Design of a VLSI Decoder for Partially Structured LDPC Codes

Hindawi Publishing Corporation
International Journal of Digital Multimedia Broadcasting
Volume 2008, Article ID 245305, 12 pages
doi:10.1155/2008/245305

Research Article

Design of a VLSI Decoder for Partially Structured LDPC Codes

Fabrizio Vacca,[1] Libero Dinoi,[2] and Guido Masera[1]

[1] *Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy*
[2] *Elettronica S.p.A., 00131 Roma, Italy*

Correspondence should be addressed to Fabrizio Vacca, fabrizio.vacca@polito.it

Received 3 April 2008; Revised 1 July 2008; Accepted 27 August 2008

Recommended by Fred Daneshgaran

The starting point of this work is the development of a new class of partially structured LDPC codes, very well suited for hardware implementation. Specifically these codes are built so that the edges of their parity matrix can be partitioned into two disjoint sets, namely, the structured and the random ones. For the proposed class of codes a constructive design method is provided. To assess the value of this method the constructed codes performance are presented. From these results, a novel decoding method called split decoding is introduced. Finally, to prove the effectiveness of the proposed approach a whole VLSI decoder is designed and characterized.

1. INTRODUCTION

Low Density Parity Check [1, 2] (LDPC) codes are among the most powerful error correcting codes available; performance and decoding algorithms have been deeply explored in the last few years and LDPC codes have been proposed for application in several standards. However, their practical implementation is still a challenging subject of investigation. There are basically two aspects of LDPC that pose serious implementation problems: (1) the huge size of parity-check matrices that are of interest for high performance applications; (2) the high irregularity of these matrices, that is, the fact that they are very sparse with ones often distributed almost randomly.

From the implementation standpoint, the first aspect implies the allocation of a large number of processing elements and larger number of communication paths among them; the second one entails a very limited adjacency of processing elements, resulting in scarcely efficient and expensive communication structures [3]. In order to limit the implementation complexity of both the processing and the interconnect resources of the decoder, several partially parallel architectures have been proposed as feasible alternatives to the fully parallel approach [4–6].

In partially parallel architectures, processing elements (PE) are shared among multiple check and variable nodes: each PE is required to sequentially serve a number of rows or columns of the parity-check matrix; since, at each instant of time, only a subset of the messages to be exchanged between variable and check nodes actually need to be moved from one PE to another one, this approach also reduces the number of physical interconnects in the decoder. Finally, instead of storing messages in independent registers, partially parallel architectures allow grouping them into more efficient memories. Of course, resource sharing implies a throughput scaling and the parallelism degree has to be selected according to the target throughput: complexity tradeoff.

While the partially parallel approach looks like a fully scalable solution able to flexibly adapt to different cost and throughput constraints, it raises the problem of collisions in the access to memories [7]. In the connection of P PEs with the same number of message memories, the possibility of simultaneously moving P messages between PEs and memories has a very low probability and in a large percentage of cases more than one required message must be read from or written to the same memory. Due to the limited adjacency of the parity-check matrix, the optimization of message partitioning among memories tends to have a poor effect on the number of occurred conflicts. PEs must then be frequently stalled to accommodate for multiple simultaneous accesses to memories and this severely affects the decoder throughput.

In order to cope with these problems, two different kinds of solution have been proposed.

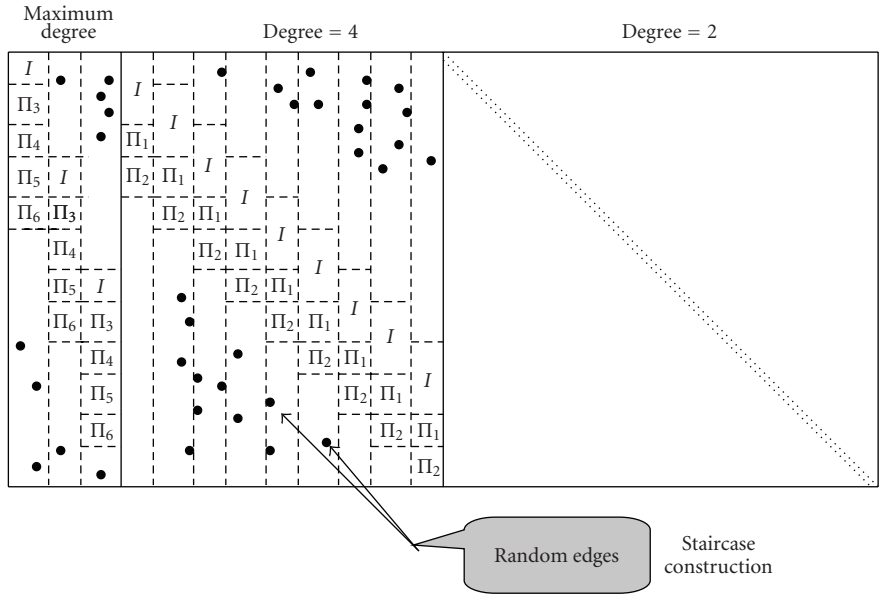

Figure 1: Parity-check matrix structure for code rate 1/2.

(1) A few code-independent approaches have been formulated aiming at avoiding collisions for a generic LDPC code. This class of solutions leads to expensive implementation architectures, such as those proposed in [5, 8, 9].

(2) Structured LDPC codes [10] have been proposed: in this approach, a structure is given to the parity-check matrix and all ones are properly distributed with the purpose of limiting the interconnect requirements while keeping good error correcting capabilities. Classes of structured codes have been studied and implemented successfully.

This paper presents a new approach based on the idea of designing parity-check matrices with edges partitioned into two classes, namely, structured ones, positioned in accordance with a repetitive fixture laid not far from the diagonal, and random ones, which can be placed freely in the whole matrix, with the purpose of achieving good performance in terms of error correcting capabilities. This approach has been previously partly presented in [11] where the authors proposed a partially structured 1/2 rate LDPC code and described the implementation of a decoder with two separate processing units: a dedicated part tailored to take advantage of the regularly placed ones, and a programmable application specific instruction set processor (ASIP) serving the whole random part of the parity-check matrix.

This work extends the previous one deeply investigating the benefits coming from the idea of designing a partitioned parity-check matrix, with a particular emphasis on the hardware architecture. This paper provides two major contributions: a decoding algorithm that takes advantage of the partitioned parity-check matrix to simplify the collision problem and the VLSI design of a decoder implementing the described algorithm.

The rest of the paper is organized as follows. Section 2 introduces LDPC and eIRA codes, reviewing some code design issues. Then the proposed partially structured eIRA code class is presented and some constructive results are given. To prove the usefulness of this approach simulation performance is presented.

Then the decoding algorithm of these codes is presented in Section 3. Due to the particular structure of the parity-check matrix, some modifications to the traditional belief propagation algorithm are possible. The obtained algorithm, called split decoding presents remarkable savings with respect to the traditional one. These savings are deeply investigated in Section 4. The modular fixture of structured ones strongly simplifies the communication structure among processing elements, while the reduced number of "random" ones enables collision-free partitioning that eliminates the need for stall cycles in the decoding process. In the same Section, logical synthesis results are given and memory footprint of the designed VLSI core is analyzed. Lastly, in Section 5 conclusions will be drawn.

2. PARTIALLY STRUCTURED eIRA CODES

2.1. LDPC and eIRA codes background

An LDPC code is a linear block code defined by an $m \times n$ sparse parity-check matrix. It can be described also in terms of its Tanner graphs: each of the bits and of the parity-check equations, defined, respectively, as variable nodes (VNs) and check nodes (CNs), is represented by a vertex in the graph. The number of edges connected to a vertex of the graph is defined as the *degree* of that node; the VN (or CN) *degree distribution* of a code specifies how the edges are distributed

Table 1: VN degree distributions (node perspective).

Code rate	$\widetilde{\lambda}_1$	$\widetilde{\lambda}_3$	$\widetilde{\lambda}_5$	$\widetilde{\lambda}_6$	$\widetilde{\lambda}_9$
1/3	0.6667	0.2222	—	—	0.1111
1/2	0.5000	0.3750	—	0.1250	—
2/3	0.3334	0.5556	—	0.1110	—

among the variable (or check) nodes of the code. In a *regular* (c, r) LDPC code all VNs have degree c and all CNs have degree r in the graph.

In [12], *irregular* LDPC codes have been shown to outperform regular LDPC codes. In this work, we will focus on eIRA codes [13–15]: a popular subset of LDPC codes. eIRA codes are characterized by the peculiar structure of the parity-check matrix corresponding to the nonsystematic bits, which is made of VNs with degree 2, arranged according to a chain-like structure, known in the literature as "staircase construction." The main advantage of this structure is that the obtained code can be encoded in linear time. In [14], it is shown that such a constraint leads to negligible performance losses.

There are some specific design issues for eIRA codes. Typically, the project of irregular LDPC codes consists of the optimization of the VN and CN degree distributions and of the design of a parity-check matrix compliant with the obtained distributions. The degree distributions are optimized by means of several techniques such as density evolution [12] and its approximated version of [16]; however, when dealing with eIRA codes, these techniques cannot be directly applied. The problem of designing an efficient eIRA code is well presented in [17], where it is shown that the loss in performance introduced by the staircase construction is about 0.1 dB, in the waterfall region.

While the VN and CN degree distributions determine the convergence behavior of the code asymptotically (for an infinite block length), the actual structure of the parity-check matrix is crucial for the performance of finite length LDPC codes. Many design algorithms can be applied to obtain a good parity-check matrix [18–20]; specific design algorithms for eIRA codes are presented in [21, 22].

2.2. Proposed code design guidelines

As a general rule, highly structured matrices and a limited connectivity lead to low code performance; this effect is not present in [10] because regular LDPC with a VN degree of 3 does not suffer from high error floors. On the contrary, eIRA codes have a significant number of degree 2 VNs. This would likely result in a code with a high error floor due to the constraints on the limited connectivity. To avoid these effects we adopt a partial structure allowing some of the edges to be placed randomly.

Let us illustrate this concept with an example: in Figure 1 we depict the parity-check matrix of a rate 1/2 code. Three sections can be identified: the rightmost contains the "staircase" parity-check VNs, the leftmost one contains high-degree VNs and the central one degree 4 VNs. The latter two sections include edges disposed according to a predefined pattern, as well as some placed in a pseudorandom fashion. The optimal degree distribution obtained as explained in [17] contains a significant number of degree-3 VNs. We deliberately decided to forbid them to lower the error floor. Details on this expedient are available in [23].

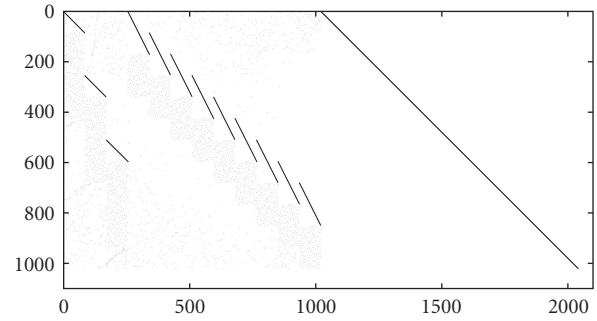

Figure 2: Parity-check matrix of a (2040, 1020) code.

Table 2: Edge categories distribution.

Code rate	Staircase	Pattern	Free
1/3	0.400	0.467	0.133
1/2	0.296	0.519	0.185
2/3	0.182	0.606	0.212

As far as the structured part of the matrix is concerned, we use permuted versions of the identity matrix. In particular, the permutations used in the blocks labelled Π_i ($i = 1, 2, \ldots$) in Figure 1 are triangular S-random interleavers built according to a tail-biting definition of the spread factor.

2.3. Three examples: rate 1/2, 2/3, and 1/3 codes

We will illustrate how the design guidelines can be applied to generate three different codes with three code rates, namely, 1/2, 2/3, and 1/3. For the rate 1/2 we consider a codeword length approximately equal to 2000 bits, in order to have a direct comparison with the code of [10], and we chose a degree distribution with a maximum VN degree of 7, which produces a total number of edges very similar to the one of [10]. The degree distribution of the code and its parity-check matrix are described, respectively, in Table 1 and in Figure 2.

The edges can be classified into three categories: they can be related to the staircase construction, to the deterministic pattern, or their position can be free. In Table 2 we list their distribution: the percentage of randomly placed edges is quite low, so that the burden on the slower ASIP part of the decoder is not excessive.

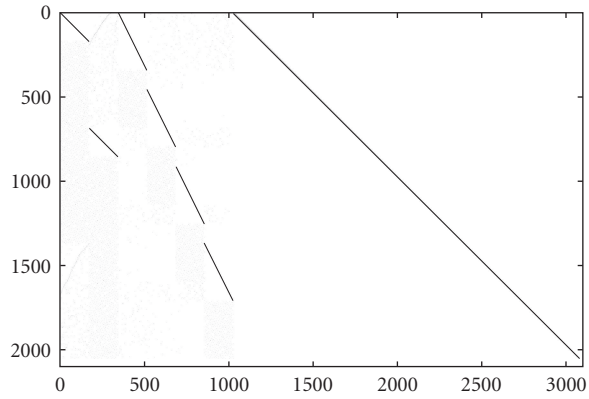

Figure 3: Parity-check matrix of a (3078, 1026) code.

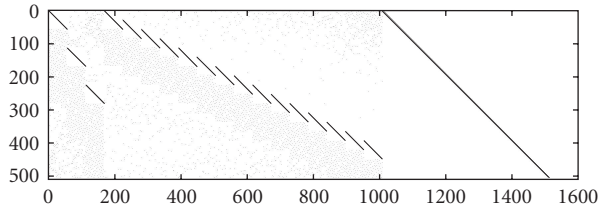

Figure 4: Parity-check matrix of a (1512, 1008) code.

According to the scheme of Figure 1, there are nine macro-columns in the central section. If $D = (n-k)/12$, for the ith $(i = 0, 1, \ldots, D-1)$ column of the jth $(j = 0, 1, \ldots, 8)$ macro-column, the three edges are placed in the following rows:

(i) $jD + 2i + (1 - \mathrm{mod}(D, 2))$;

(ii) $(2 + j)D + \Pi_1(i)$;

(ii) $(3 + j)D + \Pi_2(i)$.

Based on the equations above, we chose Π_1 and Π_2 as tail-biting S-random interleavers as explained in [11].

The same philosophy holds also for code rates different from 1/2. In particular, we present the cases 1/3 and 2/3 as other examples.

The degree distributions are reported in Table 1, while Table 2 describes the distribution of the edges in the different categories. The two generated LDPC codes have an information word length k of 1026 and 1008, respectively, while the codeword length n is 3078 and 1512. Their parity-check matrixes are depicted in Figures 3 and 4, respectively.

Also in these cases, it is crucial to use tail-biting S-random interleavers as building blocks of the parity-check matrix.

2.4. Simulation results

To verify the validity of the designed algorithm, we compared the rate 1/2 code described in Section 2.3 with two similar codes coming from IEEE 802.11n and IEEE 802.16e standards.

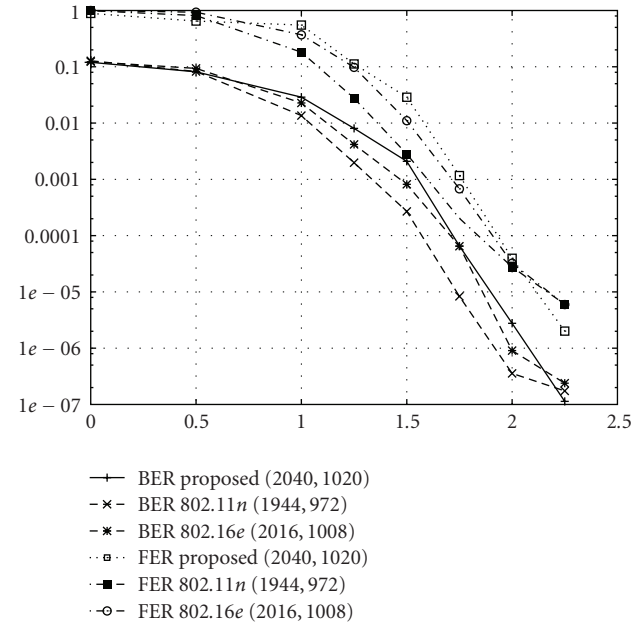

Figure 5: Simulation results for code rate 1/2.

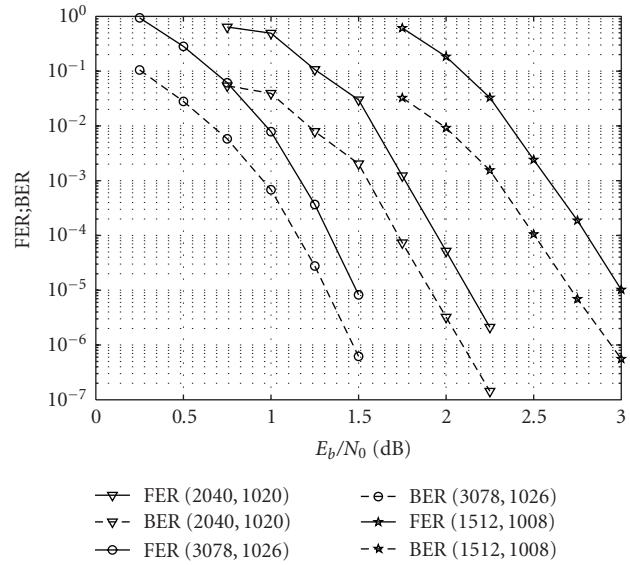

Figure 6: FER and BER simulation results for the 1/2, 1/3, and 2/3 code rates.

The simulation results of Figure 5 show that our code does not suffer from high error floor, despite its strong structure (is better than IEEE 802.11n and IEEE 802.16e at high SNR as far as FER is concerned).

Finally, in Figure 6 we provide the simulations results of the LDPC codes generated in the previous subsections. It is reasonably safe to conclude that the structured design approach suggested in this paper leads to good results for a wide range of code rates, both in terms of convergence threshold and of low error floors.

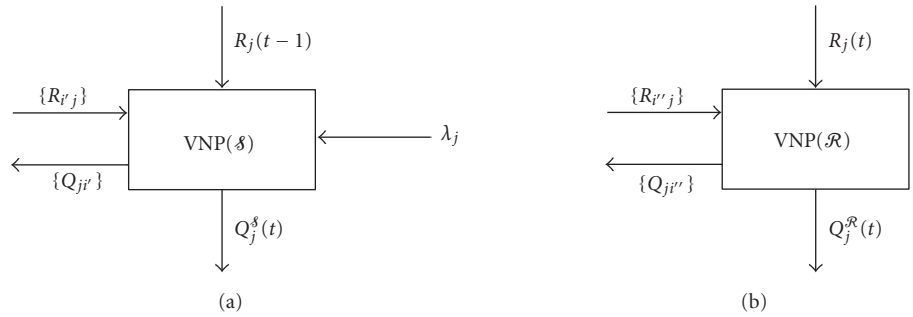

Figure 7: Variable node processor input and outputs.

3. DECODING ALGORITHM FOR PARTIALLY STRUCTURED LDPC CODES

The belief propagation (BP) algorithm is one of the most popular LDPC decoding methods, giving optimal performance in the case of H matrices with no cycles. In BP, the following update rule is applied for VN_j:

$$Q_{j,i} = \lambda_j + \sum_{k \in \mathcal{M}(j)/i} R_{k,j}, \tag{1}$$

where λ_j is the intrinsic information, which depends on the channel variance and on the jth received symbol of the codeword, $R_{i,j}$ is the message sent by CN_i to VN_j, $\mathcal{M}(j)$ is the set of nodes connected to VN_j, and $Q_{j,i}$ is the message sent by VN_j to CN_i. All but one input messages R_{ij} are summed in (1) to output a single variable to check message $Q_{j,i}$.

Check node i receives $Q_{j,i}$ messages as inputs and generates output messages according to the following equation:

$$R_{i,j} = \Omega_{k \in \mathcal{N}(i)/j}(R_{k,i}), \tag{2}$$

where $\mathcal{N}(i)$ is the set of nodes connected to CN i and the Ω operator, for two operands, is defined as

$$\Omega(a,b) = \log\left(\frac{e^a + e^b}{1 + e^{a+b}}\right). \tag{3}$$

For more than two operands, the Ω operator can be applied recursively; for example, for three operands, we have

$$\Omega_{k \in [0 \cdots 2]} a_k = \Omega(a_0, \Omega(a_1, a_2)). \tag{4}$$

The BP algorithm is usually executed in two phases, repeated at each decoding iteration: first all variable nodes sample their input messages and process them, then check nodes receive messages and generate their outputs.

We propose here a new decoding algorithm, where the same operations described in (1) and (2) are separately applied to two subsets of nodes: let us define $\mathcal{M}^{\mathcal{S}}(j)$ and $\mathcal{M}^{\mathcal{R}}(j)$ as the two subsets of nodes connected to VN_j and associated, respectively, to structured and random ones of the H matrix. The VN update rule can be rewritten as

$$Q_{j,i} = \lambda_j + \sum_{k \in \mathcal{M}^{\mathcal{S}}(j)/i} R_{k,j} + \sum_{k \in \mathcal{M}^{\mathcal{R}}(j)/i} R_{k,j}. \tag{5}$$

By also defining the following two sums:

$$\begin{aligned} R_j^{\mathcal{R}} &= \sum_{k \in \mathcal{M}^{\mathcal{R}}(j)} R_{k,j}, \\ R_j^{\mathcal{S}} &= \lambda_j + \sum_{k \in \mathcal{M}^{\mathcal{S}}(j)} R_{k,j}, \end{aligned} \tag{6}$$

the variable to check message is computed as

$$Q_{j,i} = R_j^{\mathcal{R}} + R_j^{\mathcal{S}} - R_{i,j}. \tag{7}$$

In Figure 7, a high-level view of the variable node processor operations is given. The purpose here is to better explain which variables are read and written during the execution of the decoding process. The left part of figure shows input and output signals when the VN processor is serving the structured ones (VNP($\mathcal{S}$)), while the right part refers to the exchanged signals when the same unit is processing random ones (VNP($\mathcal{R}$)).

The check node processing can be expressed in a similar way. The $\mathcal{N}(i)$ set of nodes connected to CN i is now split into $\mathcal{N}^{\mathcal{S}}(i)$ and $\mathcal{N}^{\mathcal{R}}(i)$, defined as the two subsets of nodes connected to CN i and associated, respectively, to structured and random ones. Using this two sets of nodes, we compute the check to variable messages by means of the following expression:

$$R_{i,j} = \Omega(\Omega_{k \in \mathcal{N}^{\mathcal{S}}(i)/j}(Q_{k,i}), \Omega_{k \in \mathcal{N}^{\mathcal{R}}(i)/j}(Q_{k,i})). \tag{8}$$

Two total sums of the incoming messages are defined as

$$\begin{aligned} Q_i^{\mathcal{R}} &= \Omega_{k \in \mathcal{N}^{\mathcal{R}}(i)}(Q_{k,i}), \\ Q_i^{\mathcal{S}} &= \Omega_{k \in \mathcal{N}^{\mathcal{S}}(i)}(Q_{k,i}) \end{aligned} \tag{9}$$

and the check to variable messages are then obtained as

$$R_{i,j} = \Omega(\Omega_{k \in \mathcal{N}^{\mathcal{S}}(i)/j}(Q_{k,i}), Q_i^{\mathcal{R}}) \tag{10}$$

in the case of messages associated to structured ones ($i \in \mathcal{N}^{\mathcal{S}}(i)$), and

$$R_{i,j} = \Omega(\Omega_{k \in \mathcal{N}^{\mathcal{R}}(i)/j}(Q_{k,i}), Q_i^{\mathcal{S}}) \tag{11}$$

in the case of messages associated to random ones ($i \in \mathcal{N}^{\mathcal{R}}(i)$).

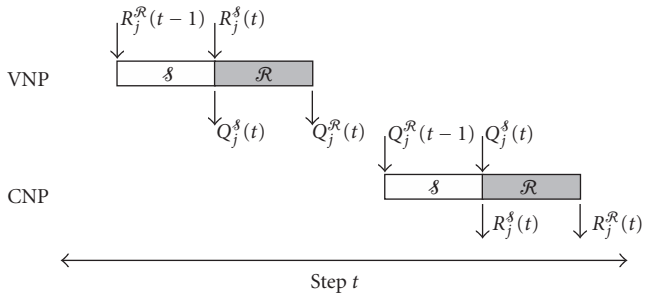

Figure 8: Scheduling of operations inside a single iteration of the split decoding algorithm.

The scheduling of the read and write operations for both a VN as well as for the CN is reported in Figure 8. As it can be seen, structured and random parts are always scheduled in sequence.

We call this modified BP algorithm split BP decoding (SBP) or briefly split decoding (SD). The new method is simply an algebraic reorganization of (1) and (2), and the code performance in terms of error correcting capabilities does not decrease when applying (7), (10), and (11). As we will detail in the following section, the key advantage provided by SD is the possibility of independently processing random and structured edges of H: while structured ones follow a modular pattern that strongly simplifies their parallel processing with no collisions in the memory access, the same characteristic does not hold for random ones, which tend to generate collisions when processed in parallel. However, differently from typical random H matrices, the subset of random ones is sparse enough to enable their efficient partitioning into separate memory banks with no collisions in the read or write accesses.

4. DECODING ARCHITECTURE

4.1. Functional description

From an architectural standpoint, the main contribution of our work lies in the different scheduling we use for the decoding process. In fact, as can be observed from Figure 9, the decoder architecture is essentially a partially parallel one very similar to those already proposed in the literature [4–6]. The decoder presented in Figure 9 supports any generic (n, m) LDPC code distributing the $n + m$ node operations over P processing elements (PEs). In particular, the example shown is for $P = 85$ but the same architecture can be used for any value of P. However, in the following analysis we will consider the case of $P = 85$ since for the code designed in Section 2.3 this is the size of the repetitive fixture.

Each PE_i is connected to a memory, called DMEM_i. Each PE can write into any memory bank exploiting a crossbar switch, while it can only read from its own memory. These memories are used to store the messages exchanged between PEs during the entire decoding process. Adopting a single crossbar switch solution allows saving of hardware resources without limiting the supported message exchange between

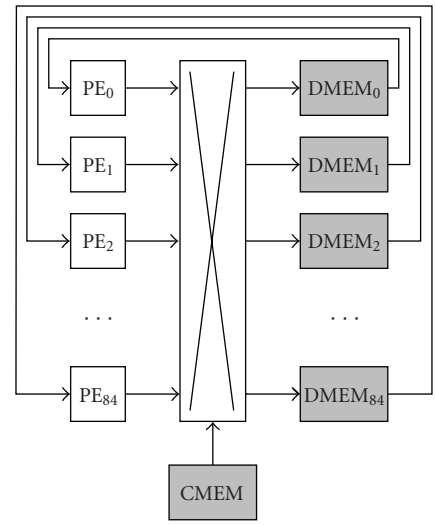

Figure 9: General partially parallel architecture.

PEs and memories: in fact, given two generic permutation laws, π_{read} and π_{write}, associated, respectively, to the reading and writing operations of PEs on the memory banks, one of the two laws can always be replaced with a fixed PE to memory coupling, provided that the other law is modified with a new one, obtained as the serial concatenation of π_{read} and π_{write}. We assume the following.

(i) Each PE is able to be used both as check as well as variable node. Let PE_i be a given processing element of the decoder: this means that through the whole decoding process PE_i will serve n_i variable nodes and m_i check nodes, respectively. If the total workload is evenly split over the PEs' set, $n_i = n/P$ and $m_i = m/P$.

(ii) Each PE receives messages from one single memory bank (DMEM) through a dedicated connection and sends updated messages to any memory through a $P \times P$ crossbar switch.

(iii) DMEM is single port memory bank, so that read and write accesses are possible only in different clock cycles.

(iv) Control values to be applied to the programming input of the crossbar switch are stored in a dedicated memory, CMEM, which is sequentially addressed by a counter.

(v) The content of CMEM is precomputed offline to resolve memory access conflicts.

Figure 10 represents the data-path architecture of a single PE. Each PE is also reused in time to compute both the structured part as well as the random one. In order to support this feature a second input and some additional logic are required.

Three memory banks are needed to fully support PE operations, namely, LLR MEM, ACC MEM, and S/R MEM. The first one, as the name suggests, is needed to store the information coming from the channel. It is intended to be

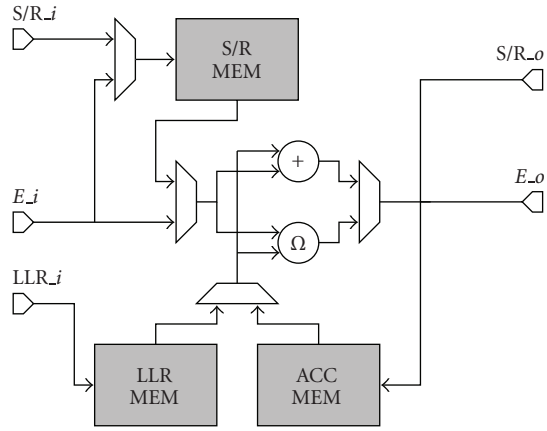

Figure 10: Architecture of the processing element dedicated to check and variable node processing.

loaded at the beginning of every new frame to be decoded. ACC MEM is used to implement a flexible accumulator scheme necessary to implement the additions needed both in VN as well as CN. The third memory bank, the so-called S/R memory is useful to allow PE to be reused for structured and random part processing. When the PE is performing the structured part, the S/R memory will hold messages coming from the random one; conversely, when the PE is dedicated to the random part, structured messages will be held into S/R RAM.

When configured to serve as CN processing unit, each PE sequentially receives messages originated by VN (here called generally E_i) from one of the DMEM banks. Partial results of this computation are then stored into the ACC MEM until all the edges for that particular CN have been received. The latency of the whole update procedure for a given CN depends on the degree of the corresponding row in the parity-check matrix. Using the data path depicted in Figure 10 one clock cycle is required for reading each incoming edge as one cycle is needed to write the result back (edge out). The same holds for the VN case, that is, the total latency is equal to twice the column degree.

4.2. Performance analysis

Given a decoder architecture as the one of Figure 9 and a flexible PE as in Figure 10 it is possible to derive a sort of *performance bound* for the decoding latency as well as for the throughput. Given an even workload distribution among the PEs and no collisions in reading message memories, the number of cycles to perform a single iteration can be expressed as

$$\hat{D} = 4 \cdot \frac{E}{P}, \tag{12}$$

where E is the total number of ones in the parity-check matrix, P is the parallelism of the system. The factor of four comes out from the structure of the PE itself. This number assumes that only single-port memories can be used and that layered decoding scheduling is not exploited.

Under these premises the maximum theoretical decoding throughput $\hat{T}$ can be expressed as

$$\hat{T} = \frac{(n-m) f_{\mathrm{CK}}}{I \cdot \hat{D}}, \tag{13}$$

where f_{CK} is the clock frequency and I the number of iterations performed. With the proposed 1/2 rate code, assuming a number of iterations equal to ten and a parallelism of $P = 85$ it turns out that $\hat{T} = 0.318 f_{\mathrm{CK}}$.

However, in practical situations memory conflicts force to delay simultaneous accesses to the same bank and then to insert stall cycles. We will then introduce a *collisions degradation factor* α able to capture this behavior. We can then express the total number of cycles needed for each iteration in case of collisions as

$$D_{\mathrm{coll}} = \alpha \cdot \hat{D}. \tag{14}$$

The resulting throughput of the decoder can be rewritten as

$$T_{\mathrm{coll}} = \frac{1}{\alpha} \cdot \hat{T}. \tag{15}$$

It is important to stress how α depends on the specific LDPC code but it is also affected by message scheduling and load partitioning between PE. Partitioning and scheduling techniques can be used to try to minimize α; however, it is known that good parity-check matrices show little adjacency making partitioning and scheduling benefits very limited.

To evaluate the impact of α we try to partition the associated Tanner graph in order to perform an initial allocation. We use the software Metis [24] freely available on the Internet. Given this allocation we implement a cycle-accurate architectural simulator using the Python language, able to report the total number of cycles needed to perform a single iteration (i.e., D_{coll}). In this way, it has been possible to derive α values for different types of LDPC codes (see Table 3).

The important peculiarity observed is that the α factor for a given LDPC tends to be as large as four. As an example, in the case the proposed code we found $\alpha = 4.28$. This means that the potential parallelism of the architecture given in Figure 9 is largely wasted.

Table 3 summarizes these values for the proposed code (reported ad code 1) as well as for other four irregular LDPC codes. In particular, we provide the values of $\hat{D}$, D_{coll}, and relative throughput figures, evaluated as a function of the clock frequency. To show how collisions are a concern also on different LDPC codes we perform the same partitioning and scheduling steps also on different codes. Code labelled as 2 is directly taken from Professor MacKay website [25], where it can be found as *4986.93i.939*. Code 3 is the IEEE 802.11n wireless local area network channel code and exhibits characteristics similar to the proposed one, while code labelled as 4 is one of the IEEE 802.16e 1/2 rate codes. Also in these three cases it is possible to observe how the presence of a significant number of collisions spoils the overall system performance. For the last two codes, however, it is important to consider that they have been specifically

Table 3: Cycle and throughput evaluation for different LDPC codes.

Code	n	m	E	$\hat{D}$	D_{coll}	α	$\hat{T}$	T_{coll}
1	2040	1020	6885	324	1387	4.28	0.32 f_{CK}	0.07 f_{CK}
2	9972	4986	14958	704	2794	3.97	0.71 f_{CK}	0.18 f_{CK}
3	1944	972	6797	320	1240	4.19	0.3 f_{CK}	0.07 f_{CK}
4	2016	1008	6384	301	1259	4.18	0.33 f_{CK}	0.08 f_{CK}

designed to be decoded avoiding collisions, provided that dual port memories are used. Despite this, we put these code in Table 3 to enhance the relevance of collisions in LDPC decoding as a severe limitation to system throughput.

The SD algorithm described in Section 3 allows to get over this limitation by exploiting the partial structure of the proposed codes. The only significant modification to the PE structure is the need for S/R MEM to store partial VN to CN and N to VN messages, $Q^{\$}_{i,j}$, $Q^{\mathcal{R}}_{i,j}$, $R^{\$}_{i,j}$, and $R^{\mathcal{R}}_{i,j}$.

The complexity increase due to these hardware modifications is negligible and it will be evaluated in Section 4.3. Instead, the reduction of required decoding cycles impacts significantly on the size of CMEM needed to control the crossbar. We derive here for the proposed approach the offered throughput, which depends on the total number of cycles required to complete all read and write memory accesses.

An iteration is divided into the following four subiterations:

(1) Φ_1: check node processing for structured ones;

(2) Φ_2: check node processing for random ones;

(3) Φ_3: variable node processing for structured ones;

(4) Φ_4: variable node processing for random ones.

In the first subiteration, variable-to-check messages associated to structured ones of H matrix are read by the PEs, as structured ones are contained into $P \times P$ submatrices concentrated along three lines in H; the number of cycles required to complete the reading is equal to the number of submatrices. It is worth noting that no conflicts are possible since each submatrix has a single one per row and per column. In the case of the code rate 1/2 matrix given in Figure 2, this number is equal to $D_{\Phi_1,\text{read}} = E_{\text{str}}/P$ being E_{str} the total number of structured edges. Subiteration Φ_1 also needs to write the generated check-to-variable messages: this operation takes the same time as the reading one, $D_{\Phi_1,\text{write}} = D_{\Phi_1,\text{read}}$. Additionally, each PE in phase Φ_1 must also read the partial VN-to-CN messages $Q^{\mathcal{R}}$ and update the partial CN-to-VN messages $R^{\$}$: both operations take a number of cycles equal to m/P. Thus, phase Φ_1 takes a total number of cycles equal to

$$D_{\Phi_1} = 2 \cdot \left(\frac{E_{\text{str}} + m}{P}\right). \tag{16}$$

The same value is obtained in the calculation of the total number of cycles needed to complete subiteration Φ_3 (variable node processing for structure ones), with the exception that the number of cycles needed to read and write CN to VN messages here is n/P, hence (16) becomes

$$D_{\Phi_3} = 2 \cdot \left(\frac{E_{\text{str}} + n}{P}\right). \tag{17}$$

In subiteration Φ_2, a number of messages equal to the number of the ones in the random part of H matrix must be read and written. Since these ones are distributed in an irregular way, there is no guaranty that they can be handled in parallel P at a time with no collisions. However, very low density of random ones enables their efficient partitioning among the P memories: experiments done with cases proposed in Section 2 and additional random matrices having the same one density show that random ones can almost always be partitioned with no collisions. Under this assumption, the total number of read or write operations is equal to E_{rnd}/P, where E_{rnd} is the number of random ones in H. Subiteration Φ_2 also needs m/P cycles to read the partial VN-to-CN messages $Q^{\$}$ and update the partial CN-to-VN messages $R^{\mathcal{R}}$. The total number of cycles for Φ_2 can then be expressed as

$$D_{\Phi_2} = 2 \cdot \left(\frac{E_{\text{rnd}} + m}{P}\right) \tag{18}$$

while Φ_4 cycles can be derived as for the case of Φ_3, hence

$$D_{\Phi_4} = 2 \cdot \left(\frac{E_{\text{rnd}} + n}{P}\right) \tag{19}$$

combining the four contributions together it turns out that

$$\begin{aligned} D_{\text{sd}} &= 4 \cdot \left(\frac{E_{\text{str}} + E_{\text{rnd}} + n + m}{P}\right) \\ &= 4 \cdot \left(\frac{E + n + m}{P}\right). \end{aligned} \tag{20}$$

We define D_{sd} as the number of cycles needed when the proposed split decoding approach is used. As already stated in (13), the resulting throughput will be

$$T_{\text{sd}} = \frac{(n-m) f_{\text{CK}}}{I \cdot D_{\text{sd}}}. \tag{21}$$

Lastly, we introduce the parameter η as a measure of the total efficiency of the decoding process with respect to the ideal case. For instance, for SD we have

$$\eta_{\text{sd}} = \frac{\hat{D}}{D_{\text{sd}}} = \frac{E}{E + n + m}. \tag{22}$$

Table 4: Memory occupation breakdown.

Split decoding						
Memory	Instances	Parallelism	Words	[Words]	Area	Total area
DMEM	85	8	$\left\lceil 2\frac{E+m+n}{P} \right\rceil$	256	12000 μm^2	1.0 mm^2
CMEM	1	$7 \cdot P = 595$	$D_{\text{sd}}/2$	512	$750 \cdot 10^3$ μm^2	0.75 mm^2
LLR MEM	85	8	$\left\lceil \frac{n}{P} \right\rceil$	32	2550 μm^2	0.22 mm^2
ACC MEM	85	8	$\max\left(\left\lceil \frac{n}{P} \right\rceil, \left\lceil \frac{m}{P} \right\rceil\right)$	32	2550 μm^2	0.22 mm^2
S/R MEM	85	8	$\max\left(\left\lceil \frac{n}{P} \right\rceil, \left\lceil \frac{m}{P} \right\rceil\right)$	32	2550 μm^2	0.22 mm^2
					Total	2.36 mm^2
Partition and scheduling (collision)						
Memory	Instances	Parallelism	Words	[Words]	Area	Total area
DMEM	85	8	$\left\lceil 2\frac{E}{P} \right\rceil$	256	12000 μm^2	1.0 mm^2
CMEM	1	$7 \cdot P = 595$	$D_{\text{coll}}/2$	2048	$2.8 \cdot 10^6$ μm^2	2.8 mm^2
LLR MEM	85	8	$\left\lceil \frac{n}{P} \right\rceil$	32	2550 μm^2	0.22 mm^2
ACC MEM	85	8	$\max\left(\left\lceil \frac{n}{P} \right\rceil, \left\lceil \frac{m}{P} \right\rceil\right)$	32	2550 μm^2	0.22 mm^2
					Total	4.24 mm^2

In the case of the proposed code, we have $E = 6885$, $n = 2040$, and $m = 1020$, hence $\eta_{\text{sd}} = 0.69$. It is extremely interesting to compare this figure with the one resulting when only partitioning and scheduling techniques are applied on the same code. In that case, we use for D_{coll} the expression as reported in (14), leading to

$$\eta_{\text{coll}} = \frac{1}{\alpha} \tag{23}$$

and with $\alpha = 4.28$ it turns out that $\eta = 0.23$, nearly three times less than the one obtainable with the proposed method.

If we are interested in the throughput for ten iterations, on the same code it turns out that $T_{\text{coll}} = \eta_{\text{coll}} \hat{T} = 0.074 f_{\text{CK}}$ while SD achieves $T_{\text{sd}} = \eta_{\text{sd}} \hat{T} = 0.22 f_{\text{CK}}$. Thus, the proposed approach achieves a throughput speedup of nearly three for the same decoding architecture and the same clock frequency, as reported in Table 3.

4.3. Synthesis results and performance

The partially parallel architecture previously discussed has been described in VHDL, synthesized and mapped on a 0.13 μm CMOS standard cell technology, considering the (2040, 1020) 1/2 rate code.

After logical synthesis and mapping, the maximum combinational delay was $t_{\text{pd}} = 2.5$ nanoseconds which corresponds to a clock frequency of $f_{\text{CK}} = 400$ MHz. As far as the decoding throughput, the split decoding solution is able to achieve 88 Mbps with 10 iterations while the direct mapping of the same code on the same architecture leads to a throughput of 29.6 Mbps. This means that the proposed solution can achieve the same throughput as a straightforward one using a clock frequency as low as one third with respect to the traditional one. Since power dissipation in CMOS circuits is directly related to the clock frequency, power dissipation of one third or a battery life of three time grater can be achieved with the same throughput.

As far as the area occupation is concerned it is possible to separate two main contributions: area that is dedicated to directly implement logical functions and area which is devoted to memory. The former can also be divided into different contributions, namely, PEs and the crossbar switch. Each PE requires 4.61 equivalent kgates leading to an area occupation of almost 392 kgates. The 85×85 crossbar switch requires 172 equivalent kgates. Hence, a total area occupation of 564 kgates is needed to implement the decoder logic. It is important to remark how this area requirement is independent from the chosen scheduling, thus both SD as well as the traditional partially parallel solution require the same area.

Memory is where things are different for the two approaches: these data are collected in Table 4. As it can be seen, some memories have the same dimensions regardless of the scheduling adopted. What really matters is the size of the memory needed to control the crossbar during an iteration. The number of words required depends directly on the number of write cycles: then each word needs to store a complete 85×85 permutation. For the sake of simplicity,

Table 5: Comparisons with state-of-art decoders implementations.

	Previously published architectures			Proposed
	[26]	[27]	[28]	
CN method	3-min	minsum	minsum	Ω
Precision	6 bit	8 bit	6 bit	8 bit
Technology	65 nm	130 nm	90 nm	130 nm
Frequency	400 MHz	83 MHz	109 MHz	400 MHz
Logic gates	520 kgates	420 kgates	380 kgates	564 kgates
Memory bits	500 kbits	106 kbits	100 kbits	544 kbits
Iterations	20	8	20	10
Net throughput	48 MBps	60 MBps	63 MBps	88 MBps
Normalized throughput	960 MBps	480 MBps	1260 MBps	880 MBps
TAR [5]	381	569	1620	321

we decided to store these control signal uncoded using 7 bits to control each crossbar line. This leads to a word length of $85 \times 7 = 595$ bits in either cases. It is also important to note how the memory increase due to S/R MEM is completely negligible when compared to the total area. Summarizing the split decoding approach needs almost 390 equivalent kgates, with respect to more than 700 kgates needed by traditional approach. This results in an area saving of more than one third, bringing also significant possibilities to reduce the overall power dissipation.

To better evaluate the validity of this approach it would be interesting to compare synthesis results to recent LDPC decoder implementations. In Table 5 some implementation results are sketched. Given the particularity of the presented approach both in terms of code design as well as hardware implementation, these comparisons are not straightforward. In order to compare our architecture to similar approaches, we select works that implement IEEE 802.16e LDPC decoder. Throughput figures are obtained considering the largest available 1/2 rate code (i.e., $n = 2304$). Even if this code presents a larger block length than the proposed one, it is important to remark that the total number of edges in this case is $E = 7296$. Comparing this number with the proposed code ($E = 6885$) it turns out that IEEE 802.16e code shows a complexity of 1.06 with respect to the proposed (2040, 1020) code. Under this premises it is then reasonable to consider the two cases almost comparable.

Additionally, it is important to consider how our results have been obtained addressing the (2040, 1020) code: given the flexibility of our partially parallel decoder the same hardware can be exploited to decode also IEEE 802.16e codes. In such a case, obviously, the advantages deriving from SD cannot be exploited anymore.

As can be seen from Table 5, the first two important things to compare are internal data representation and CN implementation: it is worth noting how our architecture represents messages using 8 bits and resorts to the Ω operator as far as the CN is concerned. While this choice tends to produce a larger PE area with respect to minsum approaches, the decoding performances are improved, enabling less decoding iterations to be implemented. In particular, [29] showed how minsum performance tends to be degraded by fixed-point implementations and in presence of irregular codes. Starting from these considerations we expect our area occupation to be larger than the other ones.

Also the technology used is quite dispersed over different values: while area occupation can be properly compared, this does not hold for delay figures. Our design is able to achieve 2.5 nanoseconds period after logical synthesis on a 130 nm technology node: we expect that resorting to a 65 nm technology asynchronous delay would be less.

As far as the area occupation is concerned, it is important to consider both the logic gates contribution as well as the memory requirements. From Table 5 it is possible to observe how our architecture exhibits a logic gate count similar to others, while memory footprint is where things are different. This difference is mainly due to two factors: the use of lookup table operations inside PEs and the use of a crossbar switch as interconnection fabric. While the former could also be relaxed, moving towards a minsum implementation, the latter cannot be avoided due to more complex code structure with respect to the 802.16e case. In that case, in fact, PEs connections can be implemented using simpler permutations that are obtained from identity matrix cyclical rotations. In our case, on the other hand, we need to support arbitrary permutations to enable collision-free decoding of the random part.

It is also interesting to analyze throughput results. It should be noted how decoding net throughput of considered cases falls between 48 and 63 MBps. Our decoder is able to achieve 88 MBps with 10 decoding iterations. If a throughput of 60 MBps is required, decoding iterations can be increased up to 14, enhancing the correction capabilities. In Table 5 we also include an additional throughput figure called *normalized throughput*: these data are extrapolated from the net throughput multiplied by the total number of decoding iterations. We decided to include also these data to better emphasize the effective throughput sustained by each compared architecture.

Finally, to better assess the throughput-area tradeoff we also report a number called TAR [5]. Under this perspective, it is important to observe how the proposed approach presents a TAR lower than others. However, one should not neglect how TAR is directly affected by technology

node, as already noted earlier. Additionally, our architecture is able to decode codes with highly irregular structure, while the others presented are limited to traditional partially structured codes.

5. CONCLUSIONS

In this paper, we present a novel class of partially structured eIRA codes. We also show how a code of this class can essentially perform equivalently to other state-of-art LDPC codes, while preserving some desirable properties that can be exploited when implementing a decoder. Then we focused on the main issues of implementing a partially parallel decoder architecture suitable for this class of codes. In this framework, we devised an alternative decoding approach, namely, the split decoding, which exhibits remarkable advantages over traditional methods. Following this approach memory requirements can be relaxed of more than one third, leading to significant reductions in power dissipation.

Additionally, split decoding enables also the possibility of achieving higher decoding throughput without any hardware impact. Thanks to this higher efficiency, the clock frequency can be reduced further reducing the total power. Finally, we compare an SD-based architecture with three state-of-art LDPC decoders. From this comparison, it can be noted how the proposed architecture presents similar decoding throughput with a larger area occupation, mainly due to internal data representation and interconnection network. Still it is our opinion that the presented approach is valuable, being able to deal with highly irregular parity-check matrices without sacrificing decoding throughput.

As far as future directions are concerned we feel that split decoding performance could be increased borrowing some ideas from shuffled decoding [30]. In particular, shuffled decoding could enhance the parallelism degree between structured and random edges processing, leading to an increased overall throughput.

ACKNOWLEDGMENT

This work was supported by the European Commission funded Network of Excellence NEWCOM++ under the 7th Framework Programme.

REFERENCES

[1] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[2] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[3] 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 Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, 2002.

[4] T. Brack, F. Kienle, and N. Wehn, "Disclosing the LDPC code decoder design space," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, vol. 1, pp. 200–205, Munich, Germany, March 2006.

[5] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Transactions on Circuits and Systems II*, vol. 54, no. 6, pp. 542–546, 2007.

[6] S.-H. Kang and I.-C. Park, "Loosely coupled memory-based edcoding architecture for low density parity check codes," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 5, pp. 1045–1056, 2006.

[7] F. Kienle, M. J. Thul, and N. Wehn, "Implementation issues of scalable LDPC-decoders," in *Proceedings of the 3rd International Symposium on Turbo-Codes & Related Topics*, pp. 291–294, Brest, France, September 2003.

[8] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2002–2009, 2004.

[9] F. Quaglio, F. Vacca, C. Castellano, A. Tarable, and G. Masera, "Interconnection framework for high-throughput, flexible LDPC decoders," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, vol. 2, pp. 124–129, Munich, Germany, March 2006.

[10] E. Kim and G. S. Choi, "Diagonal low-density parity-check code for simplified routing in decoder," in *Proceedings of IEEE Workshop on Signal Processing Systems Design and Implementation (SiPS '05)*, vol. 2005, pp. 756–761, Athens, Greece, November 2005.

[11] L. Dinoi, R. Martini, G. Masera, F. Quaglio, and F. Vacca, "ASIP design for partially structured LDPC codes," *Electronics Letters*, vol. 42, no. 18, pp. 1048–1049, 2006.

[12] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[13] L. Ping, W. K. Leung, and N. Phamdo, "Low density parity check codes with semi-random parity check matrix," *Electronics Letters*, vol. 35, no. 1, pp. 38–39, 1999.

[14] M. Yang, W. E. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate irregular LDPC codes," *IEEE Transactions on Communications*, vol. 52, no. 4, pp. 564–571, 2004.

[15] Y. Zhang, W. E. Ryan, and Y. Li, "Structured eIRA codes with low floors," in *Proceedings of IEEE International Symposium on Information Theory (ISIT '05)*, vol. 2005, pp. 174–178, Adelaide, Australia, September 2005.

[16] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 657–670, 2001.

[17] G. Durisi, L. Dinoi, and S. Benedetto, "eIRA codes for coded modulation systems," in *Proceedings of IEEE International Conference on Communications (ICC '06)*, vol. 3, pp. 1125–1130, Istanbul, Turkey, June 2006.

[18] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive edge-growth tanner graphs," in *Proceedings of IEEE Global Communications Conference (GLOBECOM '01)*, vol. 2, pp. 995–1001, San Antonio, Tex, USA, November 2001.

[19] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC codes with low error floors," in *Proceedings of IEEE International Conference on Communications (ICC '03)*, vol. 5, pp. 3125–3129, Anchorage, Alaska, USA, May 2003.

[20] A. Ramamoorthy and R. Wesel, "Construction of short block length irregular low-density parity-check codes," in *Proceedings of IEEE International Conference on Communications (ICC '04)*, vol. 1, pp. 410–414, Paris, France, June 2004.

[21] L. Dinoi, F. Sottile, and S. Benedetto, "Design of variable-rate irregular LDPC codes with low error floor," in *Proceedings of IEEE International Conference on Communications (ICC '05)*, vol. 1, pp. 647–651, Seoul, Korea, May 2005.

[22] G. Richter and A. Hof, "On a construction method of irregular LDPC codes without small stopping sets," in *Proceedings of IEEE International Conference on Communications (ICC '06)*, vol. 3, pp. 1119–1124, Istanbul, Turkey, June 2006.

[23] L. Dinoi, F. Sottile, and S. Benedetto, "Design of versatile eIRA codes for parallel decoders," to appear in *IEEE Transactions on Communications*.

[24] http://glaros.dtc.umn.edu/gkhome/views/metis.

[25] D. J. C. MacKay, "Encyclopedia of sparse graph codes," http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[26] T. Brack, M. Alles, T. Lehnigk-Emden, et al., "Low complexity LDPC code decoders for next generation standards," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*, pp. 331–336, Nice, France, April 2007.

[27] 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 Journal of Solid-State Circuits*, vol. 43, no. 3, pp. 672–683, 2008.

[28] C.-H. Liu, S.-W. Yen, C.-L. Chen, et al., "An LDPC decoder chip based on self-routing network for IEEE 802.16e applications," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 3, pp. 684–694, 2008.

[29] D. Oh and K. K. Parhi, "Performance of quantized min-sum decoding algorithms for irregular LDPC codes," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '07)*, pp. 2758–2761, New Orleans, La, USA, May 2007.

[30] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, 2005.