

PARALLEL VLSI ARCHITECTURES FOR A CLASS OF LDPC CODES

Sungwook Kim, Gerald E. Sobelman and Jaekyun Moon

Department of Electrical and Computer Engineering
 University of Minnesota
 Minneapolis, MN 55454, USA
 Phone: (612)-625-8041, FAX: (612)-625-4583
 e-mail: sobelman@ece.umn.edu

ABSTRACT

This paper presents high-performance encoder and decoder architectures for a class of Low Density Parity Check (LDPC) codes. The codes considered here are based on the Parallely Concatenated Parity Check encoder structure. A major advantage of these codes is that the generator matrix and the parity check matrix are both sparse, which leads to efficient VLSI implementations for the encoder and the decoder. Our designs use 6-bit quantization with a code rate of 8/9 and a block size of 576 bits. An evaluation of the speed and hardware complexity is given, and simulation results for the bit error rate are obtained.

1. INTRODUCTION

Low density parity check (LDPC) codes have recently become of great interest due to their ability to achieve performance near the Shannon limit [1], [2]. LDPC codes provide an attractive alternative to turbo codes because decoders for LDPC codes are much simpler than decoders for turbo codes. Some LDPC decoder chip designs have appeared in the literature recently [3], [4]. However, encoders for LDPC codes can become quite complex. This undesirable situation is due to the fact that, while the parity check matrix is sparse, the corresponding generator matrix is not normally sparse, and the encoder design is determined by the structure of the generator matrix. Recently, a new class of LDPC codes using the Parallely Concatenated Parity Check (PCPC) encoder structure have been proposed [5], [6]. These codes have the property that the generator matrix and the parity check matrix are both sparse. This means that the encoder and the decoder both have simplified structures, and therefore these codes become attractive candidates for VLSI implementation.

As an example of the structure of this class of codes, the systematic parity check matrix, H , and corresponding generator matrix, G , for a rate 8/9 code with a message size of $k=512$ bits and a block size of $n=576$ bits are shown in

Figs. 1(a) and (b), respectively. The 16×512 sub-matrix P_1 is the basic building block for both G and H . Each row of P_1 contains a block of 32 consecutive 1s, with the rest of its entries being 0s. As shown in the figure, the block of 1s in each row is offset by 32 columns from the block of 1s in the previous row. The 64×576 matrix H is constructed by concatenating the 64×64 identity matrix with a 64×512 permutation matrix. As shown in part (a) of the figure, this permutation matrix is constructed by concatenating P_1 with three random permutations of P_1 , i.e. $P_2 = \pi_1(P_1)$, $P_3 = \pi_2(P_1)$, $P_4 = \pi_3(P_1)$ where the π_i functions are independent column permutations. Since H is in systematic form, the 576×512 generator matrix G is easily obtained as a concatenation of P_1 , P_2 , P_3 and P_4 with the 512×512 identity matrix. The key point in this construction is that both G and H are sparse.

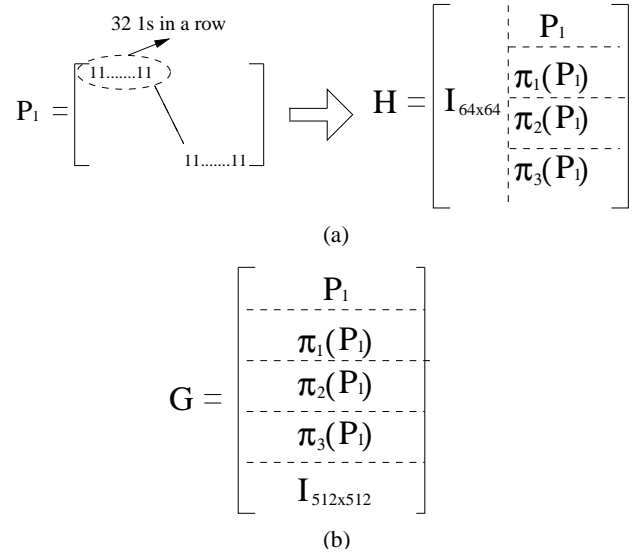


Fig. 1. Structure of the parity check and generator matrices using the PCPC construction. (a) Systematic parity check matrix. (b) Systematic generator matrix.

In this paper, we develop efficient VLSI architectures for the encoder and decoder of this code. We will show that it is possible to obtain parallel designs that offer high-throughput with moderate implementation complexity. Estimated gate counts and propagation delay values are given for the designs.

2. PARALLEL ENCODER

The block diagram for a parallel implementation of the encoder is shown in Fig. 2. It is a straightforward mapping of the previous generator matrix into hardware. The encoder contains three types of blocks: Each block labeled as π_i rearranges the incoming message bits according to the specified random permutation. This can be done directly using routing. Each block labeled as Parity-Check calculates a set of XOR functions on its input bits. Finally, the block labeled as Parallel-to-Serial converter produces a serial codeword that is sent to the channel.

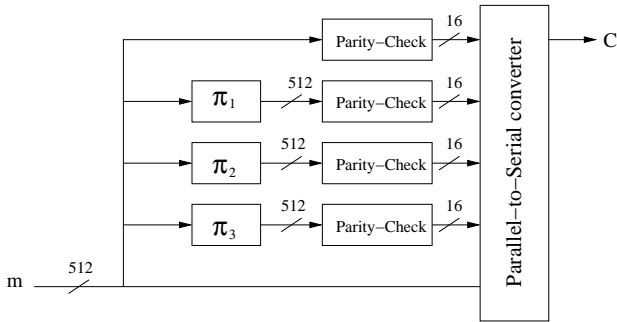


Fig. 2. Block diagram of the encoder.

3. PARALLEL DECODER

A specific LDPC code can be modeled using a bipartite graph that consists of bit nodes (corresponding to the bits of the codeword), and check nodes (corresponding to the parity checks), with an edge between a bit node and a check node for each corresponding 1 entry in the parity check matrix. Using the code parameters that we are considering, H has columns of weight 4 or 1 and rows of weight 33, with a total of $33 \times 64 = 2112$ non-zero entries. Therefore, there are 2112 edges in the bipartite graph for this code.

The most commonly used method for decoding LDPC codes is called the sum-product algorithm (SPA), as presented in [7]-[10]. In this algorithm, information is exchanged iteratively between the bit and check nodes of the bipartite graph. The algorithm may be summarized as follows:

- Step 1. Initialize. The decoder begins with prior log-likelihood ratios (LLRs) for the bits x_i .

- Step 2. Message passing from bits to checks.
- Step 3. Message passing from checks to bits.
- Step 4. Make hard-decisions.
- Step 5. Repeat steps 2-4 until either of the following termination criteria are met: (1) If $H \cdot \hat{x} = 0$, then the iterations are complete. (2) Otherwise, stop after a fixed maximum number of iterations whether they are complete or not.

As shown in Fig. 3, the parallel design of the LDPC decoder is composed of two types of computational blocks (called Cell-A and Cell-B), two interleavers (called B2C and C2B), and two pipeline registers. Cell-A computes the message from bit nodes to check nodes, as in Step 2 of the algorithm. (Note that Cell-A actually comes in two varieties, called Cell-A1 and Cell-A2, depending on whether it corresponds to a column of weight 1 or 4, respectively.) The outputs of all of the Cell-A blocks are sent through the B2C Interleaver which rearranges them using the appropriate permutations. These values are then stored in the first of two pipeline registers. Cell-B computes the message from check nodes to bit nodes, as in step 3 of the decoding algorithm. The output of all of the Cell-B blocks are stored in the second pipeline register. They are then sent to the C2B interleaver and fed back into the Cell-A blocks for another iteration.

Let (q, f) be a fixed-point number having a total of q bits, with f bits to the right of the binary point. Our designs are implemented using a $(6, 2)$ quantization, i.e. with 4 integer bits and 2 fractional bits. As will be seen, this leads to a satisfactory trade-off of implementation complexity vs. performance. Using this quantization scheme, each pipeline register requires $6 \times 2112 = 12672$ flip-flops, since the messages are represented as 6-bit quantities and there are 2112 messages.

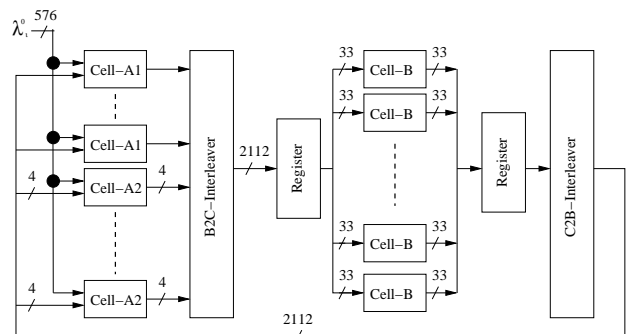


Fig. 3. Block diagram of the decoder.

3.1. Cell-A

The block diagram for Cell-A2 is shown in Figure 4. This block computes the messages from bit nodes to check nodes and makes hard decisions, based on steps 2 and 4 of the decoding algorithm. Cell-A2 accepts four messages ($r_1 - r_4$) that are represented in the (6, 2) quantization scheme. After summation of all four incoming messages and the LLR prior probability, the most-significant bit (MSB) of the LLR post probability, P^{post} , is used to obtain a decoded hard-limit information bit. The intermediate results within Cell-A2 are computed using an (8,2) quantization format and are then saturated to the (6, 2) format in the Saturation block.

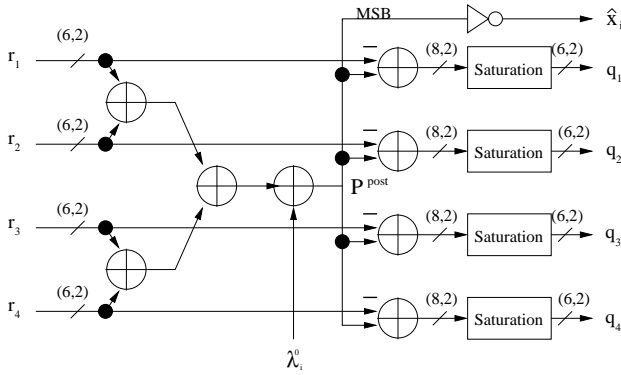


Fig. 4. Block diagram for Cell-A2, which computes the messages from bit nodes to check nodes and produces a hard-decision information bit.

The implementation of Cell-A1, which has only one incoming message, is constructed in an analogous fashion.

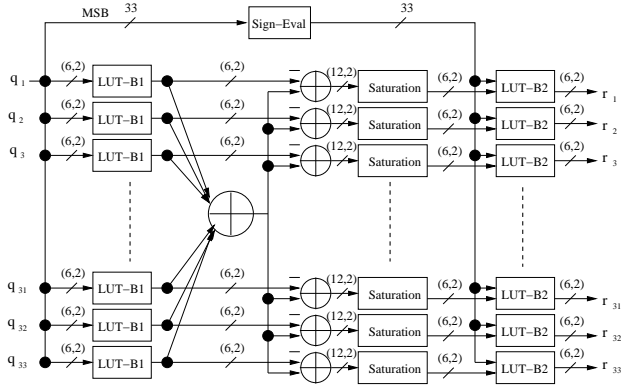


Fig. 5. Block diagram for Cell-B, which computes the messages from check nodes to bit nodes.

3.2. Cell-B

The block diagram for Cell-B, which computes the messages from check to bit nodes, is shown in Fig. 5. Cell-B is composed of an array of saturating adders, LUT-B1, Sign-Eval and LUT-B2. The 33 incoming messages ($q_1 - q_{33}$) are evaluated according to Step 3 of the decoding algorithm. The block LUT-B1 implements the nonlinear function $f(x) = \log \frac{\exp(x)+1}{\exp(x)-1}$. A plot of the function $f(x)$ (using the (6, 2) quantization scheme) is shown in Fig. 6. While this could be implemented using an SRAM-based look-up table, we assume here an implementation where each bit position of f is produced by an appropriate combinational logic function of the input bits. The array of adders produces intermediate results having a (12, 2) quantization, which are then saturated down to (6,2) in order to reduce the complexity of the subsequent LUT-B2 blocks. The sign evaluation for each message is performed by forming a collective XOR of the MSBs of all input messages. LUT-B2 takes the messages and the sign bits as inputs and computes the final messages ($r_1 - r_{33}$). Here again, we assume a dedicated combinational logic implementation of LUT-B2.

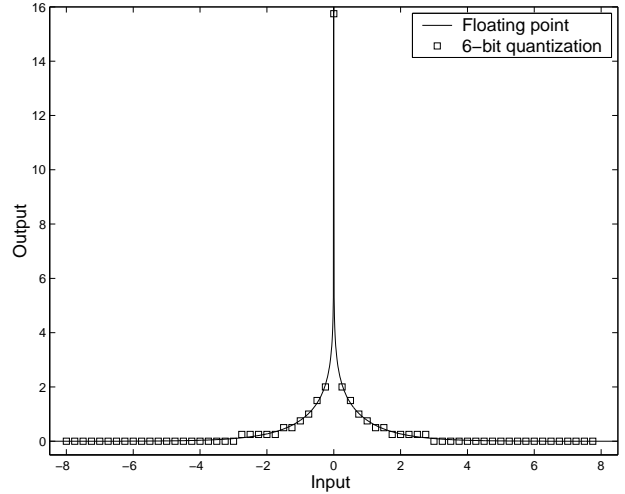


Fig. 6. The plot for the function $f(x) = \log \frac{\exp(x)+1}{\exp(x)-1}$ using the (6, 2) quantization scheme.

4. PERFORMANCE RESULTS

We specify the number of logic gates and the propagation delays in terms of a set of equivalent 2-input NAND gates. For the encoder (excluding the parallel-to-serial converter), the hardware cost is 5952 2-input NAND gates. The delay through this combinational logic is $10 \cdot T_{NAND}$, where T_{NAND} is the propagation delay of a 2-input NAND gate.

The costs of the main computational blocks (Cell-A and Cell-B) for the decoder are given in Fig. 7. These two

blocks require a total of approximately 2.4×10^6 2-input NAND gates and have a total propagation delay of $158 \cdot T_{NAND}$. Note that 92% of the total area is attributed to Cell-B, which contains the look-up tables. The simulation results for bit error rate (BER) vs. signal-to-noise ratio (SNR) are shown in Fig. 8, where an additive white Gaussian noise (AWGN) channel is assumed and the maximum number of decoding iterations is set at 20. Note that data for both the 6-bit fixed-point precision considered here and a theoretical floating-point implementation are shown in the figure. It can be seen that the loss in performance due to quantization effects is very small.

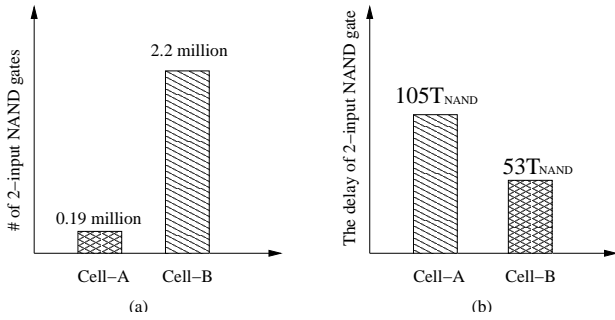


Fig. 7. Hardware cost and propagation delay for Cell-A and Cell-B in terms of equivalent 2-input NAND gates. (a) Cost. (b) Propagation delay.

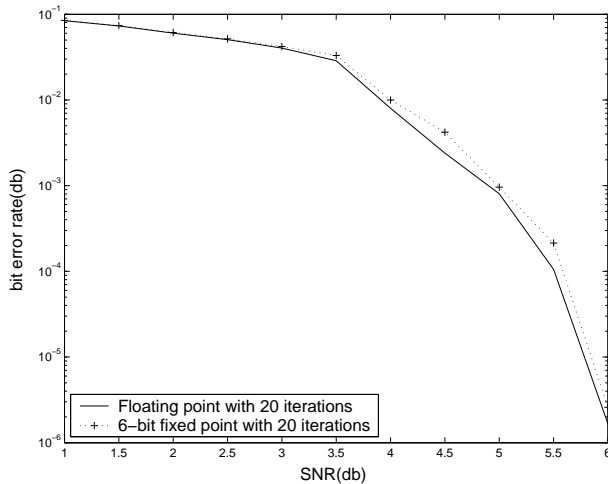


Fig. 8. Simulation results using floating-point precision and 6-bit fixed-point precision with a maximum of 20 iterations.

5. CONCLUSIONS

We have presented high-performance parallel implementations of an encoder and decoder for the PCPC class of LDPC codes. The designs are based on a code rate of 8/9, a block

size of 576 bits and a (6, 2) finite-precision implementation. The hardware cost and propagation delay of each unit has been given, and the BER performance has been shown to be very close to that for a hypothetical floating-point implementation of the same architecture. The efficient designs and the high level of performance indicate that these types of codes and structures may be of interest for a wide range of communications applications.

6. REFERENCES

- [1] R. G. Gallager, "Low Density Parity Check Codes," *IRE Trans. on Information Theory*, Vol. IT-8, pp. 21-28, Jan. 1962.
- [2] D. J. C. Mackay and R. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electron Letter*, Vol. 33, pp. 457-458, Mar. 1997.
- [3] C. Howland and A. Blanksby, "Parallel Decoding Architectures for Low Density Parity Check Codes," *Proceedings, IEEE International Symposium on Circuits and Systems*, pp. IV-742 - IV-745, 2001.
- [4] C. Howland and A. Blanksby, "A 220mW 1 Gb/s 1024-Bit Rate-1/2 Low Density Parity Check Code Decoder," *Proceedings, IEEE Custom Integrated Circuits Conference*, pp. 293-296, 2001.
- [5] T. Oenning and J. Moon "A Low-Density Generator Matrix Interpretation of Parallel Concatenated Single Bit Parity Codes," *IEEE Trans. on Magnetics*, Vol. 37, No. 2, pp. 737-741, March 2001.
- [6] L. Ping, S. Chan and K. L. Yeung, "Iterative Decoding of Multi-Dimensional Concatenated Single Parity Check Codes," *Proceedings, IEEE International Conference on Communications*, Vol. 1, pp. 131-135, 1998.
- [7] T. Oenning and J. Moon "Low Density Parity Check Coding for Magnetic Recording Channels with Media Noise," *Proceedings, IEEE International Conference on Communications*, Vol. 7, pp. 2189-2193, 2001.
- [8] J. Fan, A. Friedmann, E. Kurtas, and S. Maclaughlin, "Low Density Parity Check Codes for Magnetic Recording," *Proceedings, 37th Allerton Conf. on Communications, Control, and Computing*, 1999.
- [9] E. Yeo, P. Pakzad, B. Nikolic and V. Anantharam, "VLSI Architectures for Iterative Decoders in Magnetic Recording Channels," *IEEE Trans. on Magnetics*, Vol. 37, pp. 748-755, March 2001.
- [10] T. Zhang, Z. Wang and K. K. Parhi, "On Finite Precision Implementation of Low Density Parity Check Codes Decoder," *Proceedings, IEEE International Symposium on Circuits and Systems*, pp. IV-202 - IV-205, 2001.