A Parallelized Layered QC-LDPC Decoder for IEEE 802.11ad

Alexios Balatsoukas-Stimming*, Nicholas Preyss*, Alessandro Cevrero*, Andreas Burg*, Christoph Roth†
*Dept. of Electrical Engineering, EPFL, Lausanne, Switzerland.
†Integrated Systems Laboratory, ETHZ, Zürich, Switzerland.
Email: {alexios.balatsoukas, nicholas.preyss, alessandro.cevrero, andreas.burg}@epfl.ch, rothc@iis.ee.ethz.ch

***Abstract*—We present a doubly parallelized layered quasi-cyclic low-density parity-check decoder for the emerging IEEE 802.11ad multi-gigabit wireless standard. The decoding algorithm is equivalent to a non-parallelized layered decoder and, thus, retains its favorable convergence characteristics, which are known to be superior to those of flooding schedule based decoders. The proposed architecture was synthesized using a TSMC 40 nm CMOS technology, resulting in a cell area of 0.18 mm^2 and a clock frequency of 850 MHz. At this clock frequency, the decoder achieves a coded throughput of 3.12 Gbps, thus meeting the throughput requirements when using both the mandatory BPSK modulation and the optional QPSK modulation.**

I. Introduction

The emerging IEEE 802.11ad standard [1] aims to provide multi-gigabit wireless connectivity over relatively short distances. It promises to do so by exploiting the large amount of bandwidth available in the 60 GHz unlicensed band. A mandatory mode with a coded throughput of 1.54 Gbps and two optional modes with coded throughputs of 3.08 and 6.16 Gbps are defined, using BPSK, QPSK and 16-QAM modulations, respectively. The standard employs quasi-cyclic low-density parity-check (QC-LDPC) codes of rates 1/2, 5/8, 3/4, and 13/16, with a codeword length of 672 bits.

LDPC codes [2] are powerful capacity-approaching channel codes, which have found their way into most modern communications standards due to their exceptional error-correcting performance and the existence of highly efficient decoding algorithms. Nevertheless, the design of low power and high throughput LDPC decoders remains a challenging task. The key towards achieving multi-gigabit throughputs is parallelization. To this end, [3] presents a multi-gigabit decoder with fully parallelized variable nodes. This decoder uses the flooding schedule, which is known to have slower convergence than the layered schedule. Moreover, in [4], a *multi-layered* decoder for IEEE 802.11ad is presented, which effectively uses a hybrid between a layered and a flooding schedule. In both cases, parallelization has a negative effect on convergence speed.

Contribution and Outline: In this paper, we present a low-area layered QC-LDPC decoder specifically targeted at the codes defined by the IEEE 802.11ad standard. Our architecture adds a level of parallelism to the reference architecture of Studer *et al.* [5] by carefully splitting the parity-check matrix of the codes into parts, without any sacrifice in terms of convergence speed.

The rest of the paper is structured as follows. In Section II, we provide background on LDPC codes and the layered offset min-sum (L-OMS) decoding algorithm. Section III describes the proposed decoder architecture and some throughput enhancing optimizations. Finally, in Section IV we present our synthesis results and provide a comparison to existing work. Section V concludes the paper.

II. Background

A. QC-LDPC Codes

An LDPC code $\mathcal{C}$ is defined through its $m \times n$ sparse binary parity-check matrix $\mathbf{H}$ as

$$\mathcal{C} = \{\mathbf{c} \in \{0,1\}^n : \mathbf{Hc} = \mathbf{0}\}, \tag{1}$$

where additions are performed modulo-2 and $\mathbf{0}$ denotes the all-zeros vector of length m. A graphical representation of the code can be obtained if we regard the parity-check matrix as an adjacency matrix for a bipartite graph. This graph, which is called a *Tanner* graph, contains nodes of two types, namely variable nodes and check nodes. A variable node j is connected to a check i if, and only if, $\mathbf{H}_{ij} = 1$.

QC-LDPC codes [6] have structured parity-check matrices, which enable the design of efficient encoders and decoders. The parity-check matrix of a QC-LDPC code is an $M \times N$ block matrix defined as

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^{\alpha_{11}} & \mathbf{P}^{\alpha_{12}} & \dots & \mathbf{P}^{\alpha_{1N}} \\ \mathbf{P}^{\alpha_{21}} & \mathbf{P}^{\alpha_{22}} & \dots & \mathbf{P}^{\alpha_{2N}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}^{\alpha_{M1}} & \mathbf{P}^{\alpha_{M2}} & \dots & \mathbf{P}^{\alpha_{MN}} \end{bmatrix}, \tag{2}$$

where $\mathbf{P}$ is a $Z \times Z$ identity matrix which has been cyclically right shifted by one position, and $\alpha_{ij} \in \{1, \dots, Z-1\} \cup \infty$, with the conventions that $\mathbf{P}^0 = \mathbf{I}_{Z\times Z}$ and $\mathbf{P}^\infty = \mathbf{0}_{Z\times Z}$. Additionally, $n = ZN$ and $m = ZM$.

The parity-check matrices of the QC-LDPC codes defined in IEEE 802.11ad use blocks of size $Z = 42$. Furthermore, they have $N = 16$ block columns. The number of rows of each code defines the corresponding rate.

B. Layered Decoding

LDPC codes can be efficiently decoded by means of message-passing algorithms, which exchange messages between the variable nodes and the check nodes. Traditional LDPC decoding algorithms, such as the sum-product (SP) [7] and the min-sum (MS) [8] algorithms and its variants, such as offset MS (OMS), use a flooding schedule. This schedule first computes all messages from the variable nodes towards the check nodes and then computes the messages from the check nodes back to the variable nodes. However, it has been shown that the alternative *layered* schedule [9] can lead to a significant reduction of the number of iterations required to achieve a target bit error rate (BER) when compared with the flooding schedule, thus having the potential to provide significant energy savings.

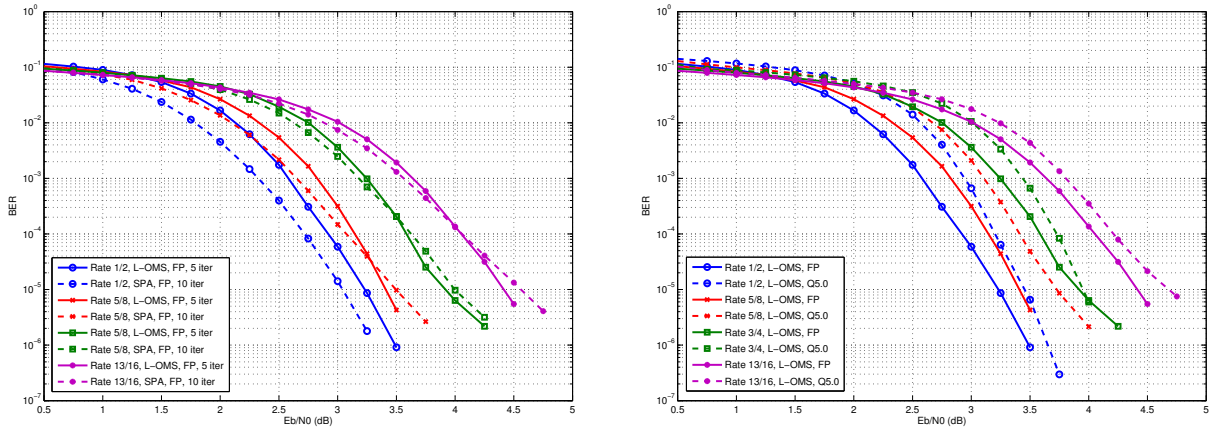
In the layered schedule, first all the messages flowing into and out of the first layer (i.e., check node) are calculated. Then, the messages flowing into and out of the second layer are calculated, possibly using the information that has already been updated by the first layer, etc. More formally, let Q_i denote the outgoing message at variable node i and let $R_{j,i}$ denote the corresponding incoming message from layer j. When processing layer j, the layered OMS (L-OMS) algorithm calculates:

$$T_i \leftarrow Q_i^{\text{old}} - R_{j,i}^{\text{old}} \tag{3}$$

$$R_{j,i}^{\text{new}} \leftarrow \max\left(0, \min_{k\in\mathcal{N}_j/i} |T_k| - \beta\right) \prod_{k\in\mathcal{N}_j/i} \text{sign}\left(T_k\right), \tag{4}$$

$$Q_i^{\text{new}} \leftarrow T_i + R_{j,i}^{\text{new}}, \tag{5}$$

for every $i \in \mathcal{N}_j$, where $\mathcal{N}_j/i$ denotes the set of all variable nodes connected to check node j except variable node i, and β is an empirical correction factor called the *offset*, which is set to 1 in the reference design. We used T_i as a temporary variable for clearer notation. After the values have been updated, we set $Q_i^{\text{old}} \leftarrow Q_i^{\text{new}}$ and $R_{i,j}^{\text{old}} \leftarrow R_{i,j}^{\text{new}}$. An iteration is completed when all layers have



(a) Floating point L-OMS vs. floating point flooding SP.

(b) Floating point L-OMS vs. fixed point L-OMS.

Fig. 1. BER performance of IEEE 802.11ad codes.

been processed. The initial values for Q_i^{old} are the channel LLRs, i.e., $Q_i^{\text{old}} = \ln\left(\frac{p(y_i|x_i=+1)}{p(y_i|x_i=-1)}\right)$, where y_i is the channel output at codeword position i and x_i is the corresponding input. All $R_{j,i}^{\text{old}}$ are initialized to 0. When the maximum number of iterations has been reached, decoding stops and hard decisions are taken based on the sign of each Q_i^{new}. It is clear that the rows of the parity-check matrix have to be processed sequentially. However, by construction, every block of the parity-check matrix of a QC-LDPC code consists of Z independent layers which can be processed in parallel.

In Fig. 1(a), we present the BER performance of the codes defined by the IEEE 802.11ad standard under floating point implementations of the L-OMS algorithm when performing 5 iterations and of the flooding SP algorithm when performing 10 iterations. The L-OMS algorithm can closely match, and in many cases even surpass, the performance of the flooding SP algorithm, even though it only performs half the number of iterations. Furthermore, in Fig. 1(b), we observe that the quantization loss for the L-OMS algorithm when using 5 bits for the representation of messages is about 0.25 dB.

The downside of the layered schedule is that decoding proceeds in a sequential fashion across the layers of $\mathbf{H}$. It is therefore very challenging to achieve high throughput because parallelization is not straightforward due to data dependencies. However, this sequential nature and the reduced number of required iterations enables LDPC decoder architectures which are based on the layered schedule to be highly efficient in terms of area and energy consumption. A multi-gigabit decoder based on a hybrid between a layered and a flooding schedule was presented in [4]. This multi-layered OMS decoder performs OMS decoding on K (not necessarily independent) layers simultaneously, providing a trade-off between parallelism (flooding) and convergence speed (layered). In the following section, we will describe our proposed architecture which parallelizes the L-OMS schedule with no negative effect on convergence speed.

III. Decoder Architecture

A. Reference Architecture

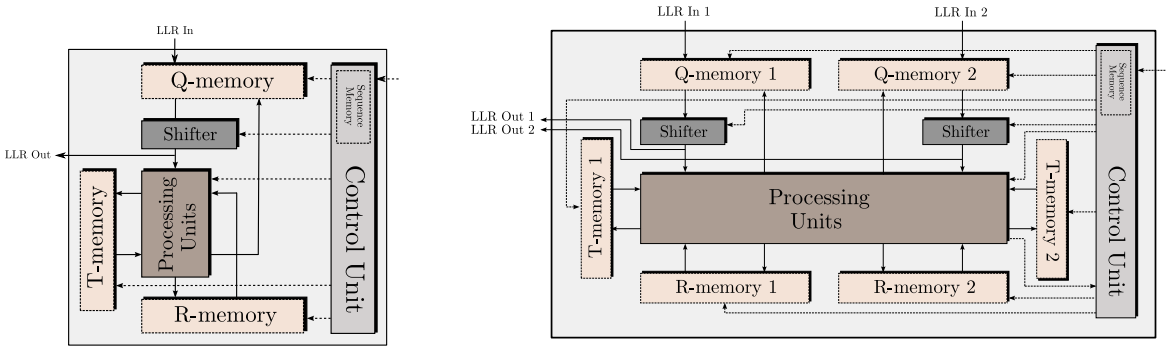
Our architecture improves the architecture presented in [5], which we will briefly review below. With the OMS algorithm, instead of calculating a different minimum, which is used in (4), for each one of the variable nodes to which a layer is connected, it suffices to find the two smallest values, denoted m_1 and m_2. We then use m_1 for the calculations of the messages towards all variable nodes except for the one where m_1 was found, where we use m_2 instead. Similarly, we can calculate the overall product of the signs and then multiply the result with the sign of the value coming from variable node i which we want to exclude. Moreover, m_1 and m_2 can be found in parallel or serially within one row. The reference architecture uses a serial approach, which reduces complexity and area significantly.

The main building blocks of the reference architecture are the MIN and SEL units, which we collectively call *processing units*. Each MIN unit operates on one non-zero element of a row of the parity-check matrix per cycle. It is responsible for finding m_1 and m_2, as well as the overall sign and the position where m_1 was found. The reference architecture contains Z MIN units which process the Z independent check nodes of every row of the block parity-check matrix simultaneously. To this end, Z Q-values are read per cycle from the Q-memory and fed into a cyclic shifter, which routes the Q-values to the correct MIN units, based on the entries of $\mathbf{H}$. The need for a second cyclic shifter which rotates the values back to their original orientation for proper storage is eliminated by using differential shifts. The shifter stores the old shift value of each block in a small memory and computes the shift required based on this value and the shift value corresponding to the current entry of $\mathbf{H}$. The temporary T-values of (3) are calculated by the MIN units and stored in the T-memory. Once the MIN units have finished processing all non-zero blocks in a row of the block parity-check matrix, the SEL units use the resulting m_1, m_2, sign and T-values to update Z R-values and Q-values according to (4) and (5). The MIN units can start processing blocks of the next row which do not have data dependencies with the rows on which the SEL units are currently operating on. Data forwarding and memory bypassing is applied whenever possible in an effort to reduce pipeline stalls and energy consumption, respectively. For example, if a Q-value which is produced at some cycle k is needed by some MIN unit in cycle $k+2$, it can be fed directly to the cyclic shifter instead of waiting for it to be written to and read from the Q-memory. An overview of the reference architecture is presented in Fig. 2(a).

The Q and T-memories have sizes $N \cdot Z \cdot N_Q$ and $N \cdot Z \cdot N_T$ bits, respectively, where N_Q and N_T is the number of bits used for quantization (in this work, $N_Q = N_T = 5$). The R-memory is slightly larger, having $N_{\text{nnz}} \cdot Z \cdot N_R$ bits, where N_{nnz} is the maximum number of non-zero blocks in the parity-check matrices defined by the standard (in this work, $N_{\text{nnz}} = 56$ and $N_R = 5$).

B. Parallelized Architecture

Due to the serial nature of the reference architecture, it is very challenging to achieve multi-gigabit throughputs. One solution would be to overlap the decoding of two codewords, as done in [3]. However, this approach would require twice the memory and it would also increase decoding latency. Since the reference architecture consists of



(a) Reference architecture.

(b) Proposed doubly parallelized architecture.

Fig. 2. High level overview of decoder architectures.

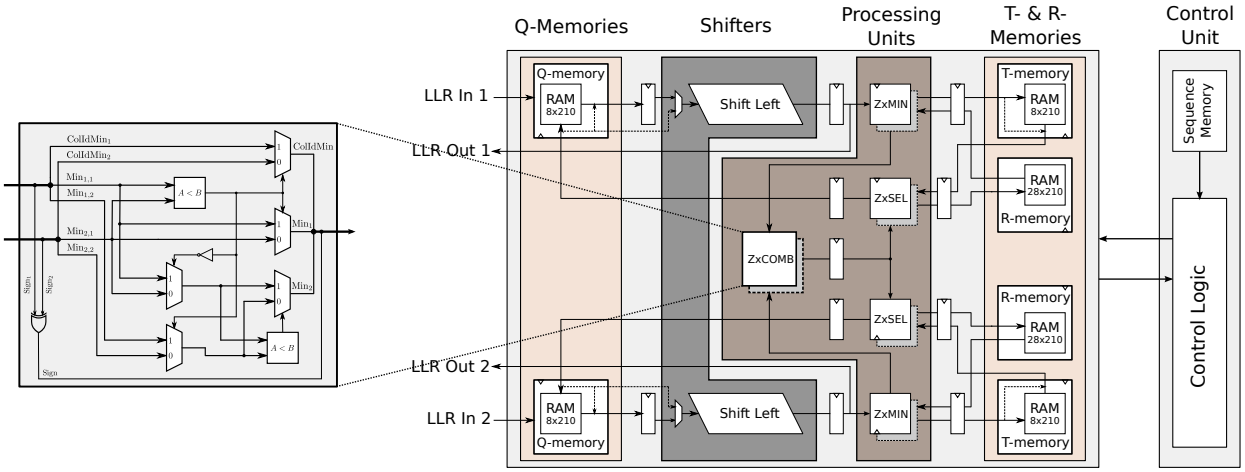


Fig. 3. detailed diagram of parallelized architecture.

almost 60% memory [5], a more elegant approach is to try to increase the number of processing elements. In order to process each layer more rapidly, we propose to split the parity-check matrix into two parts and to double the number of MIN and SEL units. Each of the two groups of Z MIN and Z SEL units operates on a different group of 8 columns of the block parity-check matrix. The grouping has to be chosen carefully in order to balance the processing load between the two groups of MIN and SEL units as much as possible, thus minimizing the number of additional pipeline stalls and maximizing hardware usage. It is possible to use more units but the returns are diminishing as data dependencies quickly become the limiting factor.

It is important to note that the total number of messages that need to be stored is the same as that of the reference architecture. However, since we are processing two blocks of $\mathbf{H}$ in parallel, memory bandwidth is doubled with respect to the reference architecture. Fortunately, the set of memory addresses that the the two groups of MIN and SEL units read and write are disjoint for all three types of memories. Therefore, we can split each memory into two independent memories of half the storage capacity, eliminating the need for a high throughput memory.

Since the MIN units in the original layered architecture compute the two minima over the entire row they are operating on, we need to combine the results of the two MIN units which operate on half a row each. This task is performed by the combiner (COMB) unit, which computes the overall minima, as well as the overall sign. Let $m_{i,j},\ i,j=1,2$, denote the j–th minimum of the i–th MIN unit. Since $m_{i,1} \leq m_{i,2},\ i=1,2$, the first overall minimum value is

$$m_1 = \min\left(m_{1,1}, m_{2,1}\right). \tag{6}$$

Let $i_1 \in \{1,2\}$ denote the value of i that minimizes (6) and let i_1^{c} denote its complement. Since $m_{i_1^{\mathsf{c}},1} \leq m_{i_1^{\mathsf{c}},2}$, finding the second minimum value is also simple, i.e.,

$$m_2 = \min\left(m_{i_1^{\mathsf{c}},1}, m_{i_1,2}\right). \tag{7}$$

For the signs, we have $s = s_1 \oplus s_2$, where s_1 and s_2 are the signs calculated by the first and second MIN unit, respectively.

The COMB unit is the key that enables our parallelized architecture to be equivalent in operation to the non-parallelized reference architecture. A high level overview of the proposed parallelized architecture can be seen in Fig. 2(b). We also provide a more detailed diagram of the architecture, where all pipeline registers are visible, in Fig. 3. Dashed lines represent places where data forwarding and/or memory bypassing is performed.

C. Sequence Length and Throughput

Every iteration of the decoder is controlled by a command sequence. The sequence is created offline based on the parity-check matrices of each one of the codes. Every command of the sequence contains all memory addresses which have to be read from and written to, as well as information on which pipeline stages have to be stalled and at which stages forwarding or memory bypassing has to be performed. One command is issued per cycle and the information contained in each command propagates through the pipeline. The sequence length L is defined as the number of commands that have to be issued in order for the decoder to complete a full iteration of the L-OMS algorithm.

For a fixed number of iterations I, coded throughput is a function

TABLE I
COMPARISON WITH EXISTING WORK

	This work	Reference Arch. [5]	ISCAS'11 [3]	JSSC'12 [10]	ACSSC'11 [11]	ISCAS'11 [4]
Standard	802.11ad	802.11n	802.11ad	802.15.3c	802.15.3c	802.11n
Technology	40 nm	180 nm	65 nm	65 nm	65 nm	45 nm
Core Area	0.18 mm^2	3.39 mm^2	1.30 mm^2	1.56 mm^2	0.72 mm^2	0.81 mm^2
Normalized Area	0.18 mm^2	0.16 mm^2	0.49 mm^2	0.59 mm^2	0.27 mm^2	0.64 mm^2
Clock Frequency	850 MHz	208 MHz	150 MHz	197 MHz	235 MHz	815 MHz
Coded $\Theta_{\min}$	3.12 Gbps	780 Mbps	3.08 Gbps	6.16 Gbps	7.90 Gbps	3.60 Gbps

TABLE II
COMPARISON OF SEQUENCE LENGTHS

	Sequence Length		
Code Rate	Natural	Optimal (Gain)	Reference/2 (Loss)
1/2	34	29 (15%)	28 (4%)
5/8	34	28 (18%)	27 (4%)
3/4	40	35 (13%)	30 (17%)
13/16	30	28 (7%)	25 (12%)

TABLE III
BEST COLUMN GROUPING FOR EACH CODE

Code	Group 1	Group 2
Rate 1/2	{1, 2, 3, 4, 7, 10, 13, 14}	{5, 6, 8, 9, 11, 12, 15, 16}
Rate 5/8	{1, 2, 3, 6, 9, 11, 13, 15}	{4, 5, 7, 8, 10, 12, 14, 16}
Rate 3/4	{1, 2, 3, 4, 5, 6, 10, 15}	{7, 8, 9, 11, 12, 13, 14, 16}
Rate 13/16	{1, 2, 3, 4, 5, 6, 7, 15}	{8, 9, 10, 11, 12, 13, 14, 16}

of the sequence length L [5]

$$\Theta(L) = \frac{Z \cdot N}{L \cdot I + 8} f_{\text{Clk}}, \tag{8}$$

where 8 cycles are needed to flush the pipeline after the last sequence command has been issued. The worst-case throughput, denoted by $\Theta_{\min}$, is the throughput achieved by using the code-rate which leads to the longest sequence, denoted $L_{\max}$, i.e., $\Theta_{\min} = \Theta(L_{\max})$. Our goal is to maximize the minimum throughput that our decoder can support, or, equivalently, to minimize the maximum sequence length.

The two lowest rate parity-check matrices of IEEE 802.11ad contain 4 and 2 groups of non-overlapping rows, respectively [3]. We re-arranged the rows of these matrices in order to group the non-overlapping rows, thus reducing data dependencies significantly. The grouping of columns can be either done naturally, i.e., by splitting the parity-check matrices into half, or with some specific goal in mind, which in our case is the minimization of the sequence length. In order to achieve this goal, for each code, we performed an exhaustive search over all possible combinations of 16 columns into two groups of 8 columns each and calculated the resulting sequence lengths using our sequence generator script. The resulting codes are always equivalent to the corresponding original ones, since by permuting the elements of $\mathbf{c}$ accordingly, (1) still holds.

From the results in Table II, we see that with the natural grouping $L_{\max} = 40$, while with the proposed transformations we obtain $L_{\max} = 35$, which is a 12.5% reduction of the maximum sequence length. As the only thing that changes is the order in which we access the Q-memory when writing the channel LLR values and when reading the output, this improvement comes at absolutely no cost. We also present the sequence lengths for the reference architecture divided by two, which could be achieved by an ideal parallelization. We see that, on average, the applied paralellization results in an average overhead of less than 10%. The column indices of the best grouping for each code can be seen in Table III. In principle, the same procedure can be applied to any QC-LDPC code in an attempt to reduce the resulting sequence length. However, for QC-LDPC codes with a larger number of block columns or for more groups, the number of combinations quickly becomes prohibitively large. Similar schedule optimizations based on column re-arrangement have been explored before, see, e.g., [12]. However, these works considered schedules for the overlapping of the two phases of the flooding schedule, whereas our method considers strictly the layered schedule.

IV. SYNTHESIS RESULTS

We synthesized the proposed architecture using a TSMC 40 nm CMOS technology with a target clock frequency of 850 MHz, which leads to a coded throughput of 3.12 Gbps. The logic area of the resulting design is 0.18 mm^2. A comparison with existing work can be seen in Table I. The core area of our decoder is significantly smaller than most previously existing multi-gigabit designs, even when they are scaled to 40 nm for fair comparison.

V. CONCLUSION

In this paper we showed that the use of a layered QC-LDPC decoder is feasible even when multi-gigabit throughputs are required. In order to achieve these throughputs, we doubled the number of processing units of an existing architecture for layered decoding, thus processing independent blocks of the parity-check matrix simultaneously. In order to maximize throughput, we provide a simple and efficient procedure which assigns blocks to processing units by carefully splitting the parity-check matrix into two parts.

ACKNOWLEDGMENT

The work of N. Preyss was supported by the Swiss National Science Foundation (SNSF) under Grant PP002-119057.

REFERENCES

[1] *Draft Standard for Information Technology, Draft Amendment 5, IEEE P802.11ad/D5.0*, IEEE Std., Sep. 2011.

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

[3] M. Weiner, B. Nikolic, and Z. Zhang, "LDPC decoder architecture for high-data rate personal-area networks," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 2011, pp. 1784–1787.

[4] Y. Sun, G. Wang, and J. Cavallaro, "Multi-layer parallel decoding algorithm and VLSI architecture for quasi-cyclic LDPC codes," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 2011, pp. 1776–1779.

[5] 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.

[6] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[7] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[8] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Linköping University, Linköping, Sweden, 1996.

[9] E. Sharon, S. Litsyn, and J. Goldberger, "Efficient serial message-passing schedules for LDPC decoding," *IEEE Trans. Inf. Theory*, vol. 53, no. 11, pp. 4076–4091, Nov. 2007.

[10] S.-W. Yen, S.-Y. Hung, C.-L. Chen, H.-C. Chang, S.-J. Jou, and C.-Y. Lee, "A 5.79-Gb/s energy-efficient multirate LDPC codec chip for IEEE 802.15.3c applications," *IEEE J. Solid-State Circuits*, vol. 47, no. 9, pp. 2246–2257, Sep. 2012.

[11] H. Shirani-Mehr, T. Mohsenin, and B. Baas, "A reduced routing network architecture for partial parallel LDPC decoders," in *Proc. 45th Asilomar Conf. on Signals, Systems and Computers*, Nov. 2011, pp. 2192–2196.

[12] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29 mm^2 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 μm CMOS process," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 672–683, Mar. 2008.