Area, Throughput, and Energy-Efficiency Trade-offs in the VLSI Implementation of LDPC Decoders

C. Roth*, A. Cevrero‡, C. Studer*, Y. Leblebici‡, and A. Burg‡

*Dept. of Information Technology and Electrical Engineering, ETH Zürich, 8092 Zürich, Switzerland
e-mail: rothc@iis.ee.ethz.ch; studerc@nari.ee.ethz.ch
‡School of Engineering, EPF Lausanne, 1015 Lausanne, Switzerland
e-mail: {alessandro.cevrero, yusuf.leblebici, andreas.burg}@epfl.ch

***Abstract*—Low-density parity-check (LDPC) codes are key ingredients for improving reliability of modern communication systems and storage devices. On the implementation side however, the design of energy-efficient and high-speed LDPC decoders with a sufficient degree of reconfigurability to meet the flexibility demands of recent standards remains challenging. This survey paper provides an overview of the state-of-the-art in the design of LDPC decoders using digital integrated circuits. To this end, we summarize available algorithms and characterize the design space. We analyze the different architectures and their connection to different codes and requirements. The advantages and disadvantages of the various choices are illustrated by comparing state-of-the-art LDPC decoder designs.**

I. Introduction

Digital communication and storage systems rely on channel coding to ensure reliable transmission and to guarantee data integrity. Besides Turbo codes, low-density parity-check (LDPC) codes [1], are among the best performing codes known today. While initially considered too complex for economic implementation, LDPC codes have been rediscovered [2] and turned out to be most suitable for implementation in modern CMOS technologies. Due to this implementation advantage, their excellent error-correction performance, and the favorable IP-licensing situation, LDPC codes are gradually replacing other well-established forward error-correction schemes.

A. LDPC Codes

Binary LDPC codes are defined through an $M \times N$ binary-valued sparse parity-check matrix $\mathbf{H}$, whose columns are associated with coded bits and rows describe the parity-check equations. A convenient representation of LDPC codes are bipartite graphs [3] (Tanner graphs) in which variable nodes (VNs) are associated with code bits and check nodes (CNs) with parity-check equations. In this graph, a VN and a CN are connected when the corresponding entry in $\mathbf{H}$ is one. The dimension and structure of $\mathbf{H}$ define the block size N, the code rate and the performance of the code, as well as the complexity of the decoding process.

B. Decoding Algorithm and Complexity

LDPC codes are commonly decoded using iterative message-passing algorithms which improve the initial estimates of the bits initializing the VNs by sending messages along the edges of the graph in an iterative manner (see Sec. II for details). The computational effort for decoding a single code block depends roughly linearly on the number of edges in the graph and on the number of iterations performed to achieve a given target error-rate. Normalizing this effort with the number of information bits contained in a code block yields the number of *processed edges per information bit*, which is a first-order approximation for the computational effort required by the decoder for a specific LDPC code. In Fig. 1, we compare the decoding complexity for prominent applications relying on LDPC codes. To this end, we plot the computational effort (assuming 10 iterations) against the throughput requirement given in information bits per second for the various operation modes. The diagonal lines in the double-logarithmic scale are representative for the overall complexity required by a corresponding decoder.

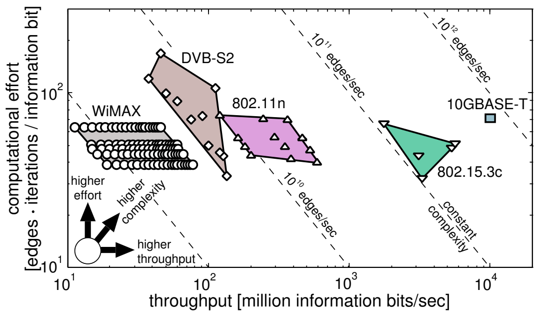

Fig. 1. Computational effort and throughput overview of typical wired and wireless communication standards employing LDPC codes.

It is interesting to see that the computational effort required per information bit remains approximately the same for all considered standards. However, the throughput requirements across standards vary by more than three orders of magnitude, whereas the throughput within a standard may also vary by one order of magnitude. Furthermore, we observe that wireless communication standards foresee a wide range of operation modes with similar—but not equal—complexity. These modes (covering different throughputs) must usually be supported by a *single flexible* decoder implementation. What is not directly visible from Fig. 1 is that some applications have a very tight energy-efficiency constraint as they target battery-powered devices.

C. Contributions and Outline

In this survey paper, we review and investigate the trade-offs associated with the design of digital LDPC decoders. To this end, Sec. II briefly summarizes the essentials and preferred choices with respect to the underlying decoding algorithms. Sec. III introduces the prototype architecture from which all other architectures are derived and describes our performance metrics to compare different state-of-the-art designs. In Sec. IV we partition the design-space into three different architecture classes. We describe similarities and differences between these architectures and highlight solutions for the main implementation challenges with reference to corresponding designs in the literature.

II. Algorithm Aspects for LDPC Decoding

Decoding of LDPC codes essentially involves three different types of values that are being updated in an iterative manner: Prior to decoding, L-values (associated to VNs) are initialized by reliability information (log-likelihood ratios) generated by the demodulator. In each iteration, Q-messages are passed from the VNs to the CNs and R-messages are sent back from the CNs to the VNs and L-values are updated. After a certain number of iterations, estimates for the information bits are computed.

A. Message-Passing Schedule

The message-passing schedule determines the order of updating the L-values, Q-, and R-messages in each iteration. With the conventional *flooding schedule* [2], each iteration comprises of a two-step update procedure: i) all VNs compute Q-messages based on their L-values and R-messages (received in the previous iteration) and send them to the connected CNs; ii) all CNs compute new R-messages based on the incoming Q-messages and pass them back to the VNs.

A superior approach to the flooding schedule is the *layered schedule* [4], which partitions the computations in each iteration into several layers, corresponding to distinct subsets of rows of $\mathbf{H}$. For each layer, only the CNs corresponding to the current layer receive Q-messages and pass their R-messages back to the VNs, which then update both, the L-values and Q-messages. Then, the next layer is processed similarly. The advantage of this schedule is that information gained in each layer will then be considered in the processing of the subsequent CNs. With proper layer selection, this schedule generally leads to faster convergence, eventually reducing the number of iterations roughly by a factor of two [4].

B. Message-Update Rules

The optimum message-update rules for Q- and R-messages and L-values are given by the *sum-product algorithm* (SPA) [2]. Unfortunately, these rules involve transcendental functions and are therefore ill-suited for VLSI implementation. A straightforward approximation to the SPA is known as the *min-sum* (MS) algorithm [5]. Here, the transcendental functions are replaced by minimum operations. The resulting complexity reduction, however, comes at the cost of a considerable loss in terms of error-rate, which can be mitigated almost entirely by either using the *normalized min-sum* (NMS) or the *offset-min-sum* (OMS) algorithm [6]. Further advantages of OMS and NMS result from their *value-reuse* properties [7], which reduces the number of minimum computations without additionally penalizing the error-rate performance.

C. Early Termination

Early termination (ET) essentially avoids redundant iterations by stopping the decoding process when further iterations are unlikely to alter the result. From an energy-efficiency perspective, ET is highly desirable since the required energy per bit grows proportionally in the number of iterations. The most prominent method is to stop decoding, whenever *all* parity-check equations are satisfied, based on the current estimates of the transmitted bits; this ensures that decoding is terminated whenever a valid code-word has been found. This approach can be implemented very efficiently for flooding schedules, but it is more involved for layered decoding [8], [9].

III. General Architecture and Performance Metrics

As illustrated in Sec. I, the architecture's portfolio for LDPC decoders must span three orders of magnitude in terms of throughput requirements, different levels of importance of energy-efficiency, and various needs for reconfigurability.

A. Prototype Architecture

From a high-level perspective, almost all implementations of message-passing LDPC decoders start from an *isomorphic architecture* [10] which is a direct mapping of the Tanner graph to three types of hardware components: VN units (VNUs) and CN units (CNUs) to compute the update equations, an interconnect network representing the edges of the graph, and storage devices for the L-values and R-messages. Starting from this prototype architecture, different implementation trade-offs are obtained through architectural

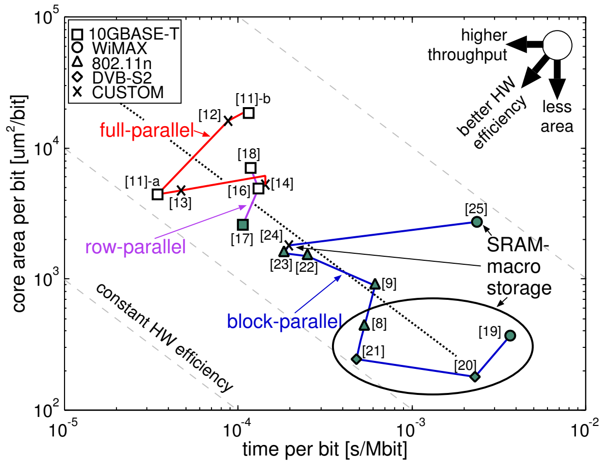

Fig. 2. Area vs. time per bit trade-offs.

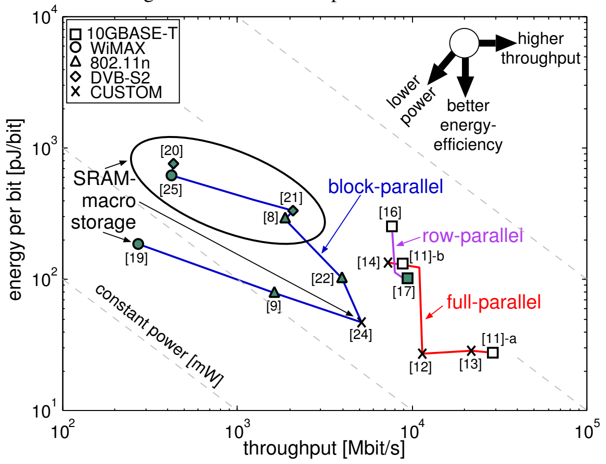

Fig. 3. Energy-efficiency vs. throughput trade-offs.

transformations such as *resource sharing* across VNUs and CNU and *iterative decomposition* of the update equations within the VNUs and CNUs together with algorithm-specific customizations to tailor the isomorphic architecture to individual application requirements. In Sec. IV we shall provide more details on the specific architectures and the associated trade-offs.

B. Performance Metrics

Besides the throughput and error-rate requirements prescribed by each standard, area- and energy-efficiency are the main performance criteria, where the importance of energy-efficiency clearly depends on the specific application. In order to evaluate the impact of architectural choices for LDPC decoding on silicon area and energy-efficiency, we collect data-points from state-of-the-art implementations highlighting designs that correspond to recent standards (cf. Fig 1). For a fair comparison of the architectural features we take technology scaling into account, i.e., we scale the area, throughput, and energy-efficiency of all designs to 90 nm CMOS assuming 1.0 V supply voltage. We use 5 and 10 iterations for architectures based on layered (filled markers) and flooding schedules, respectively, to account for the difference in convergence speed. Furthermore, we assume that all collected designs use appropriate word lengths for Q- and R-messages to satisfy the error-rate requirements prescribed by the respective standard. Fig. 2 relates the *area per bit* to the time required for decoding. The diagonal lines correspond to the area-time-per-bit (AT) product which is a common measure for hardware-efficiency. Fig. 3 relates the *energy per bit* to the maximum achievable throughput for each design. The diagonals on the double-logarithmic scale correspond to constant power consumption. In both figures, area and energy are normalized by the block length N and the throughput is measured at the input of the decoder to avoid dependencies on the code rate (which has negligible impact on the computational effort).

IV. Area and Energy-Efficiency Trade-offs

In the following, we illustrate how the design space can be divided into three different classes: *Full-parallel*, *row-parallel*, and *block-parallel* architectures (see Fig. 4)

A. Full-Parallel Architectures

Fully parallel LDPC decoder designs [11]–[14] were among the first high-throughput implementations of LDPC decoders. They have been considered for custom codes and for high-speed applications such as 10GBASE-T 10 Gb/s Ethernet. Corresponding implementations rely on the flooding schedule and directly implement the isomorphic prototype architecture of the message-passing algorithm.

1) Architecture: A high-level block diagram of a typical full-parallel design is shown in Fig 4.a). The update equations are mapped into individual VNUs and CNUs that exchange messages through a *hard-wired routing network*. The parallel processing allows each iteration to be performed in a single clock cycle in which VNUs fetch all R-messages from local (register-based) storage and compute new Q-messages that are sent through the routing network. The CNUs compute new R-messages and send them back to the VNUs where they are stored for the next iteration and are used to update the local L-value registers [12].

2) Discussion: The inherent advantage of full-parallel architectures is the ability to fully exploit the intrinsic parallelism offered by LDPC codes; this enables very high throughput since one iteration is performed per clock cycle with simple computations that allow for high clock frequencies. This performance advantage is clearly visible in Fig. 2, where designs that are based on isomorphic architectures achieve the highest throughput. Unfortunately, the complex routing network that connects CNUs and VNUs turns out to be a major implementation bottleneck for these designs [11] and straightforward implementations exhibit extremely poor area utilization and suffer from a considerable speed penalty that may even become worse when proceeding to more advanced process technologies. Proposals to mitigate this *routing bottleneck* include the serialization of multi-bit interconnect busses [14], and the rearrangement of operations between the CNs and the VNs or guided placement to reduce the number or the length of global wires [15]. A particularly interesting approach is a modification on the algorithmic level that reduces the amount of routing at the expense of a small error-rate performance degradation [11]. The impact of this modification on hardware- and energy-efficiency is clearly visible from Fig. 2 and Fig. 3, where the design with the modified algorithm [11]-(a) achieves much better results than the accompanying reference design without algorithm modifications [11]-(b).

B. Row-Parallel Architectures

Row parallel architectures [16]–[18] are a step towards less parallelism by means of resource sharing [10] across the CNUs. The objective is to reduce active silicon area and to partially alleviate the routing bottleneck while maintaining very high throughput.

1) Architecture: The architectural principle underlying such *row-parallel* architectures is depicted in Fig 4.b). In essence, the parity-check matrix is partitioned vertically into groups of parity-check equations (*layers*). The equations in each layer are executed in parallel on a subset of time-shared CNUs. An iteration now consists of multiple cycles in which the VNUs access the R-messages corresponding to the current layer sequentially from a small storage array (typically too small for economic use of SRAMs) to compute Q-messages and communicate them to the CNUs through a *programmable routing network*. Due to the sequential processing of the CNUs, the update of the L-values in the VNUs can by performed in an iteratively decomposed fashion[1], while the computation of R-messages in the CNUs is still carried out in a single cycle on a purely combinatorial network.

2) Discussion: Considering the available reference designs in Fig. 2, it is interesting to see that row-parallel architectures are still sufficiently fast to meet even the 10 Gb/s throughput requirements of high-speed wireline communication standards (e.g., 10GBASE-T). At the same time, they provide an area advantage over full-parallel designs which is partially due to a smaller number of global wires and partially due to architectural/layout measures that exploit regularities in the code [16] or circuit-level measures that reuse routing resources [17]. A key advantage of the row-parallel architectures is that they are naturally suited for a layered schedule which allows for fewer iterations (exploitable for area reduction through less stringent timing constraints). Note that additional storage to hold the L-values computed in the previous iteration, while processing layers of the current iteration, can be avoided for a layered schedule. This difference in area is visible from Fig. 2 when comparing [16] and [17].

C. Block-Parallel Architectures

The two previous architectures provide very high throughput at the cost of area and without flexibility. They are therefore ill-suited for many wireless communication standards (e.g., WiMAX, IEEE 802.11n, and DVB-S2) that require support for different code rates and block lengths at moderate to low throughput. *Block-parallel architectures* [8], [9], [19]–[25] are tailored to such applications. They rely on further resource sharing and iterative decomposition together with structured codes that facilitate reconfigurability.

1) Architecture: Fig 4.c) outlines the architectural principle, which is usually used in conjunction with the layered schedule. In essence, this architecture is obtained by starting from the row-parallel approach and by partitioning the computation of a layer further into multiple cycles, corresponding to multiple blocks. This iterative decomposition simplifies the CN processing and allows for resource sharing also across the VNUs, which are now tightly coupled to the CNUs to form a node computation unit (NCU). Due to the reduced parallel access requirements, R-messages and L-values can now be stored in area-efficient macro cells. However, a *programmable routing network* is required that connects the L-value storage array to the NCUs since in each layer a different NCU is associated with a particular L-value. The complexity and the amount of bits required to control the routing network depend on the structure of the code and on its partitioning into layers. In the worst-case, a full crossbar is needed. However, most relevant standards rely on structured codes such as *quasi-cyclic* LDPC codes [25] for which the programmable interconnect can be realized by area-efficient programmable cyclic shifters [8].

2) Discussion: Fig. 2 clearly illustrates how the block-parallel architectures cover the lower-throughput region of the design space, but it also shows that recent designs are approaching the multiple-Gb/s range [22], [23]. This is achieved by increasing the number of blocks that are processed in parallel, either within a layer or across layers. The trajectory across [8], [22], [23] shows that this transition is possible at an almost constant AT-product.

In terms of energy-efficiency, Fig. 3 reveals a disadvantage of the block-parallel architectures compared to the full-parallel and row-

[1]The degree of iterative decomposition depends on the choice of the layers and on the code. Ideally, the degree of each column of a layer should be one, which can be achieved for most of the relevant codes.

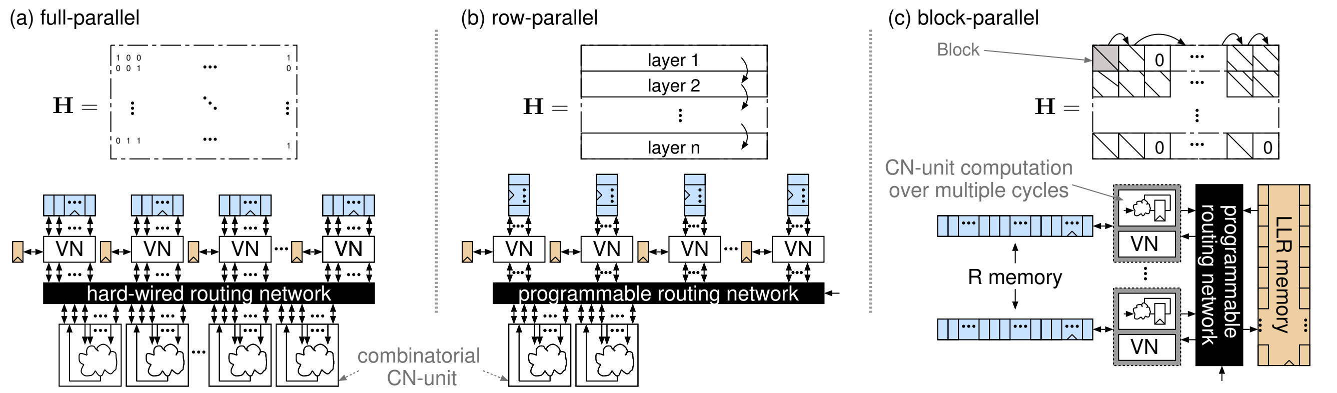
(a) full-parallel
(b) row-parallel
(c) block-parallel
H =
layer 1
layer 2
layer n
Block
0
VN
hard-wired routing network
programmable routing network
combinatorial CN-unit
CN-unit computation over multiple cycles
R memory
programmable routing network
LLR memory

Fig. 4. Parity-check matrix and corresponding high-level block diagram for LDPC-decoder implementations with different amount of resource sharing.

parallel designs. This trend can be explained by the higher degree of control and flexibility required after extensive resource sharing and iterative decomposition. Another trend that is visible from Fig. 2 and Fig. 3 is the split between two types of block-parallel designs. The first type exploits the ability of the architecture to use SRAM macro-cells to reduce core area, but suffers from poor energy-efficiency. The second type relies on standard-cell based storage at the cost of silicon area to achieve better results in terms of energy-efficiency (e.g., [9]).

V. Conclusion

State-of-the-art implementations of LDPC decoders use message passing with the normalized or offset min-sum approximation, often with a layered schedule. On the architectural side, non-programmable parallel architectures are still required to meet throughput requirements in excess of 10 Gb/s. For wireless communication applications, more serial architectures (which also approach the multiple Gb/s range) are used together with mostly quasi-cyclic codes. Interestingly, for codes with similar computational effort, most of the different architectures exhibit almost the same AT-product indicating the suitability of the message-passing algorithm for the realization of very different area/delay trade-offs.

Acknowledgment

This work was kindly supported by the Swiss National Science Foundation.

References

[1] R. G. Gallager, "Low density parity check codes," *Trans. IRE Prof. Group on Inf. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.
[2] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.
[3] H.-A. Loeliger, "An introduction to factor graphs," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 28–41, Jan. 2004.
[4] E. Sharon, S. Litsyn, and J. Goldberger, "An efficient message-passing schedule for LDPC decoding," in *Proc. 23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, Sep. 2004, pp. 223–226.
[5] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.
[6] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 7, pp. 1288–1299, Aug. 2005.
[7] K. K. Gunnam, G. S. Choi, W. Wang, and M. B. Yeary, "Multi-rate layered decoder architecture for block LDPC codes of the IEEE 802.11n wireless standard," in *Proc. IEEE Int. Symp. on Circuits and Systems*, May 2007, pp. 1645–1648.
[8] C. Studer, N. Preyss, C. Roth, and A. Burg, "Configurable high-throughput decoder architecture for quasi-cyclic LDPC codes," in *Proc. 42nd Asilomar Conf. on Signals, Systems and Computers*, Oct. 2008, pp. 1137–1142.
[9] C. Roth, P. Meinerzhagen, C. Studer, and A. Burg, "A 15.8 pJ/bit/iter quasi-cyclic LDPC decoder for IEEE 802.11n in 90 nm CMOS," in *Proc. IEEE Asian Solid-State Circuits Conf.*, Nov. 2010, pp. 313–316.
[10] H. Kaeslin, *Digital Integrated Circuit Design*. Cambridge University Press, 2008.
[11] T. Mohsenin, D. N. Truong, and B. M. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders," *IEEE Trans. Circuits Syst. I*, vol. 57, no. 5, pp. 1048–1061, May 2010.
[12] 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.
[13] N. Onizawa, T. Hanyu, and V. C. Gaudet, "Design of high-throughput fully parallel LDPC decoders based on wire partitioning," *IEEE Trans. VLSI Syst.*, vol. 18, no. 3, pp. 482–489, 2010.
[14] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A 3.3-Gbps bit-serial block-interlaced min-sum LDPC decoder in 0.13-μm CMOS," in *Proc. Custom Integrated Circuits Conf.*, Sep. 2007, pp. 459–462.
[15] A. Darabiha, A. C. Carusone, and F. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *IEEE Trans. Circuits Syst. II*, vol. 55, no. 1, pp. 74–78, Jan. 2008.
[16] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors," *IEEE J. Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, Apr. 2010.
[17] A. Cevrero, Y. Leblebici, P. Ienne, and A. Burg, "A 5.35 mm^2 10GBASE-T Ethernet LDPC decoder chip in 90 nm CMOS," in *Proc. IEEE Asian Solid-State Circuits Conf.*, Nov. 2010, pp. 317–320.
[18] L. Liu and C.-J. Shi, "Sliced message passing: high throughput overlapped decoding of high-rate low-density parity-check codes," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 11, pp. 3697–3710, Dec. 2008.
[19] T.-C. Kuo and A. Willson, "A flexible decoder IC for WiMAX QC-LDPC codes," in *Proc. IEEE Custom Integrated Circuits Conf.*, Sep. 2008, pp. 527–530.
[20] P. Urard, L. Paumier, V. Heinrich, N. Raina, and N. Chawla, "A 360mW 105Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes enabling satellite-transmission portable devices," in *Digest Techn. Papers IEEE Solid-State Circuits Conf.*, Feb. 2008, pp. 310–311.
[21] P. Urard, E. Yeo, L. Paumier, P. Georgelin, T. Michel, V. Lebars, E. Lantreibecq, and B. Gupta, "A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes," in *Digest Techn. Papers IEEE Solid-State Circuits Conf.*, vol. 1, Feb. 2005, pp. 446–609.
[22] Y. Sun and J. R. Cavallaro, "A low-power 1-Gbps reconfigurable LDPC decoder design for multiple 4G wireless standards," in *Proc. IEEE Int'l. SOC Conf.*, Sep. 2008, pp. 367–370.
[23] Y. Sun, G. Wang, and J. R. Cavallaro, "Multi-layer parallel decoding algorithm and VLSI architecture for quasi-cyclic LDPC codes," in *Proc. IEEE Int. Symp. on Circuits and Systems*, May 2011, to appear.
[24] M. M. Mansour and N. R. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, Mar. 2006.
[25] C.-H. Liu, S.-W. Yen, C.-L. Chen, H.-C. Chang, C.-Y. Lee, Y.-S. Hsu, and S.-J. Jou, "An LDPC decoder chip based on self-routing network for IEEE 802.16e applications," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 684–694, Mar. 2008.