An Alternative FPGA Implementation of Decoders for Quasi-Cyclic LDPC Codes

Murat Arabaci, *Student Member, IEEE,* and Ivan Djordjevic, *Member, IEEE*

***Abstract* — Due to their Shannon-limit-approaching performance and low-complexity decoding, low-density parity-check (LDPC) codes have been used for forward error correction in a broad-range of communication and storage systems. In addition to its low-complexity, the iterative decoding algorithm used for decoding LDPC codes is inherently parallel. To exploit the parallelism at a larger extent, a significant amount of research has been directed toward field-programmable gate-array (FPGA) implementation of LDPC decoders in recent years. In this paper, we propose an alternative FPGA implementation of decoders for quasi-cyclic LDPC codes based on MitrionC hardware programming language. We explain basic implementation steps using an example of a quasi-cyclic LDPC code particularly suited to optical communication and magnetic recording systems. We provide FPGA results and compare them against the traditional software simulation results.**

***Keywords* — Forward Error Correction, LDPC Codes, Communication Systems.**

I. Introduction

SINCE their rediscovery [1], LDPC codes have been used in a variety of applications in communication, broadcasting and storage systems due to their Shannon-limit-approaching performance and low-complexity decoding [2]. In addition to its low-complexity, the iterative decoding algorithm used for decoding LDPC codes is highly parallel in nature. Toward exploiting this advantage, there has been an important research progress on hardware implementations, in particular, FPGA implementations, of LDPC decoders [3], [4]. Since irregularity in the code's parity-check matrix increases the routing complexity, codes with structured parity-check matrices have lower routing complexity and hence smaller silicon area. In this paper, we use quasi-cyclic (QC-) LDPC codes owing to their favorable structural properties from the hardware implementation point of view.

Writing optimized applications to run on FPGA platforms requires climbing a steep learning curve for hardware description languages (HDLs) like VHDL and Verilog. Coding at the hardware level is a completely different concept than coding in software languages like C.

This paper was supported in part by StrataLight Communications, Inc., and in part by NSF under Grant IHCS-0725405.

The authors are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, USA. (e-mail: {arabaci, ivan}@ece.arizona.edu).

In this paper, we propose an alternative implementation for LDPC decoders using a hardware programming language called MitrionC, developed by Mitrionics, Inc. [5]. The basic premise of MitrionC is to build new semantics behind the familiar ANSI-C syntax and hence to isolate the user from the hardware-related concepts as much as possible. Using MitrionC, writing new codes or transforming existing ones written in software programming languages into those that can run on FPGAs is faster and the transition process is smoother.

In Section II, we introduce QC-LDPC codes. MitrionC design cycle is detailed in Section III. Different hardware implementation techniques for decoders of QC-LDPC codes are the subject of Section IV. In Section V, we present our implementation results, and conclude the paper with some remarks and future work in Section VI.

II. Quasi-Cyclic LDPC Codes

An LDPC code is a linear block code defined as the null space of a sparse parity-check matrix **H**. If every row of **H** has a row weight, i.e. number of elements in a row, of exactly ρ and every column has a column weight of exactly γ, then it is a (γ,ρ)-regular parity-check matrix; otherwise, **H** is said to be irregular. The parity-check matrix of every LDPC code can be represented as a bipartite graph, known also as the Tanner graph. The length of the shortest cycle in its corresponding Tanner graph is called as the girth of an LDPC code. QC-LDPC codes are a class of LDPC codes generated by the null space of an array of sparse circulant matrices [6].

In this paper, we use the design technique proposed in [7] for constructing large-girth QC-LDPC codes. The parity-check matrix, **H**, of a (γ,ρ)-regular QC-LDPC code is given as follows in this technique:

$$\mathbf{H} = \begin{bmatrix} I & I & I & \cdots & I \\ I & P^{S[1]} & P^{S[2]} & \cdots & P^{S[\rho-1]} \\ I & P^{2S[1]} & P^{2S[2]} & \cdots & P^{2S[\rho-1]} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I & P^{(\gamma-1)S[1]} & P^{(\gamma-1)S[2]} & \cdots & P^{(\gamma-1)S[\rho-1]} \end{bmatrix} \quad (1)$$

where I is p-by-p (p is a prime number) identity matrix, P is p-by-p permutation matrix ($P_{i,i+1} = P_{p,1} = 1$, $i = 1, 2, \ldots, p\text{-}1$; other elements of P are zeros). In this design, the set of integers S are to be carefully chosen from the set {0, 1, …, p-1} so that the cycles of short length, in Tanner graph representation of (1) are avoided. For example, by selecting $p = 1123$ and $S = \{0, 2, 5, 13, 20, 37, 58, 91, 135, 160, 220, 292, 354, 712, 830\}$, an LDPC code of rate 0.8, girth g = 10, column weight $\gamma = 3$ and length $N = 16845$ can be obtained.

III. MITRIONC DESIGN CYCLE

MitrionC is "an intrinsically parallel C-family language." Having syntax similar to ANSI-C, it aims to reduce the time required to transform codes written in traditional languages based on "order-of-execution" into those "centered on parallelism and data dependencies" [8].

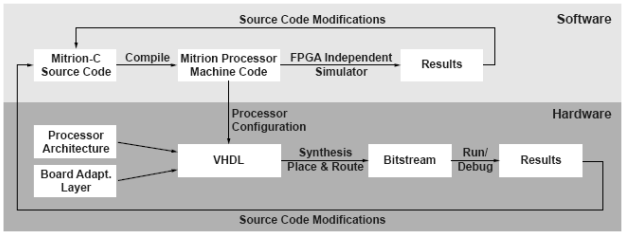

Fig. 1. MitrionC code development cycle [8].

We depicted MitrionC code development cycle in Fig. 1. First, MitrionC source code is compiled into a machine code to be interpreted by the Mitrion Simulator. The purpose of the simulator is to ensure proper functionality of the programs before downloading them onto the FPGAs. An important feature of the Mitrion Simulator is that it runs independently of the target FPGA platform. Upon completing the debugging cycle with the Mitrion Simulator and confirming the proper functionality of the program under test, we embed the MitrionC source code along with the details about the FPGA platform into a VHDL code with the help of the Processor Configuration Unit. The resulting VHDL code is then run through synthesis and place & route tools to generate the bit-stream for the target FPGA. The bit-stream can now be downloaded onto the target platform for testing. If further debugging is needed, this cycle can be repeated as needed.

IV. HARDWARE IMPLEMENTATION OF DECODERS FOR QC-LDPC CODES

A. Decoding Algorithm

LDPC codes are decoded using an iterative algorithm known as the belief-propagation algorithm or the sum-product algorithm. There are also reduced-complexity variants of the decoding algorithm that are more suitable for hardware implementation [9]. In this paper, we employ one such reduced-complexity algorithm known as the min-sum algorithm. It replaces the *log-tanh* functions in check node updates with simple comparisons while incurring only up to 0.5dB loss in performance [10]. The implementation issues of the min-sum algorithm are analyzed in detail in [11].

B. Hardware Implementation Techniques

Hardware implementations for LDPC decoders can be classified into three broad categories: fully-serial, fully-parallel and partially-parallel implementations [4]. Fully-serial implementation is similar to traditional software implementations. Even though the silicon area of the decoder is small because of sequentially utilizing the same processing element for all bit nodes (and check nodes), the throughput of the resulting decoder will be very low. On the opposite end, fully-parallel implementations provide high throughput in exchange of a larger silicon area. However, as the code length increases, the routing complexity of the decoders can get prohibitively high in a fully-parallel implementation. In order to bring together the benefits of these two schemes, we use a partially-parallel implementation. In this hybrid implementation technique, a group of bit nodes (and check nodes) are assigned to a single bit- (and check-) processing element. By changing the number of nodes mapped to a processing element, we can introduce a controlled level of parallelism into the architecture.

The most natural way of grouping the nodes in a QC-LDPC code is by partitioning it at its sub-matrix borders. We assign a bit-processing element (BPE) to each column of sub-matrices, and a check-processing element (CPE) to each row of sub-matrices in the parity-check matrix of a QC-LDPC code. Using this scheme, we obtain ρ BPEs and γ CPEs for a (γ,ρ)-regular QC-LDPC code as depicted in Fig. 2.

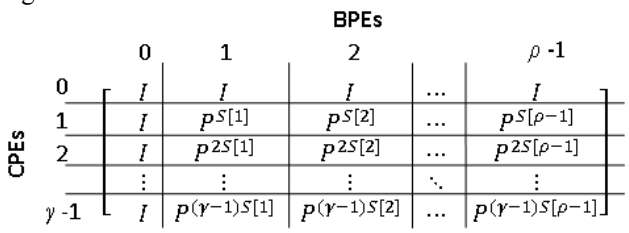

Fig. 2. BPE and CPE assignments for a (γ,ρ)-regular QC-LDPC code.

If each sub-matrix is a p-by-p matrix, then each BPE and CPE is responsible for processing p bit and check nodes, respectively. Assuming that BPEs and CPEs can access their data simultaneously, performing an iteration of the decoding algorithm takes $2p$ clock cycles. Compared to a fully-serial implementation that performs $(\gamma + \rho)p$ clock cycles per iteration, the improvement in performance is significant.

C. MitrionC Implementation

The partially parallel architecture described above can be implemented in MitrionC quite easily. However, in the current version of MitrionC, parallel access to memory blocks is not supported; and hence, instead of a simultaneous data access as alluded to above, BPEs and CPEs use a pipelined scheme while accessing to their corresponding shared memory blocks. Due to high-level abstraction capabilities of MitrionC, a code written for pipelined access to shared memory blocks can be adapted to accommodate parallel access to multiple memory blocks with slight modifications. Possessing such a feature that is common to software languages, MitrionC eases the code development and increases codes' flexibility.

Table 1 summarizes the memory allocation in our MitrionC implementation of the decoder for a (3,15)-regular LDPC(16935,13550) QC-LDPC code where we used the following notation: MEM B and MEM C denote the memory blocks used to store bit node and check node edge values, respectively; MEM E stores the codeword estimate and MEM I stores the initial log-likelihood ratios, and finally, MEM R holds the state of the random number

TABLE 1: MEMORY ALLOCATION FOR A (3,15)-REGULAR LDPC(16935, 13550) CODE

Memory Name	**MEM B**	**MEM C**	**MEM E**	**MEM I**	**MEM R**
Data width (bits)	8	11	1	8	32
Address width (bits)	16	16	15	15	10
Size (words)	50805	50805	16935	16935	625

generator, which is based on Mersenne Twister algorithm.

The pseudo MitrionC code given in Fig. 3 elaborates on how the data are transferred from MEM B to MEMC after being processed by BPEs. The code features three loop expressions of two types. The *for* loop sequentially executes its loop body for every bit node, *i*, that a BPE is responsible for. On the contrary, the *foreach* loop is a parallel loop, and hence, the operations in the loop body are applied to all the elements in its declaration simultaneously. To expatiate, due to the first *foreach* loop, all BPEs perform their operations on their corresponding *i*-th bit nodes in parallel. Since we are using a shared single memory block in our implementation to store the edge values of all check nodes, the second *foreach* loop causes a given bit node in a BPE to update its connections in MEM C in a pipelined fashion. As also shown in Fig. 3, we compute the memory addresses to read/write data from/to "on-the-fly" using the bit node ID (*i*), BPE ID (*b*) and CPE ID (*k*). This convenient calculation of addresses is possible because of the quasi-cyclic nature of the code and the way we assigned BPEs and CPEs to bit nodes and check nodes, respectively, as in Fig. 2.

```
for i = 0 to p -1 do
    foreach b = 0 to c -1 do
        - Read r data values from locations in the range [b * p * r + i
        * r, b * p * r + (i +1) * r -1] in MEM B,
        - Sum them up and store the sum,
        - Update MEM E at location (b * p + i).
        foreach k = 0 to r -1 do
            - Subtract the value located at (b * p * r + i * r + k) in
            MEM B from the sum computed above to obtain extrinsic
            information,
            - Use the result to update location (k * p + ((p - k * b) %
            p)) * c + b), where % is a modulo operator, in MEM C.
        end
    end
end
```

Fig. 3. A pseudo MitrionC code for processing steps in a BPE.

V. RESULTS

We tested our MitrionC implementations on the FPGA Subsystem located at the High Performance Computing (HPC) Center at The University of Arizona. The FPGA Subsystem consists of SGI RASC RC1000 Blade having two Virtex 4 LX2000 FPGAs.

For comparison, we ran the min-sum decoding algorithm both on the FPGA and in software. In the FPGA implementation, we used 8-bit quantization for bit nodes and 11-bit quantization for check nodes whereas in the software implementation, we used double-precision arithmetic. In Fig. 4, we present BER performances of both implementations for the (3,15)-regular, girth-10 LDPC(16935,13550) code. We observe a close agreement between the two BER curves. Furthermore, the performance of the min-sum algorithm is only 0.186 dB worse than that of the Jacobian-based sum-product algorithm at the BER of 10^{-8}, and the gap gets closer as the signal-to-noise ratio (SNR) increases. We computed the net-effective coding gain of the min-sum algorithm at BER of 10^{-12} as 10.538dB, which indicates that the code is very suitable for applications in optical communications and magnetic recording.

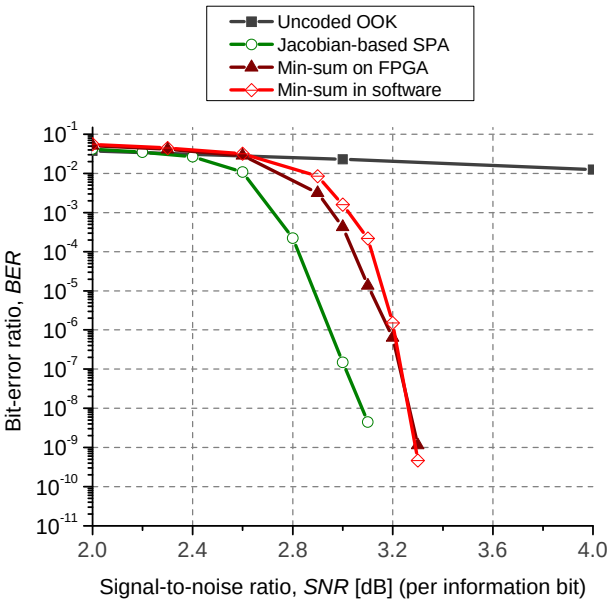

Fig. 4. BER performance comparison between FPGA and software implementations.

VI. CONCLUSION

We presented an alternative FPGA implementation of decoders for QC-LDPC codes based on the min-sum algorithm. We showed how MitrionC hardware programming language, which aims to hide almost all aspects of FPGA circuit design from the programmer, can be used to implement the decoder. With the help of this alternative implementation, it is easier to transform codes written in software languages like C into those that can run on FPGAs. We demonstrated an example implementation for a regular, girth-10 QC-LDPC code which is very suitable for applications in optical communications and magnetic recording.

Our future work includes applying the ideas on binary QC-LDPC decoders to their encoders and building a complete FEC system on the FPGAs. We then plan on extending these ideas further to encoders and decoders for non-binary QC-LDPC codes.

REFERENCES

[1] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Information Theory*, vol. 45, issue 2, pp. 399–431, March 1999.

[2] T. Ohtsuki, "LDPC Codes in Communications and Broadcasting," *IEICE Trans. Comm.*, vol. E90-B, no. 3, pp.440–453, 2007.

[3] E. Yeo, B. Nikolić, and V. Anantharam, "Iterative decoder architectures," *IEEE Comm. Mag.*, vol. 41, issue 8, pp.132–140, Aug. 2003.

[4] L. Sun and B. V. K. V. Kumar, "Field programmable gate array implementation of a generalized decoder for structured low-density parity check codes," in *Proc. of 2004 IEEE Conf. Field-Programmable Technology,* pp. 17–24.

[5] *Mitrion Product Brief*, Mitrionics Inc. Los Gatos, CA, 2008. Available: http://www.mitrionics.com/?document=Mitrion_product_brief.pdf

[6] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd ed., Upper Saddle River, NJ:Printice Hall, 2004.

[7] I. B. Djordjevic, L. Xu, T. Wang, and M. Cvijetic, "Large girth low-density parity-check codes for longhaul high-speed optical communications," in *Proc. OFC/NFOEC 2008*, San Diego, CA, Paper no. JWA53.

[8] *Mitrion Users Guide*, v1.5.0-001, Mitrionics Inc. Los Gatos, CA, 2008.

[9] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, X. -Y.Hu, "Reduced-Complexity Decoding of LDPC Codes," *IEEE Trans. Comm*., vol. 53, issue 8, pp. 1288–1299, Aug. 2005.

[10] X. -Y. Hu, E. Eleftheriou, D. -M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *Proc. of 2001 IEEE GLOBECOM*, San Antonio, TX, vol. 2, pp. 1036–1036E.

[11] F. Zarkeshvari and A. H. Banihashemi, "On implementation of min-sum algorithm for decoding low-density parity-check (LDPC) codes," in *Proc. of 2002 IEEE GLOBECOM*, Taipei, Taiwan, vol. 2, pp. 1349–1353.