Analysis and Implementation of Resource Efficient Probabilistic Gallager B LDPC Decoder

Burak Ünal*, Fakhreddine Ghaffari†, Ali Akoglu*, David Declercq†, and Bane Vasić*
*Dep. of Electrical and Computer Engineering, Uni. of Arizona, Tucson, AZ, 85719 USA
{burak, akoglu, vasic}@email.arizona.edu
†ETIS, ENSEA/CNRS UMR-8051/Uni. of Cergy-Pontoise, France
{ghaffari, declercq}@ensea.fr

***Abstract*—Low-Density-Parity-Check (LDPC) codes have gained popularity in communication systems and standards due to their capacity-approaching error-correction performance. In this paper, we first expose the tradeoff between decoding performance and hardware performance across three LDPC hard-decision decoding algorithms: Gallager B (GaB), Gradient Descent Bit Flipping (GDBF), and Probabilistic Gradient Descent Bit Flipping (PGDBF). We show that GaB architecture delivers the best throughput while using fewest Field Programmable Gate Array (FPGA) resources, however performs the worst in terms of decoding performance. We then modify the GaB architecture, introduce a new Probabilistic stimulation function (PGaB), and achieve dramatic decoding performance improvement over the GaB, exceeding the performance of GDBF, without sacrificing its superior maximum operating frequency.**

Keywords: ***high-performance probabibilistic hard-decision LDPC decoders, FPGA architectures.***

I. Introduction

Research efforts in decoding Low-Density Parity Check (LDPC) codes have led to design and implementation of a myriad of iterative message-passing decoding algorithms approaching Shannon capacity [1]. Decoding algorithms mainly differ based on the nature of iterative operations applied over the received messages. Complexity level of these operations determine the trade-off between hardware performance and decoding performance. One example decoding algorithm is the Gradient Descent Bit Flipping (GDBF [2]) and its variation Probabilistic GDBF (PGDBF [3]), which have been used as an alternative to the Min-Sum [4] algorithm to reduce its computation complexity with an acceptable decoding performance loss. Here we note that, throughout this paper, with the hardware performance, we refer to the operational clock rate and throughput as well as the resource requirements of the decoder algorithm, and with the decoder performance, we refer to the error correction capability of the decoder algorithm measured based on the Frame Error Rate (FER) metric. In this study we take a step back and revisit one of the early methods for hard-decision LDPC algorithm, called Gallager B [5]. Since its introduction in 1962, unlike other methods ([6] and [7]), hardware realization of GaB has not been of interest to researchers due to its poor decoding performance, and thus its application is limited to environments that require fast execution. In this study our first aim is to exploit the simplicity of the GaB operations to design a high-throughput decoder. We propose a resource efficient GaB architecture for widely used quasi-cyclic (QC)-LDPC codes, implement it on the FPGA, and evaluate its hardware performance with respect to GDBF and PGDBF. To the best of our knowledge there is no prior work on FPGA based GaB implementation. Our second aim is to improve the decoding performance of the GaB. For this, we introduce a probabilistic stimulation function, which stochastically ([8] and [9]) affects the state of the iterative decoding process and comes with negligible hardware overhead. We show that the modified GaB architecture archives better decoding performance without sacrificing throughput. The rest of the paper is organized as follows. In section II, we give an overview of the GaB, GDBF and PGDBF algorithms and compare their computation complexities. In section III, we present the PGaB algorithm and its hardware implementation. In Section IV, we present our simulation environment, followed by the performance analysis of the PGaB in Section V. Finally, we present in Section VI our conclusions and future work.

II. Overview of Decoding Algorithms

An LDPC code and its decoding algorithm are defined by a bipartite graph with two disjoint sets of variable node and check node vertices. To each of n variable nodes of a decoder, a so-called Variable Node Unit (VNU), is assigned. Similarly to each of m check nodes, a Check Node Unit (CNU) is assigned. Each edge in the Tanner graph (Figure 1) connects one CNU to one VNU. The degrees of VNU and CNU (dv, dc) are defined based on their numbers of adjacent vertices. The topology of the bipartite graph is represented by a parity check matrix (H matrix) as shown in Figure 1. Based on a decision function applied over the received messages from each adjacent vertex, each CNU and VNU sends a message back to its adjacent vertices. This iterative message processing between nodes recover the original data, which may have been exposed to channel noise.

A. Gallager B(GaB)

Let E(x) represent a set of edges connected to a node x which can be a VNU or a CNU, $m_i(e)$ represent the messages sent on edge e from a VNU to a CNU at iteration i, and $m'_i(e)$ represent the messages sent on edge e from a CNU to a VNU at iteration i. The received word from channel at a VNU v is denoted as $r(v)$. We express the operation of VNU and CNU

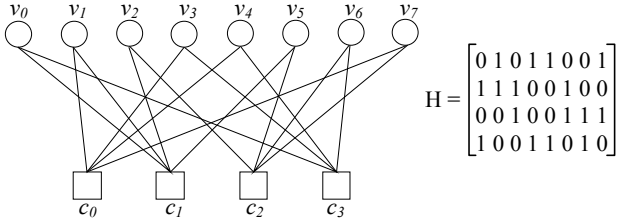


Figure 1. Tanner graph (left) and its parity check matrix (right).

using equations 1 and 2 respectively.

$$m_i(e) = \begin{cases} 1 & r(v) + \sum_{e' \in E(v)-e} m'_i(e') > b_i \\ 0 & r(v) + \sum_{e' \in E(v)-e} m'_i(e') < b_i \\ r(v) & otherwise \end{cases} \quad (1)$$

where i is the iteration count and b_i is the threshold calculated as $b_i = \lceil d_v/2 \rceil$

$$m'_i(e) = (\sum_{e' \in E(c)-e} m_i(e'))mod2 \quad (2)$$

The VNU for GaB can be implemented using *Majority* gates (based on *and* and *or* logic functions only), and does not require complex operations such as the maximum finder in GDBF and PGDBF and the random number generator in PGDBF, which will be described in the following subsection.

B. GDBF and PGDBF VNU Analysis

Energy calculations for the GDBF and PGDBF [10] are governed by equations similar to equations (1) and (2), but they involve finding the maximum value across all VNUs in each iteration of the decoding process. The size of the maximum finder unit increases linearly with the codeword size. Additionally, PGDBF architecture, in naive implementation, includes a random number generator for each VNU. Therefore, they offer limited scalability due to their large footprints and longer critical path delays for very long code sizes.

We compare the hardware performance for our implementation of the GaB decoder with respect to GDBF, PGDBF and MinSum based on the published data [10] in Table I using Tanner code. As shown in Table 1, GaB uses the least amount of resources and operates with the fastest clock rate. In the following section we will introduce our modifications to the baseline GaB algorithm in order to improve its decoding performance.

Table I. HARDWARE RESOURCE UTILIZATION, THROUGHPUT AND CLOCK RATE OF DECODING ALGORITHMS IMPLEMENTED FOR TANNER CODE ON VIRTEX6 FPGA

Algorithm	1-bit register	Slice LUTs	Fmax (MHz)	Throughput (Mbps)
GDBF[10]	946	2151	132.7	4114.3
PGDBF[10]	9161	3545	135.6	4202.5
MinSum[10]	13694	15350	237.2	197.5
GaB	811	1265	305.3	9262.6

III. PROBABILISTIC GAB ALGORITHM

During the decoding process, the interactions between CNUs and VNUs, based on the topology defined by the H matrix, may result in an oscillation phenomena due to the n^{th} order dependencies between CNUs and VNUs. In such cases, the decoding process may get trapped in a cyclic behavior. Trapping means that the decoder cannot correct the error, and that remains in cyclic sequences of states. Unfortunately, it is not easy to detect all the trapping sets during the decoding process. One may introduce large memory to keep track of the states, but that would not be hardware friendly, since the trapping set size is unknown and there can be many thousands of different trapping sets. Therefore, we randomly disturb the state of each VNU to be able to escape from the trapping set. Of course, one may question that such disturbance could adversely affect the normal behavior of the VNU, but theoretical results indicate that this side effect does not significantly increase the number of iterations [11]. If the GaB decoder does not converge within user-defined number of (k) iterations, then we apply this probabilistic strategy (Probabilistic GaB) to escape from trapping set. We modify Equation 1 by introducing a probability function p as shown in Equation 3.

$$m_i(e) = \begin{cases} 1 & p \oplus r(v) + \sum_{e' \in E(v)-e} m'_i(e') > b_i \\ 0 & p \oplus r(v) + \sum_{e' \in E(v)-e} m'_i(e') < b_i \\ r(v) & otherwise \end{cases} \quad (3)$$

We show the architecture for PGaB in Figure 2 based on a QC-LDPC code with codeword length (N) of 1296 bits. There are six 1-bit inputs for each VNU. Four of them are from CNUs (as dv=4). The other two inputs are 1-bit data received from the channel and 1-bit random value generated by the Liner Feedback Shift Register (LFSR) based random number generator (RNG). We adjust the behavior of the RNG with a user-specified distribution. In simulation based experiments we investigate the range of p between 0.1 and 0.9 and determine that 0.8 gives the best decoding performance results for the studied code. Moreover, the probability function can be applied to the decoder in various positions. For example, in PGDBF decoder the probabilistic function is applied during the final output decision of a VNU for deciding between flipping the channel value or not. In Noisy GaB [12], the randomness effect acts on both messages exchanged mutually between VNU and CNU. Our Monte-Carlo simulations show better decoding performance, for the studied LDPC codes, when we introduce randomness as an input to the VNU function for computing only messages sent from VNU to CNU.

IV. SIMULATION SETUP AND HARDWARE DESIGN

Our simulation environment includes the GaB, PGaB, GDBF, and PGDBF implementations in C programming language. For each algorithm, FER curves are plotted based on the simulation. We evaluate LDPC codes with codeword length of 1296 bits and code rates of 0.5 and 0.75. We design two architectures for each algorithm (GaB, PGaB) over two types of LDPC codes: (a) N = 1296, dv= 4, Rate 0.5, (b) N = 1296, dv = 4, Rate 0.75.

We implement each architecture on the Xilinx Virtex-6 FPGA (vc6vlx240t-2ff1156) and conduct post placement and routing analysis over hardware cost in terms of logic and register usage, and hardware performance in terms of maximum clock rate, and throughput. Similar to other studies ([4], [10]), we calculate the system throughput using Equation 4. User throughout metric takes the useful data in the codeword into account, and has been used as an alternative metric to the system throughput. Therefore, we also calculate the user throughput based on Equation 5. We also evaluate the impact of changing the code rate on hardware performance. All designs have been implemented in VHDL. Functional verification is conducted by validating iteration by iteration post-routing

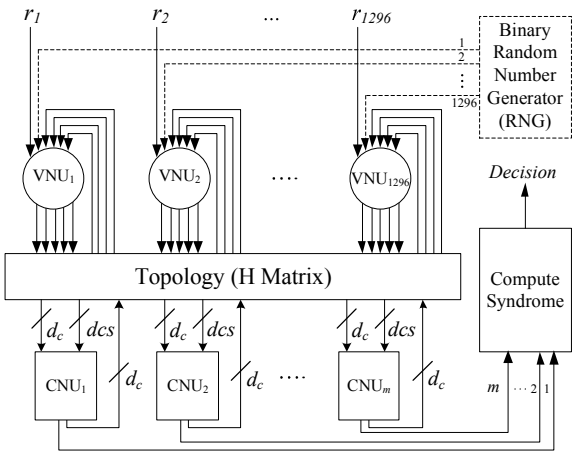


Figure 2. Our Architecture of PGaB for dv4, and N = 1296, where dc is determined by the code rate.

CNU and VNU values against the C-simulation.

$$SystemThroughput = \frac{CodeLength \times MaxClockRate}{AvgIteration \times CyclesPerIteration} \quad (4)$$

$$UserThroughput = SystemThroughput \times Rate \quad (5)$$

To the best of our knowledge there is no published work on the FPGA implementation of the GDBF and PGDBF on the LDPC codes studied in this paper. Therefore, unlike simulation based evaluations, our hardware based analysis will be limited to our GaB and PGaB implementations. In overall, simulation based experiments will help us to quantify the algorithmic contribution and hardware based analysis will help us to expose the trade-off between decoding performance and hardware cost.

V. Performance Analysis

Figure 3 shows the simulation based FER performance comparison between the GaB, PGaB, GDBF and PGDBF algorithms as a function of the cross-over probability over the binary symmetric channel (BSC) on LDPC code with the rate 0.5. This chart shows that, with the probabilistic execution, we are able to bridge the gap between the GaB and the better performing decoding algorithms. Another remarkable conclusion is our ability to perform better than the GDBF with the PGaB. The gap between PGaB and GaB in the error floor region quantifies the dramatic improvement (up to four orders of magnitude) achieved by disturbing randomly the state of the decoder.

In Table II, we present the resource usage, maximum clock rate, and throughput (system and user) for the GaB and PGaB based on their FPGA implementations with 0.5 code rate. With the probabilistic execution, we achieve dramatic improvement in the error floor over the GaB with negligible amount (0.85%) of loss in system and user throughputs. Even though modification to GaB involved including a random number generator and an additional input to each VNU, the clock rate difference between the two designs is negligible. However, the decoding performance improvement is achieved with an increase on register and slice LUT usage by 17% and 24% respectively. Register overhead of the PGaB implementation includes the 1296 bits

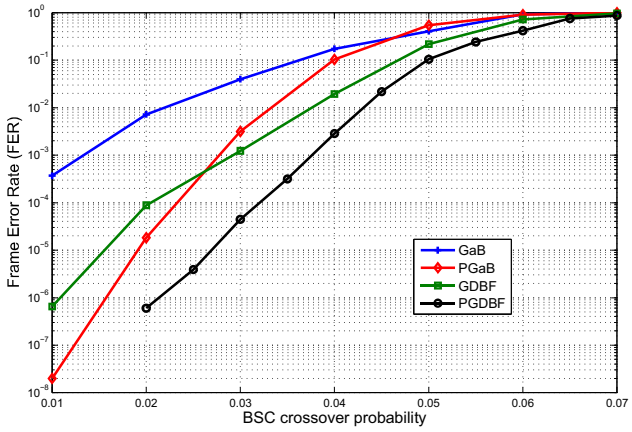


Figure 3. GaB, PGaB, GDBF, PGDBF FER comparison on LDPC code with dv= 4, dc= 8, N=1296, and Rate = 0.5

Table II. Hardware Resources and Performance for GaB and PGaB Decoders on LDPC (DV=4, DC=8, R=0.5, and N=1296)

Decoder	1-bit register	Slice LUTs	Fmax (MHz)	System Throughput (Mbps)	User Throughput (Mbps)
GaB	7812	11784	147.5	38224.5	19112.3
PGaB	9141	14605	145.2	37900.2	18950.1

to store the 1-bit random number for each VNU and the 32-bit shift register to implement the random number generator. The increase in resource usage is a reasonable tradeoff for improving the decoding performance. As shown in Table I, for even a smaller codeword, PGDBF shows an increase in registers and slice LUTs by 16% and 70% respectively compared to the GDBF. Considering that the complexity of the maximum finder unit will scale with the codeword length, the resource overhead is expected to be much higher for both GDBF and PGDBF.

We conduct an additional experiment by changing the code rate from 0.5 to 0.75 in order to evaluate its impact on both hardware and decoding performances. We implement the block diagram shown in Figure 2 on the FPGA. From hardware implementation perspective, the number of VNUs depend on the codeword length, whereas the number of CNUs depend on the code rate. In Figure 4, we compare the FER performance of the GaB and PGaB for the code rate 0.75. The PGaB consistently outperforms the GaB decoder especially in the error-floor region (more than two orders of magnitude at crossover probability of $2x10^{-3}$).

As the code rate increases from 0.5 to 0.75, based on Table II and Table III, we observe that the register and Slice LUT resources are reduced by 39% and 51% respectively for the PGaB. Similar hardware resource usage trend can be noticed

Table III. Hardware Resources and Performance for GaB and PGaB Decoders on LDPC Code With DV=4, DC=16, and R=0.75

Decoder	1-bit register	Slice LUTs	Fmax (MHz)	System Throughput (Mbps)	User Throughput (Mbps)
GaB	4596	6097	114.1	29575.6	22181.7
PGaB	5601	7133	113.7	29471.3	22103.5

for the GaB. Cost of a single CNU implementation increases with 8 additional inputs to the summation operation (XOR in Equation 2) since the degree of a CNU increases from 8 to 16. However, increasing the code rate to 0.75 reduces the number of CNUs from 648 to 324. Since the VNU operations are at and and or levels, the increase in CNU complexity is compensated by the reduction in the CNU count, which is the primary reason for reduction in the total logic block usage. Interestingly, the maximum clock rate for the new PGaB design is 113 MHz, which is 22% slower. We believe that the degree of the CNU is the primary reason for this performance loss. Placement and routing attempt to reduce the total wire length for a design by positioning the logic blocks closer and utilizing the flexibility of connection boxes and switch boxes on the FPGA to establish short paths for each net. Code rate of 0.75 results with a design that doubles the number of connections for each CNU. This in turn creates additional stress on routability, and the shorter wire segments available for the code rate of 0.5 are no longer available for the code rate of 0.75 due to congestion in regions that are densely populated with logic blocks. Therefore, nets take longer paths for routability and the critical path delay increases due to congestion. Figure 5(a) and Figure 5(b) show the layouts obtained for PGaB over the two code rates. The area expansion of the design for code rate of 0.75 is due to the router ripping up and rerouting nets through longer paths to avoid congestion. Even though the maximum clock rate is slower with the higher code rate, the user throughput shows 16% improvement over 0.5 code rate.

Based on Table II, PGaB results with an increase in 1-bit register and Slice LUT usage by 17% and 24% respectively over GaB. Similarly, from Table 3, PGaB at 0.75 code rate shows an increase in 1-bit register and Slice LUT usage by 22% and 17% respectively over GaB.

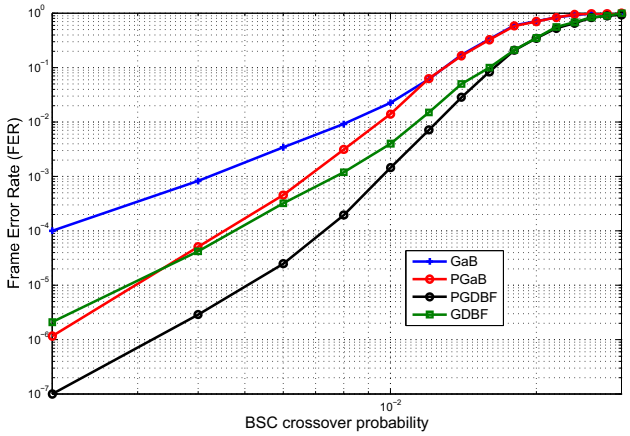
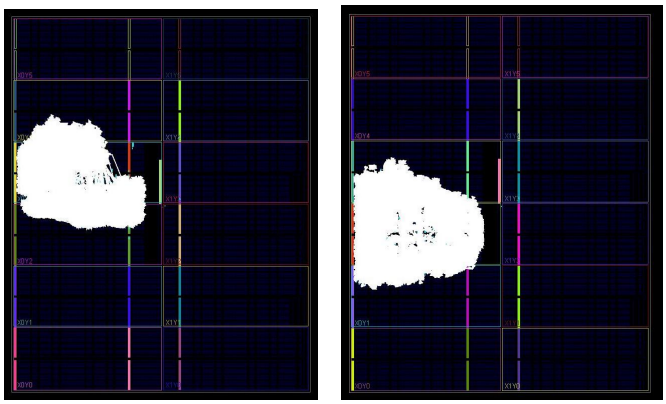


Figure 4. GaB, PGaB, GDBF, PGDBF FER comparison for Rate 0.75



(a) Code rate 0.5 (b) Code rate 0.75

Figure 5. Post-routing layout for the PGaB designs with two code rates

VI. Conclusion and Future Work

In this paper we quantified the hardware cost and performance of the GaB with probabilistic execution (PGaB decoder). We showed that without a performance loss in throughput, we improved the decoding performance of the GaB significantly and bridged the gap between GaB and other hard decision bit flipping decoding algorithms. Our simulation results showed that the PGaB now even has a better decoding performance than GDBF. In our current designs, we are generating a 1296-bit random number register. As future work, we will investigate ways to reduce this register footprint by sharing 1-bit register among a cluster of VNUs. This would reduce the size of the shift register on the datapath and have considerable impact on resource usage.

References

[1] H. D. Pfister, I. Sason, and R. Urbanke, "Capacity achieving ensembles for the binary erasure channel with bounded complexity," *IEEE Trans on Information Theory*, vol. 51, no. 7, pp. 2352–2379, July 2005.

[2] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding ldpc codes," *IEEE Trans on Comm*, vol. 58, no. 6, pp. 1610–1614, June 2010.

[3] O. Rasheed, P. Ivanis, and B. Vasic, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Comm Letters*, vol. 18, no. 9, pp. 1487–1490, September 2014.

[4] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans on Comm*, vol. 47, no. 5, pp. 673–680, May 1999.

[5] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: MIT Press, 1963.

[6] E. Boutillon, L. Conde-Canencia, and A. A. Ghouwayel, "Design of a gf(64)-ldpc decoder based on the ems algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 10, pp. 2644–2656, Oct 2013.

[7] Z. He, P. Fortier, S. Roy, and H. Xu, "An encoder/decoder with throughput over gigabits/sec for rate-compatible ldpc codes with wide code rates," in *IEEE 12th Int New Circuits and Systems (NEWCAS)*, June 2014, pp. 181–184.

[8] F. Leduc-Primeau, S. Hemati, S. Mannor, and W. J. Gross, "Dithered belief propagation decoding," *IEEE Transactions on Communications*, vol. 60, no. 8, pp. 2042 – 2047, August 2012.

[9] C. Leroux, S. Hemati, S. Mannor, and W. J. Gross, "Stochastic chase decoding of reed-solomon codes," *IEEE Communications Letters*, vol. 14, no. 9, pp. 863 – 865, September 2010.

[10] K. Le, D. Declercq, F. Ghaffari, C. Spagnol, P. Ivanis, and B. Vasic, "Efficient realization of probabilistic gradient descent bit flipping decoders," in *Proc. of the 21st 2015 IEEE Int Symp on Circuits and Systems (ISCAS)*, May 2015, pp. 1–4.

[11] B. Vasic, P. Ivanis, and D. Declercq, "Approaching maximum likelihood performance of ldpc codes by stochastic resonance in noisy iterative decoders," in *Information Theory and Applications Workshop (ITA)*, February 2016, pp. 4291–4296.

[12] B. Vasic and P. Ivanis, "Error errore eicitur: A stochastic resonance paradigm for reliable storage of information on unreliable media," *IEEE Transactions on Comm.*, vol. 64, no. 9, pp. 3596–3608, 2016.