Stochastic Implementation of LDPC Decoders

Warren J. Gross
Department of Electrical and Computer Engineering
McGill University
Montreal, Quebec, Canada, H3A 2A7
Email: wjgross@ece.mcgill.ca

Vincent C. Gaudet and Aaron Milner
Department of Electrical and Computer Engineering
University of Alberta
Edmonton, Alberta, Canada T6G 2V4
Email: vgaudet@ece.ualberta.ca

***Abstract*— LDPC codes are found in many recent communications standards such as 10GBASE-T, DVB-S2 and IEEE 802.16 (WiMAX) We present a review of a new class of "stochastic" iterative decoding architectures. Stochastic decoders represent probabilistic messages by the frequency of ones in a binary stream. This results in a simple mapping of the factor graph of the code into silicon. An FPGA implementation of a LDPC decoder with 8 information bits and 8 coded bits is described. On an Altera Cyclone FPGA, the throughput is 5 Mbps when clocked at 100 MHz and is expected to increase nearly linearly with the code length. Simulations of the decoder on an Altera Stratix FPGA indicate a potential throughput of 8 Mbps.**

I. Introduction

Low-Density Parity Check (LDPC) codes [1] are found in a variety of recent communications standards, such as 10GBASE-T, DVB-S2, and IEEE 802.16 (WiMAX) and therefore, hardware decoding architectures have recently been an active research area. In an effort to develop new, more efficient architectures, researchers have been looking "back" to reevaluate the fundamental assumptions underlying decoding hardware. A good example of this is the application of analog computing to decoders. It has been demonstrated that continuous-time, continuous-valued analog signal processing is well suited to developing high-speed, low-power, and area-efficient decoders [2–8]. In this paper we describe and present new results for a new decoding method, inspired by analog decoding, called *stochastic decoding* [9]. Stochastic decoding is based on simple digital "stochastic" computing circuits first described in the 1960s [10] and later used in implementing artificial neural networks [11]. We believe that stochastic decoders exhibit the simplicity of analog decoders, but address key limitation of analog technology such as scalability, device mismatch, and the relative difficulty of design and verification.

In this paper we describe a hardware implementation of a stochastic decoder for a LDPC code. In Section II we review the fundamentals of iterative decoding of LDPC codes. Section III describes the concept of stochastic decoding. An overview of a hardware architecture for a stochastic LDPC decoder is presented in Section IV. Results of an FPGA implementation are given in V. Conclusions are offered in Section VI.

II. Background

A. Decoding LDPC Codes

Low-Density Parity Check (LDPC) codes are a class of error-correcting codes that can approach the Shannon capacity.

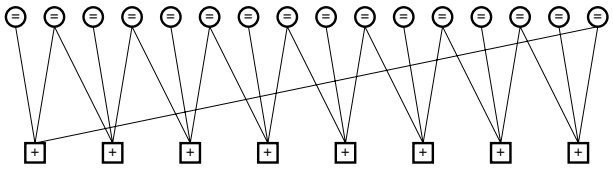

Fig. 1. Factor graph of a length 16 LDPC code.

Although discovered in the 1960's by Gallager [1], their potential was only recognized after the breakthrough discovery of turbo codes in 1993 [12]. Turbo codes, LDPC codes, and other related codes such as repeat-accumulate codes, form a class of of codes that have powerful decoding algorithms defined by iteratively passing messages on a graphical model.

The development of efficient iterative decoder hardware has been an active area of research over the last several years. From an implementation point of view, the interesting features of this type of decoder are:

1) The input and output of the decoder are "soft" values representing the probability of a bit being a '1',
2) They process soft values internally,
3) They consist of a collection of relatively simple processing elements, connected by an interconnection network,
4) Decoding consists of several rounds or, *iterations*, of passing soft *messages* between the processing elements along the interconnection network.

The graphical model for decoding LDPC codes is called a *factor graph* [13]. A factor graph is a bipartite graph that represents the code constraints and consists of two types of nodes, equality nodes and parity-check nodes. An example of a factor graph for a rate 1/2 length 16 LDPC code is shown in Figure 1.

Decoding is performed using the *sum-product algorithm* [13]. For each iteration, soft messages reflecting the probability of a bit being '1' are passed from the equality nodes to the parity check nodes along the interconnection network, then return messages are sent from the parity-check nodes back to the equality nodes. The messages are computed by functions known as "probability gates" [13] in the equality and parity check nodes. These can be viewed as the soft-information equivalent of the code constraints.

Graph nodes of arbitrary degree can always be decomposed into nodes of degree three and therefore, we only need

1-4244-0132-1/05/$20.00 ©2005 IEEE

to describe the operation of the two types of degree-three probability gates. The output on any given edge of a node is a function of the input on the other two edges. A degree-three node therefore needs to contain three copies of the basic probability gate to generate output messages on all three edges in parallel. The basic probability gates are:

- **Parity Check Probability Gate:** A parity-check node enforces even parity among the three bits represented along its edges. Given the probabilities of two input bits P_a and P_b, the probability of the output bit, P_c, is computed by the "2-input soft XOR" function:

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

- **Equality Probability Gate:** Similarly, the "2-input soft equality" function in the equality nodes for inputs P_a and P_b is:

$$P_c \quad \frac{P_a P_b}{(\quad - P_a)(\quad - P_b) + P_a P_b}, \tag{2}$$

where the division is required to normalize the messages to be the probability that c . Decoding proceeds until a codeword is found, or until a predetermined number of iterations is reached.

III. Stochastic Decoding

A stochastic decoder, introduced in [9], is a digital implementation that uses an alternate message representation to implement the probability gates as described in (1) and (2). The idea behind stochastic decoding is similar to Pearl's "stochastic simulation" of belief propagation [14], but Pearl's method is not directly applicable to error-control decoding [15].

A. Stochastic Message Representation

Messages passed along the edges of the graph in the sum-product algorithm are probabilities, with values between 0.0 and 1.0. Stochastic decoding represents these messages as *Bernoulli* sequences which encode a probability P as the frequency of '1's in a sequence of bits. For a sequence of N bits, if m bits are a '1', then the probability represented by this sequence is

$$P \quad \frac{m}{N}. \tag{3}$$

For example, a value of 0.5 can be represented by a sequence of bits, with exactly half of them being a '1'. Note that this representation is not unique, as it does not matter in what positions in the sequence the '1's are, and that the precision of representation increases with the length, N, of the sequence.

B. Stochastic Gates

The stochastic representation of messages in the graph results in very simple hardware for the probability gates in (1) and (2). The "soft XOR" gate required when the inputs are probabilities reduces to a standard "hard" XOR gate when the inputs are Bernoulli sequences. The parity-check node stochastic gate is shown in Figure 2 with the addition of a D flip-flop for timing synchronization.

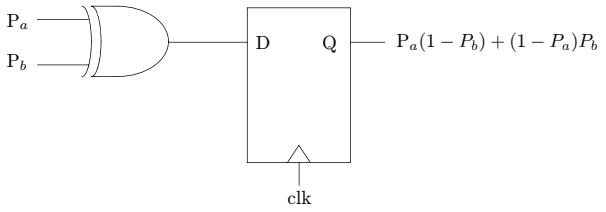

Fig. 2. Parity check stochastic gate.

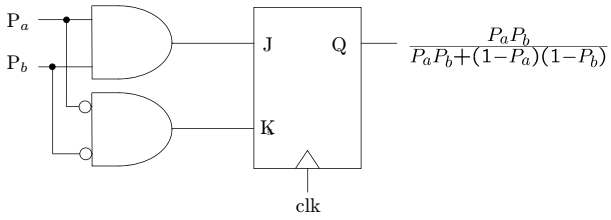

Fig. 3. Equality stochastic gate.

The equality node requires multiplications of probabilities and a division to realize the normalization. The multiplications are implemented with AND gates, and a J-K flip-flop implements the division as shown in Figure 3.

C. The Stochastic Decoding Algorithm

The stochastic decoding algorithm proceeds by the nodes exchanging a single bit along each edge in the graph at each clock cycle. We will refer to the decoding rounds as "cycles" to emphasize that they do not correspond directly to the iterations in the sum-product algorithm. One of the advantages of the stochastic approach, in addition to the simple stochastic gates, is that only two wires (one in each direction) are required to represent each edge in a fully parallel implementation of the factor graph, reducing the routing congestion problems typical of multi-bit digital representations [16].

IV. Decoder Architecture

A high-level block diagram of the decoder structure is given in Figure 4. The major components include the signal level to probability conversion, the digital to stochastic conversion, the stochastic decoder factor graph, and the final stochastic to digital conversion.

A. Input Conversion

Inputs to the stochastic decoder are assumed to be BPSK-modulated noisy samples, and must be converted into stochastic data streams. This is accomplished in two steps. The first step involves a transformation from the log-likelihood domain into the probability domain, and the second step involves generating a stochastic stream based on the appropriate probabilities.

The signal level to probability conversion is done with a small LUT. The conversion of a digital value to stochastic stream is a fairly simple process but requires a large number of independently random bits. Using multiplexers and these

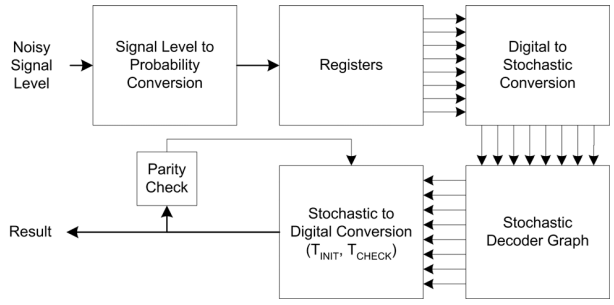

Fig. 4. Decoder block diagram.

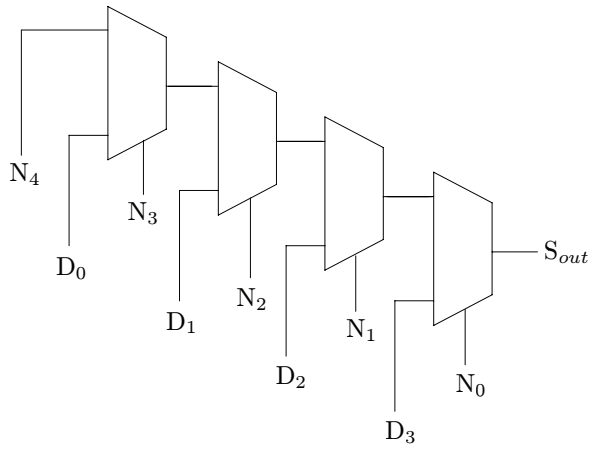

Fig. 5. Digital value to stochastic stream conversion.

random bits, the stochastic stream can be generated with a simple combinational logic structure like the one shown in Figure 5 where the vector D is the binary input value and N is the $(m+$ $)$-bit random noise vector, where m is the width of the word D. Figure 5 shows our implementation for m . This circuit is derived from that in [11] which uses m noise bits, setting N_4 . However, using a zero input biases the stochastic stream since an input of D can be exactly represented, but D can only be represented by a stream with the probability of a being / . By setting the input N_4 to be a random bit with probability / , the representation is shifted so that the smallest represented probability is / and the largest is / .

B. Supernodes

For stochastic operations to be performed effectively, each bit in an operand stream must be independent of the bits in the other operand stream. However, through some computational elements of the decoder, bit-to-bit dependencies and correlations are introduced into the output data streams. The data streams become increasingly unsuitable for further use in stochastic computation. The J-K flip-flop equality check nodes in the decoder seem to be acutely sensitive to these dependencies in the inputs and often become locked at a value for long periods of time until the right set of inputs arrives to unlock them [15].

In order to reintroduce randomness, the concept of a supernode was introduced in [15]. A supernode takes a stochastic stream and repeats the same signal value but regenerated such that the correlations and dependencies are removed. In short, it cleans up the signal and restores the useful properties of randomness permitting further stochastic computation on the stream.

The structure of a supernode is simple and consists of a counter and a digital to stochastic converter. The input stream feeds directly into the counter to tally the number of ones in the stream for a given number of cycles. This count is then given to the digital to stochastic converter to generate a new stream. For example, if the converter uses a three bit counter, the counter would count the number of ones in the input stream for eight cycles. The resulting count encodes the fraction of '1's in the regenerated stochastic stream.

For actual implementation, one must choose a width for the counter. A wider counter provides a higher degree of precision in repeating a stochastic stream with a constant value since more distinct levels can be represented. However, a counter with a smaller width requires much less time to update and can more readily propagate changes or fluctuations in the input stream's intended value. In this implementation it was found empirically that three bits was a suitable width for the repeaters in the graph. Making the counters wider impaired the cascading of calculation updates through the factor graph causing erroneous values to be calculated. Making the counters too narrow lost the precision and regenerative effect of the supernode.

C. Stochastic Decoder Graph

The stochastic probability streams are the inputs to the decoder graph. We implemented the length 16 LDPC code as shown in Figure 1. There are 16 equality nodes and 8 parity check nodes. Between the nodes of the decoder graph, supernodes were inserted to reduce correlations.

D. Stochastic to Digital Conversion

The final stage of decoding involves processing the output of the stochastic decoder graph into the final output of the decoder, which is a hard decision of the decoded bit. The stochastic stream to bit conversion is performed by an up/down counter that takes the stream as input. Every time a one is found in the stream, the counter is incremented. Every time a zero is found in the stream, the counter is decremented. At any given cycle, the sign bit of the counter indicates the hard decision, with a '0' sign bit indicating a positive value and therefore a decoded '1' and a '1' sign bit indicating a negative value and therefore a '0'.

There are two phases of computation. In the initialization phase, the stochastic decoder is operated for T_{INIT} cycles, but the output up/down counters are not updated. In the decoding phase, the decoder is run for T_{CHECK} cycles, and the up/down counters are updated on each cycle. After these T_{CHECK} cycles, decoding and up/down counting continues until a valid

codeword appears at the output, through verification of the parity check constraints of the code. Note that the number of clock cycles required to decode a codeword is not constant.

E. Generation of Noisy Bits

The conversion of a digital value into a stochastic stream requires a number of random bits. Not only are random bits required for this initial conversion, but more are required by the supernodes that regenerate the stochastic streams. It has been suggested that a standard linear feedback shift register (LFSR) provides an insufficient level of randomness for stochastic computation [11]. In the regular structure of an LFSR, there is strong correlation between adjacent bits. This degrades the quality of calculation that can be performed by the computational elements. Linear Hybrid Cellular Automata (LHCA) however, provide vectors with a higher degree of randomness. The correlation between adjacent bits is reduced significantly with relatively little structural overhead [17]. Several LHCA of different lengths and characteristic polynomials were used to generate the large number of required random bits.

F. Demonstration Framework

In addition to the decoder itself, a demonstration apparatus was built to provide the decoder with sample data, evaluate the output, and gather performance metrics. The apparatus and decoder were programmed together on the same FPGA to reduce the interface complexity and test times. In the end, it was found that there was sufficient space on the FPGA to include two copies of the test apparatus and decoder on the same chip that could be multiplexed to increase the testing throughput by a factor of two. For each variation of (T_{INIT}, T_{CHECK}) or signal-to-noise ratio (SNR), a trial of ^{26} or 67 million data bits were run through the system to get an accurate measurement of both decoder throughput and bit-error-rate (BER) performance. In addition to the components already described, the framework also included a user interface with push buttons, LEDs, and seven segment displays to facilitate the gathering of the test measurements.

V. FPGA Implementation

The final decoder design and the demonstration apparatus were synthesized onto an Altera Cyclone EP1C12F324C8 FPGA with , Logic Elements. FPGA resource usage, maximum clock frequency, bit error rate, and throughput measurements were recorded for a small (16, 8) LDPC decoder illustrated in Figure 1. The decoder occupied 1491 logic elements, or % of the FPGA. When synthesized onto the Cyclone FPGA, a maximum clock rate of just over 100 MHz was achieved. When synthesized onto a Stratix FPGA family, the maximum clock rate rose to 165 MHz. The actual bit rate, however, varies with the decoder configuration.

A. Decoder Performance

Figure 6 shows the bit error rate performance of the FPGA implementation of the decoder. Compared with a software simulation of stochastic decoding, there is approximately 0.4 dB loss at a BER of $^{-4}$, attributed to the coarse 4-bit quantization of the input probabilities. Very little difference in bit error rates was observed as the T_{INIT} and T_{CHECK} parameters were varied. In Figure 7 the decoding times for different (T_{INIT}, T_{CHECK}) variations are shown. The times shown in the graph are the average number of clock cycles required to decode a single codeword. Thus, for high SNR values the decoder is capable of running with an average period of approximately 19.5 clock cycles per uncoded information bit. On the Cyclone FPGA operating at 100 MHz, the decoder is capable of decoding at 5.1 MBit/s. Then, on a Stratix device operating at 165 MHz, the average bit rate rises to 8.4 MBit/s.

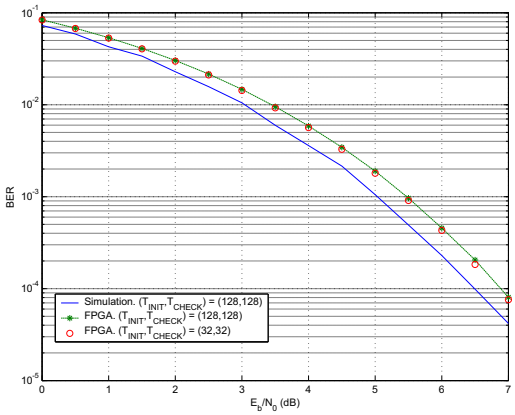

Fig. 6. The BER of a simulation of stochastic decoding and the BER of the FPGA implementation.

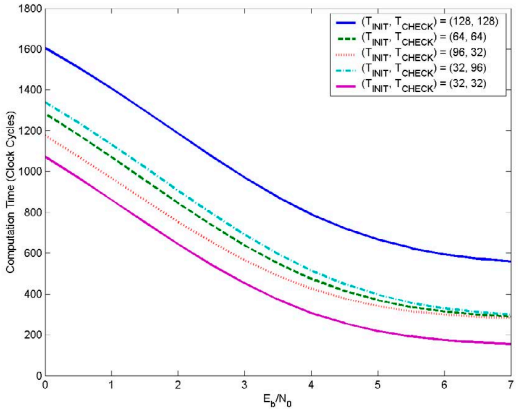

Fig. 7. Computation time to decode a single codeword for a length 16 LDPC decoder.

VI. Conclusions

This paper has reviewed a recent stochastic computation-based iterative decoding algorithm, as applied to LDPC codes, and has presented an FPGA implementation of this algorithm for a small length 16, rate 1/2 code. The throughput is

approximately 5 Mbps on a small FPGA. The throughput of fully parallel iterative decoders, such as analog and stochastic decoders, and decoders such as the one in [16], is postulated to increase nearly linearly with code size, due to the increasing number of information bits produced, and the relatively constant decoding latency. Further investigations into stochastic decoding will likely focus on the applicability of the algorithm to the larger LDPC codes found in recent communications standards, and to the suitability of the implementations to the required specifications.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, January 1962.

[2] F. Lustenberger, M. Helfenstein, H. A. Loeliger, F. Tarkoy, and G. S. Moschytz, "All-analog decoder for a binary (18,9,5) trellis code," in *Proceedings of the 1999 European Solid-state Circuits Conference (ESSIRC '99)*, September 1999, pp. 362–265.

[3] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer, "An analog 0.25μm BiCMOS tailbiting MAP decoder," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, February 2000, pp. 356–357.

[4] C. Winstead, J. Dai, S. Little, C. Myers, C. Schlegel, Y.-B. Kim, and W. J. Kim, "Analog MAP decoder for (8, 4) Hamming code in subthreshold CMOS," in *IEEE International Symposium on Information Theory*, June 2001, p. 330.

[5] C. Winstead, N. Nguyen, V. Gaudet, and C. Schlegel, "Low-voltage CMOS circuits for analog iterative decoders," accepted for publication in IEEE Transactions on Circuits and Systems I: Regular Papers.

[6] S. Hemati, A. H. Banihashemi, and C. Plett, "A high-speed analog min-sum iterative decoder," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT 2005)*, September 2005, pp. 1768–1772.

[7] D. Vogrig, A. Gerosa, A. Neviani, A. G. 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 Journal of Solid-State Circuits*, vol. 40, no. 3, pp. 753–762, March 2005.

[8] M. Arzel, C. Lahuec, F. Seguin, D. Gnaedig, and M. Jezequel, "Analoc slice turbo decoding," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 2005, pp. 332–335.

[9] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, pp. 299–301, January 6 2003.

[10] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*, J. T. Tou, Ed. Plenum Press, 1969, ch. 2, pp. 37–172.

[11] B. D. Brown and H. C. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, September 2001.

[12] C. Berrou, A. Glavieux, and P. Thitimajsjima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proceedings of the IEEE International Conference on Communications*, vol. 2, May 23-26 1993, pp. 1064–1070.

[13] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, February 2001.

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

[15] C. Winstead, A. Rapley, V. Gaudet, and C. Schlegel, "Stochastic iterative decoders," in *IEEE International Symposium on Information Theory*, September 2005, pp. 1116–1120.

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

[17] K. Cattell and J. C. Muzio, "Synthesis of one-dimensional linear hybrid cellular automata," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 3, pp. 325–335, March 1996.