A Bit-Serial Approximate Min-Sum LDPC Decoder and FPGA Implementation

Ahmad Darabiha, Anthony Chan Carusone and Frank R. Kschischang
Department of Electrical and Computer Engineering, University of Toronto
Email: {ahmadd,tcc}@eecg.utoronto.ca, frank@comm.utoronto.ca

***Abstract*— We propose a bit-serial LDPC decoding scheme to reduce interconnect complexity in fully-parallel low-density parity-check decoders. Bit-serial decoding also facilitates efficient implementation of wordlength-programmable LDPC decoding which is essential for gear shift decoding. To simplify the implementation of bit-serial decoding we propose a new approximation to the check update function in the min-sum decoding algorithm. The new check update rule computes only the absolute minimum and applies a correction to outgoing messages if required. We present a 650-Mbps bit-serial (480, 355) RS-based LDPC decoder implemented on a single Altera Stratix EP1S80 FPGA device. To our knowledge, this is the fastest FPGA-based LDPC decoder reported in the literature.**

I. Introduction

Low-density parity-check (LDPC) codes [1] have recently been adopted for several data communication applications due to their superior coding performance and parallelizable decoder architecture. LDPC codes allow a fine-level parallel message-passing decoding in which all the check and variable nodes are updated concurrently. This parallelism can potentially be used to build a decoder with Multi-Gbit/sec throughput. The major obstacle for efficient implementation of fully-parallel LDPC decoders is interconnect complexity which is the result of random location of 1's in the code's parity-check matrix.

In this paper, we propose a bit-serial scheme for fully-parallel LDPC decoders. Bit-serial computation allows variable and check nodes to communicate multi-bit messages over single wires, hence reducing the interconnect complexity. In addition, we introduce a new approximation to the check update function in min-sum decoding. In this approximation, in each check node only one minimum magnitude is calculated over all the check node inputs. Depending on the number of inputs that share the same minimum magnitude, a corrective constant is then added in order to generate the proper check outputs. We show that with 4-bit quantization this approximation reduces the check node area by 48% while introducing less than 0.1 dB loss in BER performance.

We illustrate feasibility of bit-serial LDPC decoding by implementing a (480, 355) RS-based LDPC decoder on a single Altera Stratix EP1S80 FPGA device based on the new proposed check node architecture. The decoder operates at maximum clock frequency of 61 MHz, performs 15 decoding iterations per frame and achieves 650 Mbps throughput.

This paper is organized as follows. The rest of this section briefly reviews LDPC codes and hardware implementation of iterative message-passing decoders. Section II illustrates new approximations for the check update functions in a min-sum decoder. Section III describes the internal architecture of bit-serial variable and check nodes for a fully-parallel LDPC decoder based on the approximate min-sum algorithm of Section II. Finally, in Section IV, an FPGA implementation of a bit-serial (480, 355) fully-parallel LDPC decoder is presented.

A. LDPC codes and min-sum decoding

A binary $(N, N-M)$ LDPC code, C, is the null space of a sparse $M \times N$ parity-check matrix, H. It can also be described by a bipartite graph, or Tanner graph. Tanner graph of an LDPC code consists of two set of nodes. Check nodes $\{c_1, c_2, \ldots, c_M\}$ represent the rows of H and variable nodes $\{v_1, v_2, \ldots, v_N\}$ represent the columns. An edge connects the check node c_m to the variable node v_n if and only if H_{mn} is nonzero. For a code with a full-rank $M \times N$ parity-check matrix, H, the code rate is $R = 1 - M/N$. We denote the set of variables that participate in check c_m as $N(m) = \{n : H_{mn} = 1\}$ and the set of checks in which the variable v_n participates as $M(n) = \{m : H_{mn} = 1\}$.

The following paragraphs describe min-sum (MS) decoding [2] which can be considered as an approximation to the commonly-used iterative sum-product (SP) algorithm [3]. Although the performance of MS is generally a few tenths of a dB lower than that of SP decoding, it is more robust to quantization errors when implemented with fixed-point operations [4]. Moreover, it requires much simpler hardware for the check node functions compared to SP decoding. In the MS decoding, similar to SP algorithm, the *extrinsic* messages are passed between check and variable nodes in the form of log-likelihood ratios (LLRs). Let $z_{mn}^{(i)}$ represent the LLR value for bit n, sent from variable node v_n to check node c_m in the ith iteration and similarly $\epsilon_{mn}^{(i)}$ represent the LLR value for bit n, sent from check node c_m to variable node v_n in the ith iteration. Suppose $W = (w_1, w_2, \ldots, w_N) \in C$ and $Y = (y_1, y_2, \ldots, y_N)$ are the transmitted codeword and the received sequence respectively. The MS decoding algorithm consists of the following steps:

1) Initialize the iteration counter, i, to 1 and let I_M be the maximum number of iterations allowed.
2) Initialize $z_{mn}^{(0)}$ to the *a posteriori* LLR, $\lambda_n = \log\big(P(v_n = 0|y_n)/P(v_n = 1|y_n)\big)$ for $1 \le n \le N$, $m \in M(n)$.
3) Update the check nodes, i.e., for $1 \le m \le M$, $n \in N(m)$, calculate
$$\epsilon_{mn}^{(i)} = \min_{n' \in N(m)\setminus n} |z_{mn'}^{(i)}| \prod_{n' \in N(m)\setminus n} \mathrm{sgn}(z_{mn'}^{(i)}). \qquad (1)$$
4) Update the variable nodes, i.e., for $1 \le n \le N$, $m \in M(n)$, calculate:
$$z_{mn}^{(i)} = \sum_{m' \in M(n)\setminus m} \epsilon_{m'n}^{(i)}. \qquad (2)$$
5) Apply a hard decision, i.e., compute $\hat{W} = (\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_N)$ where element $\hat{w}_n$ is calculated as
$$\hat{w}_n = \begin{cases} 0 & \text{if } \lambda_n + \sum_{m \in M(n)} \epsilon_{mn}^{(i)} \ge 0, \\ 1 & \text{otherwise.} \end{cases}$$
If $\hat{W}H^T = 0$ or $i \ge I_M$ stop decoding and go to step 6. Otherwise set $i = i + 1$ and go to step 3.
6) Output $\hat{W}^{(i)}$ as the decoder output.

B. Decoder implementation

In message-passing LDPC decoding, a large number of messages need to be updated and transfered between check and variable nodes in each iteration. Previous works have proposed several approaches for representing and updating these messages. In [5], analog signals are used to represent the extrinsic messages. In analog decoders the exponential voltage-current relationship of a transistor is used to realize the message-passing update functions. Although analog decoders have the advantage of low power consumption, they become impractical for decoding long LDPC codes due to the noise and process mismatch.

More conventional LDPC decoders often use multi-bit digital signals to represent the messages. In partially-parallel decoders [6], [7], the messages are transferred between the nodes through memory. This architecture reduces the decoder area by sharing the processing units, but this comes at the cost of reduced throughput. To achieve higher throughput, in the fully-parallel decoder presented in [8], all check and variable nodes are directly instantiated in hardware. Using this architecture, a throughput of 1 Gbps with 64 iterations per frame is reported. The major challenge in the implementation of fully-parallel LDPC decoders is the complex and random interconnection between the variable and check nodes. This problem is worsened when multi-bit buses are used to realize the edges in the code Tanner graph.

C. Bit-serial computation

To reduce the complexity of the interconnect in fully-parallel LDPC decoders, in this paper we investigate a bit-serial approach for both communicating and computing extrinsic messages. Fig. 1 shows the difference between the conventional bit-parallel scheme and a bit-serial scheme for a simple case of transferring an n-bit number, $b_n \cdots b_2 b_1$. In Fig. 1(a) all the n bits are sent over n parallel lines in one clock cycle. In contrast, in a bit-serial scheme as in Fig. 1(b), the message is sent over a single line in n clock cycles.

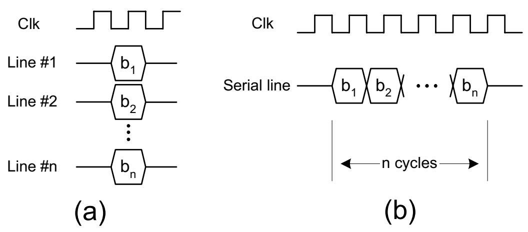

Fig. 1. Two alternatives for synchronous transmission of an n-bit number (a) bit-parallel: n bits sent in one clock cycle over n wires. (b) bit-serial: n bits sent in n clock cycles over one wire.

Stochastic computation [9] is similar to bit-serial computation in that it communicates extrinsic messages over single wires. It has a very simple check and variable node architecture but needs a significant amount of hardware overhead in oreder to translate the stochastic messages at the decoder inputs and outputs. In addition, the stochastic computation uses a redundant number representation which limits the decoder throughput.

In addition to simplifying the node-to-node interconnection, the bit-serial approach has several other advantages for fully-parallel LDPC decoders. In a bit serial scheme, the wordlength of computations can be increased simply by increasing the number of clock cycles allocated for transmitting the messages. Using this property, the precision of the decoder can be made programmable just by re-timing the node-to-node message transfers without the need for extra routing channels. Programmability of the decoder wordlength allows one to efficiently trade-off complexity for error correction performance. This in turn allows efficient implementation of gear-shift decoding [10]. Gear-shift decoding is based on the idea of changing the decoding update rule used in different iterations to simultaneously optimize hardware complexity and error correction performance. For instance, gear shift decoding often suggests applying a complex powerfull update rule in the first few iterations followed by simpler update functions in later iterations. Bit-serial computation allows efficient *shifting* between update rules by changing the computations wordlength.

Bit-serial decoding, however, imposes some challenges. The immediate effect is that it reduces the decoder throughput compared with fully-parallel implementations, as multiple clock cycles are required for transmitting a single message. Also some common check and variable update functions can not be efficiently implemented bit-serially. Although bit-serial fully-parallel LDPC decoders have a lower throughput compared with bit-parallel fully-parallel LDPC decoders we will show in this paper that their throughput can still be higher than hardware-sharing decoder schemes.

II. Simplified check update function

The MS decoding algorithm, as described in Section I, is cumbersome if implemented in a bit-serial hardware decoder. In this section, we introduce an approximation to the MS algorithm that reduces the hardware complexity of check nodes while causing minimal degradation in code performance. In fact, this approximation is also applicable to bit-parallel hardware decoders.

The first step is to replace the check update rule of (1) with

$$\epsilon_{mn}^{(i)} = \min_{n' \in N(m)} |z_{mn'}^{(i)}| \prod_{n' \in N(m) \backslash n} \mathrm{sgn}(z_{mn'}^{(i)}). \tag{3}$$

In other words, the sign of the check node outputs are calculated exactly the same as before but now the output magnitude is the minimum of magnitudes of *all* input messages. Fig. 2 compares the BER performance of original MS decoding algorithm with that of the modified MS based on (3) for two RS-based LDPC codes [11] using full-precision computations. This graph shows that with full-precision computations, the two algorithms perform almost identically. It is clear that a check update rule as in (3) significantly reduces the hardware complexity. The reason is that once the minimum among all input magnitudes is found it is sent out as the magnitude of all the outgoing messages, $\epsilon_{mn}^{(i)}$, for all $n \in N(m)$.

We have observed that although the above modification to MS results in almost no performance loss under full-precision operations, it introduces a considerable loss when performed in finite-precision. Fig. 3 shows the effect of the MS approximation when applied to quantized messages. In the following paragraphs we introduce a further change to the modified MS decoding algorithm that reduces the performance gap shown above.

The sign of the output messages in the new check update rule is the same as in (3). The magnitude of the output message is calculated as follows. First, for check node c_m, in the ith iteration, we define $M_m^{(i)} = \min_{n' \in N(m)} |z_{mn'}^{(i)}|$. We also define $1 \le T_m^{(i)} \le d_c$ as the number of inputs $z_{mj}^{(i)}$ to check node c_m that satisfy $|z_{mj}^{(i)}| = M_m^{(i)}$. The magnitude of check node outputs are calculated as

$$|\epsilon_{mn}^{(i)}| = \begin{cases} M_m^{(i)} + 1 & \text{if } T_m^{(i)} = 1 \text{ and } z_{mn}^{(i)} = M_m^{(i)} \\ M_m^{(i)} & \text{otherwise.} \end{cases} \tag{4}$$

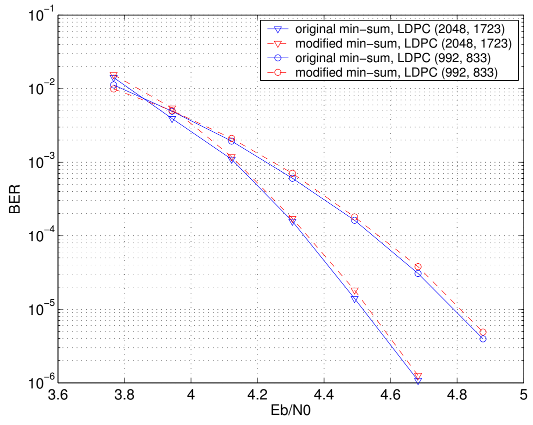

Fig. 2. Comparison between original min-sum and modified min-sum under full-precision operations for (2048, 1723) and (992, 833) LDPC codes.

Simulation results plotted in Fig. 3 show that with the new check update rule using 4-bit quantization the BER performance gap to the original 4-bit MS algorithm is reduced from 0.7 dB to less that 0.1 dB at BER of 10^{-6}. More importantly, the error floor effect is also avoided. In spite of the extra hardware needed for the correction term, VLSI implementation of a degree-15 check node using a CMOS-90nm cell library shows that a check node based on (4) is 48% smaller than a check node based on (1). This is because there is no need to calculate the second minimum among the check node inputs.

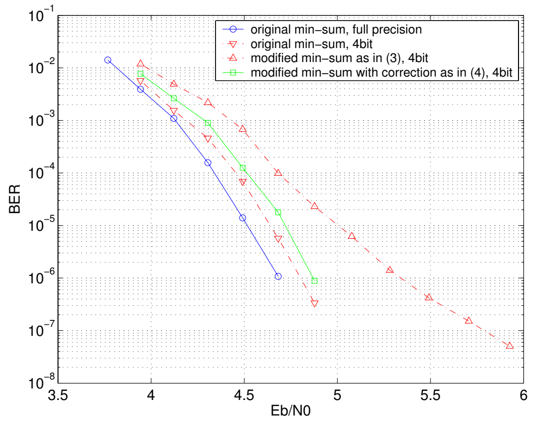

Fig. 3. Comparison between original min-sum and modified min-sums as in (3) and (4) under fixed-point operations for (2048, 1723) LDPC code.

III. Node Architecture

This section describes an internal architecture for bit-serial hardware implementation of variable and check nodes based on (2) and (4) respectively. As discussed in Section II, in modified MS algorithm only the smallest magnitude among all check inputs needs to be found. Fig. 4 shows the pipelined bit-serial module that finds the minimum of the check inputs. This module receives d_c inputs. Each input is an n-bit sign-magnitude binary number which is received bit-serially (MSB-bit first). The output is a bit-serial n-bit number which corresponds to the smallest magnitude in the inputs. Associated with each input there is a flip flop acting as status flag which indicates whether that input is still a candidate for being the minimum. At the beginning, the status flags are all reset to zero. As the MSB bits are received some flags become '1' indicating that the corresponding input is out of competition. Notice that the circuit in Fig. 4 only processes the magnitude of the check node inputs whereas the sign bit is generated separately using an XOR tree.

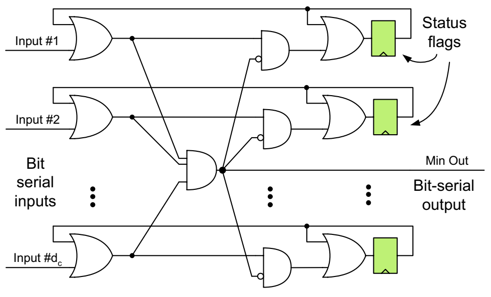

Fig. 4. A bit-serial module for detecting the minimum magnitude of the check node inputs.

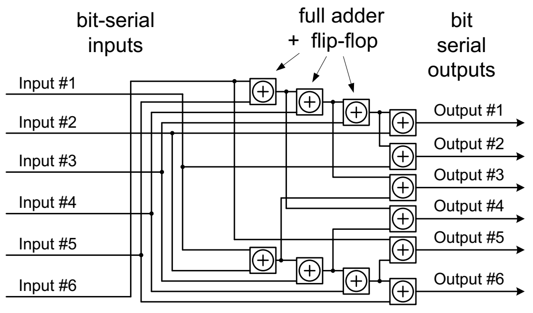

Fig. 5. A degree-6 variable node architecture for computing (2) with a forward-backward architecture [12]. Each adder box consists of a full-adder and a flip-flop to store the carry from the previous cycle.

To find an efficient bit-serial variable node architecture, we have investigated two alternatives. The first architecture, shown in Fig. 5, is based on a forward-backward computation [12]. The main difference between our approach and [12] is that here all the inputs and outputs are bit-serial. The main problem with the forward-backward architecture of Fig. 5 is that for a variable node of degree d_v the critical path consists of a chain of $(d_v - 2)$ two-input adders. For LDPC codes with relatively high d_v, this can limit the timing performance of the decoder. The second variable node architecture investigated in this paper is shown in Fig. 6. In this architecture, the bit-serial inputs are first converted to parallel inputs and then the additions are performed in one cycle using parallel adders/subtracters. The parallel outputs are finally converted back to bit-serial format before being sent to check nodes.

Table I summarizes the VLSI hardware cost and timing performance of two degree-6 variable nodes corresponding to the two above alternatives. The parameters in this table are based on the synthesis results using a CMOS 90nm cell library and with 3-bit quantization. Based on Table I, we have used the variable node architecture of Fig. 6 in the design presented in this paper since it is superior both in terms of timing and area.

Both check and variable nodes in this design are pipelined. For n-bit quantized input messages they generate n-bit output messages in n clock cycles. Each iteration of LDPC message-passing decoding consists of one check and one variable node update. As a result, using a conventional scheme, $2n$ clock cycles are needed to complete one iteration. However, in this design we adopt a block-interlaced scheme

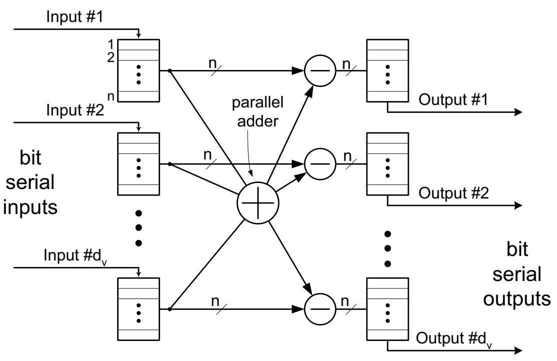

Fig. 6. A variable node architecture for computing (2) with parallel adders and parallel-serial converters at the inputs and outputs.

TABLE I

COMPARISON BETWEEN VARIABLE NODE ARCHITECTURES OF FIG. 5 (FORWARD-BACKWARD) AND FIG. 6 (PARALLEL ADDER/SUBTRACTERS) WITH $d_v = 6$ AND 3-BIT QUANTIZATION SYNTHESIZED WITH CMOS $90nm$ LIBRARY CELLS.

Architecture	Forward-backward	Parallel adder
Combinational area (μm^2)	2484	2099
Non-Combinational area (μm^2)	623	405
Total area (μm^2)	3107	2504
Minimum clock period (nsec)	3	2.20

[13] where two frames are processed in the decoder simultaneously in an interlaced fashion; while the check nodes process one frame, the variable nodes are processing the neighboring frame. So, in effect it takes only n cycles to complete one iteration, hence doubling the throughput.

IV. FPGA IMPLEMENTATION

To demonstrate the feasibility of bit-serial message-passing decoding, we have developed a fully-parallel (480, 355) RS-based LDPC decoder on a single Altera Stratix EP1S80 FPGA device using a configurable prototyping board called Transmogrifier-4 [14]. This decoder updates the extrinsic messages using the node architectures of Fig. 4 and Fig. 6. Since the updated messages are carried bit-serially over single wires, the complexity of node-to-node interconnections is less than that of conventional bit-parallel fully-parallel decoders [8]. Fig. 7 shows the measured BER performance from decoder hardware as well as the bit-true simulation. Table II summarizes the FPGA implementation results. The decoder operates at clock frequency of 61 MHz and performs 15 iterations per frame. Using the block-interlacing technique and a wordlength of 3 bits, each iteration takes 3 clock cycles to complete which results in 650 Mbps total throughput.

TABLE II

(480, 355) RS-BASED LDPC DECODER IMPLEMENTATION RESULTS ON ALTERA STRATIX EP1S80 DEVICE.

Logic elements (LEs)	66,588 (84%)
Max clock frequency (MHz)	61
Code length	480
Iterations per frame	15
Wordlength (bits)	3
Decoder throughput (Mbps)	650

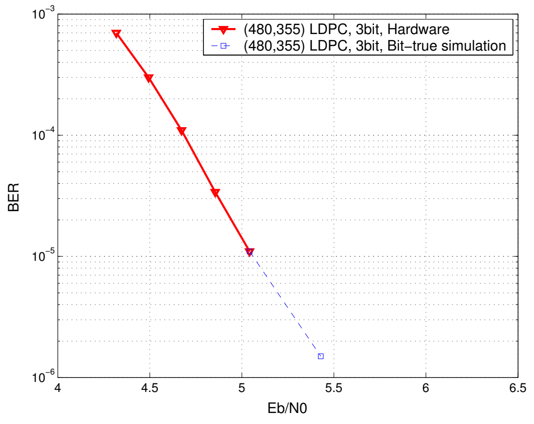

Fig. 7. FPGA hardware BER results and bit-ture software simulation.

V. CONCLUSION

In this paper we presented a bit-serial architecture for fully-parallel LDPC decoding. We also proposed a new approximation to check update function in MS decoding. A 650-Mbps FPGA-based fully-parallel LDPC decoder based on the above ideas is presented in this paper which to our knowledge is the fastest FPGA LDPC decoder reported in literature.

REFERENCES

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT press, 1963.
[2] N. Wiberg, *Codes and decoding on general graphs, PhD thesis*. Linkoping: Linkoping University, 1996.
[3] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Trans. on Information Theory*, vol. 47, pp. 498–519, Feb. 2001.
[4] A. B. F. Zarkeshvari, “On implementation of min-sum algorithm for decoding low-density parity-check (LDPC) cpdes,” in *IEEE Globecom conference*, 2002.
[5] F. Lustenberger, *On the design of analog VLSI iterative codes, PhD thesis*. Zurich: Swiss Federal Institute of Technology, 2000.
[6] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, “VLSI architectures for iterative decoders in magnetic recording channels,” *IEEE Transactions on Magnetics*, vol. 37, pp. 748–755, March 2001.
[7] T. Zhang and K. K. Parhi, “A 54 MBPS (3, 6)-regular FPGA LDPC decoder,” in *IEEE Workshop on Signal Processing Systems*, San Diego, CA, 2002.
[8] A. J. Blanksby and C. J. Howland, “A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check decoder,” *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, Mar. 2002.
[9] V. Gaudet and A. Rapley, “Iterative decoding using stochastic computation,” *Electronics Letters*, vol. 39, no. 3, pp. 299–301, February 2003.
[10] M. Ardakani and F. R. Kschischang, “Gear-shift decoding,” in *Proc. 21st Biennial Symp. on Comm.*, Queen’s University, Canada, 2002.
[11] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, “A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols,” *IEEE Comm. Letters*, vol. 7, no. 7, July 2003.
[12] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, “Efficient implementation of the sum-product algorithm for decoding LDPC codes,” in *IEEE Global Telecommunications Conference*, vol. 2, San Antonio, TX, 2001, pp. 1036–1036E.
[13] A. Darabiha, A. Chan Carusone, and F. R. Kschischang, “Block-interlaced fully-parallel LDPC decoders with reduced interconnect complexity,” *submitted to IEEE Transactions on VLSI Systems*.
[14] Transmogrifier-4, World Wide Web, http://www.eecg.utoronto.ca/~tm4.