IEICE TRANS. FUNDAMENTALS, VOL.E89–A, NO.7 JULY 2006

PAPER

VLSI Design of a Fully-Parallel High-Throughput Decoder for Turbo Gallager Codes

Luca FANUCCI[†a)], ***Member***, **Pasquale CIAO**[†], ***and*** **Giulio COLAVOLPE**[††], ***Nonmembers***

SUMMARY The most powerful channel coding schemes, namely those based on turbo codes and low-density parity-check (**LDPC**) Gallager codes, have in common the principle of iterative decoding. However, the relative coding structures and decoding algorithms are substantially different. This paper presents a 2048-bit, rate-1/2 soft decision decoder for a new class of codes known as Turbo Gallager Codes. These codes are turbo codes with properly chosen component convolutional codes such that they can be successfully decoded by means of the decoding algorithm used for LDPC codes, i.e., the belief propagation algorithm working on the code Tanner graph. These coding schemes are important in practical terms for two reasons: (i) they can be encoded as classical turbo codes, giving a solution to the encoding problem of LDPC codes; (ii) they can also be decoded in a fully parallel manner, partially overcoming the routing congestion bottleneck of parallel decoder VLSI implementations thanks to the locality of the interconnections. The implemented decoder can support up to 1 Gbit/s data rate and performs up to 48 decoding iterations ensuring both high throughput and good coding gain. In order to evaluate the performance and the gate complexity of the decoder VLSI architecture, it has been *synthesized in a 0.18 μm standard-cell CMOS technology.*

key words: *low-density parity-check (LDPC) codes, belief propagation, iterative decoding, VLSI architectures, parallel decoder architectures*

1. Introduction

In recent years the rapid developments in the field of digital transmission systems and microelectronic technologies have led to the discovery or rediscovery of more and more powerful channel coding techniques like turbo codes and low-density parity-check (LDPC) codes, which achieve near Shannon limit decoding performance.

Turbo codes were introduced for the first time in 1993 by Berrou and Glavieux [1], [2], but the problem of an efficient hardware implementation of the turbo decoder is still open. The iterative nature of the decoding algorithm, in fact, requires at each iteration the computation and storage of a large amount of information so increasing the decoder complexity. Moreover, a great number of iterations is required to obtain a high coding gain, thus it is very difficult to satisfy the high-throughput, low-power and low-latency requirements imposed by modern applications.

The growing interest in the field of iterative decoding, stimulated the rediscovery of the LDPC codes. LDPC codes have a performance, in the original regular form proposed by Gallager in 1963 [3], only slightly inferior to that of turbo codes, and even superior in their irregular version [4], [5]. A great advantage of these codes is the absence of a serial dependency in the decoding algorithm that can, therefore, be fully parallelized and potentially implemented at high speed. However, an effective hardware implementation of a parallel decoder is still an open issue since problems of gate complexity and especially routing congestion arise. In order to overcome these problems the current trend is the construction of implementation-oriented codes [6]–[10].

A new class of codes, originally proposed in [11], is considered in this paper. These codes are based on code concatenation and can be easily decoded at high speed by means of the same message-passing decoding algorithm used for LDPC codes. The resulting coding schemes and the corresponding decoding algorithms combine the advantages of turbo codes and LDPC codes. In fact, the coding structure is simply based on parallel or serial concatenation of simple convolutional codes through interleavers. The component convolutional codes are constrained to have specific algebraic properties which ensure that message-passing decoders can work successfully on the Tanner graph of the overall code. In this way a new architecture results which can be effectively implemented with simple coding and decoding operations. In terms of decoding complexity, this new class of codes represents a valid alternative to classical turbo codes and give a solution to the *encoding problem* of LDPC codes [12]. The greatest advantage of these codes is that they partially overcome the routing congestion problem which is the main bottleneck of parallel decoder VLSI implementations. In fact as classical LDPC codes, they can be decoded using a fully parallel structure, but they have an additional property known as *locality of the interconnections*, i.e., not all the links in the code Tunner graph are pseudoramdom, but some regular links are present, too. For these codes, a parallel VLSI decoder architecture that allows obtaining a very good performance in terms of bit error rate (BER) while supporting data rate up to 1 Gbit/s, is presented.

After this Introduction, Sect. 2 briefly reviews the belief propagation algorithm. Section 3 gives a description of the code. Section 4 describes the two possible categories in which the various hardware implementation architectures of the decoder can be grouped. Section 5 gives a detailed description of the implemented VLSI architecture and its sub-

Manuscript received June 8, 2005.
Manuscript revised October 25, 2005.
Final manuscript received March 22, 2006.

[†]The authors are with the University of Pisa, Dept. of Information Engineering, Via Caruso, I-56122 Pisa, Italy.
[††]The author is with the Università di Parma, Dipartimento di Ingegneria dell'Informazione Parco Area delle Scienze 181A, I-43100 Parma, Italy.
a) E-mail: luca.fanucci@iet.unipi.it

DOI: 10.1093/ietfec/e89–a.7.1976

blocks. In Sect. 6 the adopted design, verification flow and relevant CMOS implementation results are presented, while in Sect. 7 a performance comparison has been carried out with both turbo and LDPC decoders in terms of hardware requirements. Finally conclusions are drawn in Sect. 8.

2. Belief Propagation Algorithm

A Tanner graph [13], which is a graph representation of a generic linear code, is composed (Fig. 1) of two classes of nodes: variable nodes (VNs) corresponding to the code bits, and check nodes (CNs) corresponding to the code constraints. Check nodes are connected by *edges* to the variable nodes on which they depend. The number of edges of a node of each kind defines the degree of the node itself.

The message-passing decoding algorithms, originally devised in [3] for LDPC codes, are based on an iterative exchange of messages along the edges of the code Tanner graph. This class of decoding algorithms includes the infinite precision belief propagation algorithm as well as low-complexity algorithms that closely approximate in performance belief propagation. We now briefly review the belief propagation algorithm. Denoting by L the codeword length, by $\mathbf{x} = (x_1, x_2, \ldots, x_L)$ the transmitted codeword and by $\mathbf{y} = (y_1, y_2, \ldots, y_L)$ the received codeword, the so called LLRs (*Log Likelihood Ratios*) of the codeword bits, defined as

$$LLR(x_i) = \ln\left(\frac{\Pr(x_i = 0|\mathbf{y})}{\Pr(x_i = 1|\mathbf{y})}\right) \tag{1}$$

are estimated iteratively. The sign of the LLR corresponds to the estimated bit, while its magnitude provides the reliability of this estimate. Let us consider a Tanner graph with variable nodes of degree d_v and check nodes of degree d_c. At the j-th iteration, a variable node v_n processes the messages $\lambda_{c_m,v_n}^{(j-1)(i)}, i = 1, 2, \ldots, d_v$, received from the neighboring check nodes c_m at the previous iteration, and sends back to them the following messages [3], [14]:

$$\lambda_{v_n,c_m}^{(j)(k)} = \lambda_{v_n}^{(0)} + \sum_{\substack{i=1 \\ i\neq k}}^{d_v} \lambda_{c_m,v_n}^{(j-1)(i)} \tag{2}$$

where $\lambda_{v_n}^{(0)}$ is the channel output corresponding to the considered codeword bit and $\lambda_{c_m,v_n}^{(-1)(i)} = 0$. At the same time, the following estimate is produced for the LLR of the code bit corresponding to the variable node v_n:

$$LLR^j(x_n) = \lambda_{v_n}^{(0)} + \sum_{i=1}^{d_v} \lambda_{c_m,v_n}^{(j-1)(i)}. \tag{3}$$

The check node c_m processes messages $\lambda_{v_n,c_m}^{(j)(i)}, i = 1, 2, \ldots, d_c$, and sends back to the neighboring variable nodes the messages $\lambda_{c_m,v_n}^{(j)(i)}$ whose sign and magnitude are given by [3], [14]:

$$\operatorname{sign}(\lambda_{c_m,v_n}^{(j)(i)}) = \prod_{\substack{k=1 \\ k\neq i}}^{d_c} \operatorname{sign}(\lambda_{v_n,c_m}^{(j)(k)}) \tag{4}$$

$$\left|\lambda_{c_m,v_n}^{(j)(i)}\right| = \phi^{-1}\left\{\sum_{\substack{k=1 \\ k\neq i}}^{d_c} \phi\left(\left|\lambda_{v_n,c_m}^{(j)(k)}\right|\right)\right\} \tag{5}$$

where

$$\phi(x) = \log\left[\tanh\left(\frac{x}{2}\right)\right] \tag{6}$$

The decoding algorithm proceeds iteratively until the code parity-check constraints are all verified or a maximum number of iterations is reached.

Although this decoding algorithm is provably optimal for bipartite graphs without cycles [14], in practice it is only necessary to avoid cycles of length up to 4 to obtain a good performance [15]. Other message-passing algorithms can be devised based on different types of messages, possibly taking on values in some finite alphabet, and different updating rules [14]. All these algorithms can be applied to decode the codes described in the following sections.

A *message-passing schedule* is the specification of the order in which messages are updated. Usually, the so-called *flooding schedule* is adopted to decode LDPC codes: in each iteration, all variable nodes and subsequently all check nodes, pass new messages to their neighbours. As can be easily understood, this schedule is well suited for a fully parallel implementation of the decoder.

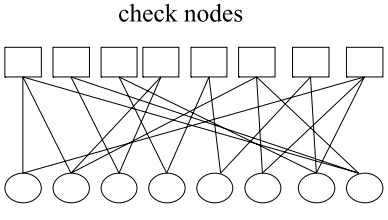


Fig. 1 Tanner graph representation.

3. Turbo Gallager Codes

3.1 Decoding of a Single Convolutional Code by Means of the BP Algorithm

In this section, we describe a class of convolutional codes that can be decoded by means of the described belief propagation algorithm [11], [16], [17]. Let us consider a convolutional code, truncated to a block code using tailbiting with no rate loss [18]. As mentioned in the previous section, a message-passing decoding algorithm is a powerful decoding technique if applied to codes whose Tanner graph has no cycles of length less than or equal to 4. In [16], [17], the conditions a convolution code must satisfy such that the corresponding Tanner graph has no cycles of length 4 are derived. We can consider as a relevant case for our purposes a rate-1/2 recursive systematic code.

A rate-1/2 recursive systematic code can be described by means of the corresponding parity-check equation. At discrete-time m, the parity bit p_m produced by the encoder, denoting by i_m the information bit, is given by:

$$p_m = \sum_{j=2}^{J_2} p_{m-\beta_j} + \sum_{j=1}^{J_1} i_{m-\alpha_j}\,,\ m = 1, 2 \ldots N \tag{7}$$

where J_ℓ, α_j and β_j are suitable integers which specify the code and N is the total number of transmitted parity bits. It is understood that, if the lower limit of a sum is greater than the upper limit, no terms must be summed. Therefore, J_1 and J_2 denote the number of information and parity bits in each parity-check, respectively. In order to simplify the notation, we collect integers α_j and β_j in the following vectors: $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_{J_1})$ and $\boldsymbol{\beta} = (\beta_1, \beta_1, \ldots, \beta_{J_2})$. The corresponding Tanner graph can be easily built. In fact, for each discrete-time instant m we have two variable nodes corresponding to the information bit i_m and the parity bit p_m, and a check node representing Eq. (7). This check node will be connected to some variable nodes according to (7). The resulting graph is regular if $J_1 = J_2$.

For the described rate-1/2 recursive systematic code the following statements apply [16]:

1. if $J_1 = 1$ ($J_1 = 2$) and $J_2 = 2$ ($J_2 = 1$), the bipartite graph has no cycles;
2. if $J_1 = 2$ and $J_2 = 2$, denoting by $\Delta\alpha = \alpha_2 - \alpha_1$ and $\Delta\beta = \beta_2 - \beta_1$, the graph has shortest cycles of length 4 if $\Delta\alpha = \Delta\beta$, of length 6 if $\Delta\alpha \neq \Delta\beta$ and $\Delta\alpha = 2\Delta\beta$ or $2\Delta\alpha = \Delta\beta$, otherwise the graph has shortest cycles of length 8;
3. if $J_1 \geq 2$ and $J_2 \geq 2$, the graph can have at most cycles of length 6 if all differences $\alpha_j - \alpha_p$ and $\beta_n - \beta_m$ ($j > p$, $n > m$) are distinct.

3.2 Concatenated Schemes

In this section, we discuss the concatenation of the convolutional codes previously described. The resulting coding schemes are classical turbo codes or serially concatenated codes but may be decoded as LDPC codes. For this reason, these new concatenated schemes are nicknamed "turbo Gallager codes" (**TGC**) [11], [16], [17].

A turbo code is the parallel concatenation of two component codes (usually recursive systematic convolutional codes) through a *nonuniform* interleaver [2]. In the literature, the decoding of these codes is based on a sub-optimal iterative process in which each component decoder takes advantage of the extrinsic information produced by the other decoder at the previous step. This iterative decoding process is made possible by employing soft-output component decoders, usually based on the BCJR algorithm [2]. More generally, code networks can be designed by concatenating a few convolutional component codes in mixed parallel and serial configurations [19], [20].

TGC are based on the following two key concepts:

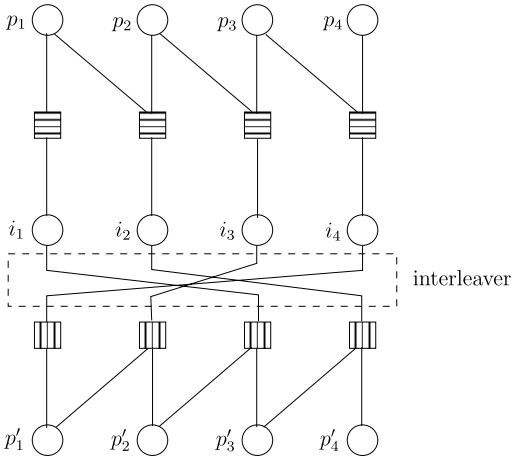


Fig. 2 Overall Tanner graph for a turbo code.

1. at the encoder, the use of the recursive convolutional codes described in the previous section in code networks as component codes;
2. at the decoder, the use of a message-passing algorithm which works on the bipartite graph of the *overall code* by adopting the flooding schedule.

Let us consider the overall Tanner graph of a turbo code. In Fig. 2 we show, as an example, the graph for a rate-1/3 code. The component codes are rate-1/2 recursive systematic codes with parity check equation

$$p_m = p_{m-1} + i_m \tag{8}$$

and, for simplicity, the code termination is ignored. In the upper part of the graph, the information bits i_m and the parity bits p_m of the first component code are connected with the corresponding check nodes. The information bits are also connected, in a permuted order, with the lower check nodes related to the second component code having parity bits p'_m. The decoding process can be performed by means of a message-passing decoder working on the Tanner graph of the overall code with the flooding schedule. In this case, the variable nodes *of both codes* and subsequently the check nodes *of both codes* are activated simultaneously.

Note that, even if the graphs of each component code have shortest cycles of length 6, the overall Tanner graph may have cycles of length 4, depending on the interleaver used. As an example, consider a turbo code composed of rate-1/2 convolutional codes. In this case, a sufficient condition to avoid the appearance of cycles of length 4 in a parallel concatenation is that information symbols which differ for less than $\max_{i,j}\{\alpha_i - \alpha_j\}$ have a distance, after the interleaver, greater than $\max_{i,j}\{\alpha_i - \alpha_j\}$. This may be obtained by the use of an *S-random* interleaver with $S = \max_{i,j}\{\alpha_i - \alpha_j\}$ [21].

An analysis of these coding schemes may be performed by the method of density evolution, which may also be employed as a design criterion to select the component codes. Density evolution is a tool for jointly analyzing the code and the message-passing decoding algorithm, evaluating

the relevant performance when averaged over the ensemble of codes with common degree distribution of variable and check nodes [14], [15]. This method, which assumes absence of cycles, analyzes the evolution of the probability density functions of the messages propagating in the graph during decoding with the aim of determining the critical channel parameter (the so-called *threshold*). The *threshold* separates the region where reliable transmission is possible from that where it is not [14].

The degree distribution of variable and check nodes, necessary for the application of the density evolution method, can be concisely described by two polynomials $\lambda(x)$ and $\rho(x)$ of the form [5], [15]

$$\lambda(x) = \sum_{i=1}^{d_\ell} \lambda_i x^{i-1}, \quad \rho(x) = \sum_{i=2}^{d_r} \rho_i x^{i-1} \tag{9}$$

where λ_i and ρ_i are the fractions of edges belonging to degree-i variable and check nodes and d_ℓ and d_r are the maximum degrees of variable and check nodes, respectively. For turbo Gallager codes, these two polynomials depend on the type of (i) concatenation, (ii) interleaver, (iii) puncturing (if any), and (iv) component codes.

3.3 A Case Example

Let us consider, as a case example, a rate-1/2 turbo code obtained from two identical rate-1/2 recursive systematic convolutional codes by means of puncturing. Let us assume that the interleaver is odd-odd, i.e., bits in odd (even) position remain in odd (even) position after the permutation, and that the odd parity bits of the first encoder and the even parity bits of the second encoder are punctured. When a bit is punctured, the corresponding variable node in the graph disappears along with the neighboring check nodes and the edges connected to them. As a consequence, by means of simple considerations on the overall Tanner graph, it can be shown that the corresponding degree polynomials are

$$\begin{aligned} \lambda(x) &= \frac{J_1}{J_1+J_2} x^{J_1-1} + \frac{J_2}{J_1+J_2} x^{J_2-1} \\ \rho(x) &= x^{J_1+J_2-1} \end{aligned} \tag{10}$$

By properly choosing the integers J_1 and J_2 we may obtain simple irregular codes. However, if $J_1 = J_2$ a regular code results. As an example, if $J_1 = J_2 = 3$ we obtain a (3,6) regular LDPC code. In order to choose the component codes and to analytically evaluate the achievable performance, the density evolution technique is used to compute the threshold of the signal-to-noise ratio. The threshold is defined as the value of the signal-to-noise ratio above which the expected fraction of incorrect messages approaches zero when the number of iterations increases, assuming absence of cycles. In the case of turbo Gallager codes such as those described by (10), these threshold values are shown in Table 1, for different values of $J = J_1 + J_2$, along with the corresponding optimal values of J_1 and J_2. It can be noted that, in this case, an optimal value of J exists and corresponds to the couple $(J_1, J_2) = (2, 4)$ (or equivalently $(J_1, J_2) = (4, 2)$). The corresponding threshold value is 0.797 dB whereas the Shannon limit for binary inputs and rate 1/2 is 0.187 dB. Lower values of the threshold (0.584 dB) may be obtained by using different component codes, i.e., an *asymmetric* turbo code, in order to make the overall code more irregular.

Table 1 Thresholds for a rate-1/2 turbo code with equal component codes, an odd-odd interleaver and an odd-even puncturing.

J	J_1	J_2	E_b/N_0(dB)
3	1 (2)	2 (1)	10.712
4	2	2	3.277
5	2 (3)	3 (2)	1.078
6	2 (4)	4 (2)	0.797
7	2 (5)	5 (2)	0.863
8	2 (6)	6 (2)	1.046
9	2 (7)	7 (2)	1.262
10	2 (8)	8 (2)	1.484

3.4 The Implemented Code

In the previous section, we derived an optimal value of the couple $(J_1, J_2) = (2, 4)$ for TGCs described by (10). However, in order to minimize the decoder complexity we can choose the corresponding optimal value of $J = 6$ and $J_1 = J_2 = 3$ so obtaining a (3,6) regular TGC code. The component codes we consider, have $\boldsymbol{\alpha} = (0, 3, 4)$ and $\boldsymbol{\beta} = (0, 14, 34)$, i.e., the parity check equation is

$$p_m = p_{m-14} + p_{m-34} + i_m + i_{m-3} + i_{m-4}\,. \tag{11}$$

We use tailbiting in both component codes, as described in [22] for recursive codes, in order to avoid a rate loss due to code termination. As both component codes are punctured, integers β_j must be all even in order to avoid that all check nodes disappear. In addition, integers α_i are chosen small in order to reduce the probability of appearance of cycles of length 4 due to the interleaver. On the contrary larger values of the integers β_j are chosen, in order to allow the propagation of a message to a wide region of the graph in a few iterations.

To demonstrate the good performance of the chosen code, we compare it with the "classical" (3,6)-regular LDPC codes [23], [24]. For the considered TGC, different codeword lengths, namely, $L = 496$, 2624, and 8000, which correspond to interleaver lengths of 248, 1312, and 4000, respectively, are considered. We emphasize the simplicity of obtaining codes with different codeword lengths: the component codes remain the same and only the interleaver size is changed. As usual for LDPC codes, computer simulations have been performed by using early detection of convergence, i.e., at each iteration the decoder controls if checks are all satisfied, stopping decoding when they are. The maximum allowed number of iterations is 400 for TGCs as well as for (3,6)-regular LDPC codes [23], [24] of length $L = 504$, 2640, and 8000, respectively, reported in Fig. 3 as a comparison [24]. Moreover, the performance of the TGC with the same component codes and codeword length of

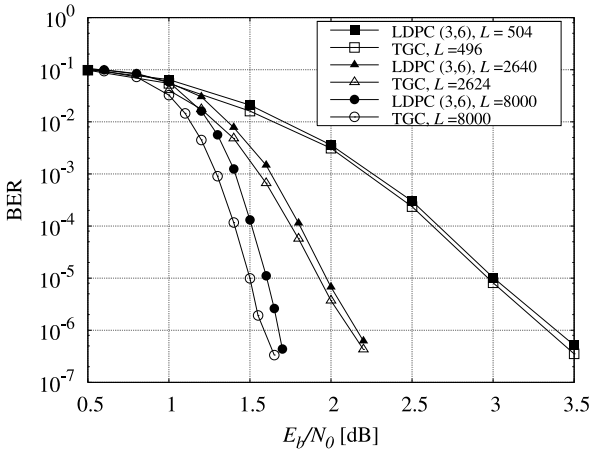


Fig. 3 BER of the considered turbo Gallager codes and comparison with regular LDPC codes for different codeword lengths L.

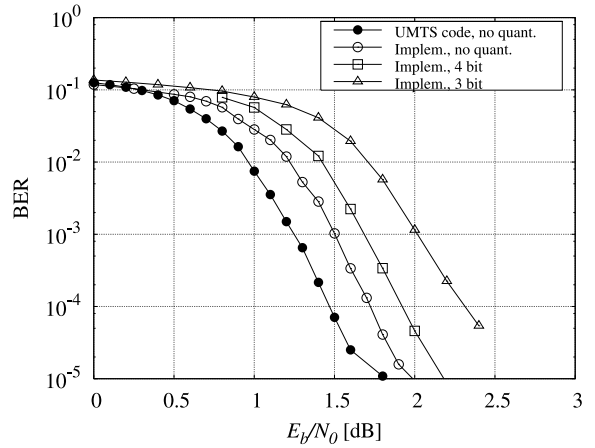


Fig. 4 Performance comparison with the turbo code used in the third generation (3G) universal mobile telecommunications service (UMTS).

$L = 2048$ bits is shown in Fig. 4. In this case, we simulated the infinite precision belief propagation algorithm as well as other two message passing decoding algorithms based on a quantization with 3 and 4 bits respectively. In the figure, the performance of a turbo code used in the third generation (3G) universal mobile telecommunications service (UMTS) [25] with the same codeword length and code rate is also shown for comparison.

We may conclude that, for a given codeword length, the considered TGCs have a performance only slight inferior to that of the turbo code used in 3G UMTS but they can be decoded in a fully parallel manner. On the other hand they perform as well as (or slightly better than) classical (3,6)-regular LDPC codes, despite their simpler construction and encoding. Moreover the Tanner graph of this code is characterized by the locality of the interconnection, which alleviates the routing congestion problem that affects the implementation of parallel decoders for classical LDPC codes [26]. For these reasons a VLSI implementation of a fully parallel decoder for the above mentioned TGC with codewords of length 2048 was carried out and will be described in the following sections.

4. Serial vs. Parallel Architectures

Since there is no serial dependency in the computations of the belief-propagation algorithm, this can be accomplished in several ways. Each node of the graph can be associated to a different processing element (PE), resulting in a "fully parallel" architecture. On the other hand, a PE can perform the functionality associated to multiple nodes of the graph (hardware sharing/multiplexing) so obtaining a serial or mixed serial-parallel architecture. In the following sections a brief comparison between these types of architectures will be presented.

4.1 Parallel Architectures

A parallel architecture directly maps the Tanner graph in hardware: each node of the graph is implemented by a different functional unit while the connectivity between them is obtained by physical edges. Typically only one flip-flop barrier is used between check and variable nodes. The greater advantage of this kind of architecture is that it can support data-rate greater than any other architecture for iterative decoding. The clock frequency can be written as follow:

$$f_c = \frac{N_{iter}}{T \cdot L \cdot r} \tag{12}$$

where L is the codeword lenght, r is the code rate, N_{iter} is the maximum number of iterations performed by the decoder, and $1/T$ is the data rate. Equation (12) shows that the clock frequency is reduced by a factor Lr/N_{iter} with respect to the data rate. For example, for 2048 variable nodes, a rate-1/2 code and a maximum of 32 iterations, a clock frequency of 32 MHz is required for obtaining a 1 Gbit/s data rate. Another advantage that can be obtained using this architecture is that the power consumption is relatively small thanks to the reduced clock frequency and the small activity factor [26].

Moreover, the amount of control logic is reduced to the one used for stopping the decoding process when the algorithm converges. No memory is required for the connectivity and the whole logic necessary for its access is so avoided. The obvious disadvantages are the overall hardware complexity of the decoder and especially the need of managing a complex routing. Moreover this architecture can be used only for the specific code for which has been designed and so it is completely not flexible both in terms of block length and node connectivity.

4.2 Serial Architectures

In serial or mixed serial-parallel architectures a principle known as hardware-sharing is applied. This consists of instantiating only one (as shown in Fig. 5) or a certain number of working units for each different functional node that must be implemented.

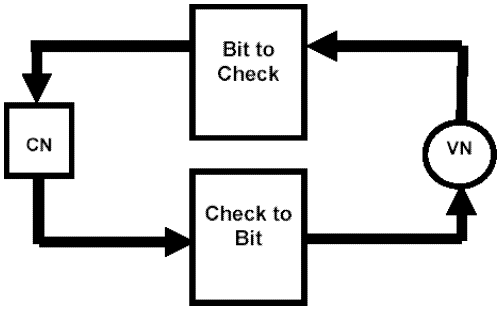


Fig. 5 Serial architecture.

This kind of solution is totally flexible and characterized by a relatively small hardware complexity. However, it suffers from the opposite problems with respect to the parallel solution: smaller data-rate and higher power consumption and complexity of the control logic. Moreover, it requires a great amount of memory to ensure the correct connectivity between check and variable nodes, since all the messages at each iteration have to be stored.

For mixed solutions another problem arises. If more PEs are instantiated for each node, it can easily happen that during one iteration more than one working unit tries to access the same memory bank (a collision happens) even if the total memory is split in more banks. This problem is known as "collision problem" and can be solved designing a code intrinsically collision-free or optimising the memory access independently from the code. The first solution causes a loss of flexibility, while the second one causes performance degradation in terms of latency, area complexity or memory size.

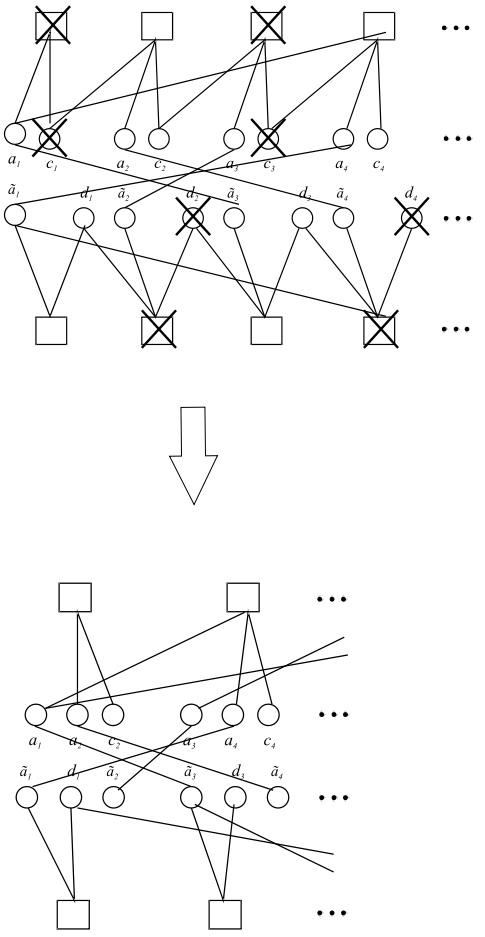


Fig. 6 Code Tanner graph.

5. VLSI Architecture of a Parallel Decoder

The main challenge when implementing a parallel architecture for the message-passing algorithm is represented by the interconnections between the functional nodes [26]. The code described in Sect. 2 has a property, known as locality, which in part allows overcoming this problem. For this reason a parallel architecture for decoding turbo Gallager codes has been implemented. To explore the performance and implementation issues of the decoder, the code described in Sect. 3 characterized by a block size $L = 2048$ and a rate $r = 1/2$ component code has been used. The overall code design is originally characterized by a rate $r = 1/3$ and a block size of 3072 bits. The Tanner graph of such a code would have 3072 variable nodes and 2048 check nodes. As already mentioned, half of the redundant bits has been punctured so reducing the code rate to 1/2. The variable nodes corresponding to the punctured bits and the associated check nodes must be erased from the Tanner graph. The final structure has been obtained by splitting the graph and duplicating the nodes corresponding to the variable nodes associated to the information bits. Figure 6 exemplifies the procedure described to obtain the code Tanner graph. The obtained structure is totally symmetric and so it is well suited to a VLSI implementation.

5.1 Decoder Architecture

The 1536 variable nodes and 512 check nodes on each side of the graph have been grouped into clusters (Fig. 7). Each cluster contains $d = 48$ variable nodes and $d/3 = 16$ check nodes, being the total number of clusters $w = 32$ for each side. The links between check and variable nodes are not all within the cluster, but there is an interconnection network between the clusters.

This interconnection network consists of 27 links in the upper side and 28 in the lower side because of the puncturing that is dual. Moreover, there is one bypass-link for connecting the cluster i to the cluster $i+1$, this solution has been pursued for not constraining too much the cluster dimension which is related to the maximum number of iterations that can be performed by the decoder.

The links between check and variable nodes are at fixed distance, while the pseudo-random links are reserved to the ones that connect the upper and lower side of the decoder (shuffle network). This propriety improves the routing-congestion.

Data input and output are performed in a block-parallel

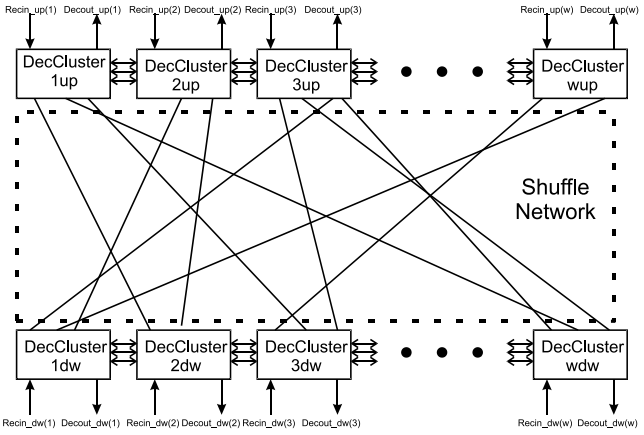


Fig. 7 Decoder structure.

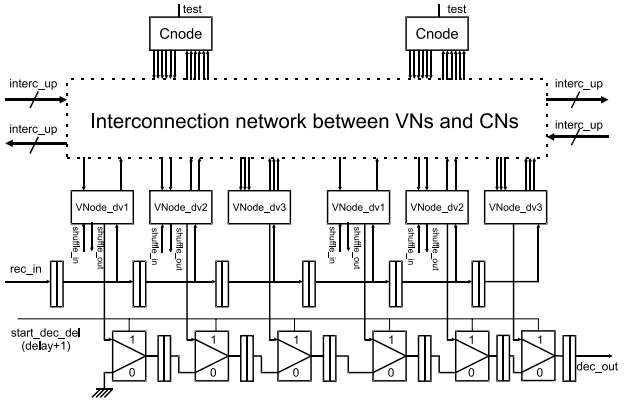


Fig. 8 Clusters internal structure.

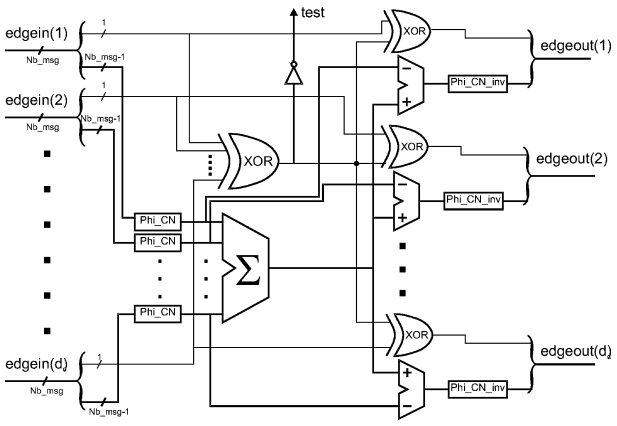


Fig. 9 Check node structure.

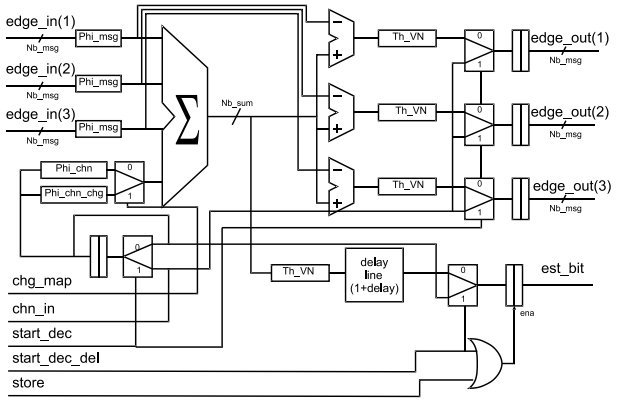


Fig. 10 Three-edge variable node structure.

manner with block of w data both in the upper and lower side of the decoder. At each clock cycle, each cluster acquires a new data to be decoded while a decoded data is read-out. This is possible because each cluster is structured on a three-level pipeline (Fig. 8). The clock cycles required for loading the $(n+1)$th decoding block are d, being wd the total number of symbols to be loaded in each side of the decoder for a single codeword and w the number of data loaded at each clock cycle. The same time is needed for reading-out the $(n-1)$th decoded block. During this time, the block n is also decoded. The decoding process is carried out in a parallel manner. Hence, each iteration involves a clock cycle and a decoding cycle can involve at most d iterations.

5.2 Check Nodes

Each check node is a combinatorial block with d_c =6 3-bit input messages coming from variable nodes and d_c 3-bit output messages going to variable nodes. Moreover, it generates a test bit which is asserted when the check is verified. Each message passed between the check and variable nodes is represented as a sign bit and two magnitude bits.

Figure 9 shows the internal structure of a check node. The magnitude of the output edge i is generated summing the mapped value of the magnitude of all the incoming messages on the edges other than i and applying to this sum a reverse map. The sign of the output edge i is obtained by XOR-ing the sign of all incoming message on the edge other than i.

The check equation is tested using the extrinsic information instead of the estimated bit at the current iteration. This causes a performance loss in terms of BER, but provides an improvement of the routing factor.

5.3 Variable Nodes

Three types of variable nodes have been implemented depending on the number of input edges. Because of the split of the Tanner graph, each node in the upper (lower) side of the graph associated to an information bit must be connected to one or two check nodes in the lower (upper) part of the graph. This connectivity is realized using the corresponding variable node in the lower (upper) part of the decoder that receives the same inputs through the shuffle network.

5.3.1 Three-Edge Variable Node

A three-edge variable node (Fig. 10) has d_v =3 3-bit input messages coming from check nodes and d_v 3-bit output messages going to check nodes. At the start signal, the decoding

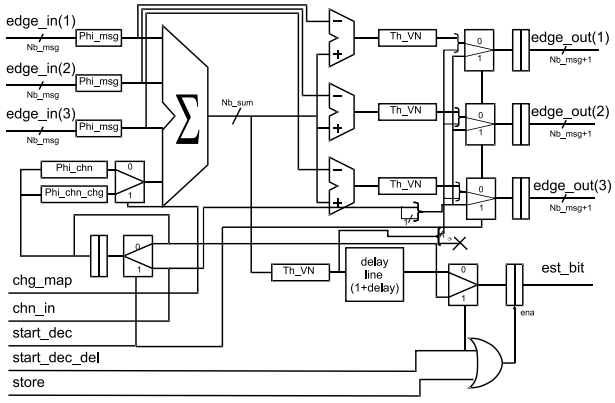


Fig. 11 Modified structure of a three-edge variable node.

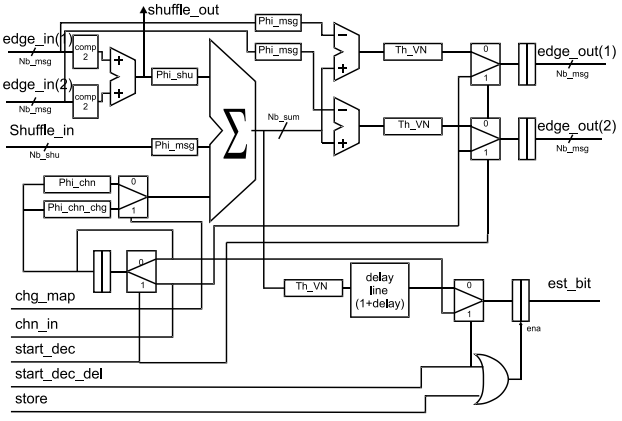


Fig. 12 Two-edge variable node structure.

of a new block is started and the channel messages are driven on the edges. For subsequent iterations, the incoming edges and the received value are mapped and summed. This sum, converted back to a sign-magnitude representation, represents the soft-output at current iteration. The message sent on an edge is computed by subtracting the message incoming on the corresponding edge and converting the resulting value back to a sign-magnitude representation.

In Fig. 11 is shown a three-edge variable node whose output edges have an additional bit passed to the check nodes with the aim of verifying the check equation using the current estimation. In this case the same modifications to the other types of variable nodes apply.

5.3.2 Two-Edge Variable Node

The functionality of the three kinds of variable nodes is substantially the same, but a two-edge variable node (Fig. 12) has two inputs coming from check nodes and one shuffle input, and the same applies to the outputs. The shuffle output is computed by summing the input edge, while the shuffle input is used as an input edge because it comes from a check node and is only bypassed by a one-edge variable node.

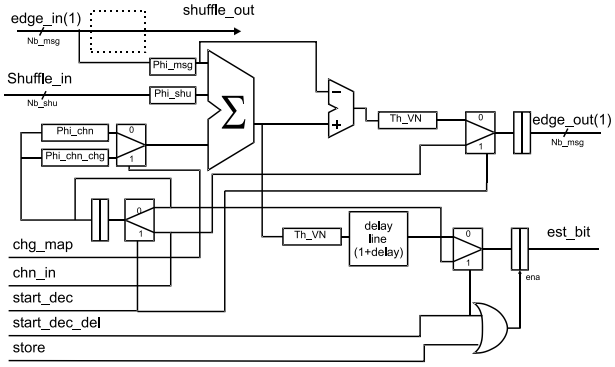


Fig. 13 One-edge variable node structure.

5.3.3 One-Edge Variable Node

A one-edge variable node (Fig. 13) has only one input coming from check nodes and one shuffle input which comes from a two-edge variable node. The shuffle input is the sum of two messages, thus a different map must be applied to it. This new map has to satisfy the following equation:

$$Phi\_shu(a+b) = Phi\_msg(a) + Phi\_msg(b) \qquad (13)$$

The same map has been used also in the two-edge variable node, even if is not necessary. This has been done to easy the implementation, as the edge sum has been already computed to evaluate the shuffle output. The shuffle output of a one-edge variable node is simply equal to the corresponding incoming message.

6. VLSI Design and Verification

The decoder has been described using VHDL hardware description language following a meet-in-the-middle approach. Due to the modularity of the architecture, all the processing elements (variable and check nodes) have been developed first and verified as stand-alone Intellectual Property (IP) macrocell. Afterwards the overall architecture has been derived by instantiating such IPs. This approach provided a speed-advantage in the simulation, synthesis, and data base management of the complex overall architecture. In the following subsections the verification methodology, the synthesis results and an upper bound estimation for the power consumption will be presented.

6.1 Verification Methodology

The design verification has been made using the test-bench set-up shown in Fig. 14. The test has been carried out starting from an high-level description of the decoding process in Fortran. The Fortran code was also in charge of generating the random sequences of coded inputs, that previously summed with additive Gaussian noise, are read in by the decoder. The soft-outputs of the two decoding processes are then compared to verify the functionality of the decoder.

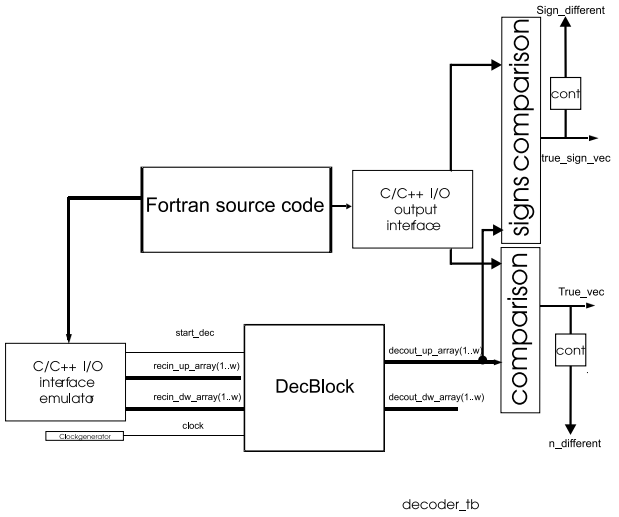


Fig. 14 Decoder testbench.

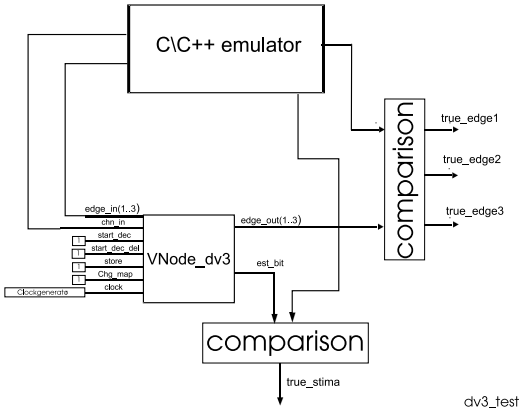


Fig. 15 Single node testbench.

Tests have been carried firstly on each single variable and check node (as in the example shown in Fig. 15). The single nodes can so be reused as stand alone IP for different implementation solutions.

6.2 VLSI Results

To evaluate its performance, the decoder has been synthesized using a 0.18 μm, 1.8 V power supply, six metal levels standard-cells CMOS technology. The obtained results are shown in Table 2, for each block in the hierarchy.

The synthesis has been carried-out targeting a system clock frequency of 50 MHz, which corresponds to a data-rate of 1 Gbit/s. The overall decoder complexity is about 2.1 million gates.

One of the advantages of the parallel solution is that it allows for reducing the power consumption. The circuit switching activity depends on the SNR and hence the frequency. It can be shown [26] that for low SNR values the decoder in not able to correct any packet, so the switching activity is relatively high. For higher values of SNR the decoder quickly corrects packets and the edge values stop changing after a small number of iterations.

Table 2 VLSI Block synthesis results.

Block	# gates	worst delay path (ns)
Cnode	284	7.6
Vnode_dv3	645	9.6
Vnode_dv2	537	10.14
Vnode_dv1	254	6.9
DecCluster_up (DecCluster_dw)	32730	18.22

According to [26], a reasonable upper bound for the switching activity is 10%. The power consumption can be then estimated using the following formula:

$$P_D = pN_{gate}f_{clock}\alpha \tag{14}$$

where α is the *switching activity* and p is a technology factor. For the used technology p = 35 nW/Gate/MHz, thus a rough estimation of the power consumption is 367.5 mW.

7. Comparison with Other Solutions

Only one of the published iterative decoder implementations [26] is characterized by a throughput of the same magnitude of the implemented decoder, while other implementations [27] are characterized by throughput at least one order of magnitude smaller.

A comparison with [26] can be directly carried out in terms of number of equivalent gates. The two solutions are characterized by the same data rate of 1 Gb/s even if there is a small technology-scaling down factor advantage because the decoder implemented in [26] was synthesized using a 0.16 μm CMOS technology. The power consumption of the decoder implemented in [26] is 690 mW at 1.5 V, which is comparable with the one obtained in spite of the disadvantage in terms of technology-scaling down and power supply. The number of gates of the implemented decoder is greater than the one obtained in [26], where the gate complexity is 1750K. On the other hand, we must take into account that 2/3 more variable nodes have been used: 1/3 due to the splitting of the Tanner graph and the others because of the larger codeword length. For the same reason, the number of check nodes is two times the one used in [26]. The great advantage of our solution is that the routing congestion, which is the real bottleneck of the VLSI implementation of a parallel decoder, can be reduced thanks to the locality of the interconnections.

A comparison can be carried out also with the solution presented in [27]. This is characterized by a 1 Mbit/s throughput, 100 mm^2 area in a 0.6 μm 3.3 V CMOS technology. Scaling down the throughput linearly with process technology feature size and the area quadratically with process technology feature size we obtained a 6.7 Mbit/s (assuming a code rate of 1/2) throughput, 135 Kgates complexity (9 mm^2 area) in a 0.18 μm 1.8 V CMOS technology characterized by an average integration density of 15 Kgates/mm^2 like the one we used. It must be noticed that

for the parallel LDPC decoder, a degradation factor should be inserted to take into account the routing factor and the pad area. The gate complexity of the parallel decoder is strictly greater, but it allows for a throughput increase of more than two orders of magnitude.

8. Conclusions

A 2048-bit rate-1/2 soft-decision decoder for a new class of codes known as "turbo Gallager codes" has been described. The decoder can support up to 1 Gbit/s data rate and performs up to 48 decoding iterations ensuring at the same time a high throughput and a good coding gain. A preliminary estimation of the power consumption lead to 367.5 mW for a 1.8 V power supply. This performance has been obtained using a fully parallel architecture for implementing the belief propagation algorithm. The considered code has a performance only slightly inferior to that of turbo codes with the same length and code rate, but it can be decoded using a fully parallel architecture which allows to obtain a throughput of at least an order of magnitude greater than the one obtainable with turbo decoders. On the other hand, it can be also shown that the considered code outperforms the regular LDPC codes, ensuring at the same time simpler encoding and a great benefit in the implementation of a parallel decoder. In fact the locality of the interconnections improves the routing congestion, which is the main bottleneck of a parallel decoder VLSI implementation.

Acknowledgments

Authors wish to thank Giovanni Vanini for his contribution to the early phases of this project. This work has been partially supported by the FIRB *"Reconfigurable platforms for wideband wireless communications"* project of the Italian Ministry for Instruction, University and Research.

References

[1] C. Berrou, A. Glavieux, and P. Thitmajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," Proc. IEEE Intern. Conf. Commun., pp.1064–1070, Geneva, Switzerland, May 1993.

[2] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," IEEE Trans. Commun., vol.44, no.10, pp.1261–1271, Oct. 1996.

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

[4] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman, "Improved low-density parity-check codes using irregular graphs," IEEE Trans. Inf. Theory, vol.47, no.2, pp.585–598, Feb. 2001.

[5] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity check codes," IEEE Trans. Inf. Theory, vol.47, no.2, pp.619–637, Feb. 2001.

[6] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," IEEE Trans. Very Large Scale Inform. Syst., vol.11, no.6, pp.976–996, Dec. 2003.

[7] T. Zhang and K. Parhi, "Join-(3,k)-regular LDPC code and decoder/encoder design," IEEE Trans. Signal Process., vol.52, no.4, pp.1065–1079, April 2004.

[8] H. Zhong and T. Zhang, "Design of VLSI implementation-oriented LDPC codes," Proc. IEEE Semiannual Vehicular Technology Conference (VTC), vol.1, pp.670–673, Oct. 2003.

[9] Y. Chen and D. Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder," Proc. IEEE Global Telecommun. Conf., pp.113–117, Dec. 2003.

[10] V. Nagarajan, N. Jayakumar, S. Khatri, and O. Milenkovic, "High-throughput VLSI implementations of iterative decoders and related code construction problems," Proc. IEEE Global Telecommun. Conf., pp.361–365, Nov.-Dec. 2004.

[11] G. Colavolpe, "Coding schemes based on convolutional codes and message-passing decoding," Italian patent n. MI2002A001438, June 2002, International Patent Application n. PCT/EP03/06337, June 2003.

[12] T.J. Richardson and R.L. Urbanke, "Efficient encoding of low-density parity-check codes," IEEE Trans. Inf. Theory, vol.47, no.2, pp.638–656, Feb. 2001.

[13] R.M. Tanner, "A recursive approach to low complexity codes," IEEE Trans. Inf. Theory, vol.27, pp.533–547, Sept. 1981.

[14] T. Richardson and R. Urbanke, "The capacity of low density parity check codes under message passing decoding," IEEE Trans. Inf. Theory, vol.47, no.2, pp.599–618, Feb. 2001.

[15] S.Y. Chung, G.D. Forney, T.J. Richardson, and R.L. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," IEEE Commun. Lett., vol.5, no.2, pp.58–60, Feb. 2001.

[16] G. Colavolpe, "Design and performance of turbo Gallager codes," IEEE Trans. Commun., vol.52, no.11, pp.1901–1908, Nov. 2004.

[17] G. Colavolpe, "Performance of turbo Gallager codes," Proc. Intern. Symp. on Turbo Codes & Relat. Topics, Brest, France, Sept. 2003.

[18] H.H. Ma and J.K. Wolf, "On tailbiting convolutional codes," IEEE Trans. Commun., vol.34, no.2, pp.104–111, Feb. 1986.

[19] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," IEEE Trans. Inf. Theory, vol.44, no.3, pp.909–926, May 1998.

[20] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-Input Soft-Output modules for the construction and distributed iterative decoding of code networks," European Trans. Telecommun., vol.9, no.2, pp.155–172, March/April 1998.

[21] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," IEEE Intern. Conf. on Commun. ICC'95, pp.54–59, Seattle, USA, June 1995.

[22] P. Ståhl, J.B. Anderson, and R. Johannesson, "A note on tailbiting codes and their feedback encoders," IEEE Trans. Inf. Theory, vol.42, no.2, pp.529–534, Feb. 2002.

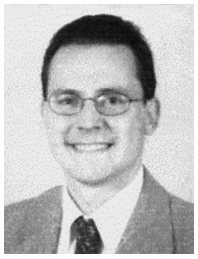
[23] D.J.C. MacKay, "Good error correcting codes based on very sparse matrices," IEEE Trans. Inf. Theory, vol.45, no.2, pp.399–431, Feb. 1999.

[24] D.J.C. MacKay, "Regular LDPC online database," available at the url http://www.inference.phy.cam.ac.uk/mackay/

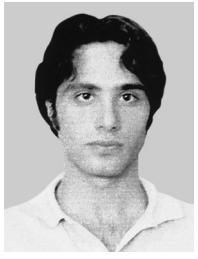
[25] 3rd Generation partnership Project (3GPP), "Technical specification group radio access network multiplexing and channel coding (TDD)," 2004.

[26] 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, March 2002.

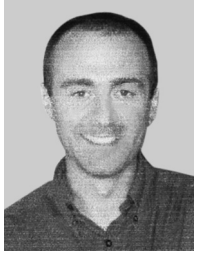
[27] S. Hong and W. Stark, "Design and implementation of a low complexity VLSI turbo-code decoder architecture for low energy mobile wireless communications," J. VLSI Signal Processing, vol.24, no.4, pp.43–57, Jan. 2000.



Luca Fanucci is an Associate Professor of Microelectronics at the Department of Information Engineering of Pisa University. He was born in Montecatini Terme, Italy, in 1965. He received the Doctor Engineer (summa cum laude) and the Research Doctor degrees, both in electronic engineering, from the University of Pisa, Pisa, Italy, in 1992 and 1996, respectively. From 1992 to 1996, he was with the European Space Agency's Research and Technology Center, Noordwijk, The Netherlands, where he was involved in several activities in the field of VLSI for digital communications. In the years 1996–2004 he was a Research Scientist of the Italian National Research Council in Pisa. His research interests include several aspects of design technologies for integrated circuits and systems, with particular emphasis on system-level design, hardware/software co-design and low-power design. The main applications areas are in the field of wireline and wireless communications, satellite communications, low power multimedia and automotive. He is co-author of more than 100 journal and conference papers and co-inventor of more than 10 patents. He is also co-author of the book "An Experimental Approach to CDMA and Interference Mitigation: From System Architecture to Hardware Testing through VLSI Design," Kluwer Academic Publishers, 2004.



Pasquale Ciao was born in Oliveto Citra (SA), Italy, in 1977. He received the Doctor Engineer degree (summa cum laude) from the University of Pisa, Pisa, Italy, in 2002. Since 2003 he has been a Ph.D. student at the Department of Information Engineering of the University of Pisa. His main interests are in the areas of System-on-Chip design, VLSI architectures for signal processing and digital communication systems.



Giulio Colavolpe was born in Cosenza, Italy, in 1969. He received the Dr. Ing. degree in Telecommunications Engineering (cum laude) from the University of Pisa, Italy, in 1994 and the Ph.D. degree in Information Technologies from the University of Parma, Italy, in 1998. Since 1997, he has been at the University of Parma, Italy, where he is now an Associate Professor of Telecommunications. In 2000, he was Visiting Scientist at the Institut Eurecom, Valbonne, France. His main research interests include digital transmission theory, adaptive signal processing, channel coding and information theory. His research activity has led to more than seventy scientific publications in leading international journals and conference proceedings and a few industrial patents. He is also co-author of the book *Detection Algorithms for Wireless Communications, with Applications to Wired and Storage Systems* (New York: John Wiley & Sons, 2004).