AN AREA-EFFICIENT FPGA-BASED ARCHITECTURE FOR FULLY-PARALLEL STOCHASTIC LDPC DECODING

Saeed Sharifi Tehrani, Shie Mannor and Warren J. Gross

Department of Electrical and Computer Engineering
McGill University
Montreal, Quebec, H3A 2A7 Canada
E-mail: {sshari9,shie,wjgross}@ece.mcgill.ca

ABSTRACT

Stochastic decoding is a new alternative method for low complexity decoding of error-correcting codes. This paper presents the first hardware architecture for stochastic decoding of practical Low-Density Parity-Check (LDPC) codes on factor graphs. The proposed architecture makes fully-parallel decoding of (long) state-of-the-art LDPC codes viable on FPGAs. Implementation results for a $(1024, 512)$ fully-parallel LDPC decoder shows an area requirement of about 36% of a Xilinx Virtex-4 XC4VLX200 device and a throughput of 706 Mbps at a bit-error-rate of about 10^{-6} with performance loss of about 0.1 dB, with respect to the nearly ideal floating-point sum-product algorithm with 32 iterations.

Index Terms— Iterative decoding, Low-Density Parity-Check (LDPC) codes, stochastic decoding

1. INTRODUCTION

Low-Density Parity-Check (LDPC) codes [1] are a class of linear block codes that includes some of the most powerful error-correcting codes with performance close to the Shanon capacity limit [2, 3]. These codes have been considered for several standards such as Digital Video Broadcast (DVB) satellite communications [4], IEEE 802.3an (10GBASE-T) [5] and IEEE 802.16 (WiMAX) [6]. LDPC codes are iteratively decoded by means of belief propagation using the Sum-Product Algorithm (SPA). The SPA involves passing messages over the edges of a bipartite *factor graph* [7]. Nodes in a factor graph are separated into two distinct groups, namely, variable nodes and parity-check nodes.

The inherent parallelism involved in LDPC decoding has motivated researchers to exploit the maximum feasible amount of parallelism in LDPC decoders for the sake of higher throughput. However, due to the routing congestion and interconnection problem (mainly associated with the interleaver in an LDPC decoder), the implementation of fully-parallel LDPC decoders is still challenging for long practical codes. The required routing for the interleaver of an LDPC code can occupy a large portion of the decoder area. In this respect, methods that alleviate the routing problem and/or reduce the complexity of nodes are favorable. The well-known min-sum algorithm is an approximate variation of SPA which offers less-complex nodes but usually with the cost of about 0.5 to 1 dB decoding loss [8, 9]. Recently, bit-serial methods have been considered for the min-sum decoding. Examples are the FPGA and ASIC implementations of LDPC decoders in [10] and [11]. The FPGA-based implementation in [10], which implements a regular $(480, 355)$ LDPC decoder is, to the best of our knowledge, the longest and fastest FPGA-based fully-parallel LDPC decoder in the literature. This decoder occupies about 84% of an Altera Stratix EP1S80 device and achieves a throughput of 650 Mbps with a maximum clock frequency of 61 MHz and 15 approximate min-sum decoding iterations [10].

Stochastic decoding is a new approach for decoding error-correcting codes. This approach results in low complexity computational nodes in the factor graph and also alleviates the routing congestion problem. Stochastic decoding is inspired by stochastic computation introduced in the 1960s to design low-precision circuits [12]. In stochastic computation, probabilities are represented by streams of stochastic bits. This representation results in low complexity computational nodes to perform operations such as multiplication and division on probabilities in a bit-serial fashion. Stochastic computation has been considered for applications such as neural networks [13] and motor controllers [14]. The application of stochastic computation for decoding error-correcting codes was first considered for decoding (16,8) LDPC and (7,4) Hamming codes [15, 16]. It was also considered for the trellis decoding of an acyclic (16,11) Hamming code and a (256,121) Turbo block (product) code based on the (16,11) acyclic Hamming decoders [17, 18]. The only hardware implementation of stochastic decoders decodes a specially constructed (16,8) tailbiting LDPC code [19].

The above-mentioned early stochastic methods could only decode special short/acyclic codes and were not applicable for decoding state-of-the-art LDPC codes on factor graphs. Stochastic decoders are sensitive to the level of switching ac-

tivity within the factor graph and they are also prone to the *latching* (lock-up) problem of stochastic nodes [18,20]. These problems can severely degrade the Bit-Error-Rate (BER) decoding performance of the decoder, particulary for practical (long) LDPC codes [18, 20]. Recently, a new method was proposed in [20] which is considered as the first successful stochastic method for decoding practical LDPC codes on factor graphs. This method is capable of providing near-optimal performance with respect to SPA with floating-point implementation.

This paper discusses the implementation issues of stochastic LDPC decoders and presents the first FPGA-based implementation of a stochastic decoder which considers a practical code. Implementation results for a fully-parallel (1024,512) regular LDPC decoder with degree-3 ($d_v = 3$) variable nodes and degree-6 ($d_c = 6$) parity-check nodes on a Xilinx Virtex-4 XC4VLX200 device are also provided in the paper. The rest of the paper is organized as follows. Section 2 provides an overview of stochastic computation and the stochastic decoding method proposed in [20]. Section 3 describes the hardware implementation issues and architecture of stochastic decoders. Synthesis and decoding performance results for the (1024,512) LDPC decoder are given in Section 4. Finally, Section 5 offers the concluding remarks and summarizes the implications of the results.

2. STOCHASTIC COMPUTATION AND LDPC DECODING

In stochastic computation, probabilities are transformed to streams of stochastic bits using Bernoulli sequences. Each bit in a stream is equal to 1 with the probability to be transformed. For instance, a frame of 10 bits with 7 bits equal to 1 represents a probability of 0.7. It should be emphasized that stochastic sequences are not necessarily frames of bits and they can be used as stochastic streams in which no framing/packetizing is required [21]. The transformation from a probability to a stochastic stream is not unique, hence, different streams are possible for the same probability. For example, Fig. 1 shows some possible streams for a probability of $7/13 \simeq 0.5385$. By using stochastic representation, complex probability operations such as multiplication and division are performed using simple circuits [12, 13]. For instance, the AND gate shown in Fig. 2 can be used for multiplication of two stochastic streams. Stochastic representation and computation can be also applied to probability operations in factor graphs. The simplicity of stochastic structures is appealing for decoding error-correcting codes such as LDPC codes.

In stochastic decoding, probabilities received from the channel are converted to stochastic streams and decoding proceeds by stochastic variable nodes and parity-check nodes exchanging bits. Let $P_a = \Pr(a_i = 1)$ and $P_a = \Pr(b_i = 1)$ be the input probabilities represented by the stochastic streams $\{a_i\}$

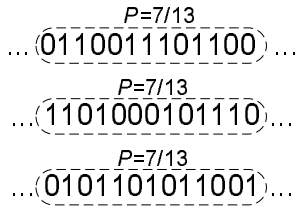

Fig. 1. Some possible stochastic streams for a probability of 7/13.

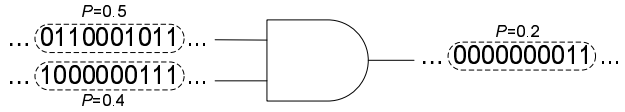

Fig. 2. An example of multiplication of two stochastic streams.

and $\{b_i\}$. The variable node operation is as follows

$$P_c = \frac{P_a P_b}{P_a P_b + (1 - P_a)(1 - P_b)}. \quad (1)$$

Similarly, the parity-check operation in a $d_c = 3$ parity-check node for inputs P_a and P_b is

$$P_c = P_a(1 - P_b) + P_b(1 - P_a). \quad (2)$$

The stochastic structures for a $d_v = 3$ variable node and a $d_c = 3$ parity-check node are shown in Fig. 3 [15]. A stochastic variable node structure uses its previous output bit (*i.e.*, $c_i = c_{i-1}$) if the input bits a_i and b_i are not equal. This is referred to as a *hold state* for a variable node.

Using the stochastic structures, decoding operation proceeds by variable nodes and parity-check nodes exchanging bits along each edge in the factor graph. Each decoding round is referred to as a Decoding Cycle (DC). A DC does not correspond to one iteration in SPA [20]. At the end of a DC, the output of a variable node is passed to an up/down counter. This counter is incremented by receiving a 1 bit and decremented by receiving a 0 bit. After a fixed number of DCs (or when all the parity-checks are satisfied), a hard-decision is applied to the contents of the counters to determine the decoded codeword. This can be done by only using the sign-bit of the up/down counters, where a 1 sign-bit determines a positive content (or $+1$ symbol) and a 0 sign-bit determines a negative content (or a -1 symbol) in a Binary Phase-Shift Keying (BPSK) transmission [16, 19, 20]. In addition to simple structures for variable and parity-check nodes, stochastic decoding significantly reduces the routing congestion problem. This is because only one bit in each direction is required to represent an edge in the factor graph.

The stochastic structures in Fig. 3 cannot be directly applied for decoding practical LDPC codes. As mentioned previously, stochastic decoders are sensitive to the level of switch-

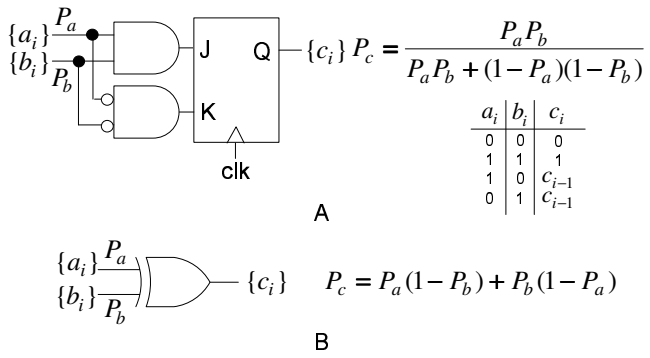

Fig. 3. The structure of (A) a degree-3 variable node and (B) a degree-3 parity-check node.

ing activity (bit transition) within the factor graph. This problem can be worse at high Signal-to-Noise Ratios (SNRs) in which probability messages are close to 0 (or 1) and hence, the corresponding stochastic bits are mostly 0 (or 1) [20]. In addition, stochastic decoding is also prone to the latching (lock-up) problem. The latching problem refers to the case where existence of cycles in the factor graph correlate the stochastic streams (messages) in such a way that stochastic nodes are stuck to a fixed state for several DCs [17, 20]. The lack of enough switching activity and the latching problem severely affects the BER performance of practical LDPC codes. To circumvent these problems two methods are suggested in [20]: noise-dependant-scaling and edge memories, which are discussed as follows.

2.1. Noise Dependant Scaling

In Noise-Dependant-Scaling (NDS), the channel Log Likelihood Ratios (LLRs) are scaled by a factor which is proportional to the level of noise that exists in the channel. Assuming BPSK transmission over an Additive White Gaussian Noise (AWGN) channel with a power-spectral density of N_0, the scaled LLR for the i-th received symbol in the block, y_i, is [20]

$$L_i' = (\frac{\alpha N_0}{Y})L_i = (\frac{4\alpha}{Y})y_i, \qquad (3)$$

where $L_i = 4y_i/N_0$ is the received channel LLR, α is a factor whose value can be chosen based on the stochastic decoding performance of the code and Y is a fixed maximum value for received bits in the block. For example, in a BPSK transmission, Y can be chosen equal to 6 [20]. The main purpose of NDS is to increase the switching activity and to provide a similar switching activity over different range of SNRs [20]. As it was shown in [20], the NDS is usually enough for stochastic decoding of short error-correcting codes such as short Hamming or LDPC codes. However, for practical long LDPC codes, edge memories are also essential [20].

2.2. "Regenerative" Bits and Edge Memories

Edge Memories (EMs) are memories assigned to outgoing edges in the graph. EMs employ a mechanism which rerandomizes and de-correlates the stochastic messages and significantly reduces the chance that stochastic variable nodes get stuck in fixed states [20, 21]. The principles of this mechanism are that (i) an EM is only updated with outgoing bits which are not produced in the hold state[1] (we refer to these bits as "regenerative" bits), (ii) when the hold state occurs for an edge, the variable node refers to the corresponding EM to produce/generate the outgoing bit and, (iii) the generation of a bit out of EM in the hold state should be done in a random/stochastic manner to rerandomize stochastic streams and break the correlation. This mechanism can be realized in different ways. For example, it is possible to count regenerative bits using up/down counters (*i.e.*, transform regenerative bits to probabilities) and then generate a new (rerandomized) stochastic bit based on the measured probabilities. Another way to realize EMs is to use shift registers with single-selectable bits. This realization does not require transformation of regenerative bits to probabilities and operates on streams of bits; it is also suitable for an FPGA implementation (see the next section). Using this realization, an EM is only updated with regenerative bits and in the case of the hold state, a bit is randomly picked from the shift-register [20].

3. DECODER ARCHITECTURE

This section describes the necessary components in an LDPC stochastic decoder and presents the implementation of these components for a (1024,512) stochastic LDPC decoder.

3.1. Generating Stochastic Streams

Upon receiving a block of symbols from the AWGN channel, the NDS is applied according to (3). NDS is based on the ratio of α and Y parameters. A good stochastic BER decoding performance can be obtained for the (1024,512) code by using an $\alpha = 3$ and $Y = 6$, (*i.e.*, $4\alpha/Y = 2$) [20]. Therefore, probabilities can be easily generated based on 1-bit shifted received values.

For the implementation of the (1024,512) decoder, we used 8-bit representation for the received channel probabilities. These probabilities are converted to stochastic streams by using the structure shown in Fig. 4. This structure consists of a comparator which compares the channel probability, P, with a (pseudo) random number, R, at each DC. P is fixed during the decoding of a block. However, R is a random number (with a uniform distribution) that is updated in every DC. The output bit of the comparator is equal to 1 if $P > R$ and it is equal to 0, otherwise. The output bit of the comparator

[1] The variable node in Fig. 3(A) is not in the hold state when $a_i = b_i = 0$ or $a_i = b_i = 1$.

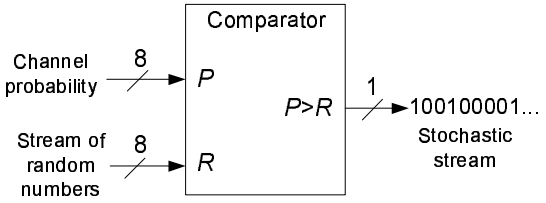

Fig. 4. Converting channel probabilities to stochastic streams.

is fed to a variable node. Since R has a uniform distribution and can take a value from 0 to $2^8 - 1$, each bit in the output stochastic stream is equal to 1 with a probability of $P/2^8$. The generation of R is performed by a randomization engine which is described in the subsection 3.5. The decoder needs one comparator for each variable node.

3.2. Variable Nodes and Edge Memories

In this implementation, each variable node uses one EM per edge. EMs operate as M-bit shift registers, however, a (single) bit in an EM is selectable by address lines in each DC. Each EM is initialized to contain zeros. Fig. 5 shows the architecture of a $d_v = 3$ variable node (only 2 inputs and the corresponding output are shown). When the input bits of a variable node are equal, the signal U in Fig. 5 is 1 and the variable node applies the equality equation and computes the outgoing bit for the edge [20]. In this case, the variable node updates the EM in a first-in-first-out manner. In the case of a *hold state* ($i.e.$, when the input bits are not equal), U is 0 and the corresponding EM is not updated. Instead, we randomly select a bit from the EM of the edge and use it as the outgoing bit [20]. In hardware, the random selection of a bit from an EM is done by generating a (pseudo) random address for each EM in each DC. This task is also done by the randomization engine described in subsection 3.5.

For the hardware implementation of the (1024,512) decoder, we used EMs with $M = 64$. The decoder uses one EM per edge. Many FPGA architectures allow efficient implementation of EMs. They allow to use small Look-Up Tables (LUTs) as Shift Register LUTs (SRLs) and accessing a single bit in the register. It is also possible to cascade any number of SRLs to form shift registers of arbitrary size. These features exactly match the operation of EMs. As an example, in our target FPGA device, Xilinx Virtex-4, a 64-bit EM can be efficiently implemented by cascading four 4-input LUTs [22]. Each 4-input LUT forms a 16-bit shift register with a single output accessed by the LUT's address line. A 64-bit EM created this way occupies only 2 slices of a Xilinx Virtex-4 FPGA.

3.3. Saturating Up/Down Counters

As described in [20], the output of a stochastic variable node is passed to an up/down counter at the end of each DC. Therefore, one counter is needed for each variable node. In this implementation, we used 6-bit signed saturating counters which count from -31 to 31. A saturating up/down counter is incremented in case of receiving 1 and decremented in case of 0. It stops decrementing/incrementing if reaches its minimum/maximum limits. As mentioned previously, the sign of the counter indicates the the hard-decision at each DC.

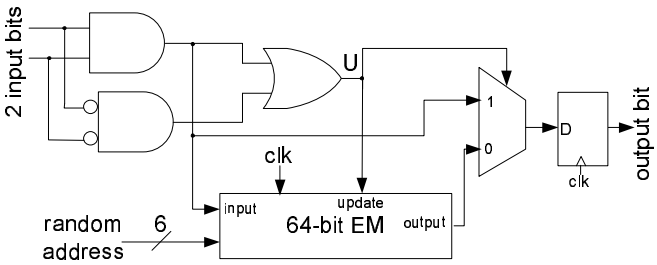

Fig. 5. The structure of a $d_v = 3$ variable node (only 2 inputs and the corresponding output are shown).

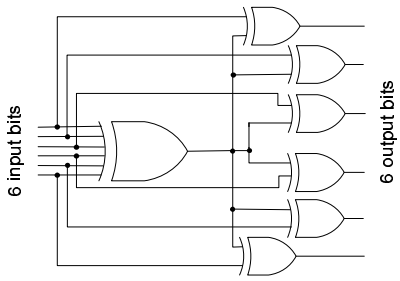

Fig. 6. The structure of a $d_c = 6$ parity-check node.

3.4. Parity-Check Nodes

The implementation of a parity-check node is straightforward. Fig. 6 shows the structure of a $d_c = 6$ parity-check used in the decoder.

3.5. The Randomization Engine

The Randomization Engine (RE) is responsible for generating random probabilities for comparators as well as generating random addresses for EMs. Although the required amount of random numbers might seem high, these random numbers can be significantly shared at two levels without a significant loss in BER performance: (i) bits in random (probability) numbers generated for comparators can also be used for addresses of EMs and, (ii) different groups of EMs can share the same random address.

The RE for the (1024,512) decoder consists of ten 16-bit Linear Feedback Shift Registers (LFSRs) in which each LFSR is associated with a prime polynomial. Each bit in an output random number is generated by XORing different bits

of these ten LFSRs. The RE generates 32 8-bit random numbers in each DC for the entire decoder.

4. IMPLEMENTATION AND PERFORMANCE RESULTS

Table 1 summarizes the area requirement for each individual module used in the decoder as well as the whole decoder (which includes the interleaver). Efficient implementation of EMs and sharing the random numbers within the decoder allow area-efficient implementation of fully-parallel stochastic LDPC decoders. Note that the area requirement for each module reported in this table is for the case where the module is individually implemented in the device. Since the interleaver only consists of routing interconnections, its area consumption is not individually reported in the table. However, the reported area for the whole decoder also includes the area consumed by the interleaver. An approximate area consumption of the interleaver (and other required routing) can be obtained by subtracting the area of all the reported components from the area of the whole decoder.

The (1024,512) regular LDPC decoder occupies 32875 out of 89088 (36%) slices available in a Xilinx XC4VLX200 Virtex-4 device. Although due to different code rates and target FPGA platforms, a precise comparison with results given in [10] is not possible, however, an approximate comparison shows that the proposed stochastic architecture offers about 53% area reduction. The approximate min-sum-based regular (480,355) decoder in [10] occupies 66588 Logic Elements (LEs) of an Altera Stratix EP1S80 device. Each LE has one 4-input LUT and one Flip-Flop (FF) [23] which results in the worst-case (FPGA area) consumption of $66588 \ / \ 480 \simeq 138.7$ LUTs and FFs per bit. The stochastic (1024,512) decoder occupies 32875 slices. Each slice in a Xilinx Virtex-4 architecture contains two 4-input LUTs and two FFs [22]. This results in the worst-case (FPGA area) consumption of $65750 \ / \ 1024 \simeq 64.2$ LUTs and FFs per bit. The (1024,512) stochastic decoder implementation achieves a clock frequency of 212 MHz after place and route step.

Table 1. (1024,512) LDPC Decoder Implementation Results on a Xilinx Virtex-4 XC4VLX200 Device

Module	Slices	Number required
Comparator	6	1024
U/D counter	6	1024
$d_v = 3$ var. node	10 (with 3 EMs)	1024
$d_c = 6$ check node	5	512
RE	394	1
Decoder	32875 (36%)	-

Fig. 7 shows the bit-true simulation of the BER performance of the stochastic decoder for decoding with a maximum DC of 6K. For the sake of comparison, bit-true simulation for decoding with maximum 1K DCs and, simulation results for SPA with double-precision implementation are also depicted. As shown, the stochastic decoder provides decoding performance close to the floating-point SPA with 32 iterations. The decoding loss is about 0.1 dB at a BER of about 10^{-6} (for decoding with a maximum DC of 6K). The stochastic decoder stops decoding as soon as all the parity-checks are satisfied or a maximum number of 6K DCs are reached. The observed average DC, however, was much lower than the maximum DC, especially for low BERs. In fact, at low BERs, there are only a few blocks that need a large number of DCs to decode. For example, the observed average DC at SNR=3 dB (BER $\simeq 10^{-6}$) was about 300 DCs which results in a throughput of about 706 Mbps at this BER.

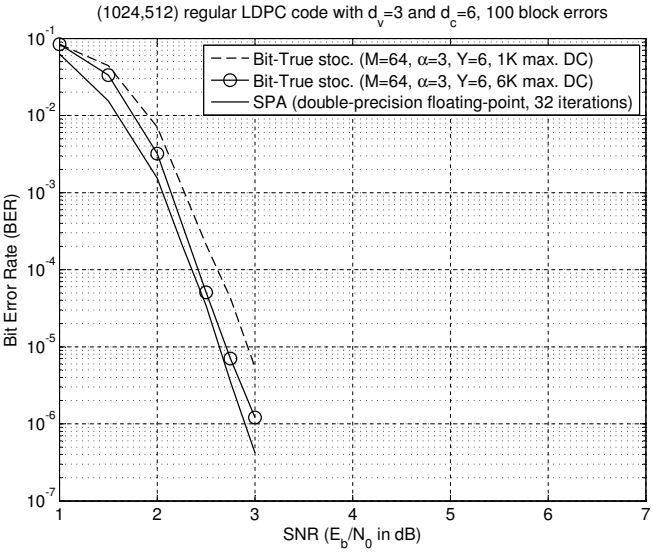

Fig. 7. Decoding performance results.

It is essential to note that the above-mentioned speed (average and maximum DC) and area requirements are for providing performance close to floating-point SPA. A stochastic decoder is able to trade-off area, speed and latency with BER performance. For example, for performance similar to the min-sum algorithm (*i.e.*, usually about 0.5 to 1 dB loss, compared to SPA [8, 9]), shorter EMs and/or fewer DCs can be used. Also, for applications in which the maximum latency requirement is strict, it is possible to reduce the maximum DC with a cost of some performance loss. This is also shown in Fig. 7 where bit-true simulations for stochastic decoding with a maximum DC of 1K is depicted.

5. CONCLUSIONS

An FPGA-based architecture for a fully-parallel (1024,512) stochastic LDPC decoder is presented. To the best of our

knowledge, this is the first hardware implementation of a stochastic decoder that considers an LDPC code with a practical length. Also, the proposed decoder is the longest FPGA-based fully-parallel LDPC decoder implemented.

The paper discussed the required components and the architecture of a stochastic LDPC decoder. It also presented ways to reduce the required amount of randomness in a stochastic decoder. It is shown that architectural features available in many FPGAs can be used for area-efficient implementation of stochastic LDPC decoders. The proposed architecture offers about 53% FPGA area improvement compared to the previous fully-parallel approximate min-sum LDPC decoder architecture.

The synthesis and performance results provided in the paper validate the potential of stochastic decoding methods for low complexity and high-throughput decoding of state-of-the-art LDPC codes with performance close to floating-point SPA.

6. REFERENCES

[1] R. G. Gallager, *Low Density Parity Check Codes*, Cambridge, MA: MIT Press, 1963.

[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, 1996.

[3] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[4] DVB-S2, http://www.dvb.org.

[5] The IEEE P802.3an 10GBASE-T Task Force, www.ieee802.org/3/an.

[6] The IEEE 802.11n Working Group, grouper.ieee.org/groups/802/11, Oct. 2005.

[7] F. Kschischang, B. Frey, and H. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, Feb 2001.

[8] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proceedings of the IEEE Global Telecommunications Conference*, Nov. 2001, vol. 2, pp. 1021–1025.

[9] F. Guilloud, E. Boutillon, and J.-L. Danger, "λ-min decoding algorithm of regular and irregular LDPC codes," in *Proceedings of 3rd International Symposium on Turbo Codes (ISTC 03)*, Brest, France, 1-5 Sept. 2003, pp. 451–454.

[10] Ahmad Darabiha, Anthony Chan Carusone, and Frank R. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *IEEE Int. Symp. on Circuits and Systems*, Greece, May 2006.

[11] T. L. Brandon, R. Hang, G. Block, V. Gaudet et al., "A 250-Mbps min-sum LDPC decoder using bit-serial message exchange," *submitted to Integration, the VLSI Journal*, Dec. 2006.

[12] B. Gaines, *Advances in Information Systems Science*, chapter 2, pp. 37–172, Plenum, New York, 1969.

[13] B. Brown and H. Card, "Stochastic neural computation I: Computational elements," Sept. 2001, vol. 50, pp. 891–905.

[14] A. A. Dinu, M. N. Cirstea, and M. McCormick, "Stochastic implementation of motor controllers," in *Proc. of the IEEE Int. Symp. on Industrial Electronics*, July 2002, pp. 639–644.

[15] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electron. Lett.*, vol. 39, no. 3, pp. 299–301, Feb. 2003.

[16] A. Rapley, C. Winstead, V. Gaudet, and C. Schlegel, "Stochastic iterative decoding on factor graphs," in *Proc. of the 3rd Int. Symp. on Turbo Codes and Related Topics*, Brest, France, Sept. 2003, pp. 507–510.

[17] C. Winstead, V. Gaudet, A. Rapley, and C. Schlegel, "Stochastic iterative decoders," in *Proc. of the IEEE Int. Symp. on Information Theory*, Sept. 2005, pp. 1116–1120.

[18] Chris Winstead, "Error-control decoders and probabilistic computation," in *Tohoku Univ. 3rd SOIM-COE Conf.*, Sendai, Japan, Oct. 2005.

[19] W. J. Gross, V. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *the 39th Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 2005.

[20] S. Sharifi Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, Oct. 2006.

[21] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Survey of stochastic computation on factor graphs," to appear in the *Proc. of the 37th International Symposium on Multiple-Valued Logic*, May 2007.

[22] Xilinx Corporation, *Virtex-4 User Guide*, www.xilinx.com, April 2005.

[23] Altera Corporation, *Stratix Device Handbook*, www.altera.com, July 2005.