Fully Parallel Stochastic LDPC Decoders

Saeed Sharifi Tehrani, *Student Member, IEEE*, Shie Mannor, *Member, IEEE*, and Warren J. Gross, *Member, IEEE*

***Abstract*—Stochastic decoding is a new approach to iterative decoding on graphs. This paper presents a hardware architecture for fully parallel stochastic low-density parity-check (LDPC) decoders. To obtain the characteristics of the proposed architecture, we apply this architecture to decode an irregular state-of-the-art (1056,528) LDPC code on a Xilinx Virtex-4 LX200 field-programmable gate-array (FPGA) device. The implemented decoder achieves a clock frequency of 222 MHz and a throughput of about 1.66 Gb/s at $E_b/N_0 = 4.25$ dB (a bit error rate of 10^{-8}). It provides decoding performance within 0.5 and 0.25 dB of the floating-point sum-product algorithm with 32 and 16 iterations, respectively, and similar error-floor behavior. The decoder uses less than 40% of the lookup tables, flip-flops, and IO ports available on the FPGA device. The results provided in this paper validate the potential of stochastic LDPC decoding as a practical and competitive fully parallel decoding approach.**

***Index Terms*—Field programmable gate arrays (FPGAs), iterative decoding, low-density parity-check (LDPC) codes , stochastic decoding.**

I. Introduction

LOW-DENSITY parity-check (LDPC) block codes [1] are powerful linear error-correcting codes with decoding performance close to the Shannon capacity limit [2]. These codes have been considered for several recent digital communication standards such as the DVB-S2 [3], the IEEE 802.3an (10GBASE-T) [4], the IEEE 802.16e (WiMAX) [5], and the IEEE 802.11n (WiFi) [6] standards. LDPC codes are usually iteratively decoded by means of belief propagation [7] using *message passing* algorithms such as the Sum–Product Algorithm (SPA) or its less-complex approximation, the Min-Sum Algorithm (MSA) [8], with the expense of some decoding loss. LDPC codes and their iterative decoding process can be graphically represented using bipartite factor graphs [9]. Factor graphs consist of two distinctive groups of nodes, variable nodes (VNs) and parity-check nodes (PNs). Each edge in a factor graph connects a VN to a PN. LDPC decoding involves message passing between VNs and PNs over the edges of the factor graph. This message passing scheme can be done concurrently, which inherently introduces a high level of parallelism in LDPC decoding. This attractive feature together with the excellent decoding performance of LDPC codes have made the efficient high-speed implementation of LDPC decoders a focal point of research in recent years.

In general, fully parallel and partially parallel architectures are two main strategies for the implementation of LDPC decoders. In the fully parallel strategy, the entire factor graph is implemented in hardware and all VNs and PNs in the graph are updated concurrently. Fully parallel decoders are usually implemented to achieve high-throughput decoding of a certain LDPC code at the cost of high area consumption. This approach is particularly considered for applications with high-speed requirements such as the IEEE 802.3an (10GBASE-T) standard [4]. The partially parallel approach instantiates a portion of the factor graph. Partially parallel decoders employ memory and hardware resource sharing to manage message passing between different portions of the factor graph. The main benefits of this approach are to minimize the area and/or to offer the flexibility to support different block lengths and code rates in applications such as IEEE 802.16e (WiMAX) [5] and IEEE 802.11n (WiFi) [6]. However, the partially parallel approach has a much lower throughput compared to the fully parallel approach. The partially parallel approach is also used for the implementation of LDPC decoders with very long block lengths where the fully parallel approach is not feasible today, such as the LDPC code for the DVB-S2 standard with a block length of 64 800 bits [10].

A major challenge in the implementation of LDPC decoders is the complexity of the interconnections between VNs and PNs. The complexity of the interleaver is due to the random-like locations of ones in the code's parity-check matrix. This problem is acute for practical fully parallel decoders (where the code block length is usually large) and results in routing congestion and interconnection problems [11]–[13]. The routing congestion causes high area consumption and low logic utilization in the decoder. For instance, with 4-bit precision of probability messages, the 52.5 mm^2 die size of the (1024,512) decoder in [11] has a logic utilization of 50% in its core; the rest of the core area is occupied by wires. In addition to high area consumption, the presence of long physical wires in the interleaver increases the power consumption and limits the maximum achievable clock frequency and thus the throughput of a fully parallel LDPC decoder (see [11]–[14]). To alleviate these problems, different approaches are investigated in the literature at both code design and hardware implementation levels. One approach is to design "implementation-aware" codes. In this approach, instead of randomly choosing the locations of ones in the parity-check matrix (at the code design stage), the parity-check matrix of an LDPC code is designed with constraints allowing a suitable structure for decoder implementation and providing acceptable decoding performance [15]–[20]. Another approach used to alleviate the routing congestion problem is to use bit-serial or digit-serial architectures to implement LDPC decoders. Examples of this approach are the recent FPGA

Manuscript received April 7, 2007; revised June 6, 2008. First published August 19, 2008; current version published Ocrober 15, 2008. The associate editor coordinating the review of this paper and approving it for publication was Prof. Jarmo Takala. The authors acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada Research Chairs (CRC), and the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT) for their financial support.

The authors are with the Department of Electrical and Computer Engineering, McGill University, Montreal, Quebec, H3A 2A7, Canada (e-mail: sshari9@ece.mcgill.ca; shie@ece.mcgill.ca; wjgross@ece.mcgill.ca).

Digital Object Identifier 10.1109/TSP.2008.929671

implementation of a bit-serial (480,355) LDPC decoder in [21], the ASIC implementation of a (660,480) LDPC decoder in [12] based on bit-serial approximate MSA, and the MSA-based bit-serial (256,128) LDPC decoder in [22]. Also, a message broadcasting technique was recently suggested to alleviate the routing congestion by reducing node-to-node communication complexity in LDPC decoders [13]. Bit-flipping (BF) decoding [1] is another approach for low-complexity LDPC decoding with the cost of some performance loss. Bit-flipping methods do not exploit message passing, they use the knowledge of unsatisfied parity-checks to iteratively correct bit errors. Recently, there has been research interest in various bit-flipping methods such as weighted BF methods (see [23] and [24]) and a newly proposed differential binary BF-based method [25]. Among conventional BF methods, the weighted BF method in [24] performs well on many LDPC codes and has a performance loss of about 0.5 to 1 dB, compared to SPA [24]. LDPC decoders can be implemented with a programmable architecture or processor, which lend themselves to Software Defined Radio (SDR). SDR is a programmable hardware platform that consists of multiple processing and memory units. SDR supports software implementations of wireless communication protocols for physical layers. SDR offers flexibility to support codes with different block lengths and rates, however, the throughput of SDR-based LDPC decoders is usually low (e.g., see [26]). In addition to digital decoders, continuous-time analog implementations have been considered for LDPC codes [27] and other error-correcting codes [28]–[33]. Compared to their digital counterparts, analog decoders offer improvements in speed and/or power. However, because of the complex and technology-dependent design process, the analog approach has been only considered for short error-correcting codes. The only reported analog LDPC decoder decodes a (32,8) code [27].

Stochastic decoding is a new alternative approach for decoding LDPC codes. Stochastic decoding is inspired by stochastic computation [34] where probabilities are converted to streams of stochastic bits and complex probability operations such as division and multiplication are performed on stochastic bits using simple bit-serial structures. Early stochastic decoding methods could only decode acyclic/simple codes such as very short LDPC codes or Hamming codes [35]–[37]. Stochastic decoding was also used for decoding a (256,121) Turbo product code based on acyclic (16,11) Hamming component trellis decoders [38], [39]. The earliest hardware implementation of a stochastic LDPC decoder belongs to a specially-constructed tail-biting (16,8) LDPC decoder [37]. The first practical stochastic method for decoding LDPC codes was proposed in [40]. It was shown that with respect to the floating-point SPA, stochastic decoding is potentially able to achieve almost identical decoding performance for short LDPC codes and near-optimal performance for practical (long) LDPC codes. The potential of this method for low-complexity and fast decoding was recently validated by an FPGA implementation of a (1024,512) regular LDPC decoder with degree-3 VNs and degree-6 PNs [41]. This decoder occupies about 36% of a Xilinx Virtex-4 LX200 FPGA device. It also achieves a throughput of 706 Mb/s at a bit error rate (BER) of 10^{-6} [41].

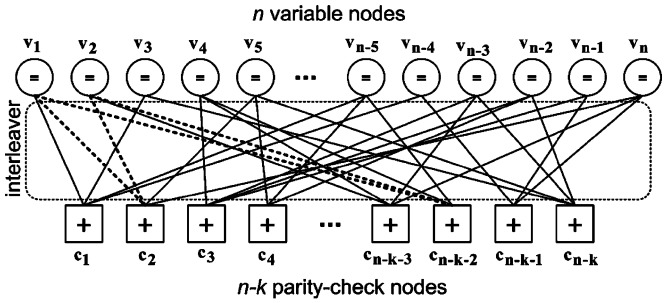

Fig. 1. Typical factor graph and the interleaver for an (n, k) LDPC code. A length-4 cycle is dashed. In a conventional implementation with W-bit representation of messages, each edge requires $2W$ wires (for two directions).

This paper presents an architecture for fully parallel stochastic LDPC decoding. Compared to our previous work in [41], this paper proposes several novel architectural techniques that improve both the hardware and decoding performance of stochastic decoders. The proposed architecture is applied to a fully parallel stochastic LDPC decoder that decodes a state-of-the-art irregular (1056,528) LDPC code on a FPGA. To show the good decoding performance and the error-floor behavior of the proposed architecture, the decoding performance of (1056,528) and (1056,704) stochastic decoders are compared to the floating-point SPA. It should be noted that the floating-point SPA is considered as a nearly-ideal case for LDPC decoding and has very high hardware complexity, hence, the BER decoding performance of hardware architectures are usually not compared to the floating-point SPA. This paper also discusses the hardware performance and various attractive features and tradeoffs offered by the proposed architecture. The rest of the paper is organized as follows. Section II provides a brief overview of LDPC codes, SPA, and stochastic decoding. Section III describes stochastic LDPC decoding. Section IV proposes the architecture for the fully parallel LDPC stochastic decoder along with various novel architectural features. Sections V and VI discuss the performance and tradeoffs, and provide comparison with the state-of-the-art work. Finally, Section VII gives the conclusions.

II. Background

A. Review of LDPC Codes and the Sum–Product Algorithm

A binary (n, k) LDPC code is defined as the null space of a sparse $(n-k) \times n$ parity-check matrix $\mathbf{H}$, $\mathbf{H}x = 0$, where x is the transmitted block containing k information and $n-k$ parity bits. This LDPC code can be represented by a factor graph with n VNs and $n-k$ PNs. The ith VN, v_i, is connected to jth PN c_j if and only if $h_{ji} = 1$ in $\mathbf{H}$ (see Fig. 1). The number of edges connected to a node (in the interleaver) is referred to as the degree of the node and represented as d_v for the VNs and d_c for the PNs. In regular codes, d_v and d_c are fixed for all VNs and PNs, respectively. In irregular LDPC codes, d_v and d_c vary for different nodes.

The SPA is an iterative algorithm for decoding LDPC codes. SPA uses soft information (probabilities) received from the channel and iteratively processes them. The SPA

makes decisions by comparing final probabilities to a threshold value (hard-decision) at the end of the decoding process. Let $x_i \in \{-1, +1\}$ and y_i respectively denote the ith sample in the transmitted and received block, in a binary phase-shift keying (BPSK) transmission over an additive white Gaussian noise (AWGN) channel. Also, let $P_{i\to j} \in [0, 1]$ be the probability message from the VN v_i with $d_v = 3$ to the PN c_j with $d_c = 3$ and $Q_{j\to i} \in [0, 1]$ be the probability message from c_j to v_i. Let $\{\mathrm{v}_i, \mathrm{v}_l, \mathrm{v}_m\}$ represent the set of VNs connected to c_j and, $\{\mathrm{c}_j, \mathrm{c}_r, \mathrm{c}_s\}$ represent the set of PNs connected to v_i.[1] The SPA steps in the probability-domain can be described as follows (see [9] and [42] for details).

1) Initialize the received probability for v_i as $P^i_{\text{init}} = \Pr(x_i = +1|y_i)$.
2) The v_i sends P^i_{init} to all connected PNs.
3) The PN c_j sends $Q_{j\to i}$ to the VN v_i:

$$Q_{j\to i} = P_{l\to j}(1 - P_{m\to j}) + P_{m\to j}(1 - P_{l\to j}). \quad (1)$$

4) The VN v_i sends $P_{i\to j}$ to the PN c_j:

$$P_{i\to j} = \frac{Q_{r\to i}Q_{s\to i}}{Q_{r\to i}Q_{s\to i} + (1 - Q_{r\to i})(1 - Q_{s\to i})}. \quad (2)$$

5) Stop decoding once a fixed number of iterations has been exceeded or the estimated vector, $\hat{x}$, satisfies $\mathbf{H}\hat{x} = 0$. Otherwise, return to Step 3.

Due to the high (hardware) complexity of the SPA operations in the probability-domain, the SPA is usually implemented in the log-domain where channel probabilities are considered as log-likelihood ratios (LLRs):

$$L_i = \log\left(\frac{\Pr(x_i = +1|y_i)}{\Pr(x_i = -1|y_i)}\right) \quad (3)$$

where x_i and y_i are the ith transmitted and received symbols, respectively, and L_i is the LLR of y_i. Using the log-domain conversion, VNs calculate the summation of LLR messages and PNs employ $\tanh(\cdot)$ processing to compute their outgoing messages [42]. In MSA, the $\tanh(\cdot)$ processing in PNs is approximated to reduce the complexity at the expense of about 0.5 to 1 dB performance loss, compared to the SPA [43], [44]. To compensate for some of the loss, different improved methods are suggested in the literature (e.g., see [45]).

B. Stochastic Computation and Decoding

In stochastic computation, probabilities received from the channel are converted to streams of bits called Bernoulli sequences [34]. In this transformation, each bit in a stochastic stream is equal to 1 with the probability to be represented. Therefore, the observation of 1's in a stream of bits, $\{a_i\}$, determines the probability, *i.e.*, $\Pr(a_i = 1) = P$. The transformation of a probability to a stochastic stream is not unique, therefore, different stochastic streams are possible for a given probability. This also implies that the order/sequence of 1's in a stochastic stream is not important. For example, Fig. 2 shows

[1]This is without loss of generality since a higher degree nodes can be converted to subgraphs containing only degree three nodes [9].

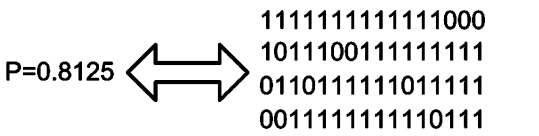

Fig. 2. Some possible streams for a probability of 0.8125.

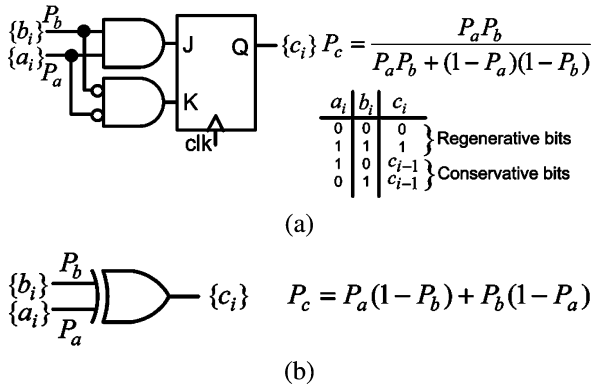

Fig. 3. Early structure for a stochastic (a) VN and (b) PN [35], [40].

some possible streams for a probability of $13/16 = 0.8125$. In each stream, 13 bits out of 16 bits are 1. Using stochastic transformation, operations such as multiplication and division on probabilities can be performed using simple structures. For example, multiplication of two probabilities represented by two stochastic streams can be performed by an AND gate and a division of two probabilities can be approximated by a JK flip-flop (see [34] and [46]).

Fig. 3(a) shows the early structure for a stochastic VN [35], [40] and Fig. 3(b) illustrates the structure for a stochastic PN [35] (see Appendix I for the proof of these operations). A stochastic VN uses its previous output bit (i.e., $c_i = c_{i-1}$) when its input bits are not equal. This is referred to as the *hold state* of the VN for the corresponding edge. The decoding process proceeds by VNs and PNs exchanging bits over the edges of the graph. Each decoding round is called a *decoding cycle* (DC). A DC includes the exchange of one bit between VNs and PNs and thus does not directly correspond to one iteration in the SPA [40]. The simplicity of the stochastic approach as well as its bit-serial nature is appealing for LDPC decoding. However, these stochastic structures are not sufficient for decoding practical codes. Stochastic decoders are prone to the latching (lock-up) problem when the code graph has cycles. The latching problem refers to the situation where the existence of cycles in the code graph correlates the stochastic streams and makes groups of stochastic nodes stick into fixed states for several DCs [38], [40]. This situation is particularly more likely to happen when the PNs connected to a VN in the cycle are in disagreement about the correctness of the received bit at the VN. In this case, the input of the stochastic VN are not equal, therefore, the VN remains in the hold state for several DCs and repeatedly outputs the same bit. Also, as shown in [40], the latching problem is more pronounced at high SNRs where the probabilities received from the channel are very close to 1 (or 0) and hence bits in the equivalent stochastic streams are mostly 1s (or 0s) and rarely change. In this situation, the lack of random bit transitions (from 0 to

1 and *vice versa*) within cycles prevents nodes from exiting the fixed states they have locked in. The latching problem prevents the affected nodes from converging to right decisions and hence disrupt the convergence of the decoder to the right codeword and severely degrades the BER decoding performance. The latching problem is particularly acute for long LDPC codes because the code graph has many cycles, particularly, short cycles such as length-6 cycles (namely, 6-cycles) or length-4 cycles (4-cycles). Due to these problems, early stochastic decoders could only decode simple/short or acyclic error-correcting codes.

III. Stochastic LDPC Decoding

In order to circumvent the switching activity problem and reduce the latching problem, the noise-dependent-scaling method and edge memories (EMs) are suggested in [40]. Scaling and EMs are essential for decoding practical LDPC codes. In this paper, we also propose internal memories (IMs) to improve the BER performance of stochastic LDPC decoders, especially when high-degree VNs are used.

A. Scaling Channel Reliabilities

Scaling methods have been previously suggested in the literature for the performance improvement of SPA (e.g., see [47] for details). In [40], A new and essential scaling method for stochastic decoders (called noise-dependent-scaling) was suggested. This method was used to provide a similar level of switching activity over different ranges of SNRs, which results in improved BER performance for stochastic decoders. In this scaling method the received channel reliabilities are scaled by a factor which is proportional to the noise level in the channel. The scaled LLRs are, however, independent of channel noise and thus the decoder does not need to estimate the noise in the channel. Assuming a BPSK transmission over an AWGN channel, the scaled LLR L'_i for the ith received symbol in the block y_i is [40]

$$L'_i = \left(\frac{\alpha N_0}{Y}\right) L_i \overset{L_i = 4y_i/N_0}{\longrightarrow} L'_i = \left(\frac{4\alpha}{Y}\right) y_i = \frac{4\alpha}{Y}(x_i + n_i) \tag{4}$$

where x_i is the ith transmitted symbol in the block, n_i is the zero-mean AWGN sample with a power-spectral-density of N_0 and, $L_i = 4y_i/N_0$ is the soft output of the channel (in the form of a LLR). Y is a fixed parameter which is used as a maximum magnitude for the received symbols. For example, for a BPSK modulation Y can be set to 6. Also, α is a fixed factor whose value is chosen based on the best BER performance of the stochastic decoder [40].

B. Edge Memories and Regenerative Bits

EMs are memories assigned to edges in the factor graph. EMs are used to break the correlation between stochastic streams using re-randomization to circumvent the latching problem. In this respect, stochastic bits generated by a VN are categorized into two groups: *regenerative* bits [41] and *conservative* bits. Conservative bits are output bits which are produced in the hold state and regenerative bits are output bits which are produced in nonhold states [see Fig. 3(a)]. The essentials of the operation of EMs are as follows.

1) EM are only updated with the regenerative bits. Therefore, when a VN is not in the hold state, the newly produced regenerative bit is used as the outgoing bit of the edge and the EM is updated with this new bit. When the VN is in the hold state for an edge, a bit is randomly chosen from the corresponding EM and is used as the outgoing bit. This mechanism breaks the correlation of stochastic streams by rerandomizing stochastic bits and also reducing the correlation caused by the hold state in a stochastic stream. The reason is that every time the hold state happens, a bit is randomly chosen from previous regenerative bits (which are not generated in the hold state).
2) In order to facilitate the convergence of the decoder, EMs need to have a time-decaying reliance (forgetting mechanism) on previous regenerative bits and only rely on most recent regenerative bits.

Different embodiments can be used to implement EMs. One implementation which is used in this paper is to use an M-bit shift-register with a single selectable bit. In this implementation, the shift register is updated with regenerative bits and in the case of hold state a bit is (pseudo) randomly chosen from the shift register using a (pseudo) randomly generated address. Clearly, the length of the shift-register, M, guarantees the time-decaying reliance mechanism needed for an EM. Another possible implementation for EMs is to transform regenerative bits to the probability domain using up/down counters and then regenerate the new stochastic bits based on the measured probability by the counter. This implementation also needs to exploit time-decaying mechanisms such as saturating limits and feedback to rely on recent (regenerative) bits.

C. Internal Memories

Regenerative bits are important for the proper operation of a stochastic decoder. The lack of enough regenerative bits propagating in the decoder results in less switching activity and a higher possibility of latching. For this reason, the structure used for constructing VNs is crucial for BER performance of stochastic decoders. It is known that in general a VN can be constructed based on subgraphs of lower degree nodes [9], [35]. In this paper, the construction of stochastic VNs is also based on subgraphs of low-degree subnodes (with $d_v \leq 3$). We also propose to use internal memories (IMs) for each subnode in high-degree VNs to significantly decrease the chance of the hold state in a high-degree VN. This structure is shown in Fig. 4 where each IM is assigned to one subnode. The operation of IMs is similar to EMs but the IM length L is much shorter than M (it is only a few bits). An IM is updated with regenerative bits produced by the subnode and in the case of the hold state for a sub-node a bit is randomly chosen as the outgoing bit of the subnode.

D. Summary of Stochastic LDPC Decoding

Upon receiving a block from an AWGN channel, channel reliabilities are scaled as in (4) and then transformed to stochastic streams. Each VN receives one bit per DC and propagates its outgoing 1-bit messages to the connected PNs. PNs check the

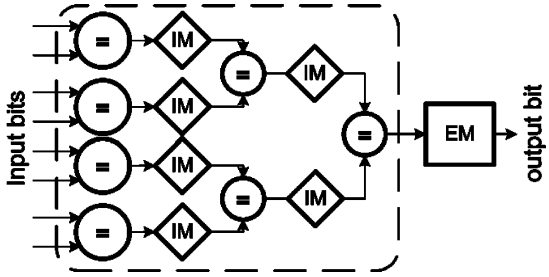

Fig. 4. Construction of a VN based on IMs used for low-degree sub-VNs. An EM is only used for the exit edge.

TABLE I
IRREGULAR LDPC CODES CHOSEN FROM THE IEEE 802.16e STANDARD

(n,k)	d_v distribution	d_c distribution
(1056,528)	(2,3,6)=$\{11/24, 1/3, 5/24\}$	(6,7)=$\{2/3, 1/3\}$
(1056,704)	(2,3,4)=$\{7/24, 1/24, 2/3\}$	(10,11)=$\{7/8, 1/8\}$

parities and send their 1-bit messages to VNs. The output of each VN at the end of a DC is passed to an up/down counter in which its sign-bit determines the hard-decision. This exchange of bits between VNs and PNs will be stopped as soon as all the parity-checks are satisfied or a maximum number of DCs is exceeded.

IV. DECODER ARCHITECTURE

Table I summarizes the characteristics of two LDPC codes considered in this paper. Both codes are irregular and belong to the IEEE 802.16e (WiMAX) standard [5]. The code used for implementation is the (1056,512) code. The (1056,704) code is only used to study performance behavior. The reader should note that this paper does not propose an LDPC decoder for the WiMAX standard and the main reason to choose these codes was to show the applicability of the stochastic approach to decode state-of-the-art irregular LDPC codes with high-degree nodes designed for recent applications (using the fully parallel design approach).

A. Scaling

We used lookup tables to apply scaling to the symbols received from the AWGN channel. The input of each lookup table is a 6-bit received symbol and the output is the corresponding probability, represented in 7 bits. Probabilities in each lookup table are calculated as

$$P_i = \frac{e^{L'_i}}{e^{L'_i}+1} \tag{5}$$

where L'_i is the scaled LLR according to (4). Note that due to the symmetry in (5), a lookup table can store only half of the probabilities. For example, it is possible to only store probabilities for positive y_i's (i.e., probabilities ≥ 0.5). When a y_i is negative, an additional NOT operation can be performed on the stochastic stream (during probability to stochastic stream conversion). Using this scheme the size of each lookup table is $2^{6-1}\times 7$ bits or 28 bytes. The implemented stochastic decoder employs 44 lookup tables to apply scaling, and each lookup table serially generates probabilities for $1056/44 = 24$ VNs. This uses $T_{\mathrm{IO}} = 24$ DCs,[2] where T_{IO} is the number of DCs the decoder spends to input channel reliabilities, apply scaling and output the decoded bits.

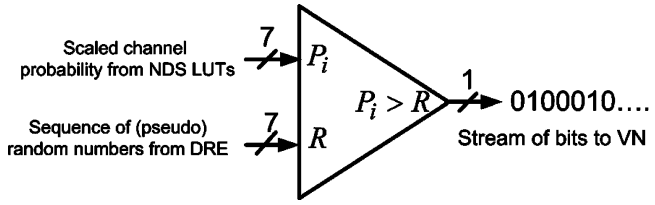

Fig. 5. Conversion of channel probabilities to stochastic streams.

B. Probability to Stochastic Stream Conversion

The conversion of each P_i to the corresponding stochastic stream is done by employing a 7-bit comparator as shown in Fig. 5. In this structure, P_i is fixed during the decoding operation and is compared to a (pseudo) random number, R, which changes at every DC. The output bit of the comparator is 1 when $P_i > R$ and 0 otherwise. Therefore, the density of 1's in the output stochastic stream is $P_i/2^7$. The random number in the figure is generated by a distributed randomization engine (DRE) which is described in Section IV-G. The output of each comparator is fed to one VN in each DC. The decoder, hence, needs one comparator per VN.

An attractive advantage of using lookup tables to apply scaling in stochastic decoders is that the precision of the lookup tables' output probabilities can be increased without a significant change in the decoder complexity. This is so because the precision of probabilities does not affect the interleaver, VNs or PNs. It only affects the size of lookup tables used for scaling, the comparators and the DRE. This, however, is not the case for SPA or MSA-based decoders where changing the precision means significantly increasing the number of wires in the interleaver. For the case of bit-serial SPA or MSA-based decoders, increasing the precision increases the latency of each iteration and hence reduces the throughput, because more clock cycles are needed to bit-serially send messages between nodes.

C. Stochastic Variable Nodes

Fig. 6 depicts the architecture of VNs in the (1056,528) stochastic decoder (only one output and its corresponding inputs are shown). EM lengths of $M = 32$, $M = 48$, and $M = 64$ bits are used for $d_v = 2$, $d_v = 3$, and $d_v = 6$ VNs, respectively. EMs are implemented as shift-registers with a single selectable bit using shift register lookup tables available in Xilinx Virtex architectures. The architecture of $d_v = 3$ VNs are based on two $d_v = 2$ subnodes. The architecture of $d_v = 6$ VNs are based on two $d_v = 3$ and one $d_v = 2$ subnodes. IM lengths of $L = 1$ and $L = 2$ are used for $d_v = 3$ and $d_v = 6$ VNs, respectively.

A VN has two modes of operation, described as follows.

1) Partial Initialization Mode: Prior to the decoding operation and when the channel probabilities are all loaded into

[2]Here and in the rest of the paper, it is assumed that each DC takes one clock cycle.

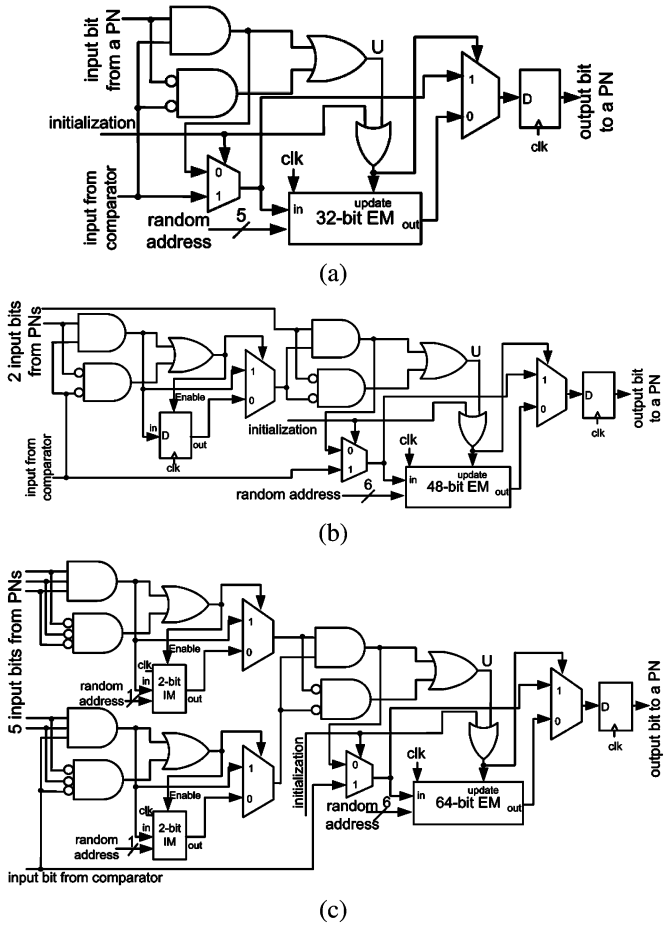

Fig. 6. Architecture for (a) a $d_v = 2$ VN, (b) a $d_v = 3$ VN and, (c) a $d_v = 6$ VN based on IMs and an EM (in each figure, only one output and its corresponding inputs are shown).

the decoder, VNs start to initialize their EMs according to the received probability. Although it is possible for EMs to start from zero state [41], however, the initialization of EMs improves the convergence of the stochastic decoder. In the implemented decoder, we consider partially initializing the EMs to 16 bits. During the partial initialization, the EMs of each VN are bit-serially updated with the output of the node comparator for $T_{\mathrm{LOAD}_{\mathrm{EMs}}} = 16$ DCs.

2) Decoding Mode: After the partial initialization phase, the decoding operation starts. Each VN in the Fig. 6, uses a signal $\mathbf{U}$ to determine if the VN is in the hold state ($\mathbf{U} = 0$) or not ($\mathbf{U} = 1$). When the VN is not in the hold state, the new regenerative bit is used as the output bit and also to update the EM. In the case of the hold state, a bit is randomly chosen from the EM. This scheme is also employed in each subnode to update the IMs. The random selection of bits in EMs and IMs are done by (pseudo) random addresses which vary in each DC. These addresses are also provided by the DRE in Section IV-G.

Due to the partial initialization scheme at the beginning of the decoder operation, the range of (pseudo) random addresses is limited to 4 bits (i.e., 0 to 15) for 40 DCs. This ensures that during the hold state, a valid bit is picked from EMs. When decoding proceeds for 40 DCs and EMs are updated, the DRE produces full range addresses for EMs.

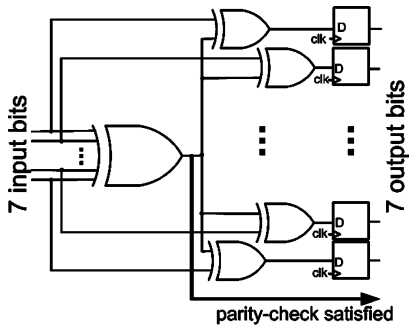

Fig. 7. Architecture of a stochastic $d_c = 7$ PN. The "parity-check satisfied" signal is used for termination criteria.

D. Hard-Decision Using Saturating Up/Down Counters

The output bit of each VN at the end of every DC is passed to an up/down counter. Each counter is incremented when receiving 1 and decremented when receiving a 0 bit. The counters are implemented as saturating counters which stop incrementing/decrementing when they reach their maximum/minimum limits. For this implementation, we used 4-bit saturating counters that count from -7 to 7. The sign-bit of each counter determines the hard-decision, i.e., in a BPSK transmission a 0 sign-bit of the counters determines a "$+1$" decoded bit and a 1 sign-bit determines a "-1" decoded bit.

Based on our observation, we discovered that for the case of stochastic LDPC decoders, up/down counters are mostly effective at low SNRs (high BERs). At high SNRs, up/down counters can be neglected and replaced by 1-bit flip-flops. In this case, the last output bit of each VN directly determines the hard-decision. The reason is that at high SNRs where convergence of stochastic decoders is fast, the counters easily become saturated (i.e., high reliability) which implies that they mostly receive constant output bits from VNs. The output bits of VNs at low SNRs are, however, less reliable and are more varying.

E. Stochastic Parity-Check Nodes

The construction of a PN is based on XORing the input bits received from VNs. Fig. 7 shows the structure of a $d_c = 7$ PN used in the implemented stochastic decoder. The construction of the $d_c = 6$ in the decoder is similar. PNs send their output bits to VNs. In addition, each PN produces a "parity-check satisfied" output signal which determines if the corresponding parity-check is satisfied. This signal is used to terminate the decoding as will be discussed in Section IV-H.

F. Asynchronous Pipelining and Interleaver Design

As mentioned earlier, the structure of interleavers in LDPC decoders results in (long) wires and forces a bottleneck on the speed and throughput of decoders. For this reason, fully parallel architectures can use pipelining to break the wires and increase the speed/throughput. However, pipelining the interleaver in conventional SPA or MSA-based decoders has a drawback: it increases the number of clock cycles required per iteration by a factor of P, the number of pipeline stages. The reason is that in SPA and MSA, there is a data dependency between iterations and the output of nodes at each iteration depends on their outputs at the previous iteration. For instance,

assume that the nonpipelined decoder runs for I iterations and uses 1 clock cycle per iteration. In the decoder with pipelined interleaver, P clocks are needed to pass messages generated in the previous iteration. Therefore, although the pipelined decoder is faster than the nonpipelined decoder, it needs $P \times I$ clock cycles to provide the same BER decoding performance. To reduce this inefficiency and to increase the utilization of decoder, the pipelined LDPC decoders need to decode more than one codeword in the pipelined interleaver at the expense of more hardware complexity [22]. Another suggested technique in the literature is block interlacing [13]. This technique is used to increase the throughput of the decoder by processing two consecutive blocks simultaneously. In [13], the block interlacing technique together with a message broadcasting technique provided high throughput for ASIC LDPC decoders using 16 iterations of MSA and 32 iterations of hard-decision message-passing decoding algorithm [48]. In addition to these methods, the flooding-type update-schedule algorithm is suggested in [49]. This algorithm allows limited partitioning of some of the long wires in the decoder using flip-flops [49] without affecting the required clock cycles per iteration. This relies on the similarity of time-consecutive messages which limitedly let nodes tolerate operating with messages produced in recent iterations. However, in this algorithm the degree of freedom for partitioning wires is limited. In [49], only messages from two consecutive iterations are used at VNs.

We claim that the above-mentioned drawbacks and limitations do not apply to stochastic decoders. The operation of stochastic nodes does not depend on the output bits produced in the previous DC. In fact, the order of bits in stochastic streams is not important for the nodes. That is why EMs with random bit selection and different lengths can be used at VNs. Therefore, if a stochastic decoder needs to operate for D DCs to decode a codeword, the P-stage pipelined stochastic decoder needs a maximum of $D + P$ DCs to decode the codeword. This interesting characteristic introduces a high degree of freedom for partitioning wires in stochastic decoders, which is especially advantageous for ASIC implementations.

1) In principle, an "arbitrary" number of pipeline stages can be used in the interleaver to break the wires and increase the clock rate to a "desired" speed.
2) Pipelining in a stochastic decoder does not need to be uniform in the entire factor graph. Different stages of pipelining can be used for different edges. It is also possible to only pipeline some (critical) wires in the interleaver with an arbitrary number of pipeline stages.

It should be noted that because stochastic decoders (and other bit-serial approaches) require less wires to represent the factor graph, pipelining the interleaver in stochastic decoders requires less hardware resources (registers) compared to the conventional SPA or MSA-based decoders. For the implemented stochastic decoder we used a four-stage pipeline interleaver.

G. Distributed Randomization Engine

The randomization engine is responsible for providing random numbers in the decoder. In the proposed architecture, (pseudo) random numbers are used in comparators and also as the addresses of EMs and IMs. Although this amount of random numbers for the entire decoder might seem high, as shown in [41], (pseudo) random numbers can be significantly shared at two levels without having a considerable impact on the decoding performance of the decoder: 1) different EMs can share the same random address and 2) random numbers used in comparators and random numbers used as the addresses of EMs and IMs can share bits. Sharing random numbers significantly reduces the complexity of the randomization engine.

In this paper, we propose a distributed architecture to generate random numbers. The DRE consists of 48 independent randomization engines (REs). Each RE generates the required random numbers for a portion of the factor graph and consists of only two 10-bit linear feedback shift registers (LFSRs) associated with prime polynomials. Random bits in each RE are generated by XORing different bits of the two LFSRs. The main reason to use a distributed structure is to reduce the routing required by DRE. Note that by using the asynchronous pipelining technique in Section IV-F the interleaver is no longer a bottleneck for the speed of a stochastic decoder. This is so because an arbitrary number of registers can be used to break long wires in a pipelined stochastic interleaver. In this case, the routing required by REs becomes a limiting factor for the speed and hence using a distributed architecture for generating random numbers becomes essential. It should be noted that the asynchronous pipelining technique is also applicable for DRE because the sequence/order of random numbers is not important for comparators, EMs and IMs.

H. Termination Criteria

The stochastic decoder checks two criteria in each DC to terminate the decoding operation: 1) it checks if all the PNs are satisfied or 2) if a maximum number of DCs has been exceeded. As soon as one of the criteria is satisfied, the decoder outputs the sign-bit of each saturating up/down counters as the decoded codeword and starts loading the probabilities for the next received block. Checking the first criterion is done by NORing "parity-satisfied" signals from all PNs (i.e., decoding is terminated if all the 528 parity-checks are satisfied). This is implemented as a three-stage pipelined NOR tree. The latter criterion is checked using a counter.

I. Input/Output Unit

As mentioned previously, the decoder uses $T_{\mathrm{IO}} = 24$ DCs to load 1056 received symbols (each with 6-bits precision) into the decoder and apply scaling. To do so, the decoder employs 264 input pins. While loading the probabilities, the decoder also outputs the previous 1056 bit decoded codeword using 44 pins (in $1056/44 = 24$ DCs). Therefore, the total IO overhead is $T_{\mathrm{IO}} = 24$ DCs.

V. Performance and Tradeoffs

Table II lists the parameters used for each code. To obtain the characteristics of the proposed architecture, the (1056,528) irregular LDPC decoder is implemented on a Xilinx Virtex-4 XC4VLX200-11FF1513 device using Xilinx ISE 9.2 tool. The Sections V-A–D discuss the performance of the decoder.

TABLE II
DECODING PARAMETERS USED

Code	EM length (M)	IM length (L)	α/Y	Max. DCs
(1056,528)	$\{32, 48, 64\}$ for $d_v = (2, 3, 6)$	$\{1, 2\}$ for $d_v = (3, 6)$	3.0/6	700
(1056,704)	$\{32, 48, 48\}$ for $d_v = (2, 3, 4)$	$\{1, 1\}$ for $d_v = (3, 4)$	4.5/6	700

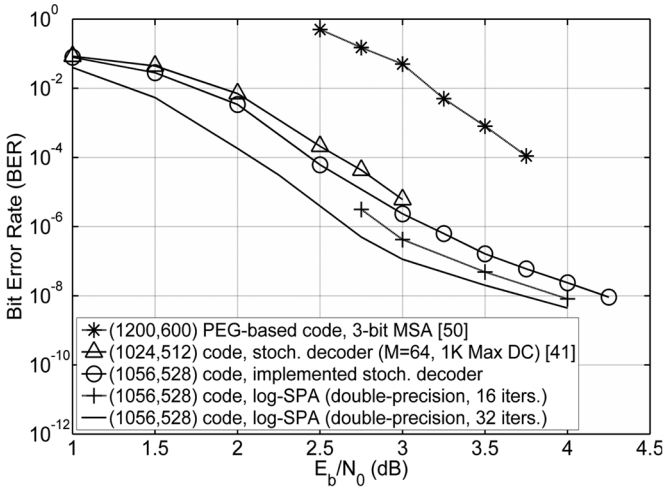

Fig. 8. BER performance of the implemented (1056,528) irregular stochastic decoder and decoders in [41] and [50].

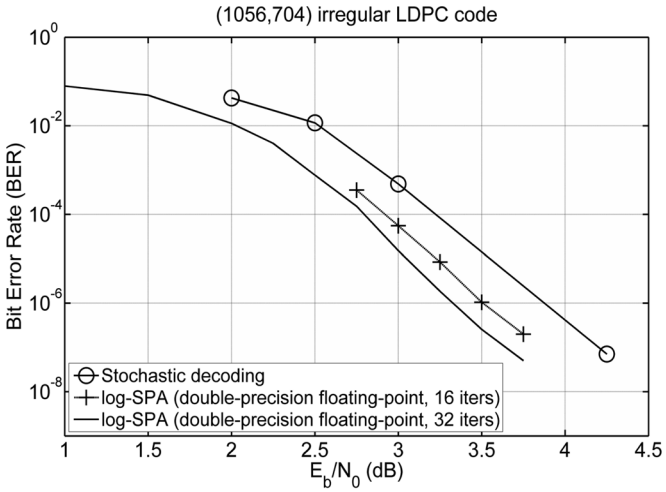

Fig. 9. BER performance of the (1056,704) irregular stochastic decoder.

A. Bit Error Rate Performance

Figs. 8 and 9 show the BER performance. These figures also depict the performance of the floating-point SPA. Also depicted in Fig. 8 is the performance of the decoder in [50] whose implementation values will be discussed in Section VI. Compared to floating-point SPA with 32 and 16 iterations, the irregular stochastic decoders only have a loss of about 0.5 and 0.25 dB, respectively, at low BERs. It should be highlighted that, in the shown BER region, a similar error-floor behavior to that of floating-point SPA is observed. Note that the floating-point implementation outperforms the fixed-point implementation which is usually considered in hardware implementations. In fact, due to the complexity/area concerns, in most fully parallel decoders fixed-point implementation with limited precision (usually $\leq$4 bits) is considered which causes additional decoding loss and higher error-floors.

TABLE III
XILINX VIRTEX-4 XC4VLX200-11FF1513 DEVICE UTILIZATION (LUT: 4-INPUT LOOKUP TABLE, FF: FLIP-FLOP)

Resources	Occupied	Available	Utilization
Slice LUTs	68163	178176	38%
Slice FFs	44502	178176	24%
IOBs	308	960	32%
Slices	46097	89088	51%

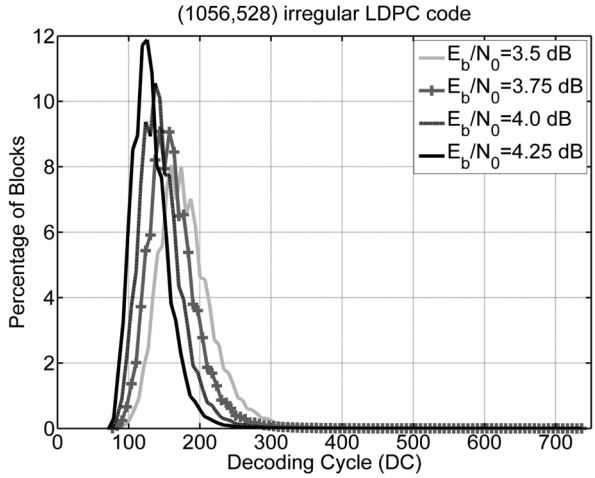

Fig. 10. Histograms of T_{AVG} at different SNRs (based on 1 million blocks).

B. Area and Clock Frequency

Table III summarizes the area consumption of the (1056,528) decoder on the FPGA device. The decoder occupies about 38% of the four-input lookup tables and 24% of the flip-flops available on the device. These occupied resources are distributed in 51% of the device slices. The decoder uses one clock cycle per DC and achieves a clock rate of 222 MHz after place-and-route.

C. Throughput

As mentioned in Section IV-H, the decoder terminates the decoding and starts loading the next codeword when all the parity check signals are satisfied or, when a maximum number of DCs has been exceeded. Due to these termination criteria T_{AVG}, the average number of DCs used to load, decode and output codewords determines the throughput of the decoder. For the sake of brevity, we refer to T_{AVG} as the average number of DCs in the rest of the paper. T_{AVG} is equal to

$$T_{\mathrm{AVG}} = T_{\mathrm{AVG_{CORE}}} + T_{\mathrm{IO}} + T_{\mathrm{LOAD_{EMs}}} \tag{6}$$

where $T_{\mathrm{AVG_{CORE}}}$ is the average number of DCs used by the core decoder to decode codewords and, as mentioned before, $T_{\mathrm{IO}} = 24$ and $T_{\mathrm{LOAD_{EMs}}} = 16$. It should be noted that at high SNRs (low BERs), T_{AVG} is much less than the maximum DC used by the core decoder ($T_{\mathrm{MAX_{CORE}}} = 700$ DCs). In fact at low BERs, only a few codewords require a high number of DCs to decode. This is shown in Fig. 10 where the histograms of

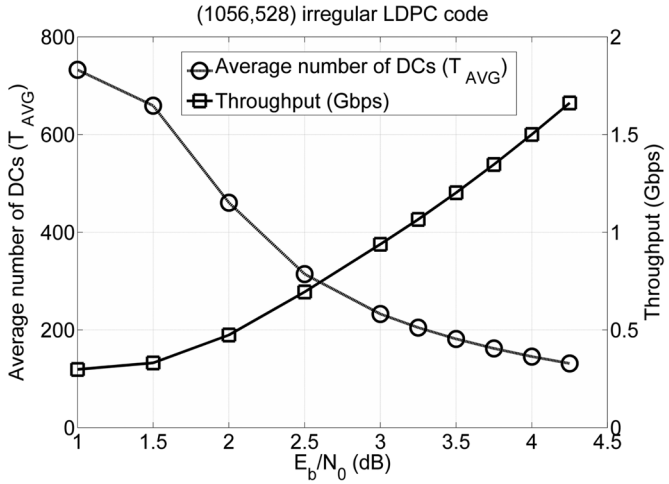

Fig. 11. T_{AVG} and throughput of the decoder at different SNRs (based on 1 million blocks).

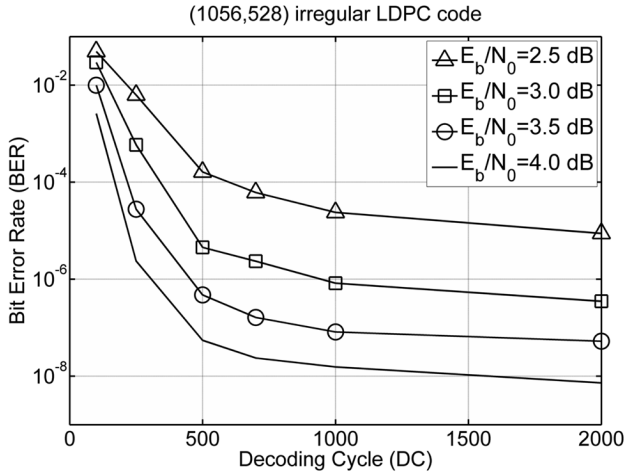

Fig. 12. BER performance of the (1056,528) stochastic decoder over DCs.

T_{AVG} over different SNRs are depicted. These histograms are based on observation of 1 million blocks.

Fig. 11 shows the observed T_{AVG} over different SNRs. It also shows the throughput of the decoder based on T_{AVG} at different SNRs for the achieved clock rate of 222 MHz. T_{AVG} and the throughput of the decoder vary at different BERs. As Fig. 11 shows, at high SNRs (low BERs) the throughput of the decoder is higher than the requirements of many applications. The decoder provides a throughput of more than 1 Gb/s for $E_b/N_0 > 3.5$ dB. The throughput of the decoder at $E_b/N_0 = 4.25$ dB is about 1.66 Gb/s.

D. Latency

Fig. 12 shows the BER performance of the stochastic decoder versus DC. The maximum number of DCs used for decoding the (1056,528) code was $T_{\mathrm{MAX_{CORE}}} = 700$ DCs. Due to the termination criteria of the decoder, $T_{\mathrm{MAX_{CORE}}}$ only influences the latency of the decoder. The maximum latency of the decoder is determined by T_{MAX} which is calculated as

$$T_{\mathrm{MAX}} = T_{\mathrm{MAX_{CORE}}} + T_{\mathrm{IO}} + T_{\mathrm{LOAD_{EMs}}}. \qquad (7)$$

For the (1056,528) decoder, T_{MAX} is 740 DCs. With the achieved clock rate of 222 MHz, this results in a maximum latency of 3.3 μs which is in an acceptable range for most applications such as the IEEE 802.16e (WiMAX) standard. In addition, as Fig. 12 suggests, for applications which have a strict latency requirement, it is possible to tradeoff the latency with some BER performance.

VI. Comparison

A. Comparison With FPGA Fully Parallel Decoders

Table IV compares different aspects of the most recent FPGA-based fully parallel LDPC decoders. To our knowledge, the decoders in [50] and [21] are the fastest and the second fastest (nonstochastic) FPGA-based fully parallel LDPC decoders, respectively. The decoder in [50] decodes a (1200,600) regular code which is constructed based on the Progressive-Edge-Growth (PEG) method. The throughput of the decoder is 12 Gb/s. This throughput was achieved by employing 3-bit fixed-point MSA with ten iterations. The decoder in [21] decodes a (480,355) regular code with a throughput of 650 Mb/s using bit-serial approximate MSA with 15 iterations. Also, the decoder in [41] is a (1024,512) stochastic decoder with a throughput of 706 Mb/s at a BER of 10^{-6}. Compared to the regular code used in [41], the (1056,528) code in this paper has a much more complex structure due to its irregularity and high degree nodes.

Table IV gives the throughput efficiency per information bit for each decoder. As discussed in Section V-C, the throughput of the (1056,528) decoder varies at different SNRs. For example, the decoder provides a throughput of 694 Mb/s at a $E_b/N_0 = 2.5$ dB ($\mathrm{BER} \simeq 10^{-4}$) and at $E_b/N_0 = 4.25$ dB ($\mathrm{BER} \simeq 10^{-8}$) the throughput is about 1.66 Gb/s. This throughput is beyond the requirements of most applications. Compared to [50] and [21], the stochastic decoder has a higher latency. This latency is however within an acceptable range for many applications. Usually, a latency limit of about 6 μs is assumed for channel decoders in applications such as WiMAX. In addition, as mentioned before, for applications with more strict latency requirements, it is possible to tradeoff the decoding performance and the latency (see Fig. 12).

Table IV also gives the absolute area as well as area efficiency based on the number of four-input lookup tables and flip-flops per coded bit and, slices per coded bit. Note that a Logic Element in the Altera Stratix architecture has one four-input lookup table and one flip-flop [51] which is half of the resources of a slice in a Xilinx Virtex-4 architecture [52]. Since the number of lookup tables and flip-flops were not reported in [21], the comparison with this decoder is based on the approximate slice per coded bit efficiency. The area efficiency of the stochastic decoder is better than the bit-serial decoder in [21]. Compared to [50], the stochastic decoder needs more lookup tables and flip-flops per coded bit (but offers about 1.3 dB decoding gain as shown in Fig. 8). The majority of this difference is due to the higher degree of VNs. As shown, the stochastic decoder in [41], with the same rate and node degrees as in [50], needs much less resources than this work and offers a better slice per coded bit efficiency compared to [50].

TABLE IV
COMPARISON OF RECENT FPGA-BASED FULLY PARALLEL LDPC DECODERS (LUT: 4-INPUT LOOKUP TABLE, FF: FLIP-FLOP, LE: LOGIC ELEMENT)

	Fastest non-stochastic FPGA-based decoders		Stochastic decoders	
	[21]	[50]	[41]	this work
Code	(480,355)	(1200,600)	(1024,512)	(1056,528)
Code structure	Regular, RS-based	Regular, PEG-based	Regular	Irregular, WiMAX code
Max. (d_v,d_c)	(4,15)	(3,6)	(3,6)	(6,7)
Decoding	Bit-serial approx. MSA	3-bit fixed-point MSA	Stochastic	Stochastic
Iterations or DCs	15 iters.	10 iters.	6K and 1K (max DCs)	700 (max DCs)
Input quantization	3 bits	3 bits	8 bits	6 bits
FPGA device	Stratix EP1S80	Virtex-4 XC4VLX200	Virtex-4 XC4VLX200	Virtex-4 XC4VLX200
Max. clock	61 MHz	100 MHz	212 MHz	222 MHz
Clocks per iter. or DC	3	1	1	1
Max. latency (μs)	0.73	0.1	28.30 (for 6K max. DCs) 4.71 (for 1K max. DCs)	3.3
Throughput	650 Mbps	12 Gbps	706 Mbps	1.66 Gbps
Throughput per information bit	1.83 Mbps	10 Mbps	1.38 Mbps (at E_b/N_0=3 dB)	3.14 Mbps (at E_b/N_0=4.25 dB)
Absolute area (in slices/LEs)	66588 LEs ($\simeq$33294 slices)	40613 slices	32875 slices	46097 slices
4-input LUTs and FFs per coded bit	not reported	57.5 LUTs and 15.7 FFs	46.0 LUTs and 20.1 FFs	64.5 LUTs and 42.1 FFs
Slices/LEs per coded bit	138.7 LEs ($\simeq$69.3 slices)	33.8 slices	32.1 slices	43.6 slices
Relative decoding gain (at BER=10^{-4})	not comparable	–	$\simeq$ 1.1 dB gain, compared to [50]	$\simeq$ 1.3 dB gain, compared to [50]
Hardware decoding loss (at BER=10^{-4})	not reported	$\simeq$ 0.25 dB loss from floating-point MSA (10 iters.)	$\simeq$ 0.2 dB loss from floating-point SPA (32 iters.)	$\simeq$ 0.4 dB loss from floating-point SPA (32 iters.)

Compared to other fully parallel approaches, an important advantage of the stochastic approach is its good decoding performance and error-floor behavior. Fig. 8 compares the performance of decoders in [50] and [41]. Both codes are regular and have the same rate and node degree. As shown, while the PEG-based code in [50] is longer, stochastic decoders outperform this decoder by more than 1 dB. It should be noted that the reported area efficiency for stochastic decoders is for providing a performance close to the floating-point SPA. The stochastic decoding approach is able to easily tradeoff the hardware complexity and decoding performance. For example, if a performance close to fixed-point MSA with limited precision is required, it is possible to significantly increase the area efficiency and/or reduce the latency of the stochastic decoder by using much shorter EMs/IMs, simpler DRE and by reducing the precision of comparators/counters.

B. Potentials for ASIC Implementation

The proposed architecture provides a significant potential for high-throughput fully parallel decoding on ASIC. The routing congestion problem caused by the large number of wires in the interleaver is a daunting problem in fully parallel ASIC LDPC decoders. In both ASIC and FPGA implementations, the number of wires in the interleaver significantly affects the area consumption [11], [12], [22]. As mentioned before, 50% of the core area in [11] with a die size of 52.5 mm^2 is occupied by wires using 4-bit precision. Another example is [50] where the authors report that with 5-bit precision the (1200,600) fully parallel decoder could not fit in a Virtex-4 LX200 FPGA device.[3] Therefore, the conventional fully parallel LDPC decoders have to use limited precision (usually, 3 or 4 bits) [11], [12], [14], [21], [50] to make the number of wires low and/or to prevent having large nodes with the expense of decoding loss. In stochastic decoding, the number of wires in the interleaver is minimized, i.e., one wire per direction. Also, the precision of the input probabilities does not affect the interleaver and the complexity of the majority of components in the decoder such as PNs, VNs and saturating up/down counters. In addition, stochastic decoding offers the asynchronous pipelining technique which provides a high degree of freedom for pipelining the interleaver and breaking long wires in the decoder with negligible effect on the number of DCs used. These interesting features enable stochastic decoders to easily achieve high speeds and provide a good decoding performance while having a better/comparable area consumption compared to conventional fully parallel methods with limited precision. The achieved high clock frequency of 222 MHz and the throughput of 1.66 Gb/s (at $E_b/N_0 = 4.25$ dB) on an FPGA device for a complex (1056,528) irregular code with high degree nodes shows this potential.

[3]Using 5-bit precision, this decoder would use $1200 \times 3 \times 5 \times 2 = 36000$ physical wires (in 2 directions) in the interleaver.

C. Comparison With Partially Parallel Decoders

As mentioned in Section I, partially parallel decoders use memory and share hardware to tradeoff area/flexibility with throughput. Fully parallel decoders, however, occupy much larger area but provide much higher throughput. In this respect, partially parallel and fully parallel decoders occupy a different place on the tradeoff curve. This is also the case for the proposed fully parallel architecture. Compared to the recent FPGA and ASIC partially parallel decoders, the (1056,528) stochastic decoder occupies much more (absolute) area but corrects more errors at a much higher speed. For example, the multirate partially parallel decoder in [53] occupies 1640 to 6568 slices and uses more than 60K bits of RAM, and provides a throughput of 41 to 278 Mb/s on a Xilinx Virtex-II 2V8000

device. Also, the partially parallel (8176,7154) decoder in [54] uses about 23 K to 27 K slices and 128 block RAMs of a Xilinx VirtexII-6000 FPGA device and, provides a throughput of 172 Mb/s with 15 decoding iterations. Another recent example is the ASIC implementation of a partially parallel (1024,506) decoder in [55] with a throughput of 31.2 Mb/s with eight decoding iterations.

VII. Conclusion

We proposed an architecture for fully parallel stochastic LDPC decoders. Using this architecture, we demonstrated the performance of a fully parallel stochastic LDPC decoder that decodes a state-of-the-art irregular (1056,528) code on a Xilinx Virtex-4 LX200 FPGA device. This decoder exploits several novel architectural techniques, provides a throughput of 1.66 Gb/s at $E_b/N_0 = 4.25$ dB (BER of 10^{-8}) and, achieves decoding performance within 0.5 and 0.25 dB loss of floating-point SPA with 32 and 16 iterations, respectively. The decoder also shows similar error-floor behavior as floating-point SPA with 32 iterations. To our knowledge, this decoder is the first stochastic LDPC decoder which decodes a state-of-the-art code and it is the first irregular and one of the fastest and most area-efficient fully parallel LDPC decoders implemented on FPGA.

Appendix
Proof of Stochastic VN and PN Operations

Consider the XOR gate in Fig. 3(b) and its input stochastic streams with $P_a = \Pr(a_i = 1)$ and $P_b = \Pr(b_i = 1)$. The output bit c_i is 1 when $a_i = 1$ and $b_i = 0$, or when $a_i = 0$ and $b_i = 1$. Therefore, $P_c = \Pr(c_i = 1) = P_a(1-P_b)+P_b(1-P_a)$. The output of other boolean operations is calculated similarly.

Now consider the VN structure in Fig. 3(a). This structure can be represented as a Markov chain with two states (0 state and 1 state). The probability (transition) matrix of this chain is

$$\mathbf{T} = \begin{bmatrix} 1 - P_a P_b & P_a P_b \\ (1 - P_a)(1 - P_b) & 1 - (1 - P_a)(1 - P_b) \end{bmatrix}$$

where t_{ij} in $\mathbf{T}$ is the probability of transition from state i to j $(i, j \in \{0, 1\})$. The probability of having 1 in the steady state of the chain (*i.e.*, $P_c = \Pr(c_i = 1)$) is calculated by computing the eigenvector of $\mathbf{T}$ with respect to an eigenvalue of 1, which is equal to $P_c = P_a P_b/(P_a P_b + (1 - P_a)(1 - P_b))$.

Acknowledgment

The authors would like to thank the reviewers and the associate editor for providing helpful comments.

References

[1] R. G. Gallager, "Low density parity check codes," *IRE Trans. Inf. Theory*, vol. 8, no. , 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, 1996.

[3] The Digital Video Broadcasting Standard [Online]. Available: www.dvb.org

[4] The IEEE P802.3an 10GBASE-T Task Force [Online]. Available: www.ieee802.org/3/an

[5] The IEEE 802.16 Working Group [Online]. Available: http://www.ieee802.org/16/

[6] The IEEE 802.11n Working Group [Online]. Available: http://www.ieee802.org/11/

[7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.

[8] N. Wiberg, "Codes and Decoding on General Graphs," Ph.D. dissertation, Electrical Engineering Dept., Linkoping Univ,, Linkoping, Sweden, 1996.

[9] F. Kschischang, B. Frey, and H. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, pp. 498–519, Feb. 2001.

[10] P. Urard *et al.*, "A 135 Mb/s DVB-S2 compliant codec based on 64,800b LDPC and BCH codes," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2005.

[11] A. Blanksby and C. 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.

[12] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A 3.3-Gbps bit-serial block-interlaced min-sum LDPC decoder in 0.13-μm CMOS," in *Proc. Custom Integrated Circuits Conf.*, Sep. 2007, pp. 459–462.

[13] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 1, pp. 74–78, Jan. 2008.

[14] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Multi-Gbit/Sec low density parity check decoders with reduced interconnect complexity," in *IEEE Int. Symp. Circuits Syst.*, San Jose, CA, May 2005, pp. 5194–5197.

[15] E. Boutillon, J. Castura, and F. Kschischang, "Decoder-first code design," in *Proc. 2nd Int. Symp. Turbo Codes Related Topics*, Brest, France, Sep. 2002, pp. 459–462.

[16] E. Yeo, B. Nikolic, and V. Anantharam, "Iterative decoder architectures," *IEEE Commun. Mag.*, vol. 41, no. 8, pp. 132–140, Aug. 2003.

[17] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Commun. Mag.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

[18] M. Mansour and N. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE J. Solid-State Circuits*, vol. 41, pp. 684–698, Mar. 2006.

[19] J. Chen *et al.*, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[20] T. Zhang and K. Parhi, "Joint (3,k)-regular LDPC code and decoder/encoder design," *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp. 1065–1079, Apr. 2004.

[21] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Greece, May 2006, pp. 149–152.

[22] T. L. Brandon *et al.*, "A scalable LDPC decoder ASIC architecture with bit-serial message exchange," *Integration, The VLSI J.*, vol. 41, no. 3, pp. 385–398, May 2008.

[23] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," *IEEE Commun. Lett.*, vol. 8, pp. 165–167, Mar. 2004.

[24] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," *IEEE Commun. Lett.*, vol. 9, pp. 814–816, Sep. 2005.

[25] N. Mobini, A. H. Banihashemi, and S. Hemati, "A differential binary message-passing LDPC decoder," in *Proc. IEEE Global Telecomm. Conf. (IEEE GLOBECOM)*, Washington, DC, Nov. 2007, pp. 1561–1565.

[26] S. Seo, T. Mudge, Y. Zhu, and C. Chakrabarti, "Design and analysis of LDPC decoders for software defined radio," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, Shanghai, China, Oct. 2007, pp. 210–215.

[27] S. Hemati, A. Banihashemi, and C. Plett, "A 0.18 μm analog min-sum iterative decoder for a (32,8) low-density parity-check (LDPC) code," *IEEE J. Solid-State Circuits*, vol. 41, pp. 2531–2540, Nov. 2006.

[28] F. Lustenberger *et al.*, "All-analog decoder for a binary (18,9,5) tail-biting trellis code," in *Proc. Eur. Solid-State Circuits Conf.*, 1999, pp. 362–365.

[29] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer, "An analog 0.25 μm biCMOS tailbiting map decoder," in *Proc. IEEE Custom Integrated Circuits Conf.*, Feb. 2000, pp. 356–357.

[30] V. Gaudet and G. Gulak, "A 13.3-Mb/s 0.35 μm CMOS analog turbo decoder IC with a configurable interleaver," *IEEE J. Solid-State Circuits*, vol. 38, pp. 2010–2015, Nov. 2003.

[31] C. Winstead, J. Dai, S. Yu, C. Myers, R. Harrison, and C. Schlegel, "CMOS analog map decoder for (8,4) Hamming code," *IEEE J. Solid-State Circuits*, vol. 39, pp. 122–131, Jan. 2004.
[32] D. Vogrig, A. Gerosa, A. Neviani, A. G. I. Amat, G. Montorsi, and S. Benedetto, "A 0.35 μm CMOS analog turbo decoder for the 40-bit rate 1/3 UMTS channel code," *IEEE J. Solid-State Circuits*, vol. 40, pp. 753–762, Mar. 2005.
[33] M. Arzel *et al.*, "Analog slice turbo decoding," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2005, pp. 332–335.
[34] B. Gaines, *Advances in Information Systems Science*. New York: Plenum, 1969, ch. 2, pp. 37–172.
[35] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electron. Lett*, vol. 39, no. 3, pp. 299–301, Feb. 2003.
[36] A. Rapley, C. Winstead, V. Gaudet, and C. Schlegel, "Stochastic iterative decoding on factor graphs," in *Proc. 3rd Int. Symp. Turbo Codes Related Topics*, Brest, France, Sep. 2003, pp. 507–510.
[37] W. J. Gross, V. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *Proc. 39th Asilomar Conf. Signals, Systems, Computers*, Pacific Grove, CA, Nov. 2005, pp. 713–717.
[38] C. Winstead, V. Gaudet, A. Rapley, and C. Schlegel, "Stochastic iterative decoders," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Sep. 2005, pp. 1116–1120.
[39] C. Winstead, "Error-control decoders and probabilistic computation," in *Proc. Tohoku Univ. 3rd Student-Organizing Int. Mini-Conf. Information Electronics System (SOIM-COE)*, Sendai, Japan, Oct. 2005, pp. 349–352.
[40] S. Sharifi Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, Oct. 2006.
[41] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "An area-efficient FPGA-based architecture for fully-parallel stochastic LDPC decoding," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, Shanghai, China, Oct. 2007, pp. 255–260.
[42] C. B. Schlegel and L. C. Perez, *Trellis and Turbo Coding*. Piscataway, NJ: IEEE Press, 2004.
[43] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proc. IEEE Global Telecomm. Conf. (IEEE GLOBECOM)*, Nov. 2001, vol. 2, pp. 1021–1025.
[44] F. Guilloud, E. Boutillon, and J.-L. Danger, "λ-min decoding algorithm of regular and irregular LDPC codes," in *Proc. 3rd Int. Symp. Turbo Codes (ISTC)*, Brest, France, Sep. 1–5, 2003, pp. 451–454.
[45] S. Howard, C. Schlegel, and V. Gaudet, "A degree-matched check node approximation for LDPC decoding," in *Proc. IEEE Int. Symp. Information Theory*, Adelaide, Australia, Sep. 4–9, 2005, pp. 1131–1135.
[46] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Survey of stochastic computation on factor graphs," in *Proc. 37th IEEE Int. Symp. Multiple-Valued Logic*, Oslo, Norway, May 2007, pp. 54–59.
[47] M. R. Yazdani, S. Hemati, and A. Banihashemi, "Improving belief propagation on graphs with cycles," *IEEE Commun. Lett.*, vol. 8, no. 1, pp. 57–59, Jan. 2004.
[48] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, pp. 599–618, Feb. 2001.
[49] N. Onizawa, T. Ikeda, T. Hanyu, and V. Gaudet, "3.2-Gb/s 1024-b rate-1/2 LDPC decoder chip using a flooding-type update-schedule algorithm," in *Proc. 50th IEEE Midwest Symp. Circuits Systems*, Brest, France, Aug. 2007, pp. 217–220.
[50] R. Zarubica, S. G. Wilson, and E. Hall, "Multi-Gbps FPGA-based low density parity check (LDPC) decoder design," presented at the IEEE Global Telecomm. Conf. (IEEE GLOBECOM), Washington DC, Nov. 2007.
[51] Stratix Device Handbook, Altera Corporation, San Jose, CA [Online]. Available: www.altera.com
[52] Virtex-4 User Guide, Xilinx Corp., San Jose, CA [Online]. Available: www.xilinx.com
[53] K. K. Gunnam, G. S. Choi, M. B. Yeary, and M. Atiquzzaman, "VLSI architectures for layered decoding for irregular LDPC codes of WiMax," in *Proc. IEEE Int. Conf. Communications*, Jun. 2007, pp. 4542–4547.
[54] Z. Wang and Z. Cui, "Low-complexity high-speed decoder design for quasi-cyclic LDPC codes," *IEEE Trans. VLSI Systems*, vol. 15, no. 1, pp. 104–114, Jan. 2007.
[55] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Trans. VLSI Syst.*, vol. 54, no. 6, pp. 542–546, Jun. 2007.

Saeed Sharifi Tehrani (S'05) received the B.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2002 and the M.Sc. degree in electrical and computer engineering from the University of Alberta, Edmonton, AB, Canada, in 2005. He is currently working towards the Ph.D. degree at the Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada.

His research interests include digital signal processing systems, low-complexity error-control-coding techniques, and design and hardware implementation of iterative decoders.

Mr. Sharifi Tehrani received the Post Graduate Scholarship Award from the Alberta Informatics Circle of Research Excellence (iCORE) during his M.Sc. study at the University of Alberta. He is also an awardee of the doctoral research scholarship from the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT) as well as a recipient of the Alexander Graham Bell Canada Graduate Scholarship Doctoral Award (CGS D) from the Natural Science and Engineering Council of Canada (NSERC).

Shie Mannor (S'00–M'03) received the B.Sc. degree in electrical engineering, the B.A. degree in mathematics (both *summa cum laude*), and the Ph.D. degree in electrical engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 1996, 1996, and 2002, respectively.

During spring semester 2002, he was a Lecturer at the Electrical Engineering Department of the Technion. From 2002 to 2004, he was a Postdoctoral Associate with the Massachusetts Institute of Technology (MIT), Cambridge. He is currently an Assistant Professor of Electrical and Computer Engineering at McGill University, Montreal, QC, Canada. His research interests include machine learning and pattern recognition, planning and control, multiagent systems, and communications.

Dr. Mannor was a Fulbright scholar in 2002, and he is currently a Canada Research Chair in Machine Learning.

Warren J. Gross (S'92–M'04) received the B.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1996, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada, in 1999 and 2003, respectively.

Currently, he is an Assistant Professor with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada. His research interests are in the design and application of signal processing microsystems and custom computer architectures. In summers 2004 and 2005, he was a Visiting Professor at the Université de Bretagne-Sud, Lorient, France.

Dr. Gross is a member of the Design and Implementation of Signal Processing Systems Technical Committee of the IEEE Signal Processing Society. He served as the General Chair of the Sixth Analog Decoding Workshop. He has served on the Program Committees for the IEEE Workshop on Signal Processing Systems, the IEEE Symposium on Field-Programmable Custom Computing Machines, and the International Conference on Field-Programmable Logic and Applications. He is a Member of the IEEE and a licensed Professional Engineer in the Province of Ontario.