A Low-Cost Serial Decoder Architecture for Low-Density Parity-Check Convolutional Codes

Stephen Bates, *Senior Member, IEEE*, Zhengang Chen, *Student Member, IEEE*, Logan Gunthorpe, Ali Emre Pusane, *Student Member, IEEE*, Kamil Sh. Zigangirov, *Fellow, IEEE*, and Daniel J. Costello, Jr., *Life Fellow, IEEE*

***Abstract*—We propose a low-cost serial decoder architecture for low-density parity-check convolutional codes (LDPC-CCs). It has been shown that LDPC-CCs can achieve comparable performance to LDPC block codes with constraint length much less than the block length. The proposed serial decoder architecture for LDPC-CCs uses a single decoding processor. Terminated data frames are sent through the processor iteratively until correctly decoded or a maximum number of iterations is reached. This architecture saves memory consumption and uses a very small number of logic elements, making it especially suitable for strong LDPC-CCs with large code memory. The proposed architecture is realized for a (2048,3,6) regular LDPC-CC on an Altera Stratix FPGA. With a maximum of 100 iterations, the design achieves up to 9-Mb/s throughput using only a very small portion of the field-programmable gate array resources.**

***Index Terms*—Convolutional codes (CCs), data communication, error correction codes, high-speed integrated circuits.**

I. Introduction

LOW-DENSITY parity-check (LDPC) codes were first proposed by Gallager in the 1960s [1] and were then rediscovered by, among others, MacKay and Neal in 1996 [2]. Since their rediscovery, LDPC codes have become very popular and have been incorporated into emerging communication standards such as 10GBASE-T, DVB-S2, and IEEE 802.16e. The reason for this is that LDPC codes can perform very close to the Shannon limit under a suboptimal yet computationally feasible message-passing decoding algorithm, called belief propagation (BP) [3], [4]. The BP decoding algorithm is based on a bipartite graph interpretation of the code, called the Tanner graph [5]. Research on LDPC codes has, to date, focused almost exclusively upon LDPC block codes (LDPC-BCs). However, LDPC convolutional codes (LDPC-CCs) [6], [7] may be better suited to certain applications than their block-code counterparts. This is because LDPC-CCs are able to encode and decode arbitrary lengths of data without the need to fragment them into fixed-sized blocks. Many packet switching networks, including those based on the Ethernet frame format, utilize a Protocol Data Unit (PDU) that can vary in size. For example, in the IEEE 802.11 wireless standards, variable-length frames are used.

Although the underlying decoding algorithm for LDPC-CCs is intrinsically the same as for LDPC-BCs, it assumes a very different decoder architecture [6], [8]. For LDPC-BCs, a great deal of effort has been devoted to implementing the iterative decoder using both field-programmable gate arrays (FPGAs) [9]–[12] and application-specific integrated circuits (ASICs) [13]–[16]. Early work focused on a fully parallel architecture [13], where each variable/check node of the Tanner graph is physically realized on the circuit, and the decoding messages are processed at such nodes and passed between them. Despite its high throughput (500-Mb/s information throughput in [13]), the fully parallel architecture does not scale well with code length. Also, as the block length grows, long interconnections between nodes cannot be totally avoided, causing routing congestion. In high-throughput application scenarios, a fully parallel architecture is desirable, and certain implementation optimization measures can be taken to alleviate the routing problem [16]. For applications where the throughput is not critical, larger LDPC-BCs can be implemented by customizing the code design to better fit a partially parallel decoder architecture [9]–[11], [14], [17].

In an LDPC-CC decoder, multiple iterations are translated into multiple processors to deal with belief propagation on an infinite Tanner graph. The statistical messages, in the form of log-likelihood ratios (LLRs), are passed from one processor to the next. The processors are identical, and each of them contains storage elements for LLRs plus some number of variable node units (VNUs) and check node units (CNUs). The concatenated processors form a naturally pipelined architecture [6]. As the inputs arrive from the channel, the LDPC-CC decoder can output decoded bits continuously after an initial delay. Both FPGA and ASIC architectures have been proposed for LDPC-CC decoders [18], [19].

In this paper, we propose a low-cost serial decoder architecture for LDPC-CCs. Instead of mapping multiple decoding iterations into the same number of processors, we use a single processor. The data frames are sent through this processor iteratively until correctly decoded or a maximum iteration number is reached. To make serial decoding possible, encoded data

Manuscript received April 11, 2007; revised August 27, 2007. This work was supported in part by the NSERC through a Discovery Grant, the SRC under Grant 1170.001, the National Science Foundation under Grant CCR02-05310 and Grant CCF05-15012, and NASA under Grant NNG05GH736. This paper was presented in part at the IEEE International Symposium on Circuits and Systems, Island of Kos, Greece, May 2006. This paper was recommended by Associate Editor V. Öwall.

S. Bates, Z. Chen, and L. Gunthorpe are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G 2V4 Canada (e-mail: zgchen@ece.ualberta.ca).

A. E. Pusane, K. S. Zigangirov, and D. J. Costello Jr. are with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA.

Digital Object Identifier 10.1109/TCSI.2008.918002

frames must be terminated before being transmitted through the channel. This serial architecture is proposed based on the following considerations.

- LDPC-CCs demonstrate comparable performance to LDPC-BCs with a much smaller constraint length (a code property defined in Section II) than the block length [6]. The constraint length of LDPC-CCs is directly related to the actual memory consumption of a decoding processor in circuits. So LDPC-CCs potentially use less memory than LDPC-BCs with the same performance. However, using many processors in the LDPC-CC decoder cancels out this advantage. By using a single processor, the memory consumption is minimized; but, as will be discussed later, a frame buffer is required in the serial decoder, which reduces the memory savings.
- An LDPC-CC decoder with a single processor consumes minimal hardware resources. This architecture leads to a low-cost design. As a tradeoff, the decoding throughput is lower.
- The serial architecture is particularly suitable for LDPC-CCs with large code memory (a code property defined in Section II). Such codes demonstrate very good error performance. However, mapping each iteration to a processor for these codes would consume too many hardware resources.
- In real systems, data is usually organized in frames. LDPC-CCs can be applied to a variety of frame lengths when encoder termination is applied. The proposed serial architecture naturally fits such a scenario.

The proposed low-cost serial architecture is implemented for a (2048,3,6) LDPC-CC (see Section II) on an Altera Stratix FPGA. This implementation achieves excellent frame error rate (FER) performance and a moderate throughput using only a fraction of the targeted FPGA resources. Therefore, the architecture is suitable for low-cost, moderate throughput applications where good error correction capabilities are required, such as wireless mobile telephone networks and space and satellite communications.

The remainder of this paper is organized as follows. In Section II, we present a brief overview of LDPC-CCs. In Section III, the performance of these codes is demonstrated via simulation. In Section IV, we discuss the termination of data streams encoded with LDPC-CCs. The architecture of the proposed decoder is discussed in Section V. Synthesis and performance results for the implementation of the (2048,3,6) LDPC-CC decoder are presented in Section VI. Section VII concludes the paper.

II. Low-Density Parity-Check Convolutional Codes

An LDPC-CC[1] is defined by an infinite but periodically time-varying parity-check matrix. Suppose the parity-check matrix is $\mathbf{H}$. The code is defined as the null space of $\mathbf{H}$, i.e., $\boldsymbol{v}\mathbf{H}^T = \mathbf{0}$, where $\boldsymbol{v}$ is the codeword row vector. The infinite parity-check matrix of a rate $b/c\,(b < c)$ LDPC-CC can be written as (in transposed form, also known as the syndrome former) shown in (1), at the bottom of the page, where the $\mathbf{H}_i^T(t)$ $(i = 0, 1, \cdots, m_s)$ are periodically time-varying submatrices of binary values. If the period is T_s, then $\mathbf{H}_i^T(t) = \mathbf{H}_i^T(t + T_s)$ for all i and t. An important parameter of the code is the (syndrome-former) memory m_s, which determines the maximum number of previous bits that are involved in the encoding of the current bit. The *constraint length* of the LDPC-CC is defined as $\nu = c \cdot (m_s + 1)$. For regular LDPC-CCs, both the column weight (K) and the row weight (J) of $\mathbf{H}^T$ are constant,[2] and the code is referred to as an (m_s, J, K) LDPC-CC. For fixed J and K, the code performance improves with increasing m_s. In this paper, we use the term *code memory* to represent m_s, which is a property of the code, different from the term memory usually used in circuit definitions. Although the parity-check matrix $\mathbf{H}$ is infinite, the code memory m_s is finite. LDPC-CCs are applicable to data frames of finite length through encoder presetting and termination, which will be discussed in Section IV.

The submatrices in (1) are of size $c \times (c - b)$, depending on the code rate. For instance, the submatrix of a rate 5/6 LDPC-CC ($b = 5$, $c = 6$) can be written as $\mathbf{H}_i^T(t) = [h_i^{(5)}(t), h_i^{(4)}(t), h_i^{(3)}(t), h_i^{(2)}(t), h_i^{(1)}(t), h_i^{(0)}(t)]^T$. Let the coded sequence $\mathbf{v}$ be written as $\mathbf{v} = \{\cdots, \mathbf{v}(0), \cdots, \mathbf{v}(t), \cdots\}$, where $\mathbf{v}(t)$ is a sub-block of size $1 \times c$. From the definition $\mathbf{v}\mathbf{H}^T = \mathbf{0}$, the parity-check constraints can be written as

$$\mathbf{v}(t)\mathbf{H}_0^T(t) + \mathbf{v}(t-1)\mathbf{H}_1^T(t) + \cdots + \mathbf{v}(t-m_s)\mathbf{H}_{m_s}^T(t) = \mathbf{0}, \quad t = \cdots, 0, 1, \cdots. \tag{2}$$

This parity-check equation can be directly used to encode LDPC-CCs. As long as $\mathbf{H}_0^T(t)$ is full rank, (2) can be solved

[1]The nomenclature used in this paper is based, in part, upon that presented in [6].

[2]In fact, the column weights for the first $(c-b)\cdot m_s$ columns of the syndrome former can be less than K when it is truncated to be applied to frames of finite lengths.

$$\mathbf{H}^T = \begin{bmatrix} \ddots & & \ddots & & & & \\ & \mathbf{H}_0^T(t-m_s) & \cdots & \mathbf{H}_{m_s}^T(t) & & & \\ & & \ddots & \vdots & \ddots & & \\ & & & \mathbf{H}_0^T(t) & \cdots & \mathbf{H}_{m_s}^T(t+m_s) & \\ & & & & \ddots & \vdots & \ddots \end{bmatrix} \tag{1}$$

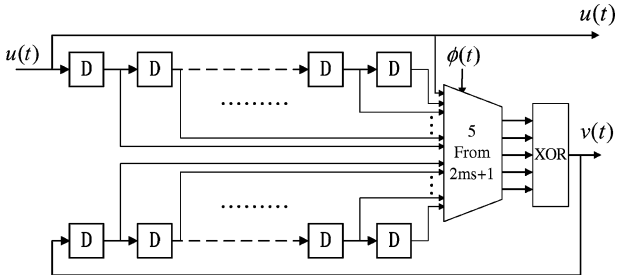


Fig. 1. Encoder realization for an $(m_s, 3, 6)$ rate 1/2 LDPC-CC.

and the subblock $\mathbf{v}(t)$ can be represented as a linear combination of previous subblocks $\mathbf{v}(t-1), \cdots, \mathbf{v}(t-m_s)$. For simplicity, we take rate 1/2 codes as an example, i.e., $b = 1, c = 2$. The constituent submatrices are then

$$\mathbf{H}_i^T(t) = \begin{bmatrix} h_i^u(t) \\ h_i^v(t) \end{bmatrix}, \quad i = 0, \cdots, m_s,\ t = \cdots, 0, 1, \cdots \tag{3}$$

where choosing $h_0^v(t) = 1$ for all t satisfies the full rank condition and ensures systematic encoding. Now, assuming the subblocks of a codeword are $\mathbf{v}(t) = (u(t), v(t))$, where the symbol u corresponds to systematic information bits and v corresponds to parity bits, and combining the above equations, we obtain the following encoding equation for rate 1/2 systematic LDPC-CCs:

$$v(t) = \sum_{i=0}^{m_s} h_i^u(t) u(t-i) + \sum_{i=1}^{m_s} h_i^v(t) v(t-i). \tag{4}$$

Letting the input data sequence be $\mathbf{u} = \{u(0), u(1), \cdots, u(t), \cdots\}$, the coded sequence is then given by $\mathbf{v} = \{u(0), v(0), u(1), v(1), \cdots, u(t), v(t), \cdots\}$. The coefficients $h_i^u(t)$ and $h_i^v(t)$ in (4) are periodically time-varying elements in $\mathbf{H}^T$. Each possible combination of the coefficients represents a parity-check equation (column) in $\mathbf{H}^T$. Due to the sparsity of $\mathbf{H}^T$, only a few of the coefficients are nonzero. For example, for a regular LDPC-CC with check node degree $K = 6$, only $5 = 6 - 1$ coefficients are nonzero, i.e., only five of the previous information and parity bits and the current information bit are needed to generate the current parity bit.

A block diagram of the encoder for an $(m_s, 3, 6)$ rate 1/2 LDPC-CC is shown in Fig. 1. The encoder can be implemented using shift registers, multiplexers, and XOR gates. It is a finite state machine (FSM) whose state bits are the previous m_s information bits $(\{u(t-1), \cdots, u(t-m_s)\})$ and the previous m_s parity bits $(\{v(t-1), \cdots, v(t-m_s)\})$. The inputs are the information bit sequence and the phase select signal $\phi(t)$, which is introduced to address the periodicity of the encoder. The coefficients $(h_i^u(t), h_i^v(t))$ in (4) are time varying with period T_s, and the phase signal $\phi(t)$ picks the right combination from the $2m_s$ state bits and the current input information bit.

III. Performance of LDPC-CCs

In Fig. 2, the bit error rate (BER) performance of a number of regular LDPC-BCs and LDPC-CCs is compared. Two turbo codes are also included for reference. All codes were simulated over the additive white Gaussian noise (AWGN) channel with binary phase-shift keying (BPSK) modulation. The LDPC codes were all decoded using the belief propagation algorithm, and the turbo codes were decoded with a MAP-based iterative decoder. The parameters of the codes are summarized in Table I. It can be seen that both randomly and algebraically constructed LDPC-CCs have the ability to outperform LDPC-BCs, even when the constraint length is much smaller than the block code length. The rate 1/2 (2048,3,6) LDPC-CC (with constraint length $\nu = 4098$) performs slightly better than the rate 1/2 (100000,3,6) LDPC-BC and the rate 5/6 (128,3,18) LDPC-CC (with $\nu = 774$) performs the same as the (2048,6,32) LDPC-BC.[3] Also, the (512,3,6) LDPC-CC (with $\nu = 1026$) performs slightly better than the $(\mathrm{N} = 1530)$ turbo code,[4] but the (2048,3,6) LDPC-CC (with $\nu = 4098$) performs slightly worse than the $(\mathrm{N} = 5000)$ turbo code. This is because

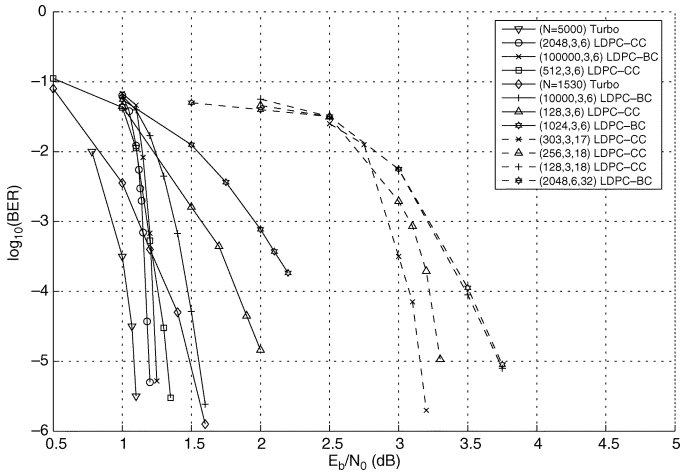


Fig. 2. Performance of a variety of channel coding schemes. For clarity the curves are labeled in order of increasing E_b/N_0 required to achieve a BER of 10^{-5}. Turbo codes used 20 iterations. LDPC codes used up to 100 iterations. For the LDPC-CCs, the BER was obtained in a streaming fashion without termination.

TABLE I
KEY PARAMETERS OF THE CODES PRESENTED IN FIG. 2. THE (303,3,17) LDPC-CC AND THE (2048,6,32) LDPC-BC WERE OBTAINED THROUGH ALGEBRAIC CONSTRUCTION, WHILE ALL OF THE OTHER LDPC CODES WERE OBTAINED BY RANDOM CONSTRUCTION. FOR TURBO CODES, N REPRESENTS THE INTERLEAVER SIZE

Code Id.	Rate	Reference
(N=5000)Turbo	1/2	[24]
(2048,3,6) LDPC-CC	1/2	[25]
(100000,3,6) LDPC-BC	1/2	[26]
(512,3,6) LDPC-CC	1/2	
(N=1530)Turbo	1/2	[27]
(10000,3,6) LDPC-BC	1/2	[25]
(128,3,6) LDPC-CC	1/2	
(1024,3,6) LDPC-BC	1/2	
(303,3,17) LDPC-CC	14/17	[7]
(256,3,18) LDPC-CC	5/6	[28]
(128,3,18) LDPC-CC	5/6	[28]
(2048,6,32) LDPC-BC	1723/2048	[20]

[3]Note that this LDPC-BC is not full rank and therefore its rate is slightly more than $1 - (6/32)$. This code has been selected for the IEEE 802.3an standard [16], [20].

[4]This turbo code is used in the CDMA2000 standard, where the 1/2 code rate is obtained through puncturing.

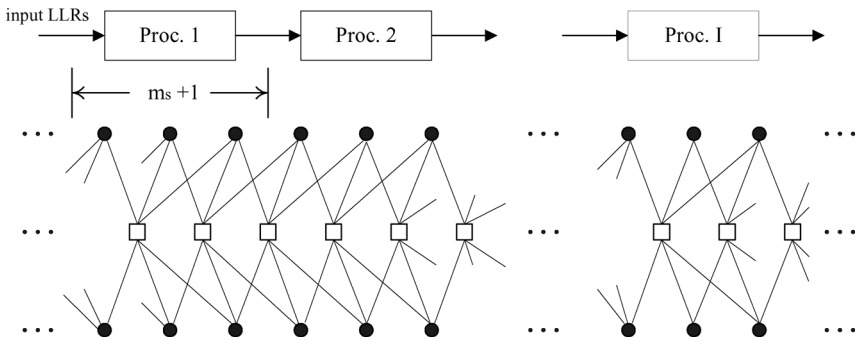


Fig. 3. Rate 1/2 (3,6) LDPC-CC decoder based on its infinite Tanner graph. Each processor deals with one part of the Tanner graph. Round black nodes are variable nodes and square nodes are check nodes. Note that this is a trivial example of an LDPC-CC with memory $m_s = 2$ and period $T_s = 1$.

only regular LDPC-CCs were considered here. Optimized irregular LDPC-BCs [21] and irregular LDPC-CCs [22], [23] both perform better than turbo codes. Based on Fig. 2, it is evident that LDPC-CCs exhibit very good performance over a range of code rates.

IV. LDPC-CC Encoder Termination

To apply LDPC-CCs with infinite parity-check matrices to data frames of finite length, the encoder must be properly initialized and terminated. Encoding of a frame can be initialized by assuming all previous bits to be zeros, i.e., the encoder in Fig. 1 starts from the all-zero state. We call this *encoder presetting*, which is commonly used for convolutional codes. Presetting is trivial and not often mentioned in the literature because it can be easily done by setting the encoder to the all-zero state. Similarly, convolutional encoders must be terminated properly at the end of the frame. To terminate a conventional convolutional encoder, either a few tail bits are fed into the encoder, or tail-biting is used [29]. For LDPC-CCs, to be used on frames of finite length, termination is as necessary as presetting. We use tail bits (called the termination sequence) to terminate an LDPC-CC encoder. For a properly initialized and terminated frame, extending it in both directions with an infinite number of zeros would correspond to an infinitely long LDPC-CC codeword. We briefly discuss termination in this section because it is important in generating finite length frames and decoding them iteratively.

Consider the encoder description in Section II. An LDPC-CC encoder is an FSM. In this case, a termination sequence is defined as a sequence of information bits that takes the encoder from any given state back to the all-zero state, which is defined as the state in which all m_s information bits and parity bits stored in memory are zero.

It has been shown in [30] that almost all $(m_s, J, 2J)$ rate 1/2 regular LDPC-CCs can be terminated with a tail length no greater than $2(m_s + 1)$. Actually, all of the rate 1/2 LDPC-CCs we simulated require a termination sequence of only $L = m_s + \epsilon$ information bits $(\epsilon \ll m_s)$. (The extra ϵ bits are needed because sometimes the LDPC-CC cannot be terminated with exactly m_s bits. For the (2048,3,6) LDPC-CC, experiments show that ϵ is less than $0.02m_s$.) Generating the termination sequence from any given encoder state requires a special termination circuit to be added to the encoder in Fig. 1 (see [31], [32]). The implementation complexity of an LDPC-CC encoder with termination is still small compared to the decoder [33]. However, the termination sequence has an impact on the BER vs Signal-to-Noise Ratio (SNR or E_b/N_0) performance of an LDPC-CC. Since the termination sequence does not convey information, it reduces the overall rate of the code. Therefore, for rate 1/2 codes, if the data frame to be encoded is of length N_f bits, the effective code rate R' is given by $R' = N_f/(N_f + L)R \approx N_f/(N_f + m_s)R$. The rate loss due to termination is directly related to the frame length N_f. When N_f is small with respect to m_s, the impact of termination is quite large. However, when $N_f \gg m_s$, the impact of termination is minimal. LDPC-CC performance curves with termination are presented in Section VI.

V. Decoder Architecture

An LDPC-CC decoder is based on the same message-passing algorithm as an LDPC-BC decoder. However, the decoder architecture is fundamentally different. In the original BP decoding algorithm for LDPC-BCs, the decoding iterations are executed one after another. For a given code block, the next iteration cannot begin until the current iteration is completed for all of the variable/check nodes. To accelerate the iterations, both horizontal partitioning [15] and vertical partitioning [34] methods have been proposed. In the latter case, shuffled iterative decoding was used, where the BP algorithm takes a serial form to make the check node updating partly utilize the information obtained from other nodes in the same round of decoding.

For LDPC-CCs, different decoding iterations can be executed at the same time for different nodes. For example, if node $V(t)$ is undergoing iteration i, an earlier node $V(t - m_s - 1)$ can be undergoing decoding iteration $i + 1$, because these two nodes do not participate in a common parity-check equation. It follows that LDPC-CCs can be decoded using a sliding window of size $(m_s + 1)I$, where I is the number of decoding iterations to be performed. The decoding window contains I identical sections, with each section consisting of $(m_s + 1)c$ variable nodes and $(m_s + 1)(c - b)$ check nodes, and a separate processor can be assigned to each section. Fig. 3 shows a rate 1/2 LDPC-CC decoder with I processors. In the original decoding algorithm proposed in [6], each processor contains a few first-in first-out (FIFO) delay lines and a few VNUs and CNUs. Decoding statistics (LLRs) are stored in these FIFOs and updated by the node units. The processors are identical and concatenated in series, forming a pipelined architecture (as shown in Fig. 3). In this

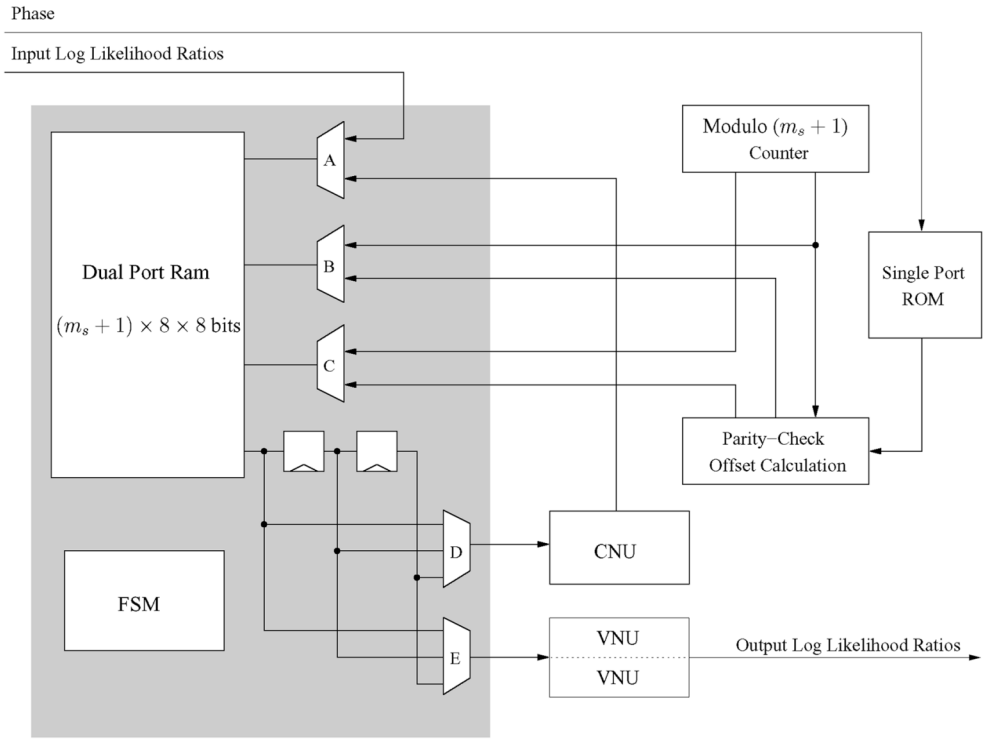


Fig. 4. Design of a single memory-based processor for (3,6) regular LDPC-CC decoding. The gray area is clocked using the $\times 3$ clock. One CNU and two VNUs are used for a single processor.

case, the message passing is localized and routing is only required inside one processor. For the same performance, a single LDPC-CC processor is much smaller than an LDPC-BC decoder [8]. It can be seen that in an LDPC-CC decoder, the iterative decoding process in the temporal dimension is "unwrapped" and mapped into processors in the spatial dimension.

A. Memory-Based Processor Design

The design of a single processor is considered first. The FIFOs in a processor can be realized with registers [19]. However, this is not a good choice for FPGA devices because it requires too many logic elements. A more balanced memory-based design is chosen instead because modern FPGAs carry on-chip RAMs [18]. The FIFOs in the processor are implemented as a circular buffer. The LLRs are stored and tracked with pointers.

For the rate 1/2 (3,6) regular LDPC-CC discussed in this paper, each variable node has four edges (one from the channel LLR and three from check nodes) and requires four LLRs to be saved in memory. To reduce the number of read and write operations per memory unit, we split the circular buffer into eight columnar memory banks. Half of them are for information-bit LLRs and half are for parity-bit LLRs. The height of these RAMs should be at least m_s+1. The width of these RAMs is 8 b for each LLR. Using pointers, the oldest LLR in the buffer is read as the output, and the input LLR can be written into the buffer accordingly. Similarly, the VNUs and CNUs update the LLRs at their proper addresses in the buffer. The calculation of these addresses is dependent on the parity-check matrix and is realized with a single-port ROM plus a simple calculation circuit.

TABLE II
FPGA RESOURCE REQUIREMENTS OF AN ENTIRE (2048,3,6) LDPC-CC DECODER WITH A SINGLE PROCESSOR. THE WORD-LENGTH OF THE LLRS IS $W = 8$ bits. THE FIGURES IN PARENTHESES REFER TO THE PERCENTAGE UTILIZATION OF RESOURCES ON THE TARGETED FPGA

Parameter	Value
Device Name	Altera Stratix EP1S80
Logic Elements	4004 (5.05%)
Memory Bits for the processor (RAM and ROM)	278984 (3.46%)
Memory Bits for the frame buffer	516096 (6.40%)
Max Clock Frequency	60MHz
Max $\times 3$ Clock Frequency	180MHz

Here a modified min-sum approximation for check node processing in [35] is used. This approximation is an improvement over the min-sum approach and still has a simple hardware implementation, but it entails some loss compared to sum-product processing, as noted in Section VI-B on decoder performance.

Fig. 4 shows a diagram of the processor design. We note below a few features of the design.

- With a dual-port RAM, we can read and write LLRs in a single clock cycle.
- Each check node update may take up to two clock cycles to simultaneously read/write the memory banks composing the circular buffer, and each variable node update (including I/O) takes one clock cycle because the memory banks are partitioned based on variable node edges. The circular buffer is over-clocked to $3\times$ the base clock frequency [18] (see Table II) so that one round of message exchange between variable nodes and check nodes can be completed within one base clock cycle. To synchronize the

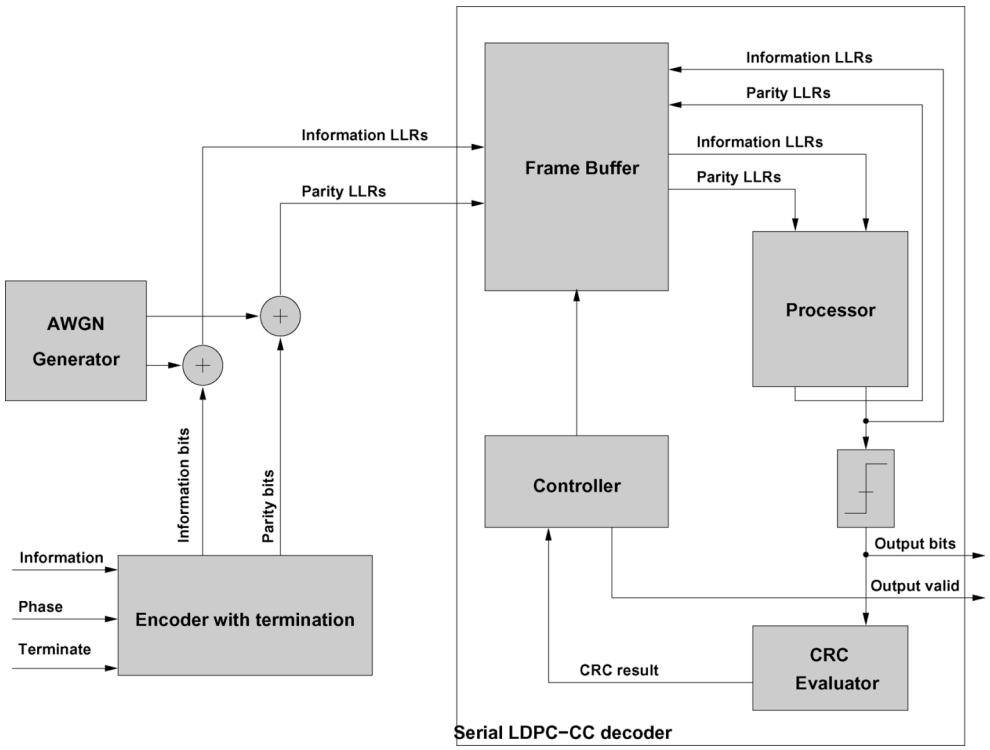


Fig. 5. Decoder architecture implemented on an FPGA. This includes the LDPC-CC encoder, decoder, and noise generators. The decoder consists of a frame buffer, a single processor, a decoder controller, and a CRC evaluator.

data transfer between the over-clocked domain and the rest of the circuit, simple parallel-to-serial and serial-to-parallel conversion circuits are used.

- The modulo $(m_s + 1)$ counter and the parity-check offset calculation unit are used to generate the right addresses for LLR updates in the memory banks.
- A simple FSM combined with a few multiplexers coordinates the circular buffer operations.

B. Serial Decoding Architecture

With a single processor designed, it is easy to tile them as needed. To achieve a high throughput, many processors can be pipelined. However, in applications where throughput is not critical, this may not be the best choice because each additional processor consumes more resources. Moreover, for strong LDPC-CCs with large code memory, using many processors can be a very expensive solution.

Take the (2048,3,6) LDPC-CC as an example. For such a code, the decoder requires more iterations to converge than smaller codes, especially at low SNRs. If 100 iterations are required, then 100 processors would be needed to maximize throughput. The required storage space for the LLRs in decoding is then $100 \times (2048 + 1) \times 8 \times 8 \approx 13\ \mathrm{Mbits}$.

To avoid this expense and to achieve a low-cost design, we propose a serial decoding architecture. A single processor is used for the LDPC-CC decoder. Each encoded data frame passes through this processor multiple times to be decoded. The data frames are first encoded and terminated. A decoding iteration is accomplished by using a controller that manages the flow of frames through the processor (see Fig. 5). The controller takes a single frame as its input and places it in a frame buffer.[5] The bit LLRs are then updated using the processor and the outputs are returned to the frame buffer. After the last bit of each frame is processed, the controller can either output the data or send it through the processor again, depending on the result of a cyclic redundancy check (CRC). The frame buffer enables the looping of the frame LLRs for serial decoding, but it also adds an extra memory cost, as will be discussed in Section VI.

In our design, we used the Ethernet frame format as the basis for the finite length data. This is because Ethernet is one of the most prevalent frame formats in data communications. Since our frames are Ethernet-compliant, we use a 4-byte CRC after every loop through the processor. We can therefore determine if the frame has been decoded correctly and use this indicator as a stopping criterion. The frame will loop through the decoder as many times as necessary until it passes the CRC check or the maximum number of iterations is reached.

C. Parallelization and Serialization of LDPC Decoding

The proposed architecture in this paper indicates that LDPC-CC decoding can have a parallel or serial architecture. This may be easily confused with the often-mentioned parallelization of LDPC-BC decoding. However, they are different. Actually, LDPC-CCs reveal another dimension of parallelization in LDPC decoding.

[5]The frame buffer is a memory unit reserved for frame data storage and its size should be large enough to accommodate the maximum frame size. It is a separate memory unit from the circular buffer in the processor, whose size is determined by m_s.

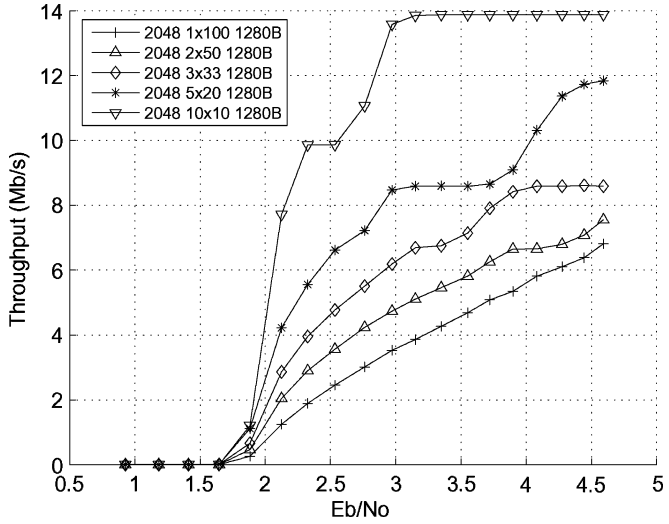


Fig. 6. Decoding throughput for different decoder configurations. The clock frequency is 50 MHz and the frame size is 1280 bytes.

For LDPC-BCs, the node updates in the Tanner graph can be done at the same time within the same iteration. Therefore, LDPC-BC decoding can be parallelized in the node dimension. The fully parallel and partially parallel architectures mentioned in Section I are based on node parallelization.

For LDPC-CCs, it is also possible to parallelize decoding in the node dimension ([36], [37] for time-invariant codes and [38] for time-varying codes). In addition, the iterations can be performed in parallel. This is another dimension of parallelization. The decoding architecture proposed in this paper is serial in the iteration dimension. But performing iterations fully or partially in parallel is also possible for LDPC-CC decoding, i.e., using more than one processor and less looping through each processor.

To investigate this possibility, we approached the design in such a way that the number of processors and the maximum number of loops can be chosen as variables during synthesis. This allows us to test many different configurations. We use the naming scheme "$m_s\, p \times n$" to designate a design using an LDPC-CC with code memory m_s, p processors, and a maximum of n loops.[6] Fig. 6 shows decoding throughputs using one, two, three, five, and tenprocessor arrays. The cost of the decoder (not including the frame buffer, the CRC evaluator, and the looping logic) increases linearly with the number of processors used. However, the throughput does not increase in the same fashion. With $p > 1$ processors in the decoder, the actual number of decoding iterations performed on a frame can only be an integer multiple of p. This causes some power inefficiency, since decoding may not stop as early as possible. As the SNR increases, fewer iterations are required to correctly decode the frame. However, if there are $p > 1$ processors, a reduction in the number of iterations (e.g., from p to $p-1$, or from $2p$ to $2p-1$) may not increase the throughput. This is reflected in the throughput curves in Fig. 6, which exhibit plateaus when $p > 1$. For instance, when $p = 2$, the decoding processor array cost is twice that of the serial decoder $(p = 1)$, but the throughput gain is less than a factor of two, especially at high SNRs.

The decoding latency of this LDPC-CC decoder architecture is also dependent on p and n, because it is proportional to the product of p and the number of loops used. Therefore, for two configurations with the same maximum number of iterations (pn), the maximum decoding latency is almost the same, i.e., using a single processor will not increase the maximum decoding latency. However, using a single processor decreases the average decoding latency because $p = 1$ and the average number of loops is much smaller than 100 at high SNRs, as will be shown in Section VI-B. Finally, we note that there is a slight latency penalty in the CRC check, which is negligible compared to the decoding processor latency.

From the above discussion, we see that it is advantageous to use a single processor in the decoder for the (2048,3,6) LDPC-CC. With $p = 1$, a frame is processed only as many times as needed to properly decode it, due to the intermediate CRC verification of frames. This improves both the decoding throughput and the average decoding latency at high SNRs where most frames take only 2 to 3 iterations to decode. The single processor decoder is also very small in terms of area and power consumption. Such a decoder can be integrated with an Ethernet Media Access Control (MAC) to produce a very low-power data-link layer solution. At low SNRs, multiprocessor designs have higher throughputs, but higher cost. Making the best choice in this tradeoff is application dependent and also code dependent. The serial decoder design with a single processor can be considered as a minimum cost special case of a more general partially parallel structure.

VI. Decoder Implementation

A. Specifications

The single processor decoder was implemented for the (2048,3,6) LDPC-CC on an Altera Stratix EP1S80 FPGA. This is a mid-sized FPGA with on-chip RAM and dedicated arithmetic blocks. The architecture of the decoder is as described in Section V-B.

As can be seen from Table II, the decoder occupies only a few percent of the targeted FPGA resources. Therefore, it is expected that an ASIC implementation of the same design would occupy very little area and consume very small amounts of power. The processor memory consumption and the memory for the frame buffer are listed separately to provide a better picture of the cost of the decoder. Firstly, for a partially parallel decoder with p processors, both the logic elements and the processor memory size increase linearly with p, but the frame buffer size is independent of p and depends only on the maximum frame length. In Table II, the frame buffer accounts for most of the memory consumption in the decoder because of the large maximum frame length. Secondly, the processor memory serves as an "internal" memory which must be continuously updated, while the frame buffer is a "global" memory used only for buffering.

[6]Here one "loop" means the coded frame going through the processor array for one round, while "iteration" refers to the actual message passing decoding iteration. For a p-processor array, a maximum number of loops of n means the maximum number of iterations is $p \cdot n$. In the discussion of the serial decoder with $p = 1$, one loop equals one iteration, and the maximum number of iterations is n.

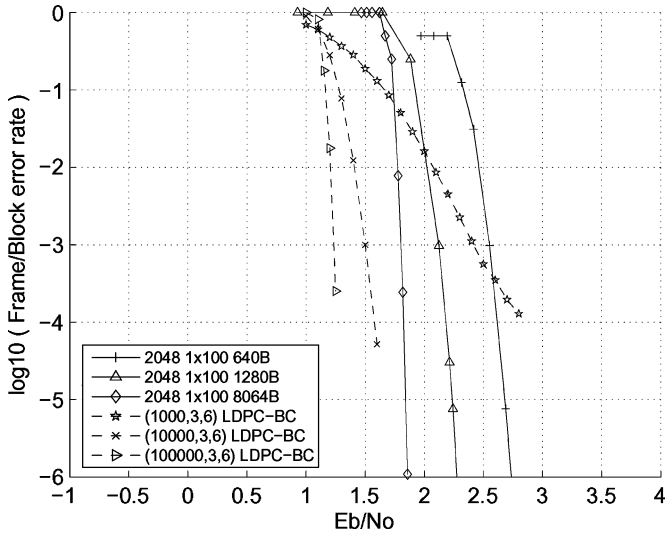


Fig. 7. FER performance of the LDPC-CC decoder for a variety of frame sizes and the BLER performance of several LDPC-BCs.

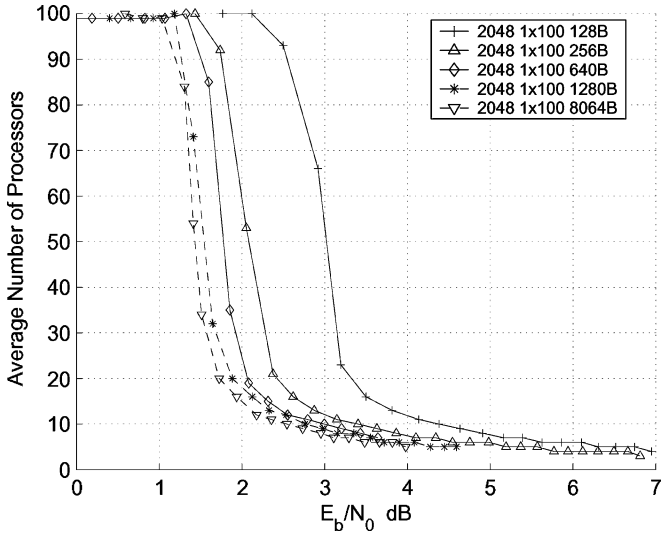


Fig. 8. Average number of processor iterations for a variety of frame sizes.

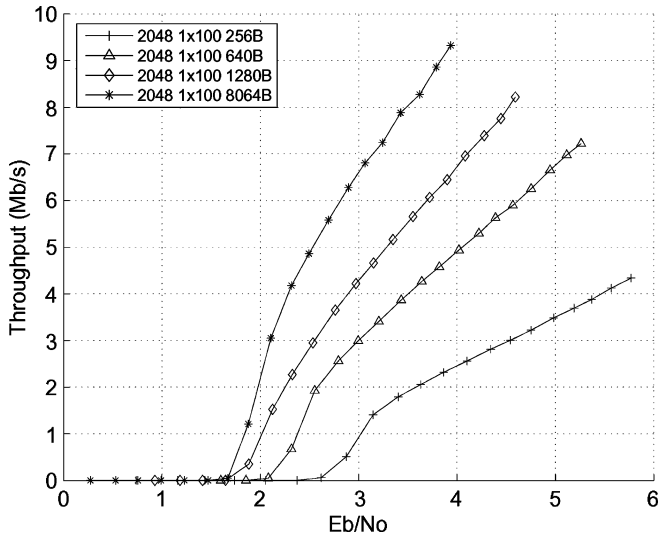


Fig. 9. Average throughput for a variety of frame sizes.

B. Decoder Performance

The decoder was tested using different frame sizes ranging from 640 bytes (640B) to 8064 bytes (8064B). This allowed us to compare the performance for small and large frame sizes. As noted in Section V, the decoder looped the received data through the single processor until either the CRC was evaluated correctly or the maximum number of iterations was reached. For all frame sizes the FER, the average number of iterations, and the average throughput were measured.

In Fig. 7, the error correcting capabilities of the LDPC-CC can be seen. As long as a certain threshold in E_b/N_0 is exceeded, then small increases in signal power lead to large drops in FER. When the frame size is small, the overhead of termination has a large impact, since the effective rate of the code is much less than 1/2. At very large frame sizes such as 8064B, this overhead is only around 0.1 dB.

The software simulated block error rate (BLER) performance of three LDPC-BCs is also plotted. Our FPGA decoder outperforms the (1000,3,6) LDPC-BC for most frame sizes. For the frame size of 8064B, the FER performance is about 0.1–0.2 dB worse than the BLER of the (10000,3,6) LDPC-BC, which is somewhat degraded compared with the simulated BER performance shown in Fig. 2. Besides termination overhead, this is mainly caused by the parity-check node approximation used in hardware. Despite these factors, the (2048,3,6) LDPC-CC demonstrates a very steep waterfall region in the FER curves, and no error floor was observed as low as an FER of 10^{-6}. Also note that, in Fig. 7, the LDPC-BC performance curves are for different codes with different block lengths and different decoders, whereas the LDPC-CC curves are all for the same code and the same decoder, but just with different frame sizes.

Fig. 8 shows that the average number of iterations through the single processor in the decoder decreases as E_b/N_0 increases. This has a direct impact on the throughput, since we can proceed to the next frame in a much shorter time. Fig. 9 shows that the maximum throughput occurs when the frame size is large and the E_b/N_0 is high. In this case, we can reach a maximum 9-Mb/s information throughput. The average decoding latency is also significantly less at high SNRs for similar reasons. Note that as the frame size is reduced, the potential throughput of the decoder is also decreased. This is due to the fact that the overhead of termination is a fixed number of bits and hence begins to dominate at small frame sizes. For higher rate LDPC-CCs, the throughput would be higher.

C. Discussion

LDPC-CCs explore a new design space for LDPC decoding and the proposed serial decoding architecture differs from previous designs in that both the iteration and node dimensions are serial, targeting a low-cost implementation.

As shown in Section III, the implemented (2048,3,6) regular LDPC-CC outperforms regular (3,6) LDPC-BCs of length up to 100 000 b. Even with the termination overhead, the FER performance obtained in Section VI-B with large frame sizes is still very good. Fully parallel LDPC-BC decoder implementations usually use codes of moderate block lengths (1000–3000

b) [13], [16]. Implementation of rate 1/2 LDPC-BCs with large block lengths ($\sim$10 000 b) appeared in [9] and [14], where information throughputs of about 50 Mb/s on FPGA devices were achieved. A more recent implementation [11] on an FPGA employed an irregular rate 1/2 LDPC-BC of block length 9036, whose BER performance is roughly the same as for the (2048,3,6) regular LDPC-CC. Using this as a comparison will give us a better picture of the cost of the serial LDPC-CC decoder architecture.

The LDPC-BC decoder in [11] was implemented on a Xilinx Virtex-II FPGA device. With the maximum number of iterations set to 60, the irregular 1/2 LDPC-BC achieved a maximum 15-Mb/s information throughput. About 34 000 slices and about 1.8 Mb of on-chip block RAM were used. Because it was a design accommodating three code rates (1/2, 5/8, and 7/8), the actual slice usage of the rate 1/2 code would be somewhat less. Also, because the RAM in [11] is used to store exchanged messages between nodes, the memory is heavily reused across the three code rates, whereas the rate 1/2 code has a larger number of nodes and messages to handle. In our serial LDPC-CC decoder design, the maximum throughput is relatively low: 9 Mb/s with the maximum number of iterations set to 100. However, only a few thousand logic elements are used and the total memory used including the frame buffer is only about 800 Kb. The significant savings in logic gates is due to the fact that the serial decoder only uses a single processor with 1 CNU, 2 VNUs, and some control logic. As for memory consumption, although 800 Kb is a savings compared with 1.8 Mb, the memory savings obtained by using a single processor in a serial decoder is mitigated by having to use a frame buffer. The processor memory itself only requires about 300 Kb. For an ASIC implementation of the serial decoder architecture proposed in this paper, the frame buffer cost can be significantly reduced, because it is not a time critical part of the decoding circuit (using a $1\times$ clock frequency while the processor memory uses a $3\times$ clock frequency) and only serves as a circular or FIFO buffer. For example, a DRAM can be used for the frame buffer to minimize area. In this way, the actual cost per bit for the frame buffer is much less than for the processor memory.

VII. Conclusion

We have proposed a memory-based serial decoder architecture for LDPC-CCs, which is suitable for LDPC-CCs with large code memory and for low-cost low-power designs. A (2048,3,6) LDPC-CC decoder was implemented on an Altera FPGA device. Only a small fraction of hardware resources were used. A maximum of 9-Mb/s decoding throughput was achieved with a frame length of 8064 bytes, and the results suggest that, to achieve optimum performance, small frames should be avoided whenever possible.

The FPGA results indicate that an ASIC implementation of such a serial decoder could yield a low-cost design with low power consumption, which is the subject of ongoing work. Future work also includes implementing high rate LDPC-CCs, irregular LDPC-CCs, and parallelizing node operations, all of which will lead to substantial improvements in throughput and/or performance.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[2] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996.

[3] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[4] S.-Y. Chung, G. D. 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 Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[5] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981.

[6] A. Jiménez-Felström and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2181–2191, Sep. 1999.

[7] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, Jr., "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[8] D. J. Costello, Jr., A. E. Pusane, S. Bates, and K. S. Zigangirov, "A comparison between LDPC block and convolutional codes," in *Proc. Inf. Theory Applications Workshop*, San Diego, CA, Feb. 2006.

[9] T. Zhang and K. K. Parhi, "A 54 MBPS (3,6)-regular FPGA LDPC decoder," in *Proc. IEEE Workshop Signal Process. Syst.*, San Diego, CA, Oct. 2002, pp. 127–132.

[10] P. Bhagawat, M. Uppal, and G. Choi, "FPGA based implementation of decoder for array low-density parity-check codes," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Mar. 2005, vol. 5, pp. 29–32.

[11] L. Yang, H. Liu, and C.-J. R. Shi, "Code construction and FPGA implementation of a low-error-floor multi-rate low-density parity-check code decoder," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 53, no. 4, pp. 892–904, Apr. 2006.

[12] L. Sun, H. Song, Kumar, B. V. K. Vijaya, and Z. Keirn, "Field-programmable gate-array-based investigation of the error floor of low-density paritycheck codes for magnetic recording channels," *IEEE Trans. Magn.*, vol. 41, no. 10, pp. 2983–2985, Oct. 2005.

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

[14] Y. Chen and D. Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder," in *Proc. IEEE Global Telecommun. Conf.*, San Francisco, CA, 2003, vol. 1, pp. 113–117.

[15] M. M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

[16] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity," in *Proc. IEEE Int. Symp. Circuits Syst.*, Kobe, Japan, May 2005, pp. 5194–5197.

[17] H. Zhong and T. Zhang, "Block-LDPC: A practical LDPC coding system," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 4, pp. 766–775, Apr. 2005.

[18] S. Bates and G. Block, "A memory-based architecture for low-density parity-check convolutional decoders," in *Proc. IEEE Int. Symp. Circuits Syst.*, Kobe, Japan, May 2005, pp. 336–339.

[19] R. Swamy, S. Bates, and T. L. Brandon, "Architectures for ASIC implementations of low-density parity-check convolutional codes," in *Proc. IEEE Int. Symp. Circuits Syst.*, Kobe, Japan, May 2005, pp. 4513–4516.

[20] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on Reed–Solomon codes with two information symbols," *IEEE Commun. Lett.*, vol. 7, no. 7, pp. 317–319, Jul. 2003.

[21] T. J. Richardson, M. A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[22] A. E. Pusane, K. S. Zigangirov, and D. J. Costello Jr., "Construction of irregular LDPC convolutional codes with fast encoding," in *Proc. IEEE Int. Conf. Commun.*, Istanbul, Turkey, Jun. 2006, pp. 1160–1165.

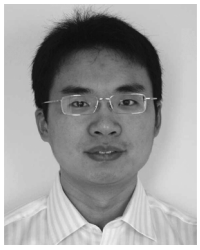
[23] G. Richter, M. Kaupper, and K. S. Zigangirov, "Irregular low-density parity-check convolutional codes based on protographs," in *Proc. IEEE Int. Symp. Inf. Theory*, Seattle, WA, Jul. 2006, pp. 1633–1637.

[24] C. Schlegel and L. Perez, *Trellis and Turbo Coding*, 1st ed. New York: IEEE/Wiley, 2004.

[25] A. Sridharan and D. J. Costello, Jr., "A new construction method for low density parity check convolutional codes," in *Proc. IEEE Inf. Theory Workshop*, Bangalore, India, Oct. 2002, p. 212.
[26] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.
[27] F. Dowla, *Handbook of RF and Wireless Technologies*, 1st ed. Boston, MA: Newnes, 2003.
[28] Z. Chen and S. Bates, "Construction of low-density parity-check convolutional codes through progressive edge-growth," *IEEE Commun. Lett.*, vol. 9, no. 2, pp. 1058–1060, Dec. 2005.
[29] H. H. Ma and J. K. Wolf, "On tail biting convolutional codes," *IEEE Trans. Commun.*, vol. COM-34, no. 2, pp. 104–111, Feb. 1986.
[30] A. Sridharan, "Design and analysis of LDPC convolutional codes," Ph.D. dissertation, Dept. Elect. Eng., Univ. of Notre Dame, Notre Dame, IN, May 2005.
[31] Z. Chen, S. Bates, and X. Dong, "Low-density parity-check convolutional codes applied to packet based communication systems," in *Proc. IEEE Global Telecommun. Conf.*, St. Louis, MO, 2005, vol. 3, pp. 1250–1254.
[32] Z. Chen, S. Bates, D. Elliott, and T. Brandon, "Efficient encoding and termination of low-density parity-check convolutional codes," in *Proc. IEEE Global Telecommun. Conf.*, San Francisco, CA, 2006, pp. 1–5.
[33] S. Bates, D. G. Elliott, and R. Swamy, "Termination sequence generation circuits for low-density parity-check convolutional codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 9, pp. 1909–1917, Sep. 2006.
[34] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, Feb. 2005.
[35] S. Howard, C. Schlegel, and V. Gaudet, "A degree-matched check node approximation for LDPC decoding," in *Proc. IEEE Int. Symp. Inf. Theory*, Adelaide, Australia, Sep. 2005, pp. 1131–1135.
[36] E. Matúš, M. B. Tavares, M. Bimberg, and G. P. Fettweis, "Towards a Gbits/s programmable decoder for LDPC convolutional codes," in *Proc. IEEE Int. Symp. Circuits Syst.*, New Orleans, LA, May 2007, pp. 1657–1660.
[37] M. Bimberg, M. B. Tavares, E. Matúš, and G. P. Fettweis, "A high-throughput programmable decoder for LDPC convolutional codes," in *Proc. IEEE Int. Conf. Application-Specific Syst., Architectures and Processors (ASAP)*, Montreal, QC, Canada, Jul. 2007, pp. 239–246.
[38] Z. Chen, S. Bates, and W. Krzymień, "High throughput parallel decoder design for LDPC convolutional codes," in *Proc. IEEE Int. Conf. Circuits Syst. Commun.*, New Orleans, LA, May 2008, pp. 35–39.

Stephen Bates (S'94–M'03–SM'06) received the Ph.D. degree from Edinburgh University, Edinburgh, Scotland, in 1997.

He then joined Massana, an Irish Fabless Semiconductor company specializing in high-speed Ethernet transceiver design. In 2003, he joined the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada, as an Assistant Professor. His research interests include circuits and systems for high-speed wireline communication protocols, circuits for advanced forward error correction schemes, and the use of reconfigurable logic to solve problems related to communications and bio-informatics.



Zhengang Chen (S'01) received the B.S. and M.S. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1999 and 2002, respectively. He is currently working toward the Ph.D. degree in electrical and computer engineering at the University of Alberta, Edmonton, AB, Canada.

His research interests include error control coding, communication systems, digital application-specific integrated circuit (ASIC) design, and pattern recognition. He is currently working on VLSI architectures for advanced channel coding schemes such as low-density parity-check codes.

Logan Gunthorpe received the B.Sc. degree in electrical and computer engineering from the University of Alberta, Edmonton, AB, Canada, in 2007.

He is currently performing embedded design with Deltatee Enterprises Ltd., Calgary, AB, Canada.

Mr. Gunthorpe is an Engineer in Training (EIT) in APPEGA.

Ali Emre Pusane (S'99) was born in Istanbul, Turkey, in 1978. He received the B.Sc. and M.Sc. degrees in electronics and communications engineering from Istanbul Technical University, Istanbul, in 1999 and 2002, respectively, and the M.Sc. degrees in electrical engineering and applied mathematics and the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 2004, 2006, and 2008, respectively.

His research interests include error control coding, coded modulation, and information theory.

Kamil Sh. Zigangirov (M'95–F'01) was born in the U.S.S.R. in 1938. He received the M.S. degree from the Moscow Institute for Physics and Technology, Moscow, U.S.S.R., in 1962 and the Ph.D. degree from the Institute of Radio Engineering and Electronics, U.S.S.R. Academy of Sciences, Moscow, in 1966

From 1965 to 1991, he held various research positions with the Institute for Problems of Information Transmission, U.S.S.R. Academy of Sciences, Moscow, first as a Junior Scientist and later as a Main Scientist. During this period, he visited several universities in the United States, Sweden, Italy, and Switzerland as a Guest Researcher. He organized several symposia on information theory in the U.S.S.R. In 1994, he received the Chair of Telecommunication Theory at Lund University, Lund, Sweden. In 2003 and 2004, he was a Visiting Professor with the University of Notre Dame, Notre Dame, IN, and with the University of Alberta, Edmonton, AB, Canada. His scientific interests include information theory, coding theory, detection theory, and mathematical statistics. In addition to papers in these areas, he published a book on sequential decoding of convolutional codes (in Russian) in 1974. With R. Johannesson, he coauthored the textbook *Fundamentals of Convolutional Coding* (IEEE, 1999). He authored another book, *Theory of CDMA Communication* (IEEE, 2004).

Daniel J. Costello, Jr. (M'69–F'85) received the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 1969.

He then joined the Illinois Institute of Technology as an Assistant Professor. In 1985, he became a Professor with the University of Notre Dame and later served as Department Chair. In 1999, he received the Humboldt Research Prize from Germany, and in 2000 he was named Bettex Professor of Electrical Engineering at Notre Dame. His research interests are in error control coding and coded modulation. He has authored or coauthored more than 300 technical publications and has coauthored a textbook entitled *Error Control Coding: Fundamentals and Applications* (Pearson Prentice–Hall, 2004, 2nd ed.).

Dr. Costello has been a member of the Information Theory Society Board of Governors since 1983, and in 1986 he served as President. He also served as Associate Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS and the IEEE TRANSACTIONS ON INFORMATION THEORY and as Co-Chair of the ISIT's in Kobe, Japan (1988), Ulm, Germany (1997), and Chicago, IL (2004). In 2000 he was selected by the IT Society as a recipient of a Third-Millennium Medal.