Efficient Realization Of Probabilistic Gradient Descent Bit Flipping Decoders

Khoa Le, David Declercq
Fakhreddine Ghaffari
ETIS, ENSEA
Univ. Cergy-Pontoise
CNRS UMR-8051
95014 Cergy-Pontoise Cedex, France
{le.khoa, ghaffari}@ensea.fr

Christian Spagnol
Emmanuel Popovici
Department of Elec.
and Electronic Eng.
Univ. College Cork
email:E.Popovici@ucc.ie

Predrag Ivanis
Faculty of Elec. Eng.,
University of Belgrade,
Serbia
predrag.ivanis@etf.rs

Bane Vasic
Department of Elec.
and Comp. Eng.
University of Arizona
Tucson, AZ, 85721, USA
vasic@ece.arizona.edu

***Abstract*—In this paper, several implementations of the recently introduced PGDBF decoder for LDPC codes are proposed. In [2], the authors show that using randomness in bit-flipping decoders can greatly improve the error correction performance. In this paper, two models of random generators are proposed and compared through hardware implementation and performance simulation. A conventional implementation of the random generator through LFSR as a first design, and a new approach using binary sequences that are produced by the LDPC decoder, named IVRG, as second design. We show that both implementation of the PGDBF improve greatly the error correction performance, while maintaining the same large throughtput. However, the performance gain requires a large hardware overhead in the case of LFSR-PGDBF, while the overhead is limited to only 10% in the case of the IVRG-PGDBF.**

***Keywords*—LDPC decoders, bit-flipping, PGDBF, random generators.**

I. Introduction

Low-Density Parity-Check (LDPC) codes have been intensively studied in the past several years due to their excellent performance under iterative decoding. Their practical iterative decoders vary from Belief Propagation (BP) [6] which offers the best error correction performance, but at the cost of intensive computation, to simple hard-decision algorithms such as Bit-Flipping (BF) decoders [3][4][5]. All iterative LDPC decoders share the same general concept of passing the information between Variable Nodes (VNs) and Check Nodes (CNs). The difference between BP-based decoders and BF-based decoders lies in the computation of iteratively passed messages. Due to their simple computation units, BF algorithms significantly reduce the hardware resources needed for implementation. The drawback of this simplification is a non-negligible performance loss compared to BP and its variants Min-Sum (MS), normalized MS [1]. As a consequence, many generalization of BF algorithms have been proposed, with the objective of reducing the performance loss while keeping the hardware complexity low as in weighted BF (WBF) [3], modified weighted BF (MWBF) [4], Gradient-Descent BF [1][2] algorithms.

Gradient Descent Bit Flipping (GDBF) algorithm for binary LDPC decoders have been first proposed by Wadayama et al [1]. This algorithm is derived from gradient descent formulation and its principle consists in finding the best suitable bit (or group of bits) to be flipped in the VN processing in order to maximize a pre-defined objective function. GDBF algorithm shows error correction performance far better than other BF variants and very close to normalized MS algorithm [1]. Inspired by GDBF algorithm and the Probabilistic BF algorithm in [5], Rasheed et al. proposed another variant of GDBF called Probabilistic GDBF (PGDBF) [2] which has even better performance than the original GDBF. Instead of flipping all bits satisfying the gradient descent condition, PGDBF takes the flipping decision in a *probabilistic* manner. The results in [2] show that the randomness introduced in PGDBF makes this decoder superior to other BF decoding algorithms.

The performance improvement of PGDBF comes at the cost of extra hardware resources, since hardware-exhausted random generators blocks have to be implemented, for each and every VNs units. In this paper, we present two different hardware implementations of probabilistic GDBF with a study of their trade-offs in term of performance versus hardware resource required. A comparison with the non-probabilistic GDBF decoder is also presented. The basic difference of the two proposed PGDBF realizations comes from the implementation methods of random binary sequence generators. The first method is conventional, and makes use of linear feedback shift register (LFSR) with a modification in using different length of registers. The second method, called Intrinsic-Value Random Generator (IVRG), uses the value of Check Node (CN) units and interprets these values as a random source of bits. In IVRG, a function G is designed to obtain, from the CN outputs, random binary sequences with a controlled probability distribution. This method, to the best of our knowledge, is original and has not been proposed in the literature. By using the intrinsic values which are already generated by the existing hardware block of the GDBF decoder, the IVRG-PGDBF significantly reduces the hardware resource needed.

The paper is organized as follows. In section II, general notations on LDPC codes and decoders are recalled and a short description of the PGDBF [2] is made. We also highlight the main difference between the probabilistic GDBF and non-probabilistic GDBF which motivates the requirements of binary random generators in the hardware implementation. In section III, the two methods for the hardware design of the random generators are presented. In section IV, our proposed global architecture of the PGDBF is presented and synthesis results are produced, for different cases of the random generator use. Additionally to our IVRG-PGDBF model, we also consider the case of a partial use of random generators in the LFSR-PGDBF approach, in which the random generators are applied to only part of the VN units. Finally, in section V, we plot simulation results of the two PGDBF decoders implementation, and discuss the trade-off between error correction performance and hardware complexity. It is in particular shown that with only 9.7% of slice registers and 12.1% of slice LUTs overhead compared to the non-probabilistic PGDBF, the IVRG-PGDBF has performance results approaching the Min-Sum decoder. Section VI concludes the paper.

II. Probabilistic Gradient Descent Bit flipping

An LDPC code is defined by a sparse parity-check matrix H with size of (M,N), where $N > M$. A code word is a vector $x = (x_1, x_2, ...x_N) \in \{0,1\}^N$ which satisfies $H.x = 0$. We denote by $y = \{y_1, y_2, ..., y_N\} \in \{0,1\}^N$ the output of a binary symmetric channel (BSC), in which the bits of the transmitted codeword x have been flipped with crossover probability p_0. The decoders presented in this paper are dedicated for BSC channel. Let $\mathcal{N}(v(i))$ denotes the set of CNs connected to the VN $v(i)$, with connexion degree $d_{v(i)}$. Let also define $\mathcal{N}(c(i))$ as the set of VNs connected to the CN $c(j)$, with connexion degree $d_{c(i)}$.

In BF decoders, the value of variable nodes can change over the iterations, and we denote in this paper by $v^{(k)}(i)$ the value of the variable node at the k-th iteration. We correspondly denote by $c^{(k)}(j)$ the value of the parity checks at iteration k.

The CN calculation in BF algorithms is defined by checking whether the parity check is satisfied or not. It can be written as: $c^{(k)}(j) = \mathbf{XOR}_{v(i) \in \mathcal{N}(c(j))} v^{(k-1)}(i)$, ($\mathbf{XOR}$ is the bit-wise Exclusive-OR operation). In the case of gradient descent BF algorithms, a function called *inversion function*, is defined for each VN unit, and used to evaluate that the value $v^{(k)}(i)$ should be flipped or not.

The original GDBF is designed for the Additive White Gaussian Noise (AWGN) channel and the inversion function is defined as in (1). In GDBF [1], only the VN having the smallest inversion function's value will be flipped, and sent for the next iteration.

$$\Lambda_{v(i)}^{(k)} = (1 - 2v(i)^{(k)})\gamma_i + \sum_{c(j) \in \mathcal{N}(v(i))} (1 - 2c(j)^{(k)}) \qquad (1)$$

where γ_i is the received value from AWGN channel. In [2], the authors proposed an inversion function to apply GDBF algorithm for the Binary Symmetric Channel (BSC) [2]. The inversion function for BSC is modified, and the bits having the maximum value of $\Delta_{v(i)}^{(k)}$ in (2) are flipped.

$$\Delta_{v(i)}^{(k)} = v(i)^{(k)} y_i + \sum_{c(j) \in \mathcal{N}(v(i))} c(j)^{(k)} \quad (2) \qquad (2)$$

In [2], the inversion function's value is an integer and varies from 0 to $d_v(i)+1$. Due to the integer representation of inversion function, many bits can be flipped in one iteration. This fact may induce a negative impact to the convergence of the algorithm as the analysis of [2] shows. To avoid this effect, the PGDBF has been proposed with the idea that, instead of flipping all the bits with maximum inversion function value, only a random fraction of those bits are flipped. The random fraction is fixed by a pre-defined probability $p_i^{(k)}$, which could be different for each VN and each iteration. In this work, we restrict ourself in keeping $p_i^{(k)}$ constant for all iteration and all VNs (denoted as p' hereafter). The details of the PGDBF are explained in Algorithm 1.

For the hardware implementation, it can be seen that the non-probabilistic GDBF and PGDBF have the same structure for the CN units and for the maximum-finder. The maximum-finder is in charge of finding the maximum value of inversion functions. In this work, we follow the conventional method which uses the binary comparator tree to implement the maximum-finder. In VN units of PGDBF, extra blocks which generates sequences of random bits, denoted as $R_i^{(k)}$ in algorithm 1, are needed. Those blocks are the main difference between PGDBF and non-probabilistic GDBF and are required in order to improve the error correction performance. In [2], it is also shown that the optimum probability mass function for the random binary sequence is $p' = 0.9$. Two solutions for the analysis and design of random generators having a fixed value of p' are presented in the next section.

Algorithm 1 Probabilistic Gradient Descent Bit-Flipping

```
**Initialization** $v_i^{(init)} \leftarrow y_i, i \in [1, N]$
$syndrome = H.v^{(init)}$
**while** $syndrome \neq 0 \quad and \quad k < I_{max}$ **do**
  $\forall i \in [1, N]$
  $\Delta_{v(i)}^{(k)} = v_i^{(k)} y_i + \sum_{c_j \in \mathcal{N}(v(i))} \mathbf{XOR}_{v_u \in \mathcal{N}(c(j))} v_u^{(k)},$
  $Threshold = max(\Delta_{v(i)}^{(k)})$
  **if** $\Delta_{v(i)}^{(k)} = Threshold$ **then**
    **if** $R_i^{(k)} = 1$ **then**
      $x_i^{(k)} = \mathbf{NOT}(v_i^{(k)}); \{p(R_i^{(k)} = 1) = p_i^{(k)}\}$
    **end if**
  **end if**
  $syndrome = H.v^{(k)}$
  $k = k + 1$
**end while**
**Outputs :** $v^{(k)}$
```

III. Analysis and design of random binary generators

A. LFSR random generator

The first design that we study is based on linear feedback shift registers, with controlled probability of getting zero or ones, that we consider for inclusion is each instantiated VNU. The generic architecture for the random binary sequence generator is not presented in the paper because of the lack of space, but is briefly described thereafter. We make use of LFSR with maximum length feedback polynomial to generate an integer number, and the generated number is compared with a threshold to decide if the new bit in the random sequence should be a 0 (higher than threshold) or 1 (lower than threshold). Two aspects are of interest when designing a variable threshold random binary sequence generator, first the period of the random sequence, second the granularity with which the threshold can be programmed. In the proposed architecture the period depends on the length of the LFSRs and the granularity from the number of LFSRs (i.e. bits) implemented. The chosen parameters guarantee a granularity of 2^{-8}. Moreover, having different length for each LFSR ensures a higher period than the period of the longest LFSR. In particular the least common multiple of all the periods $lcm([2^l - 1])$ with $l \in \{3 \ldots 10\}$ is around 17 billions, which ensures that the binary sequence will appear random to the decoding process. Having different length LFSRs also reduces the total number of register required.

B. Intrinsic-value random generator

An alternative solution is presented in this section that reduces the cost of generating random binary sequences by means of subtituting all local RBSGs (one per VNU) with a global one. We name this new method *intrinsic-value random generator* (IVRG), which makes use of the value of the CNs inside decoder as its inputs. In an LDPC iterative decoder, the values of the CNs depend both on the BSC crossover probability of p_0, the degree of check nodes d_c and the iteration number. Typically, the number of CN which are unsatisfied (value '1') is large during the first iterations, while it becomes smaller as the iteration number increases. We denote by $p(c_j^{(k)} = 1) = F(p_0, k, d_c)$ the probability that a CN is unsatisfied, as a function of the three mentioned parameters.

In this paper, we will use only the CN values produced at the first iteration $k = 1$ in order to generate sequences of random bits. At the first iteration, the probability mass function is given by $p(c_j^{(1)} = 1) = F(p_0, 1, d_c) = \frac{1}{2} - \frac{1}{2}(1 - 2p_0)^{d_c}$. Figure 1 shows the probability $p(c_j^{(1)} = 1)$ versus p_0 when the CN degree d_c changes. In order to control the random generator probability p', we propose to use a function G of the CN values $G(c_{j1}^{(0)}, c_{j2}^{(0)}, ..), j1, j2 \in [1, M]$ that controls the desired probability p'. We briefly describe the function G in the following.

Let c_{j1} and c_{j2} be two binary random variables with $p(c_{j1} = 1) = p(c_{j2} = 1) = p$, it can be proved that $p(c_{j1}\mathbf{OR}c_{j2} = 1) = 2p + p^2 > p$ and $p(c_{j1}\mathbf{AND}c_{j2} = 1) = p^2 < p$. More specifically, $p(c_{j1} = 1) = p(c_{j2} = 1) = p = \frac{1}{2} - \frac{1}{2}(1 - 2p_0)^{\mathbf{d_c}}$, $p(c_{j1}\mathbf{XOR}c_{j2} = 1) = \frac{1}{2} - \frac{1}{2}(1 - 2p_0)^{\mathbf{2d_c}}$. Using these transformations of probability, and a function G implemented as described in figure 2, we can transform the CN output sequence into a longer binary pseudo-random sequence with a desired probability p'. The CNs values at first iteration are stored in the chain of Flip-Flops and are cyclically shifted at each iteration and assigned to be the inputs of the input-selectable-OR gates through an interconnexion network in order to ensure randomness of the output sequence. The value of crossover probability triggers the selectable-OR gates, in order to control the value of p'. The more precision is put on p_0, the finer is the control on p', but at the cost of larger hardware resource.

This IVRG has been realized and verified, for the case of p_0 that triggers the IVRG output stored on 2 bits. As a result, in running time of the PGDBF decoder, the probability of the IVRG output is not exactly tuned along the iterations and varies around the target value $0.88 < p' < 0.92$.

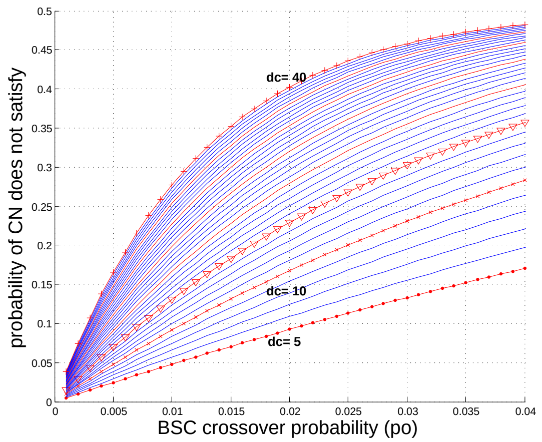

Fig. 1. Statistics of the CN values as a function of the BSC crossover probability.

IV. Global Architecture of Probabilistic GDBF

The top level architecture of the decoder is presented in Figure 3. This architecture differs from the generic LDPC decoder architecture in several aspects. First, the presence of a global block that takes inputs from all VNUs (the *Lambdas*) and computes the maximum, and second the presence of binary random generators. The LFSR approach can be seen as a distributed random generator due to the fact that it is implemented inside every VN unit.

The complexity of the interconnection network depends on the type and size of LDPC code used as well as the chosen level of parallelism.

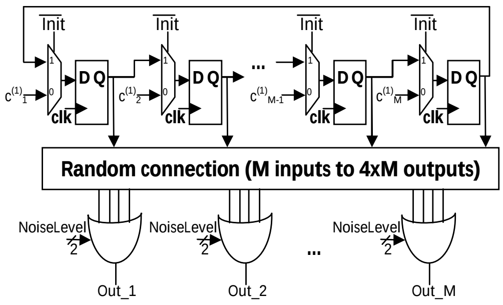

Fig. 2. Architecture of Intrinsic-valued Random Generator

These aspects have been widely discussed in the literature [7] and are not discussed here. Implementation of the RG have been discussed in the section III.

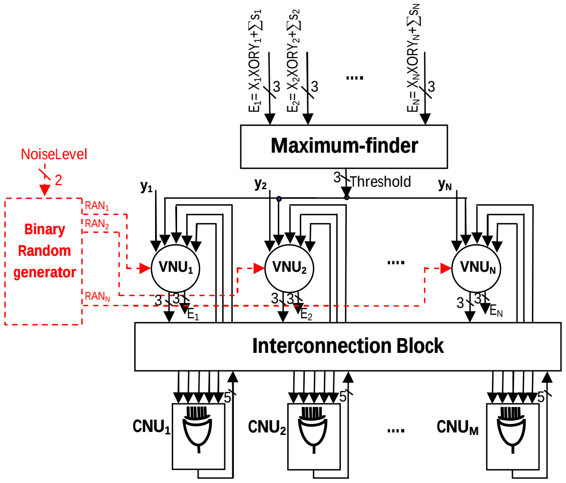

Fig. 3. Global architecture of PGDBF compared to the original GDBF

We have made the synthesis of the different solutions for the case of a small LDPC code that has been proposed in the literature [8]: a regular, quasi-cyclic LDPC code with regular connexion degrees $d_v = 3$ and $d_c = 5$, with codeword length $N = 155$, called the *Tanner code*. Table I shows the hardware resources needed to implement the two different PGDBF structures. As benchmarks, the resources for the non-probabilistic GDBF and 6 bits Min-Sum decoder are shown as well. The maximum frequency and the estimated throughput have been obtained from an implementation using FPGA Xilinx virtex 6 of 40nm technology, after place and route. For the throughput calculation, we use the following definition: $Throughput = f_{max} * N/(I_{aver} * S)$ where f_{max}, I_{aver}, S are respectively the maximum frequency, the average iteration number and the number of clock cycles needed for one iteration. We obtained $S = 1$ for the PGDBF algorithms and $S = 10$ for the offset Min-Sum.

The IVRG-PGDBF needs an additional 92 1-bit registers (9.7% overhead) compared to the non-probabilistic while the LFSR-PGDBF needs 8215 1-bit registers overhead (868.4% overhead). This large overhead emphasize the advantage of IVRG over LFSR in terms of implementation. Comparing the Slice LUTs required, the IVRG-PGDBF requires 261 more slices than the non-probabilistic (12.1%)

and this number for LFSR-PGDBF is 1394 (64.8%). The extra complexity brought by the RG implementation has moreover a negligible impact on the obtained throughput (less than 2%) in all PGDBF impementations. We can also see that the offset min-sum decoder is far more complex than the BF type decoders, and cannot compete in terms of decoding speed.

TABLE I. HARDWARE AND THROUGHPUT ESTIMATION FOR PGDBF WITH DIFFERENT RG IMPLEMENTATIONS AND FOR OFFSET MIN-SUM

	1-bit Register	Slice LUTs	F_{max} (MHz)	Throughput (Mbps)
Non-Probabilistic GDBF	946	2151	132.721	4114.3
PGDBF with IVRG	1038	2412	132.721	4114.3
PGDBF with LFSR	9161	3545	135.56	4202.36
offset min-sum (6 bits)	13694	15350	237.185	197.5

In order to reduce the hardware resources used in the LFSR-PGDBF we propose to apply the RG only in a subset of the VN, and not everywhere. The reason for this study is that putting RGs in all the variable units might not be necessary in order to obtain good decoding results, especially in the case of Quasi-cyclic LDPC codes. We report in table II the synthesis results for different fractions of VNs using LSFR-RG, from 0% (non-probabilistic) to 100%. As expected, the complexity in registers and slices grows linearly with the number of LSFR-RG considered. Even with only 20% of VNs incorporating the LSFR-RG, the complexity is larger than the one of the IVRG approach (see table I). The performance results of these different cases are reported in the next section.

TABLE II. HARDWARE AND THROUGHPUT ESTIMATION FOR PGDBF WITH DIFFERENT NUMBER OF LFSR IN PGDBF

	1-bit Register	Slice LUTs	F_{max} (MHz)	Throughput (Mbps)
Non-Probabilistic GDBF	946	2151	132.721	4114.3
PGDBF with LFSR in 20% VNs	2589	2429	133.886	4150.466
PGDBF with LFSR in 40% VNs	4232	2708	134.426	4167.206
PGDBF with LFSR in 60% VNs	5875	2987	132.117	4095.627
PGDBF with LFSR in 80% VNs	7518	3266	134.21	4160.51
PGDBF with LFSR in 100% VNs	9161	3545	135.56	4202.36

V. NUMERICAL RESULTS

Figure 4 shows the frame error rate of the PGDBF algorithms and non-probabilistic GDBF as a function of the channel crossover probability. The two solutions proposed in this paper for PGDBF algorithms produces a significant gain in performance comparing to non-probabilistic GDBF. The IVRG-PGDBF shows a performance loss in the error floor region (flattening) compared to the LFSR-PGDBF. However, the performance in the waterfall region are strictly similar. This performance loss in the error floor could come from the correlation induced by the imprecise random generation implemented with IVRG. We will continue the characterization of the IVRG approach in future papers. We can also notice that the partial use of LFSR (20%) is not sufficient to obtain good results. As a conclusion the IVRG-PGDBF appears as the best performance vs. complexity trade-off for the implementation of the PGDBF decoder. As expected, hard decision decoder are still far from soft-decision ones (min-sum), as can be seen in figure 4.

VI. CONCLUSION

In this paper, several implementations of the PGDBF decoder for LDPC codes have been proposed. A conventional implementation of the random generator through LFSR as a first design, and a new approach using binary sequences that are produced by the LDPC decoder, named IVRG, as second design. We showed that both implementation of the PGDBF improve greatly the error correction performance compared to the non-probabilistic version, while maintaining the same large throughtput. However, the performance gain requires a large hardware overhead in the case of LFSR-PGDBF, while the overhead is limited to only 10% in the case of the IVRG-PGDBF, which appears then as a promising solution for very high throughtput application.

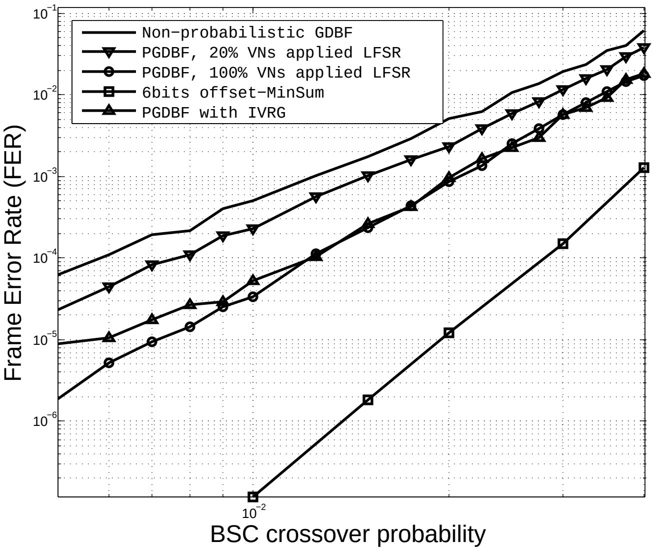

Fig. 4. FER performance comparison of the different decoders on the (N=155,K=64) Tanner code

ACKNOWLEDGMENT

the authors wish to thank Marc Aysu from ENSICAEN, France, who wrote early version of the LSFR random generators. This work was supported by the Seventh Framework Programme of the European Union, under Grant Agreement number 309129 (i-RISC project).

REFERENCES

[1] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami and I. Takumi, “Gradient descent bit flipping algorithms for decoding LDPC codes”, IEEE Transactions on Communications, vol. 58, no. 6, pp. 1610–1614, June 2010.

[2] O.A. Rasheed, P. Ivanis and B. Vasic, “Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoder”, Communications Letters, IEEE, vol. 18, no. 9, pp. 1487–1490, September 2014.

[3] J. Zhang, M.P.C. Fossorier, “A modified weighted bit-flipping decoding of low-density Parity-check codes”, IEEE Communications Letters, vol. 8, no. 3, pp. 165–167, March 2004.

[4] M. Jiang, C. Zhao, Z. Shi and Y. Chen, “An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes”, IEEE Communications Letters, vol 9, no 9, pp. 814–816, September 2005.

[5] N. Miladinovic and M.P.C. Fossorier, “Improved bit-flipping decoding of low-density parity-check codes”, IEEE Transactions on Information Theory, vol. 51, no. 4, pp. 1594–1606, April 2005.

[6] T. Richardson and R. Urbanke, “Modern Coding Theory”, Cambridge University Press, 2008.

[7] A. Darabiha, A.C. Carusone, F.R. Kschischang, “Block-Interlaced LDPC Decoders With Reduced Interconnect Complexity,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 55, no. 1, pp. 74–78, Jan. 2008.

[8] M. Tanner, D. Srkdhara, and T. Fuja, “A class of group-structured LDPC codes,” 2001. [Online]. Available: citeseer.ist.psu.edu/tanner01class.html