed Adaptive Threshold Bit Flipping (ATBF), designed to intrinsically operate in multi-step or single-step mode with a simple dynamic shifting of the threshold value. The ATBF algorithm avoids the additional complexity of switching between modes and use of a minimum function associated with the multi-step GDBF algorithm.

B. Adaptive Threshold Bit Flipping (ATBF)

Let $\lambda_k, k \in \{1, \dots, N\}$ 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 improved bit flipping algorithm can then be stated as shown in Algorithm 2, 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, $\Delta_k^{(GD)}$, is not below the threshold, λ_k, the threshold is lowered by scaling 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.

III. Power Reduction Method

With each iteration of the ATBF algorithm, step 4 will result in either a change of sign of x_k or the lowering of λ_k. As $\lambda_k \longrightarrow 0$ the probability of a sign change will become smaller

Algorithm 2 Adaptive Threshold Bit Flipping (ATBF)

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

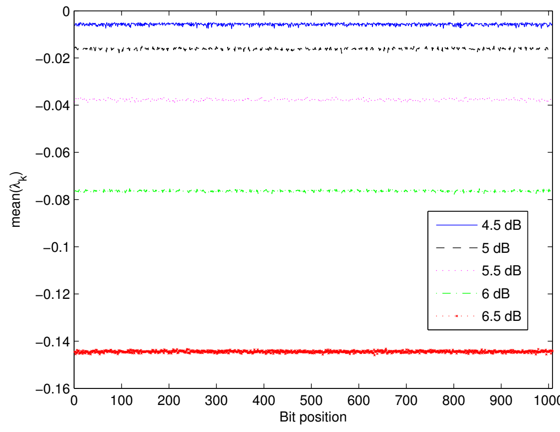

Fig. 1. Mean λ value as a function of SNR, $\theta = 0.25$, $\lambda_0 = -10$

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 power savings.

To determine when a bit processor is unlikely to change the sign of a bit we introduce a flipping threshold, ϕ_1, against which a given λ_k is compared. If $\lambda_k \geq \phi_1$ the bit processor is placed in a quiescent state and no further operations are undertaken. Determing the optimal threshold value was deemed too difficult as it is a function of the Signal-to-Noise Ratio (SNR), LDPC code characteristics and modulation scheme therefore we adopted a statistical approach. 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 a regular-(3,6), half-rate (504,1008) code taken from [14] referred to as PEGReg504x1008. Fig. 1 shows how the mean, over at least 1000 blocks, of each of the 1008 threshold values changes as a function of SNR.

Let $\bar{\lambda}_k = \frac{1}{S}\sum_{j=1}^{S} \lambda_j$, $k \in \{1, \dots, N\}$ and S is the number of blocks simulated. For a given signal-to-noise ratio we define $\bar{\lambda}_{SNR} = \sum_{k=1}^{N} \bar{\lambda}_k$. Using the $\bar{\lambda}_{SNR}$ values we apply a least squares polynomial fit to derive a relationship between the mean threshold value reached in decoding and the SNR value. We define a flipping threshold, ϕ_{SNR}, based on the polynomial shown in (6) as the threshold value for which a bit processor is put in a quiescent state, where a is the SNR in dB.

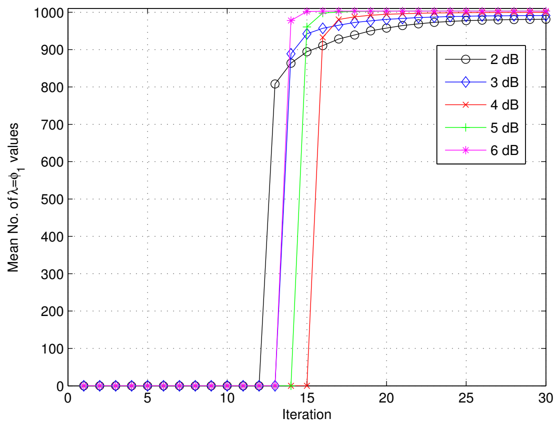

Fig. 2. Mean no. of $\lambda = \phi_1$ values as a function of iterations at different SNR values, $\theta = 0.25$, $\lambda_0 = -10$

$$\phi_{SNR} = -0.005a^4 + 0.089a^3 - 0.666a^2 + 2.258a - 2.897 \quad (6)$$

For a practical implementation the scaling factor applied to 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_1, can be derived from ϕ_{SNR} by setting ϕ_1 to the nearest value to ϕ_{SNR} achievable by a θ scaling, as shown in (7c).

$$z = \left\lceil -log_2\left(\frac{\phi_{SNR}}{\lambda_0}\right)\right\rceil \quad (7a)$$

$$z' = -(z + mod(z, d)) \quad (7b)$$

$$\phi_1 = \lambda_0 2^{z'} \quad (7c)$$

Fig. 2 shows the average number of values with $\lambda_k = \phi_1$ 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 Fig. 2 at which the mean number of $\lambda_k = \phi_1$ values becomes non-zero for a given SNR. All prevous 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 Fig. 5, which gives the average number of decoding iterations as a function of SNR for different decoding algorithms. Fig. 3 shows the estimated percentage power saving as a function of SNR (dB) when using the power reduction method described. At 3 dB

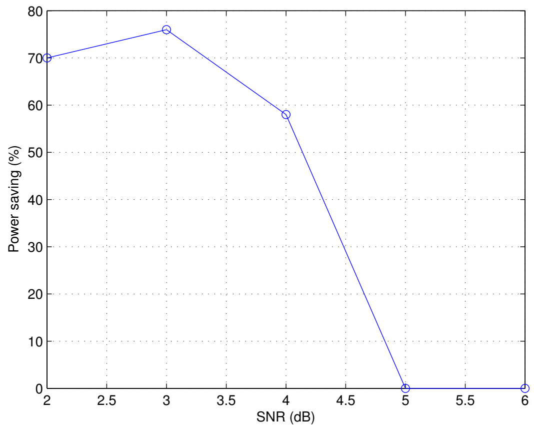

Fig. 3. Estimated saving in power as a function of SNR for PEGReg504x1008 code using the power reduction method

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.

IV. 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 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 (BP) 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].

Building on the idea of using a flipping threshold to determine when a bit processor is placed in a quiescent state, we further investigate behaviour of the threshold values, λ_k. From Fig. 2, for the first 13 iterations no threshold values drop to the flipping threshold, between 13-15 iterations we observe a large step where the majority of threshold values reach the flipping threshold. As Fig. 1 shows this corresponds to the average threshold value with the maximum number of iterations is set to a high value. Thus, terminating the decoding operation at this point should result in little change in the error correcting capability whilst reducing the number of required iterations.

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 Fig. 2 we define a simple, bit-local, metric for determining when to terminate the decoding process. When any $\lambda_k \geq \phi_1$ the decoding process is terminated, this is a straightforward edge detection of the transition from zero values with $\lambda_k \geq \phi_1$ to many values for which $\lambda_k \geq \phi_1$. The ATBF algorithm is modified to incorporate the early stopping criterion as shown in Algorithm 3.

Algorithm 3 Early Stopping Adaptive Threshold Bit Flipping (ES-ATBF)

```
1) Initialise $\lambda_k = \lambda_0, k \in \{1, \dots, N\}$
2) For $j = 1 : n$
       $x_j = sign(y)$
3) If $\prod_{j \in N(i)} x_j = +1$
       output x and stop
4) For $k = 1 : n$
       If $\Delta_k^{(GD)} < \lambda_k$
           flip bit $x_k$
       otherwise
           $\lambda_k = \theta\lambda_k$
5) If,
           $\exists k \in \{1, \dots, N\} : \lambda_k \geq \phi_1$
   Or the maximum number of iterations is reached
       output x and stop
   otherwise
       goto step 3
```

V. 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 is compared to the WBF, MWBF, single-step GDBF and multi-step GDBF algorithms in Fig. 4 and Fig. 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 multi-step 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%.

VI. Conclusion

We have presented two new bit flipping algorithms designed to have low computational complexity and, importantly for high throughput, low latency. The ATBF algorithm describes a thresholding technique operating at bit-level thereby removing the minimum operation over the whole block, resulting in reduced latency. The ES-ATBF algorithm permits early termination of the decoding process using 89% fewer iterations than the ATBF algorithm, without loss in error correcting performance and without any significant increase in latency.

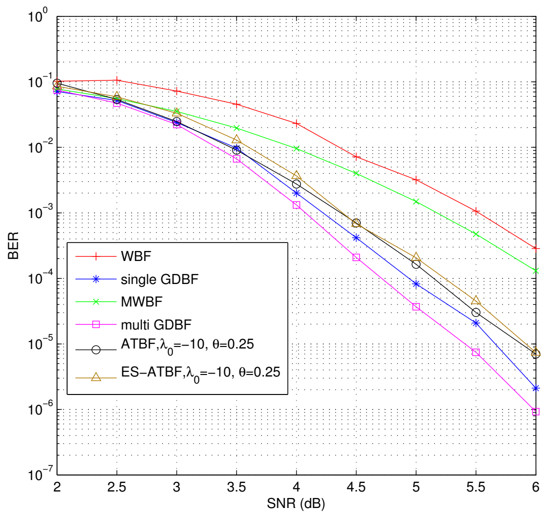

Fig. 4. Bit error rate of bit flipping algorithms using PEGReg504x1008 code in an AWGN channel, maximum iterations limited to 100

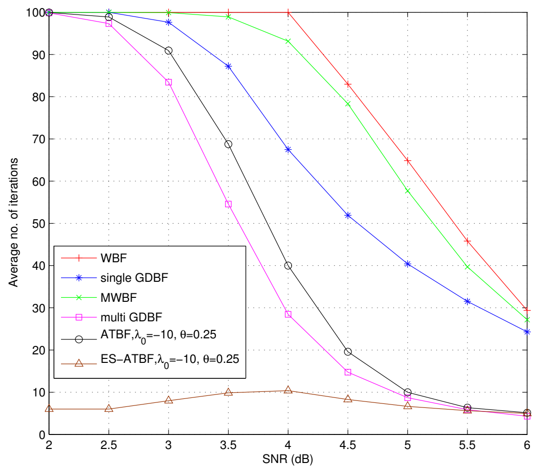

Fig. 5. Average number of iterations of bit flipping algorithms using PEGReg504x1008 code in an AWGN channel, maximum iterations limited to 100

Additionally, we detail an easy to implement metric resulting in power savings of upto 76% for the inner loop of the ATBF algorithm.

Acknowledgement

The authors would like to thank Toshiba TRL and its directors for supporting this work.

References

[1] R. G. Gallager, *Low-Density Parity-Check Codes*, M. Cambridge, Ed. MIT Press, 1963.

[2] D. J. C. MacKay, “Good Codes Based on Very Sparse Matrices,” in *Proceedings of the 5th IMA Conference on Cryptography and Coding*. London, UK: Springer-Verlag, 1995, pp. 100–111.

[3] S.-Y. Chung, J. Forney, G.D., T. Richardson, and R. Urbanke, “On the design of low-density parity-check codes within 0.0045 db of the Shannon limit,” *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, 2001.

[4] 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)*, Brest, France, September 2003, pp. 137–140.

[5] ——, “High-throughput LDPC decoders,” *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 976–996, 2003.

[6] D. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes,” in *Proc. IEEE Workshop on Signal Processing Systems SIPS 2004*, 2004, pp. 107–112.

[7] Y. Kou, S. Lin, and M. Fossorier, “Low density parity check codes based on finite geometries: a rediscovery,” in *Proc. IEEE International Symposium on Information Theory*, 2000, pp. 200–.

[8] J. Zhang and M. Fossorier, “A modified weighted bit-flipping decoding of low-density parity-check codes,” *IEEE Commun. Lett.*, vol. 8, no. 3, pp. 165–167, 2004.

[9] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, “Gradient descent bit flipping algorithms for decoding LDPC codes,” in *Proc. International Symposium on Information Theory and Its Applications ISITA 2008*, 2008, pp. 1–6.

[10] M. Shan, C. Zhao, and M. Jiang, “Improved weighted bit-flipping algorithm for decoding LDPC codes,” vol. 152, no. 6, Dec. 2005, pp. 919 – 922.

[11] T. Magloire, N. Ngatched, M. Bossert, A. Fahrner, and F. Takawira, “Two bit-flipping decoding algorithms for low-density parity-check codes,” *Communications, IEEE Transactions on*, vol. 57, no. 3, pp. 591 –596, March 2009.

[12] Z. Liu and D. A. Pados, “A Decoding Algorithm for Finite-Geometry LDPC Codes,” *Communications, IEEE Transactions on*, vol. 53, no. 2, pp. 380 – 380, Feb. 2005.

[13] W. Yu, M. Ardakani, B. Smith, and F. Kschischang, “Complexity-optimized low-density parity-check codes for Gallager decoding algorithm B,” in *Proc. International Symposium on Information Theory ISIT 2005*, 2005, pp. 1488–1492.

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

[15] Z. Wang, Y. Zhang, and K. Parhi, “Study of Early Stopping Criteria for Turbo Decoding and Their Applications in WCDMA Systems,” vol. 3, May 2006, pp. III –III.

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

[17] F. Kienle and N. Wehn, “Low complexity stopping criterion for LDPC code decoders,” vol. 1, May 2005, pp. 606 – 609 Vol. 1.

[18] Z. Cui, L. Chen, and Z. Wang, “An efficient early stopping scheme for LDPC decoding,” *13th NASA Symposium on VLSI Design, Idaho, Jun. 2007*, 2007. [Online]. Available: http://www.cambr.uidaho.edu/symposiums/13TH_NASA_VLSI_Proceedings