A Comparison Between LDPC Block and Convolutional Codes

Daniel J. Costello, Jr.*, Ali Emre Pusane*, Stephen Bates†, and Kamil Sh. Zigangirov*
*Department of Electrical Engineering
University of Notre Dame, Notre Dame, IN 46556, U.S.A.
†Department of Electrical and Computer Engineering,
University of Alberta, Edmonton, Canada.
E-mail: {Costello.2@nd.edu, Pusane.1@nd.edu, Stephen.Bates@ece.ualberta.ca, Zigangirov.1@nd.edu.}

***Abstract*— LDPC convolutional codes have been shown to be capable of achieving the same capacity-approaching performance as LDPC block codes with iterative message-passing decoding. However, traditional means of comparing block and convolutional codes tied to the implementation complexity of trellis-based decoding are irrelevant for message-passing decoders. In this paper, we undertake a comparison of LDPC block and convolutional codes based on several factors: encoding complexity, decoding computational complexity, decoding hardware complexity, decoding memory requirements, decoding delay, and VLSI implementation requirements.**

I. Introduction

A few years after the invention of turbo codes, researchers became aware that Gallager's low-density parity-check (LDPC) block codes, first introduced in [1], were also capable of capacity-approaching performance on a variety of channels. Analysis and design of these codes quickly attracted considerable attention in the literature, beginning with the work of Wiberg [2], MacKay and Neal [3], and many others. The convolutional counterparts of LDPC block codes, namely LDPC convolutional codes, were subsequently proposed in [4]. Analogous to LDPC block codes, LDPC convolutional codes are defined by sparse parity-check matrices that allow them to be decoded using a sliding window-based iterative message-passing decoder.

Recent studies have shown that LDPC convolutional codes are suitable for practical implementation in a number of different communication scenarios, including continuous transmission as well as block transmission in frames of arbitrary size [5], [6], [7]. They are also known for their encoding simplicity, since the original code construction method proposed in [4] yields a shift-register based systematic encoder for real time encoding of continuous data. This is an advantage when compared to randomly constructed LDPC block codes.

Given their excellent bit error rate (BER) performance along with their simplicity of encoding, it is quite natural to compare LDPC convolutional codes with corresponding LDPC block codes. In this paper, we compare these codes under several different assumptions: equal decoding computational complexity, equal decoding processor (hardware) complexity, equal decoding memory requirements, and equal decoding delay.

The paper is organized as follows. In Section II, we provide a brief overview of LDPC convolutional codes. The main contribution of the paper is Section III, where comparisons of LDPC block and convolutional codes based on several criteria are presented. In the next section, we focus on finite block length comparisons between LDPC block codes and terminated LDPC convolutional codes. Finally, we provide some conclusions in Section V.

II. An Overview of LDPC Convolutional Codes

An $(m_{\rm s}, J, K)$ regular LDPC convolutional code is the set of sequences $\mathbf{v}$ satisfying the equation $\mathbf{v}\mathbf{H}^{\rm T} = 0$, where

$$\mathbf{H}^{\rm T} = \begin{bmatrix} \mathbf{H}_0^{\rm T}(0) & \cdots & \mathbf{H}_{m_{\rm s}}^{\rm T}(m_{\rm s}) & & \\ & \ddots & & \ddots & \\ & & \mathbf{H}_0^{\rm T}(t) & \cdots & \mathbf{H}_{m_{\rm s}}^{\rm T}(t+m_{\rm s}) \\ & & \ddots & & \ddots \end{bmatrix}. \tag{1}$$

Here, $\mathbf{H}^{\rm T}$ is the (time-varying) semi-infinite syndrome former (transposed parity-check) matrix. For a rate $R = b/c$, $b < c$, LDPC convolutional code, the elements $\mathbf{H}_i^{\rm T}(t)$, $i = 0, 1, \cdots, m_{\rm s}$, are binary $c \times (c-b)$ submatrices defined as

$$\mathbf{H}_i^{\rm T}(t) = \begin{bmatrix} h_i^{(1,1)}(t) & \cdots & h_i^{(1,c-b)}(t) \\ \vdots & & \vdots \\ h_i^{(c,1)}(t) & \cdots & h_i^{(c,c-b)}(t) \end{bmatrix}. \tag{2}$$

Starting from the $m_{\rm s} \cdot (c-b)$-th column, $\mathbf{H}^{\rm T}$ has J ones in each row and K ones in each column. The value $m_{\rm s}$, called the syndrome former memory, is determined by the maximal width of the nonzero area in the matrix $\mathbf{H}^{\rm T}$, and the associated constraint length is defined as $\nu_{\rm s} = (m_{\rm s}+1) \cdot c$. In practical applications, periodic syndrome former matrices are of interest. Periodic syndrome formers are said to have a period T if they satisfy $\mathbf{H}_i^{\rm T}(t) = \mathbf{H}_i^{\rm T}(t+T)$, $i = 0, 1, \ldots, m_{\rm s}$, $t \in \mathbb{Z}$.

Although the corresponding Tanner graph has an infinite number of nodes, the distance between two variable nodes that are connected to the same check node is limited by the syndrome former memory of the code. This allows continuous

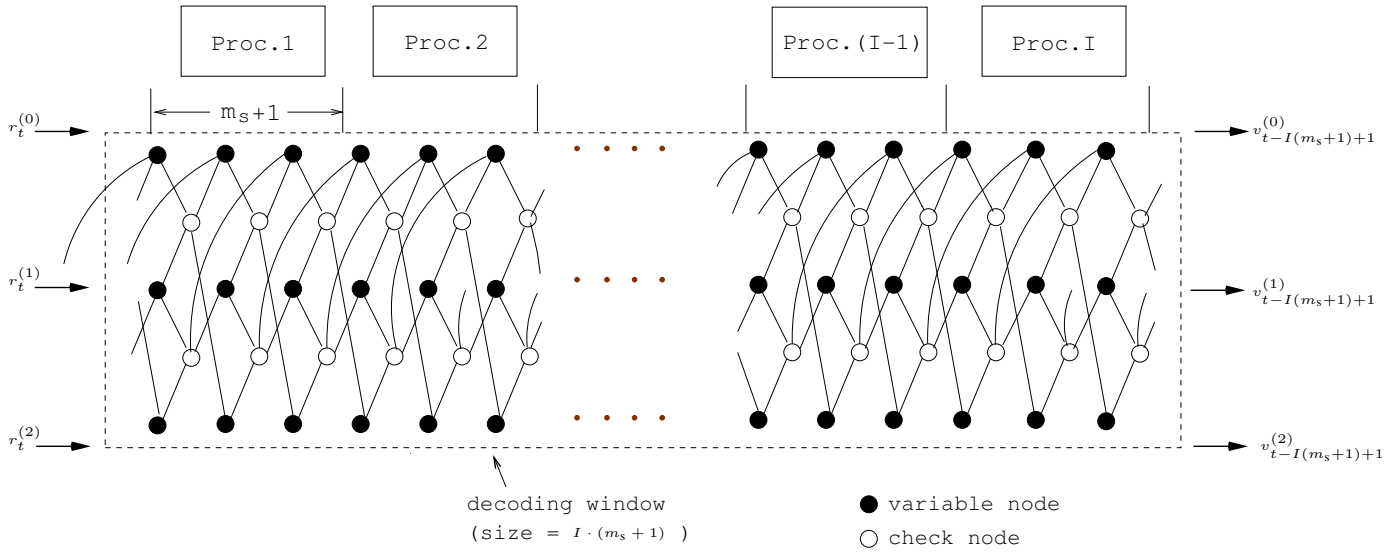

Fig. 1. Tanner graph of an R=1/3 LDPC convolutional code and an illustration of pipeline decoding.

decoding that operates on a finite window sliding along the received sequence, similar to a Viterbi decoder with finite path memory [8]. The decoding of two variable nodes that are at least (m_s+1) time units apart can be performed independently, since the corresponding bits cannot participate in the same parity-check equation. This allows the parallelization of the I iterations by employing I independent identical processors working on different regions of the Tanner graph simultaneously. Alternatively, since the processors implemented in the decoder hardware are identical, a single "hopping" processor that runs on different regions of the decoder memory successively can also be employed.

A pipeline decoding architecture that is based on the ideas summarized in the previous paragraph was introduced by Jiménez Felström and Zigangirov in [4]. The pipeline decoder outputs a continuous stream of decoded data once an initial decoding delay has elapsed. The operation of this decoder on the Tanner graph for a simple time-invariant rate $R = 1/3$ LDPC convolutional code with $m_s = 2$ is shown in Figure 1. (Note that, to achieve capacity-approaching performance, an LDPC convolutional code must have a large value of m_s.)

III. Implementation complexity comparisons of LDPC Block and Convolutional Codes

In this section, we compare several aspects of decoding LDPC convolutional and block codes.

A. Computational Complexity

Let C_{check} (C_{var}) denote the number of computations required for a check (variable) node update for a check (variable) node of degree K (J). Regardless of the code structure, C_{check} and C_{var} only depend on the values J and K.

For a rate $R = b/c$, (m_s, J, K)-LDPC convolutional code decoded using a pipeline decoder with I iterations/processors, at every time instant each processor activates $c - b$ check nodes and c variable nodes. The computational complexity per decoded bit is therefore given by

$$\begin{aligned} C_{bit}^{conv} &= ((c-b)\cdot C_{check} + c\cdot C_{var})\cdot I/c \qquad (3)\\ &= ((1-R)\cdot C_{check} + C_{var})\cdot I, \end{aligned}$$

which is independent of the constraint length ν_s.

Similarly, the decoding complexity for an (N, J, K)-LDPC block code is given by

$$\begin{aligned} C_{bit}^{block} &= (N\cdot\frac{J}{K}\cdot C_{check} + N\cdot C_{var})\cdot I/N \qquad (4)\\ &= (\frac{J}{K}\cdot C_{check} + C_{var})\cdot I\\ &= ((1-R)\cdot C_{check} + C_{var})\cdot I, \end{aligned}$$

which is again independent of the code length N. Thus, there is no difference between block and convolutional LDPC codes with respect to computational complexity.

B. Processor (Hardware) Complexity

The sliding window decoder implementation of an LDPC convolutional code operates on $I\cdot\nu_s$ symbols. However, decoding can be carried out by using I identical independent parallel processors, each capable of handling only ν_s symbols. Hence it is sufficient to design the processor hardware for ν_s symbols. For an LDPC block code of length N, the processor must be capable of handling all N symbols. Therefore, for the same processor complexity, the block length of an LDPC block code must be chosen to satisfy $N = \nu_s$.

C. Memory Requirements

For the pipeline decoder, we need a storage element for each edge in the corresponding Tanner graph. Each variable node also needs a storage element for the channel value. Thus a total of $I\cdot(J+1)\cdot\nu_s$ storage elements are required for I iterations of decoding. Similarly, we need $N\cdot(J+1)$ storage

elements for the decoding of an LDPC block code of length N. Thus, for the same memory requirements, an LDPC block code must satisfy $N = I \cdot \nu_{\mathrm{s}}$.

D. Decoding Delay

Let T_s denote the time between the arrival of successive symbols, i.e., the symbol rate is $1/T_s$. Then the maximum time from the arrival of a symbol until it is decoded is given by

$$\Delta_{io}^{conv} = ((c-1) + (m_{\mathrm{s}}+1) \cdot c \cdot I) \cdot T_s. \tag{5}$$

The first term $(c-1)$ in (5) represents the time between the arrival of the first and last of the c encoded symbols output by a rate $R = b/c$ convolutional encoder in each encoding interval. The dominant second term $(m_{\mathrm{s}}+1) \cdot c \cdot I$ is the time each symbol spends in the decoding window. Since c symbols are loaded into the decoder simultaneously, the pipeline decoder also requires a buffer to hold the first $(c-1)$ symbols.

With LDPC block codes, data is typically transmitted in a sequence of blocks. Depending on the data rate and the processor speed, several scenarios are possible. We consider the best case for block codes, i.e., each block is decoded by the time the first bit of the next block arrives. This results in a maximum input-output delay of $\Delta_{io}^{block} = N \cdot T_s$[1]. Thus, for equal decoding delays, the block length must satisfy $N = (c-1) + \nu_s \cdot I$, assuming the least possible delay for block codes.

E. VLSI implementation requirements

As previously noted, both LDPC block and convolutional codes can be decoded using message passing algorithms. Therefore decoder implementations in both cases consist of identical processing elements, namely variable nodes and check nodes. What differs between the two decoders is the total number of these elements and the way in which they are interconnected.

It is well known that VLSI implementations of parallel LDPC block decoders suffer from an interconnection problem [9]. This is due to the fact that processing nodes must be placed on the silicon at specific locations and connected as defined by $\mathbf{H}$. Regardless of how the rows and columns of $\mathbf{H}$ are permuted, long interconnections are still required. The same observation was also noted for LDPC block codes constructed using algebraic techniques [10].

However, VLSI implementations of LDPC convolutional decoders are based on replicating identical units, termed processors. As illustrated in Fig. 2, the complete decoder can be constructed by concatenating a number of these processors together. For comparable BER performance, the size of an LDPC convolutional code processor needs to be about an order of magnitude less than the block length of an LDPC block code

[1]Note that the block decoder does not need any buffering under such conditions. Let $T_{dec}(N, I)$ denote the time required to perform I iterations of message-passing decoding on a block of N symbols. Thus we require $T_{dec}(N, I) \leq T_s$, i.e., this scenario requires extremely fast processors or very low data rates. By contrast, $T_{dec}(c, I = 1) \leq c \cdot T_s$ for convolutional codes, implying that they can achieve much higher data rates under this assumption.

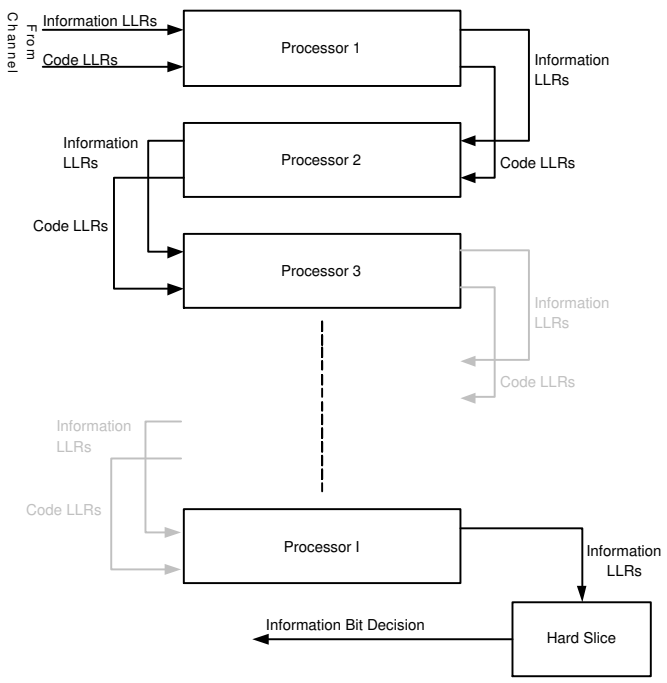

Fig. 2. LDPC convolutional code decoders can be implemented by concatenating sub-units termed processors.

[11]. Therefore the routing complexity within a processor is also an order of magnitude less than for a block code.

There are a wealth of other considerations than impact upon VLSI implementations of LDPC codes. These include the following.

- Any fully parallel LDPC block code decoder may suffer from routing congestion [9]. If so, the total area of such a decoder will be quite large and the maximum clock frequency will be limited by wiring delays.
- The fully parallel LDPC block code decoder can be replaced with a smaller decoder that implements a fraction of the circuit per clock cycle over a number of cycles. This reduces power, area, and throughput in a linear fashion.
- The LDPC convolutional code architecture is more amenable to pipelining because it is inherently a feed-forward architecture. Therefore it may achieve higher clock speeds.
- Since LDPC convolutional code decoders require fewer check and variable processing elements, the marginal cost of optimizing their critical paths in return for increasing their area is less than for LDPC block codes.
- Memory-based architectures have been proposed for both LDPC block codes [12] and LDPC convolutional codes [13].
- Algebraic code construction techniques can simplify LDPC block code decoder implementations [14], [15]. However since LDPC convolutional codes can often be constructed using the same methods these benefits may be applied to both types of codes [11].

- The choice may be affected by system level issues such as variable frame size, variable code rates, latency budgets, and target frame error rate (FER).

In Table I we present a brief summary of some existing LDPC block code and LDPC convolutional code VLSI implementations. In [16] a 54 MBPS decoder was implemented on a Xilinx Virtex-E FPGA. In [9] the decoder throughput was much higher (500 MBPS) because the design was implemented as an ASIC. The BER was about 2×10^{-5} at 2 dB. The decoder presented in [10] targets the IEEE 802.3an standard [17], but it is a hard-decision decoder. This permits a very high throughput but compromises performance. Also note that the codes in [12] and [10] are high-rate, which makes it easier to achieve higher throughput since less processing is required per information bit. The decoder presented in [7] is for a very powerful LDPC convolutional code, and as such it achieves very good performance at the cost of relatively high-complexity and low throughput. In [18] the first ever LDPC convolutional code ASIC decoder is presented. It occupies three times less area (in a larger process) than the LDPC block code ASIC in [9], however it has about three times less throughput and the BER performance is worse.

The discussion in this section illustrates the point that making a comparison between LDPC block codes and LDPC convolutional codes that includes BER/FER performance and VLSI implementation complexity is not simple. It depends on code choice, throughput, power, area, clock speeds, processing node sizes, and system considerations. It is very possible that there is no single best choice between LDPC block codes and LDPC convolutional codes. Instead, LDPC block codes and LDPC convolutional codes may provide complementary solutions and the appropriate choice may vary from one system to another. It is worth noting that LDPC block codes have been the focus of extensive research for several years whereas LDPC convolutional codes are relatively understudied. The first attempts to implement decoders for LDPC convolutional codes noted here are encouraging enough to suggest that further investigation is warranted.

IV. BER-FER Performance Comparison of LDPC Block and Convolutional Codes

In order to test the comparisons given in the previous section, in Figure 3 we plot the performance of a rate $R = 1/2$, (2048,3,6)-LDPC convolutional code with $I = 50$ iterations on an AWGN channel. Also shown is the performance of two $J = 3$, $K = 6$ LDPC block codes with a maximum of 50 iterations. The block lengths were chosen so that in one case the decoders have the same processor complexity, i.e., $N = \nu_{\mathrm{s}}$, and in the other case the same memory requirements, i.e., $N = \nu_{\mathrm{s}} \cdot I$. For the same processor complexity, the convolutional code outperforms the block code by about 0.6 dB at a bit error rate of 10^{-5}. For the same memory requirements, the convolutional and block code performance is nearly identical.

LDPC convolutional codes are very efficient for the transmission of streaming data since they allow continuous encoding/decoding. However, in some applications, it is preferable

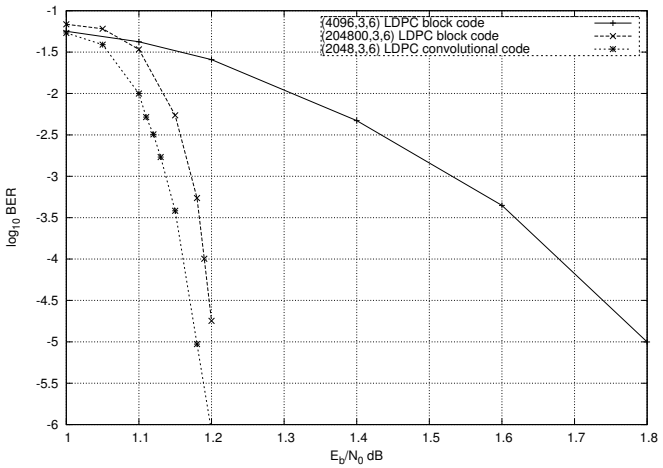

Fig. 3. BER performance comparison of LDPC block and convolutional codes.

to have the data encoded in frames of pre-determined size in order to maintain compatibility with some standard format. Therefore, we now consider the performance of terminated LDPC convolutional codes in this context.

The information sequence must be terminated with a tail of symbols to force the encoder to the zero state at the end of the encoding process. For conventional polynomial convolutional encoders, the terminating tail consists of a sequence of zeros. For LDPC convolutional code encoders, the tail is, generally speaking, non-zero and depends on the encoded information bits. Therefore, a system of linear equations must be solved [19].

In Figure 4, we show FER performance comparisons of terminated LDPC convolutional codes versus LDPC block codes, assuming 100 decoding iterations. We terminate a rate $R = 1/2$ $(2048, 3, 6)$ LDPC convolutional code at various frame lengths, resulting in a variety of terminated block lengths and rates. We also provide simulation results for LDPC block codes of comparable block lengths. As shown in Figure 4, a single LDPC convolutional code can be employed to construct a family of codes of varying frame length and error performance via termination. This is an advantage in terms of flexibility compared to LDPC block codes, where a new code must be constructed each time a new transmission frame length is required.

Figure 4 also shows that, even though the LDPC convolutional code with syndrome former memory $m_{\mathrm{s}} = 2048$ has a hardware complexity comparable to that of a length $N = 4096$ LDPC block code, its performance is similar to much longer LDPC block codes. In particular, for a terminated frame length of $N = 64512$, the LDPC convolutional code outperforms the LDPC block code of length $N = 10000$ and performs almost as well as the LDPC block code of length $N = 100000$.

V. Conclusions

In this paper, we have provided a comparison of LDPC block and convolutional codes based on several criteria,

TABLE I

A COMPARISON OF EXISTING BLOCK AND CONVOLUTIONAL LDPC CODE IMPLEMENTATIONS.NOTE PERFORMANCE FIGURES, WHERE AVAILABLE, ARE GIVEN AT A CERTAIN $\frac{E_b}{N_0}$.

Ref.	Type	Code Params.	Code Rate	Device	Perf.	Throughput
[9]	Block	(1024,3,6)	0.5	52.5mm^2 in 0.16um CMOS	BER = 2.0e-5 @ 2dB	500 MBPS
[16]	Block	(9216,3,6)	0.5	FPGA	BER = 1.0e-6 @ 2dB	54 MBPS
[12]	Block	(8176,7154)	0.875	FPGA	n/a	169 MBPS
[10]	Block	(2048,1723)	0.841	17.64mm^2 in 0.18um CMOS	BER = 5.0e-5 @ 6dB	3200 MBPS
[13]	Conv.	(128,3,6)	0.5	FPGA	BER = 1.0e-3 (1.0e-4) @ 2dB	75 MBPS (25 MBPS)
[7]	Conv	(2048,3,6)	0.5	FPGA	BER = 1.0e-10 @ 2dB	2 MBPS
[18]	Conv	(128,3,6)	0.5	16mm^2 in 0.18um CMOS	BER = 3.0e-4 @ 3dB	150 MBPS

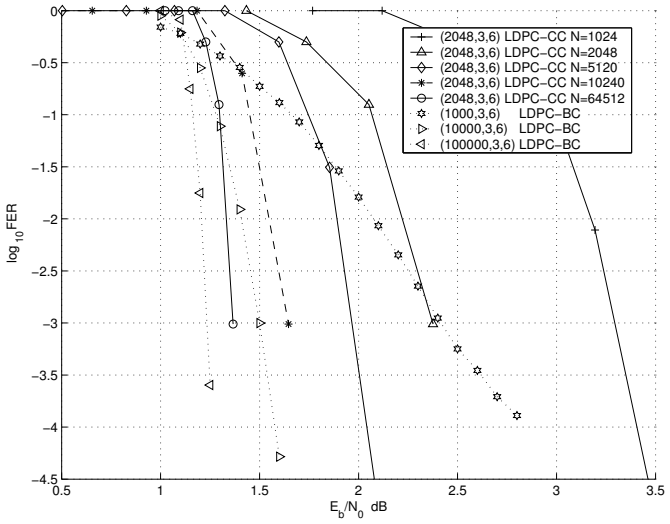

Fig. 4. FER performance comparison of terminated LDPC convolutional and block codes.

including computational complexity, hardware complexity, memory requirements, decoding delay, and BER/FER performance. It has been shown via computer simulations that LDPC convolutional codes have an error performance comparable to that of their block code counterparts. In addition, several interesting tradeoffs have been identified between the two different types of codes with respect to VLSI implementation.

ACKNOWLEDGEMENT

This work was supported in part by NSF Grant CCR02-05310, NASA Grant NNG05GH73G, and SRC Grant 1170.001.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[2] N. Wiberg, *Codes and Decoding on General Graphs*. PhD thesis, Linkoping University, Sweden, 1996.

[3] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, pp. 1645–1646, August 1996.

[4] A. Jiménez-Feltström and K. Sh. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inform. Theory*, vol. IT-45, pp. 2181–2191, Sept. 1999.

[5] S. Bates, Z. Chen, and X. Dong, "Low-density parity check convolutional codes for Ethernet networks," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, (Victoria, B.C., Canada), Aug. 2005.

[6] S. Bates, D. Elliot, and R. Swamy, "Termination sequence generation circuits for low-density parity-check convolutional codes," *submitted to IEEE Trans. Circuits and Systems I*, March 2005.

[7] S. Bates, L. Guntorphe, A. E. Pusane, Z. Chen, K. Sh. Zigangirov, and D. J. Costello, Jr., "Decoders for low-density parity-check convolutional codes with large memory," in *Proc. 12th NASA Symposium on VLSI Design*, (Coeur d'Alene, Idaho, U.S.A.), Oct. 2005.

[8] S. Lin and D. J. Costello, Jr., *Error Control Coding*. Englewood Cliffs, NJ: Prentice-Hall, 2nd ed., 2004.

[9] C. J. Howland and A. J. Blanksby, "A 690-mW 1-Gb/s 1024-b, rate 1/2 low density parity check decoder," *IEEE Trans. Solid-State Circuits*, vol. 37, pp. 404–412, March 2002.

[10] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Multi-gbit/sec low density parity check decoders with reduced interconnect complexity," in *Proc. IEEE Intl. Symposium on Circuits and Systems*, (Kobe, Japan), May 2005.

[11] R. Tanner, D. Sridhara, A. Sridharan, T. Fuja, and D. Costello Jr., "LDPC block and convolutional codes based on circulant matricies," *IEEE Trans. Information Theory*, vol. 50, pp. 2966–2984, December 2004.

[12] Z. Wang and Q. Jia, "Low complexity, high speed decoder architecture for quasi-cyclic LDPC codes," in *Proc. IEEE Intl. Symposium on Circuits and Systems*, (Kobe, Japan), May 2005.

[13] S. Bates and G. Block, "A memory based architecture for low-density parity-check convolutional decoders," in *Proc. IEEE Intl. Symposium on Circuits and Systems*, (Kobe, Japan), May 2005.

[14] H. Zhong and T. Zhang, "Block-LDPC: a practical LDPC coding system design approach," *IEEE Trans. Circuits and Systems I: Fundamental Theory and Applications*, vol. 52, pp. 766–775, April 2005.

[15] K. Yoshida, J. B. Brockman, D. J. Costello, Jr., T. E. Fuja, and M. R. Tanner, "VLSI implementation of quasi-cyclic LDPC codes," in *Proc. Intl. Symposium on Information Theory and its Applications*, (Parma, Italy), pp. 551–556, Oct. 2004.

[16] T. Zhang and K. Parhi, "A 54 MBPS (3,6)-regular FPGA LDPC decoder," in *Proc. IEEE Workshop on Signal Processing*, pp. 127–132, October 2002.

[17] The Institute of Electrical and Electronic Engineers, *IEEE Standard 802.3-2002 : Carrier Sense Multiple Access with Collision Detection CSMA/CD Access Methods and Physical Layer Specification*. The IEEE, 2002.

[18] R. Swamy and S. Bates, "A (128,3,6) low density parity check convolutional code encoder and decoder implemented in 180nm cmos," *in preparation*.

[19] A. E. Pusane, A. Jiménez-Feltström, A. Sridharan, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., "Implementation aspects of LDPC convolutional codes," *submitted to IEEE Trans. Commun.*, Nov. 2005.