Low Complexity, Flexible LDPC Decoders

Federico Quaglio
Dipartimento di Elettronica
Politecnico di Torino – Italy
Email: federico.quaglio@polito.it

Fabrizio Vacca
Dipartimento di Elettronica
Politecnico di Torino – Italy
Email: fabrizio.vacca@polito.it

Guido Masera
Dipartimento di Elettronica
Politecnico di Torino – Italy
Email: guido.masera@polito.it

***Abstract*—The design and implementation of a flexible LDPC decoder able to cope with different codes, is gathering an increasing interest in the scientific community. Roughly speaking, a flexible decoder exhibits the remarkable ability of being able to decode different codes resorting to the same hardware. In this paper we present a novel decoder architecture specifically devised to present an high flexibility and to being able to properly trade-off decoding performances with reconfigurability and overall system complexity.**

I. Introduction

Low Density Parity-Check (LDPC) Codes [1], are a class of powerful linear block codes able to approach Shannon AWGN channel capacity bound, [2]–[4], with very impressive decoding throughputs [5], [6]. These two peculiarities have significantly contributed to make LDPC codes among one of the most promising candidates for next generation communication standards (e.g. DVB-S2, IEEE 802.16e ...).

LDPC decoders are usually represented as bipartite graphs, also called *Tanner Graph*, where two class of Processing Elements (PEs) are mapped to two classes of graph's vertices. These processing elements are usually referred as *Bit Nodes* (BNs) and *Check Nodes* (CNs) and are strictly related to the $N \times M$ parity check matrix $\mathbf{H}$ associated with the code. Edges of the Tanner Graph represent interconnections between the processing elements. BNs are related to the N columns of $\mathbf{H}$ matrix and represent the "code-block" of the specified LDPC code. The M CNs (which correspond with the rows of $\mathbf{H}$) represent the M parity-check equation of the code. Given two distinct processing elements BN$_j$ and CN$_i$, they are interconnected through an edge on the Tanner graph only if $\mathbf{H}(i,j) = 1$. This explains how the edges on Tanner graph are a sort of "pictorial" representation of the "1s" in $\mathbf{H}$.

As for *Turbo Codes*, [7], the decoding process is applied iteratively sending messages back and forth from BNs to CNs (and vice-versa) updating the received soft-information at each PE. To avoid probability multiplications, decoders work in logarithmic domain with "Log-Likelihood Ratio (LLR)" λ rather than the received bit leading to the "Sum-Product" implementation of the more general "Belief Propagation" algorithm. According to the extrinsic information principle [7]–[9], BNs update the LLR values with the information received from the CNs. This process is expressed in equation (1) where k represents the current iteration, Q_{ji} is the message being sent from BN$_j$ to CN$_i$, $R_{i'j}$ is a incoming message (produced by CN$_{i'}$ and directed to BN$_j$ while $C[j]$ is the whole set of incoming messages for BN$_j$.

$$Q_{ji}[k] = \lambda_j + \sum_{i' \in C[j] \setminus \{i\}} R_{i'j}[k-1] \tag{1}$$

On the other hand, Check Nodes perform the parity check on the updated values. In a significant number of implementations, this parity check is performed applying iteratively the so-called Ω function defined as in equation (2).

$$\Omega(I_0, I_1) = \ln\left(\frac{e^{I_0} + e_1^I}{1 + e^{I_0 + I_1}}\right) \tag{2}$$

In (2), with I_0 and I_1 we refer to two incoming messages, from the BNs, or a message and the result from a previous Ω evaluation. Ω function is usually preferred due to its "numeric stability" since it is a limited function. Moreover, it can be easily approximated taking the minimum of the two operands corrected with two logarithmic terms stored in two Look-Up Tables (LUTs) as in eq. (3), [9]–[11].

$$\begin{aligned} I_0 \oplus I_1 &= -\mathrm{sgn}(I_0)\mathrm{sgn}(I_1)\min(|I_0|, |I_1|) \\ &+ \ln(1 + e^{-|I_0 - I_1|}) - \ln(1 + e^{-|I_0 + I_1|}) \end{aligned} \tag{3}$$

To achieve better performances, "parallel" implementation of the CNs (able to evaluate all the outgoing messages at the same time) are desirable. Nevertheless, this will lead to high complex structures.

Moreover, according to Tanner Graph representation, lots of interconnections between Bit & Check Nodes are involved in exchanging informations. In fact, considering the average edge-degree of the BNs as be and with ce the correspondent edge-count for CNs, there are barely $N \times be = M \times ce$ messages that have to be delivered. Thus, interconnection structures become predominant over computation blocks.

Under these premises, it is clear how any actual implementation of an LDPC decoder have to cope with two main issues. On one hand the "numeric stability" of CN update equations can be achieved at the expenses of an increased computational complexity. On the other hand, the interconnections' complexity tend to explode as the code–dimensions (N and M) increase. This consideration is even more severe if one considers how practical LDPC codes often consist of thousands BNs and CNs: efficient message exchanging structures and techniques ought to be investigated in order to mitigate the total amount of area penalties due to "wiring".

In this paper we present an analysis for the design of a new class of LDPC decoders able to be retargetable to different codes, with a reduced amount of interconnections and low CN complexity. The paper is organized as follows: section II describes the proposed interconnection structure able to cope with different codes. Section III presents some consideration to design low-complexity decoder analyzing the trade-offs between performances and complexity. Finally in section IV some conclusions are drawn.

II. Novel Flexible Decoder Implementation

LDPC decoders able to be retargetable to different codes represent an attractive solution since they can address different systems without the need for a change in hardware. Although a fully parallel decoder implementation (e.g. [5]) can not achieve the aforementioned flexibility since the entire solution is built around the specific code it addresses.

Starting from this statement one can be attracted by serial implementations. A serial implementation of the decoder is composed by a single BN and a single CN that emulates all the processing elements of the associated Tanner graph. The interconnection structure is simulated by an interleaving memory between the BN and CN. Such a structure exhibits, without any doubts, an high flexibility and reconfigurability since there is no theoretical limit to the number of BNs or CNs to be simulated. However it severely suffers under the throughput standpoint, which usually tends to be too poor for practical implementations. Beside the poor throughput, serial LDPC decoders are known to be prone to the memory *collision* phenomena. Different memory locations are likely to be accessed at the same time in the same memory bank when storing or reading an updated message.

Thus, a reasonable trade-off between flexibility and throughput is obtainable by *partially-parallel* decoders where only a subset of the whole decoder PEs are instantiated. Proper hardware support and data organization must be added in order to complete the decoding process on the whole codes in a certain number of cycles. Moreover some memory resources are required to store the partial results decoding: unfortunately the collision problem may occur also for these solutions.

Two main different approaches have been proposed in literature to cope with collisions:

1) To design "hardware-aware" codes (collision free codes).
2) To design a decoder architecture capable to avoid collisions.

Codes of the first class are properly designed to lead to decoders where messages will never collide into memories. Moreover, the final decoder can achieve high throughputs, [6], [12], [13]. On the other hand these decoder works only on specific classes of structured LDPC codes but not on all possible realizations.

Decoders of the second class rely on costly routing networks (i.e. crossbar switches) to avoid collision for generally defined codes, [14]. On the other hand, there is still a lack of hardware implementation for this class of decoders. In [15] a novel hardware architecture able to avoid collisions is presented. This architecture relies on circular buffer, called *ring interleaver*, where messages are stored only when a collision occurs. This means, that in case of collision, some memory access are delayed until collisions are resolved. Due to the intrinsic properties of LDPC codes and proper design of the circular buffer depth, collision problem is mitigated with very little penalties.

In this work, we present a novel flexible decoder implementation able to work on generally defined codes trying to reduce the total amount of interconnections. In particular rearranging eq. (1) we are able to reduce the amount of the exchanged messages requiring less routing resources. Analyzing equation (1) it can be inferred that message $Q_{ji}[k]$ can be computed adding all the incoming messages $R[k-1]$ and the received LLR and then subtracting the message received on the same edge ij. From (1) we obtain (4).

$$\begin{aligned} Q_{ji}[k] &= \lambda_j + \sum_{l \in C[j]} R_{lj}[k-1] - R_{ij}[k-1] \\ &= S_j - R_{ij}[k-1] \end{aligned} \tag{4}$$

If each CN stores locally the messages sent to the BNs, each BN can be alleviated of the final subtraction to compute the extrinsic informations and could send to all the CNs only the global sum S_j. Instead of sending be messages (where be is the number of outgoing edges from the generic bit node), each BN could send only a broadcast message (S_j) to all the CNs.

Through a similar consideration, it is possible to modify also the CN computation. In order to easily deal with CN, it is better to rely on an alternative representation of CNs' parity check equation as in (5).

$$R_{ij}[k] = \psi^{-1}\left[\sum_{\tilde{j} \in R[i] \setminus \{j\}} \psi(Q_{\tilde{j}i}[k])\right] \cdot \delta_{ij} \tag{5}$$

In eq (5), R_{ij} are the outgoing message from CN$_i$ to BN$_j$, Q_{ji} is a generic incoming message from CNs, $R[i]$ is the set of the whole incoming messages and $\delta_{ij} = \prod \operatorname{sgn}(Q)$ is the product of the signs of the incoming messages from the BNs. Moreover $\psi = \psi^{-1}$ is a non-linear function as in equation (6).

$$\psi = -\ln\left(\tanh\left|\frac{x}{2}\right|\right) \tag{6}$$

With this representation, equation (5) can be rearranged as in (7). In figure 1 an 8-edge check node (according to eq (7)) is depicted.

$$\begin{aligned} R_{ij}[k] &= \psi^{-1}\left[\sum_{\tilde{j} \in R[i] \setminus \{j\}} \psi(Q_{\tilde{j}i}[k])\right] \cdot \delta_{ij} \\ &= \psi^{-1}\left[\sum_{l \in R[i]} \psi(Q_{li}[k]) - \psi(Q_{ji}[k])\right] \cdot \delta_{ij} \\ &= \psi^{-1}\left[PS_i - \psi(Q_{ji}[k])\right] \cdot \delta_{ij} \end{aligned} \tag{7}$$

Analyzing (7), it can be noted that if also BNs store local copies of the messages (more precisely a $\psi(Q_{ji})$ version of the messages) also each check node could send a broadcasted message PS_i rather than ce (where ce is the number of outgoing edge from a generic CN).

This implies little modification both for bit nodes and check nodes. Subtraction should be added to locally computed extrinsic information at the input of each PEs. Additionally the BN hardware block need to be modified to compute ψ function. In the end, two new architectures for BN and CN have to be designed obtaining *Modified BN* (MBN) and *Modified Check Node* (MCN). In particular if MBN$_1$ received broadcasted values PS_2 and PS_3, it should compute the new summation S_1 according to (8).

$$\begin{aligned} R_{21}[k-1] &= \psi(PS_2 - \psi(Q_{12}[k-1])) \\ R_{31}[k-1] &= \psi(PS_3 - \psi(Q_{13}[k-1])) \\ S_1 &= \lambda_j + R_{21}[k-1] + R_{31}[k-1] \\ Q_{12}[k] &= S_1 - R_{21}[k-1] \\ Q_{13}[k] &= S_1 - R_{31}[k-1] \end{aligned} \quad (8)$$

In (8), Q_{12} and Q_{13} are the extrinsic value locally stored at iteration $k-1$ and k.

Similarly, if MCN$_1$ receives the broadcasted sums P_2 and P_3 it should compute PS_1 and the local messages R_{21} and R_{31} according to (9).

$$\begin{aligned} Q_{12}[k] &= S_2 - R_{21}[k-1] \\ Q_{13}[k] &= S_3 - R_{31}[k-1] \\ PS_1 &= \psi(Q_{12}[k]) + \psi(Q_{13}[k]) \\ R_{21}[k] &= \psi(PS_1 - \psi(Q_{12}[k])) \\ R_{31}[k] &= \psi(PS_1 - \psi(Q_{13}[k])) \end{aligned} \quad (9)$$

In figure 2, an 8-edge MCN is depicted.

With respect to eq. (4) and (7), MBN and MCN both require at least k (where k is the number of incoming edge) more subtractions (see equations 8,9 and fig. 2) implying an increase in area and little degradation in maximum operating frequency. In table I we give a preliminary estimation of the area increase MBN and MCN compared to ordinary BN and CN. In these evaluations we consider only the total amount of adder/subtracter neglecting the area increments due to ψ function blocks and registers to store local messages.

These modification are general and are completely independent from the overall decoder architecture (serial, fully-parallel or partially-parallel). Each of the possible implementation can receive beneficial improvements using MBN and MCN rather than "classical" BN and CN. For example, supposing to have a flexible serial LDPC decoder able to cope codes with up to $N = 1024$ BNs and $M = 512$ CNs where each PE has 8 incoming edge ($be = ce = 8$). With simple BNs and CNs, to avoid collision in the interleaving memory each PE requires up to 8 read cycles and 8 write cycles. This means that $16 \times (1024 + 512)$ clock cycles (supposing embedded RAM memory) are required just for accessing memory in 1 iteration. With MBNs and MCNs, only 2 cycles (1 for reading and 1 for writing) are needed for each PE, requiring only $2 \times (1024 + 512)$ memory access with an $8\times$ improvements. This means that to achieve the same throughput MBN and MCN could work at a lower frequency or conversely achieving higher throughputs when working at the same frequency of BNs and CNs.

Based on this reformulated algorithm a novel partially-parallel architecture for LDPC decoder can be derived. The main idea behind this implementation is to map directly a portion of the **H** matrix onto an array of programmable processing elements. This idea is illustrated in fig 3 where a portion of the decoder with 3 MBN and 3 MCN are depicted.

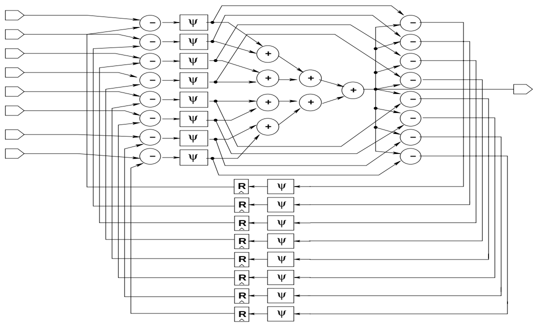

Fig. 2. 8-edge Modified Check Node

TABLE I
AREA INCREASE IN MBN & MCN

Architecture	Area [Add/Sub]	Increase [%]
BN	$2 \cdot N$	–
MBN	$3 \cdot N$	+50%
CN	$2 \cdot N - 1$	–
MBN	$3 \cdot N - 1$	$\sim$ +50%

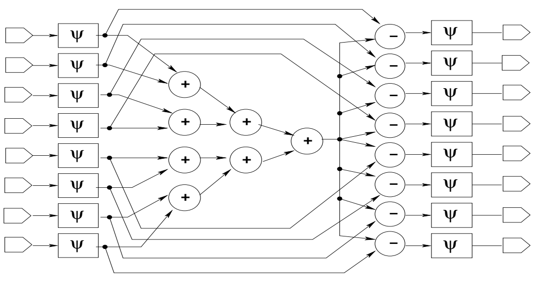

Fig. 1. 8-edge CN architecture (the sign correction part has been omitted for the sake of clearness).

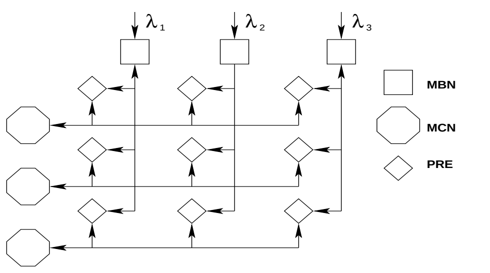

Fig. 3. Flexible Partially-Parallel Decoder

Each MBN and MCN send broadcasted messages and a routing network of *Programmable Routing Elements* (PRE) dispatches these messages to the proper destination. Decoding process, at each iteration, is organized as follow:

1) MBNs update messages and send broadcast message S_j along columns (1 cycle)
2) S_j are stored in PRE corresponding to 1s in the **H** matrix. These values are then dispatched to MCNs (be cycles)
3) MCNs update incoming informations and send broadcast message PS_i along rows (1 cycle)
4) PS_i value are stored in PRE corresponding to 1s in the **H** matrix and then send to the proper MBNs (ce cycles)

This solution requires $be + ce + 2$ cycles to complete an iteration, but if p multiple busses outgoing from PRE (either for rows and columns) are allocated, then an overall improvements in latency approximately of the same factor is possible.

With this approach different codes are allowed since PREs are programmed to dispatch messages only when they are correspondent to a 1 in the **H** matrix. Moreover only a subset of this matrix (N' MBNs and M' MCN) can be actually allocated; then changing the values stored into PRE the whole **H** matrix can be "scanned" to compute the entire code. Preliminary synthesis test showed a PRE node complexity of few hundred gates. Mapped on a Xilinx Virtex2 FPGA a 64×64 routing matrix. can reach operating frequencies higher than 200 MHz. This means that any 1024-bits LDPC code with rate 1/2 and $be = 5$ and $ce = 10$ can be decoded, with the aforementioned matrix, with a throughput of 512 Mbps per iteration. Better performances could be achieved relying on standard cell technology rather than FPGA.

III. Novel ψ function Piece-Wise Linear Approximation

The MBNs and MCNs have been derived from direct manipulation of equations (1), (5) and they both need computation of ψ function as was shown in (8) and (9). This non-linear function can be easily implemented in hardware as LUT, [16], but suffers from "numerical instability" problems when it is represented over finite precision fixed point notation due to the vertical asymptote when its argument is zero.

On the other hand, for a 2-input CN, eq. (5) can be rearranged to obtain Ω function, eq (2), or its Min_Star representation as in eq. (3). Unfortunately, Min_Star (Min*) operator cannot be combined in order to obtain the *inverse operator* needed if we want to "sum" up all the incoming messages and doing a final "subtraction" (or better a "Min_Star^{-1}" function) similarly as what is done with ψ function in equation (7). In [11], authors inverted eq. (2) and then they were able to approximate it as in [9] to obtain an inverse operator similar to the Min* one. This solution, nevertheless, contains two logarithmic terms $\ln(1 - e^{-|x|})$ that are not limited leading the same problem of numerical instability as for the direct implementation of ψ function. Moreover both Min* operator and this approximation suffers from high hardware complexity and low operating frequency.

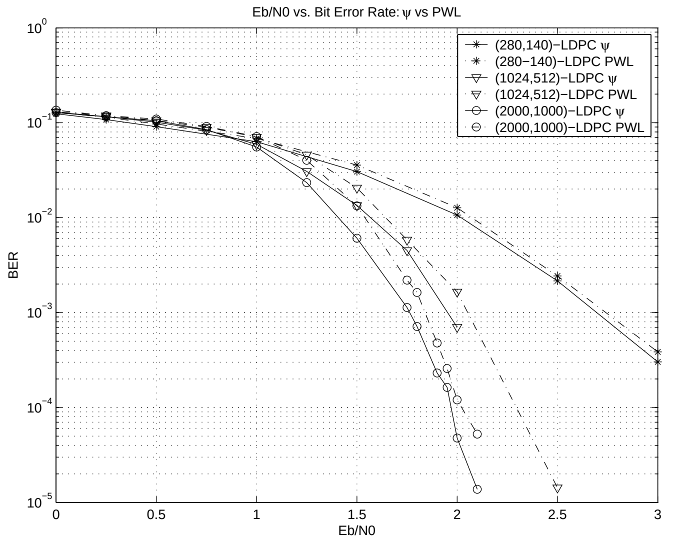

Fig. 4. Bit Error Rate vs. Eb/N$_0$ Different Approx.

Starting from these considerations, it can be easily inferred that a flexible decoder cannot rely on Min* operator even if it performs better than ψ when fixed point is dealt with. We tried to define a novel approximation for the ψ operator in order to implement MBN and MCN, with little degradation in error correction performances, but saving in area and improvements in operating frequency. For these reasons we defined a Piece-Wise Linear (PWL) approximation designed according to two main principles. Firstly, it has to be as accurate as possible with respect to the original ψ function. Thus more linear pieces has been allocated where ψ is less linear. Secondly, coefficients were chosen in order to be easily represented in fixed point notation, trying to avoid multiplications. This lead to the final PWL approximation reported in (10).

$$PWL = \begin{cases} -48x + 7.9375 & iff & |x| < 0.125 \\ -7.5x + 3.875 & iff & |x| \leq 0.25 \\ -2x + 2.5625 & iff & |x| < 0.75 \\ -x + 1.75 & iff & |x| \leq 1 \\ -0.5x + 1.25 & iff & |x| \leq 2 \\ -0.125x + 0.5 & iff & |x| \leq 2.8125 \\ -0.0625x + 0.3125 & iff & |x| \leq 3.75 \\ -0.0625 & iff & |x| \leq 6.0625 \\ 0 & otherwise & \end{cases} \tag{10}$$

The coefficients in (10) were chosen to be easily obtained with "shift", or sum of shifted data (i.e $48x = 32x + 16x$) of the input value to improve hardware implementation. The resulting coefficients were also fine tuned to obtain reasonable decoding performances. In figure 4 we depicted the Bit Error Rate (BER) vs. Eb/N_0 curves for some rate 1/2 LDPC codes obtained with ψ function and the same results for our PWL approximation. These results were obtained by means of Radford Neal's LDPC simulator, [17]. Further investigation are undergoing to estimate PWL behavior when data are represented in fixed point notation.

In figure 5 a block scheme of our novel PWL approximation is depicted. We first calculate the reciprocal value of the input (if it is already negative, no modification occur), then the obtained values are properly shifted according to (10). Actually all the shifts are performed in parallel and then the proper one is selected to the results of the comparisons (also comparisons are performed in parallel). Then a final addition is required; to improve its performances a fast Han & Carlson carry-lookahead-adder [18] was implemented. In

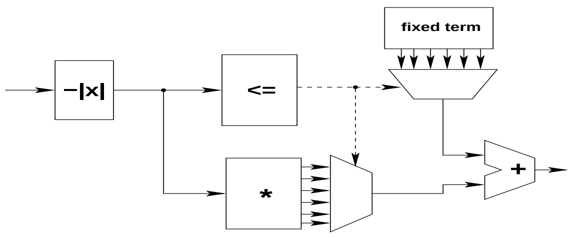

Fig. 5. Block Scheme of PWL circuit

table II some preliminary synthesis results (on a 0.13 μm standard cell technology) are presented compared with a LUT implementation. Our approach requires little more area and run little slower, but it is less "sensible" to data representation. Thus, it is a good compromise respect to error correcting capabilities, decoder complexity and operating frequencies.

TABLE II
PWL ψ VS. LUT-BASED - 0.13 μM -

Architecture	Max. Freq. [MHz]	Area [μm^2]	Cells
PWL	1020.41	3279.96	254
LUT	1111.11	1855.82	153

In table III we present synthesis results for an 8-incoming edge Modified Check Node compared with an implementations based on fig. 1 and on Min* blocks. As can be seen our implementation overpass the "classical" both in terms of maximum frequency and complexity with little degradation in correction capabilities. When working at the same frequencies it was noticed that MCN has an area increase of approximately 50%, according to table I, respect to classical CN.

IV. CONCLUSION

In this paper a novel flexible architecture for generally defined LDPC codes is presented. The presented architecture is able to successfully decode a broad variety of different codes without significantly constrain the LDPC code itself. Additionally it seems very well suited to being employed in actual reconfigurable decoders.

The desired flexibility has been reached through a "smart" rearrangement of BN and CN functionalities in order to reduce the number of interconnections. Additionally, a novel PWL approximation of ψ function is derived in order to trade-off complexity and decoding performances. Some preliminary decoding performances and hardware synthesis results are also reported to validate our approach.

TABLE III
MCN VS MIN* CN COMPARISONS - ST 0.13 μM -

Arch.	Freq. [MHz]	Area [μm^2]	Cells	ΔF	ΔA
CN	307.69	51109.80	2635	0%	0%
MCN	264.55	59208.86	3050	-14.02%	+15.84%
Min* CN	101.52	60481.71	3106	-67%	+18%

ACKNOWLEDGMENT

This work was partially supported by FIRB PRIMO and Newcom

REFERENCES

[1] R. G. Gallager, "Low Density Parity Check Codes," *IRE Trans. Information Theory*, vol. IT-8, no. 1, pp. 21–28, 1962.
[2] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, 1997.
[3] D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.
[4] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of Capacity-Approching Irregular Low-Density Parity-Check Codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.
[5] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, Rate 1/2 Low-Density Parity-Check Code Decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.
[6] M. M. Mansour and N. R. Shanbhag, "High Throughput LDPC Decoders," *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.
[7] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo Codes (1)," in *Proc. IEEE Int. Conf. Communications (ICC'93)*, May 1993, pp. 1064–1070.
[8] S. Benedetto and E. Biglieri, *Principles of Digital Transmission With Wireless Applications*. New York, USA: Kluwer Academic / Plenum Publisher, 1999.
[9] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.
[10] F. Guilloud, E. Boutillon, and J. Danger, "λ-Min Decoding Algorithm of Regular and Irregular LDPC Codes," in *Proc. 3rd Int. Symp. on Turbo Codes & Related Topics*, Brest, France, Sept. 2003.
[11] T. Clevorn and P. Vary, "Low-Complexity Belief Propagation Decoding by Approximations with Looup-Tables," in *Proc. 5th Int. ITG Conference on Source and Channel Coding (SCC 2004)*, Erlangen, Germany, Jan. 2004.
[12] T. Zhang and K. K. Parhi, "A 54 Mbps (3,6)-regular FPGA LDPC decoder," in *Proc. IEEE Workshop on Signal Processing Systems 2002 (SIPS '02)*, Oct. 2002, pp. 127–132.
[13] D. E. Hocevar, "LDPC code construction with flexible hardware implementation," in *Proc. IEEE Int. Conf. on Communications 2003, (ICC 2003)*, vol. 4, Anchoarage, USA, May 2003, pp. 2708–2712.
[14] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaver laws to parallel turbo and LDPC decoders architectures," *IEEE Trans. Inform. Theory*, vol. 50, no. 9, pp. 2002–2009, Sept. 2004.
[15] F. Kienle, M. J. Thul, and N. Wehn, "Implemntation Issue of Scalable LDPC-Decoders," in *Proc. 3rd Int. Symp. on Turbo Codes & Related Topics*, Brest, France, Sept. 2003, pp. 291–294.
[16] E. Yeo, P. Pakzad, B. Nicolic, and V. Anantharam, "High Throughput Low-Density Parity-Check Decoders Architectures," in *Proc. IEEE Global Telecommunications Conference, 2001. GLOBECOM '01*, vol. 5, Nov. 2001, pp. 3019–3024.
[17] R. Neal. (2001, Nov.) LDPC CoDec C Code package. Release of 2001-11-18. [Online]. Available: http://www.cs.toronto.edu/~radford/ldpc.software.html
[18] T. Han and D. A. Carlson, "Fast area-efficient VLSI adder," in *Proc. 8th Symp. on Comp. Arithmetic*, May 1987.