A Low-Cost Parallel Scalable FPGA Architecture for Regular and Irregular LDPC Decoding

François Verdier and David Declercq

***Abstract*—We present in this paper an architectural model for implementing parallel and scalable low-density parity-check (LDPC) decoders. This model has been developed for targeting field-programmable gate array devices and system-on-chip (SoC) platforms. We present first the motivations of investigating a new hardware model for regular and irregular LDPC decoders. The code flexibility, the memory usage optimization, and an easy hardware integration have been taken into account. The construction of a specific class of codes (*hardware-constrained* LDPC codes) is then presented. Parallelization and pseudorandomness constraints of codes are particularly detailed. A complete description of our parallel and scalable hardware model suitable for reprogrammable architectures is then given. Simulation results are presented showing the efficiency of this model with both (3,6) regular and irregular codes.**

***Index Terms*—Field-programmable gate arrays (FPGAs), hardware-constrained low-density parity-check (LDPC) codes, parallel implementation.**

I. Introduction

WE ARE concerned in this paper with the development of methodologies for designing highly integrated architectures dedicated to digital communications. The very large integration level must be understood here as putting all digital blocks of a wireless transmitter/receiver together on a single electronic chip. This means that both high-level processing functions (source coding/decoding, front-end interface, application-specific programs) and signal processing functions (matched filtering, channel coding/decoding, etc.) should interact closely and efficiently. In the scope of highly integrated architectures, we present a solution for a simple integration of a parallel channel decoder using low-density parity-check (LDPC) codes. The first motivation of this work relies on the fact that today's demand for small and powerful digital communication appliances creates more and more complicated constraints to meet. For example, future generations of smart phones or personal digital assistants will require more and more processing needs, and will be, in the same way, integrated in very restricted silicon surfaces and powered by constantly decreasing battery voltages. Moreover, *flexibility* seems to be an important capability of such systems. Small, mobile, and low-power communication devices delivered with very short delays are now available with system-on-chip (SoC) and platform-based design methodologies. Both SoCs and platform-based concepts try to fill the gap between common hardware "platforms" and flexible (programmable) applications.

The second motivation of our work is that error-correcting methods based on LDPC codes, presented by Gallager in 1963 [1] and rediscovered by MacKay and Neal in 1995 [2], have been widely studied, and significant works have shown their potential closeness to the Shannon limit. LDPC codes are decoded iteratively by belief propagation (BP) on their associated factor graph [3]. Both the high performance of these codes and the simplicity of decoding (iterative) algorithms are attractive, and LDPC solutions have recently been proposed as error-correcting codes in several communication standards (DSL [4], magnetic storage [5], digital video broadcast [6]). A significant advantage of LDPC decoders over competitors' turbo codes [7] resides in their possible parallel implementation, and several hardware solutions have already been published. The reader can refer to [8] for a good introduction to the many aspects of implementing LDPC architectures and the tradeoffs between performance and hardware costs.

Unfortunately, many proposed hardware solutions of LDPC decoders suffer from a serious lack of flexibility. Some of them constitute fully parallel implementations of the LDPC factor graph in the form of a huge number of simple processing units (PUs) connected through complex routing networks [9]–[11]. Among the most significant works are an application-specified integrated circuit (ASIC) realization reaching a 1-Gb/s throughput [12] and a 54-Mb/s field-programmable gate array (FPGA) decoder [11]. The main advantage of such approaches is a very short decoding duration with the drawback of very complex (pseudorandom) and unflexible routing networks. Neither modification of codes nor scalability can be achieved with these solutions.

Both scalability and flexibility, however, could be achieved when implementing the decoding algorithm on partly parallel structures. Code construction and hardware resources must, therefore, be optimized jointly. Partly parallel architectures have been studied for both regular [13], [14] and irregular [15] LDPC codes. The work of Hocevar [16] [17] is a good example of such a solution. Starting from a structured decoding matrix H and a matrix expansion technique, only a small number of functional units can implement the whole decoding computation. However, good scalability and flexibility properties are not clear when programmable code lengths and rates are expected.

Other approachs exist with simultaneous code, algorithmic, and architectural optimizations [18], [19], and give very good

Paper approved by E. Ayanoglu, the Editor for Communication Theory and Coding Application of the IEEE Communications Society. Manuscript received August 11, 2003; revised August 5, 2004.

The authors are with the ETIS Laboratory, CNRS UMR 8051/University of Cergy-Pontoise/ENSEA, 95014 Cergy-Pontoise Cedex, France (e-mail: Francois.Verdier@ensea.fr; David.Declercq@ensea.fr).

Digital Object Identifier 10.1109/TCOMM.2006.877980

performance and speed results. Turbo-like algorithms are used for decoding, but are far from the context of our work.

The aim of this paper is to illustrate a methodology for designing hardware solutions exploiting reconfigurability of new available architectures for an efficient integration of LDPC decoders in SoC systems. We propose a low-cost reconfigurable and scalable parallel realization of a hardware LDPC decoder. This parallel architectural model does not depend on the decoding algorithm, because it relies only on the LDPC parity-matrix structure. Actually, all iterative decoding algorithms derived from the BP algorithm (min-sum and its modified versions [20]) can be considered with our model. This paper presents the motivation and details of this architectural model. Our architectural model and the LDPC code family it implements have been optimized jointly under realistic hardware constraints. The resulting subclass of LDPC codes has been denoted *hardware-constrained LDPC* (HC-LDPC) and is presented in this paper. Theory and fundamentals of HC-LDPC codes can be found in [21].

This paper is organized as follows. Section II presents signal processing aspects of the LDPC decoder and gives details about the BP decoding algorithm as an example. The problem of facing the pseudorandomness of the codes and our solution for parallelizing LDPC decoders is also presented in this section. Then we present in Section III the architectural model and how it has been implemented in hardware. Section IV gives some performance results, and a discussion is given in Section V.

II. Architectural Issues of LDPC Decoders

For the sake of clarity, we present first the notations that will be used throughout this paper. A regular or irregular LDPC code is defined by a block of N symbols (the codeword) and a set of M parity-check equations. Each parity function applies to a subset of symbols of the codeword. In the case of a regular LDPC code, all symbols participate exactly to the same number of parity functions, and all parity functions apply to the same number of symbols. In the case of an irregular LDPC code, these quantities are conveniently expressed by two polynomials $\lambda(x)$ and $\rho(x)$, detailed later. The codeword $\underline{C}$ of length N contains $K = N - M$ information symbols and M redundancy symbols. The rate of this code is at most $R \leq (K/N)$.

An LDPC code is represented by its sparse parity-check matrix H of size $M \times N$ and of density defined as $N_e/(M \times N)$ (N_e is the number of nonzero values in the matrix). A value $h_{mn} = 1$ indicates that the nth symbol is used in the mth parity function. A regular LDPC code is defined by a matrix containing exactly the same number of ones in each row and the same number of ones in each column. The decoding matrix may also be represented by a bipartite graph with N symbols (or data nodes) and M parity (or check) nodes. Using the notation of [22], the connectivity of these nodes are expressed by the polynomials $\lambda(x)$ and $\rho(x)$, where

$$\lambda(x) = \sum_{i=2}^{t_{c_{\max}}} \lambda_i x^{i-1} \tag{1}$$

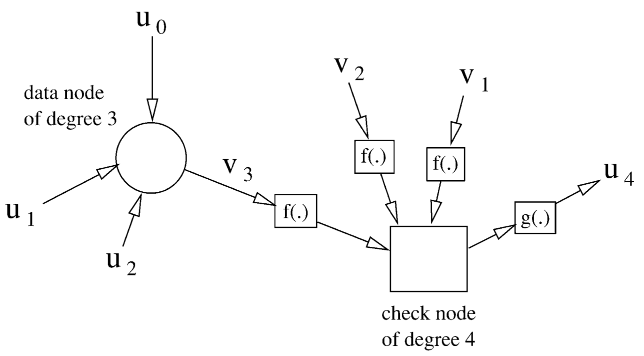

Fig. 1 Representation and notations for nodes and messages. Arrows represent the message flow.

is used to indicate the fraction of edges (λ_i) that are connected to data nodes of degree i, and

$$\rho(x) = \sum_{j=2}^{t_{r_{\max}}} \rho_j x^{j-1} \tag{2}$$

gives the fraction of edges (ρ_j) connected to check nodes of degree j. A degree i data node (respectively, a degree j check node) is a node connected to exactly i (resp. j) edges. Quantities $t_{c_{\max}}$ and $t_{r_{\max}}$ are the upper bounds of edge connectivity. The decoding matrix is typically constructed at random, leading to a random topology of the corresponding graph.

A. BP Decoding Algorithm

Several approaches exist for decoding LDPC codes, all derived from the BP algorithm [20]. Insisting on the fact that our model does not depend on the decoding solution, we will only present the BP algorithm in the log domain (log-BP).

Decoding a LDPC code with log-BP consists of propagating messages along all the edges of the factor graph. We denote v_k as the data to check messages and u_k as the check to data messages. Messages represent probabilities (or "beliefs") of symbols on parity-check values and parity-check values on symbols, respectively. In addition to these quantities, a log-likelihood ratio (LLR) u_0 is associated with each data node. This ratio feeds the decoding algorithm and comes from the channel receiver.

The following notations are used for the description of the log-BP algorithm.

- The binary codeword to be transmitted on the channel is $\underline{C} = [c_0 \cdots c_{N-1}]^T$.
- The output of the modulation is $\underline{X} = \text{BPSK}[\underline{C}]$.
- The transfer function of the channel is modeled by $\underline{Y} = \underline{X} + \underline{W}$, where $\underline{W}$ is an additive white Gaussian noise (AWGN).
- The decoding algorithm is fed by the LLR $u_0[n] = \log(p(c_n = 0|y_n)/p(c_n = 1|y_n))$.

The decoding algorithm consists of three computing steps, computed iteratively (see Fig. 1 for the notations).

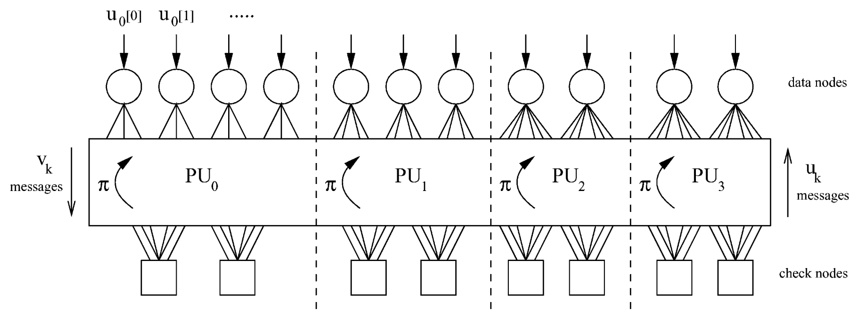

Fig. 2 Partitioning of the bipartite representation of a regular or irregular LDPC code. Each partition is associated with an independent PU.

1) The data-node process (*data update*) computes v_k messages from u_k and LLRs. For each data node (i being the degree of the node and u_0 the LLR entering this node)

$$v_k = u_0 + \sum_{l=1, l\neq k}^{i} u_l. \tag{3}$$

2) The check-node process (*check update*) computes u_k messages. For each check node (j being the degree of the node)

$$u_k = g\left(\sum_{l=1, l\neq k}^{j} f(v_l)\right) \times \prod_{l=1, l\neq k}^{j} \operatorname{sgn}(v_l). \tag{4}$$

3) The hard-decision process that computes the estimated value of each symbol of the code. For each data node

$$\hat{c} = \operatorname{sgn}\left(u_0 + \sum_{l=1}^{i} u_l\right). \tag{5}$$

Note that steps 1 and 3 could easily be done simultaneously. Step 3 needs a full sum of incoming messages (plus a sign calculation), while step 1 needs extra partial differences. Two math functions are used in step 2 for transposing the initial needed convolution in the log-Fourier domain. These functions are defined as

$$f(x) = \log\left(\tanh\left(\frac{|x|}{2}\right)\right) \tag{6}$$
$$g(x) = 2 \times \tanh^{-1}(\exp(x)). \tag{7}$$

A complete description of the log-BP algorithm can be found in [20].

An important point that must be considered when implementing LDPC decoders in hardware consists of optimizing resource usage (gate count, memory sizes). We discuss in the following how to manage memory needs by parallelizing hardware implementation.

B. The Problem of Handling Randomness

Let an LDPC code be represented by its bipartite graph, as shown in Fig. 2. In this representation, each connection between a data node and a check node can be represented by a unique number A_e, which is the address in two memories containing the u_k and v_k messages propagated along this connection. In this model, edges are numbered in a natural order from $A_e = 0$ to $A_e = N_e - 1$ (from the data-nodes side of the graph). Equivalently, the ones in the decoding matrix are numbered from top to bottom by columns first.

Decoding this LDPC code thus relies on the iteration of two processing steps:

1) the data-to-check process (*data update*), which calculates v_k messages from data nodes to check nodes using (3);
2) the check-to-data process (*check update*), which calculates u_k messages using (4).

As long as all messages are contained in memories, the two processing steps are equivalent to reading from (and writing to) specific memory locations. More precisely, due to the edge numbering, the first step consists of accessing memories in a natural order (from 0 to $N_e - 1$), while the second step accesses memories in a pseudorandom order. Any graph topology can then be obtained by a unique interleaving set $\pi = \{\pi_n\}$ of size N_e, where π_n is the nth check-node connection number (i.e., the nth memory location) treated by the check-node process. A particular regular or irregular LDPC code is then completely defined by the set of parameters $\{N, M, \lambda(x), \rho(x), \pi\}$. The two irregularity profiles $\lambda(x)$ and $\rho(x)$ reduce to the two numbers $t_{r_{\max}}$ and $t_{c_{\max}}$ for regular codes.

The interleaver may be implemented, for example, by using a memory containing all π_n values. For large block-length codes, this solution is very expensive in terms of memory cost. Optimizing the interleaver complexity is then a major challenge when considering hardware implementation of large block-length codes. Parallelization of such large codes is a possible solution to this critical issue if the interleaver size could be kept constant when the length of the code increases. In this context, we developed a subclass of LDPC codes, namely, HC-LDPC codes, well adapted for the parallelization of LDPC decoders.

In the next section, we present a brief overview of HC-LDPC code construction, and introduce the general concepts and notations useful for the rest of the paper. For a more detailed presentation of the HC-LDPC code construction, we refer readers to [21].

C. Regular and Irregular HC-LDPC Codes Construction

The construction of an HC-LDPC code parallelized on N_p *partitions* starts with a highly structured block-diagonal rect-

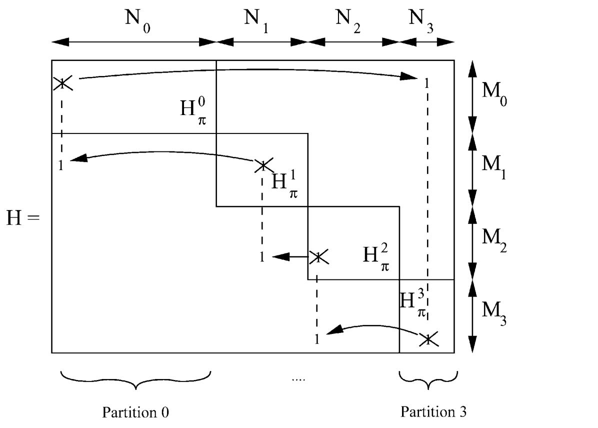

Fig. 3 Example of the two-step construction (diagonal construction followed by moving nonzero values) with an irregular parallel HC-LDPC code implemented on $N_p = 4$ partitions. The HC-LDPC code is irregular in column weight only in this example.

TABLE I
BEST CODES OF RATE $R \approx 0.5$ WITH $\rho(x) = x^7$ AND $N_p = 8$ PARTITIONS. δ IS THE CODE THRESHOLD COMPUTED BY DE, C IS THE CHANNEL CAPACITY

Code	$\underline{\lambda}$	Rate	$(\delta - C)$
Code40	$\lambda(x) = \frac{1}{4}x + \frac{1}{4}x^2 + \frac{1}{8}x^4 + \frac{3}{8}x^{10}$	0.533	0.1754
Code41	$\lambda(x) = \frac{1}{4}x + \frac{1}{4}x^2 + \frac{1}{8}x^4 + \frac{3}{8}x^{11}$	0.528	0.1533
Code42	$\lambda(x) = \frac{1}{4}x + \frac{1}{4}x^2 + \frac{1}{4}x^5 + \frac{1}{4}x^{13}$	0.533	0.1638
Code43	$\lambda(x) = \frac{1}{4}x + \frac{1}{4}x^2 + \frac{1}{4}x^5 + \frac{1}{4}x^{14}$	0.531	0.1478
Code44	$\lambda(x) = \frac{1}{4}x + \frac{1}{4}x^2 + \frac{1}{4}x^5 + \frac{1}{4}x^{15}$	0.529	0.1535
Code45	$\lambda(x) = \frac{1}{4}x + \frac{1}{4}x^2 + \frac{1}{4}x^5 + \frac{1}{4}x^{16}$	0.528	0.1682
Code46	$\lambda(x) = \frac{1}{4}x + \frac{1}{4}x^2 + \frac{1}{4}x^5 + \frac{1}{4}x^{17}$	0.526	0.1832

angular matrix $H_d = \mathrm{diag}[H_\pi^0, H_\pi^1, \ldots, H_\pi^{N_p-1}]$ (see Fig. 3). The $(M \times N)$ matrix H_d meets the following constraints: 1) each submatrix has strictly regular connections both data-wise and check-wise; 2) the total number of ones N_e in H_d is uniformly divided onto the N_p submatrices (all matrices H_π^p have the same number of ones N_e/N_p); and 3) each submatrix structure leads to the same interleaver $\pi = \{\pi_n\}$. In the particular case of a regular code, all submatrices are identical. For irregular codes, submatrices H_π^p have different sizes, column, and/or row degrees according to the polynomials $\lambda(x)$ and $\rho(x)$.

Note that with this method, we suppose that the LDPC matrix is bit-irregular, but check-regular. Moreover, there are certain issues in designing irregular HC-LDPC codes, and especially the condition 3) is sometimes rather difficult to fulfill. Since the interleaver is build at random and is very constrained (it must be equal for different degrees and submatrices sizes), our algorithm often gets stuck because of edge conflicts. However, the number of constraints involved in 3) is limited to the number of different degrees in the LDPC irregularity profile. In our simulations, with only four different degrees (see Table I), our constrained interleaver construction succeeds 40% of the time.

The number of partitions N_p thus defines the maximum number of different data- and check-node degrees in the HC-LDPC code. At this point, the parallel code is the "concatenation" of N_p independant partitions (disjoints graphs) that can be mapped independently on a set of N_p PUs (for the messages computations) and memories (for storing messages), as illustrated in Fig. 2.

The next step consists of connecting these graphs with *inter-partition* edges. This is done by moving a subset of ones from the diagonal matrices H_π^p to the other partitions. In order to respect both column and row weights (data and check degrees), as well as the number of edges in all partitions, these moves must be done in the following way. A subset of values inside the interleaving sequence are tagged by a circular partition shift s. Given a particular value of the sequence π_n and its tag s_n in the partition p, this edge is moved into the partition number $(p + s_n) \mathrm{mod} N_p$. Fig. 3 illustrates some moves in this construction.

By construction, the resulting decoding matrix has exactly the given irregularity profiles $\lambda(x)$ and $\rho(x)$ as pointed out in [21]. The performance of the code built this way depends on the number of moved edges. This number is chosen to be a $(N_p - 1/N_p)$ proportion of the number of edges in each partition, so that the distribution of ones in the resulting matrix is as uniform as possible.

It is obvious to say that this method applies on any number N_p of partitions and any code block-length multiple of N_p.

This parallel construction of the code applies to irregular codes, as well as to regular ones. The only limitation of this construction resides in the restriction on the irregularity profiles. More precisely, the number of different (data) node degrees is limited to the number of partitions. For the rest of this paper, we restrict the irregular examples to the codes that have irregular profiles in column degrees only (with regular polynomials $\rho(x)$). Table I shows, however, that such codes have very good performance even with a limited number of partitions. The coding rates have been chosen for a comparison purpose with the regular $(3, 6)$ LDPC code. Thresholds δ have been computed by applying an approximate density-evolution (DE) method (in floating-point precision) and with codes of infinite lengths (refer to [23] for a detailed presentation of a finite precision version of DE).

We present in the next section the hardware implementation of the HC-LDPC decoder that we propose based on all these architectural issues.

III. PARALLEL HARDWARE IMPLEMENTATION

A. Architectural Model for HC-LDPC Decoders

Our architectural solution consists of a representation of the decoding graph where all messages are located in specific separate memories and PUs are in charge of all decoding functions. This model implements a sequential scheduling of all node operations, and is valid for any iterative message-passing algorithm. The sequential scheduling is known to be expensive in terms of memory requirements, but leads to simple and highly optimizable (pipelined) data paths. Pseudorandom topologies are here realized through address calculations instead of routing

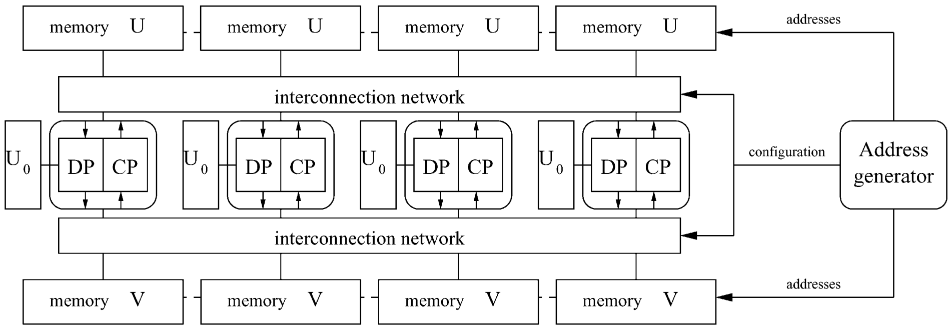

Fig. 4 Parallel hardware model of an HC-LDPC decoder. Code is divided into multiple partitions with independent PUs associated with a couple of local memories (U and V). Memory accesses are routed by using two communication networks configured by an address generator.

networks. Scalability could also be achieved by raising memory sizes (and PU numbers).

In this architectural model, both hardware complexity (in terms of memory requirements) and execution time are mainly proportional to the number of ones N_e in the decoding matrix. In order to cope with large codeword lengths, we propose a parallel architecture based on the same sequential scheduling of operations and a *partitioning* of the whole codeword into several partitions. Motivations for such a parallel model are twofold:

- if some specific synchronization constraints are met, the decoding duration should be divided by the number of available PUs (the number of partitions);
- implementing irregular codes is done by partitioning the factor graph into several *regular* partitions. All data (resp. check) nodes belonging to the same partition have the same connectivity degree.

Based on the factor-graph representation of an LDPC code (see Fig. 2), we propose an architectural model for an efficient LDPC decoder suitable for FPGA implementation.

The key points of the architecture we have presented are summarized below.

- The decoder uses a minimal amount of memory for storing messages and interleaver. This constraint is a critical issue when targeting FPGA architectures. Both memory hierarchy and fixed-point representations have been optimized.
- This model allows a parallel implementation and reduces the decoding duration by a factor of N_p (the number of available processors) without the need of extra memory.
- The whole architecture is able to operate in a pipelined fashion for a minimal decoding duration. A complete iteration of decoding a codeword with N_e edges is done in $T_p + 2 \times N_e$ clock cycles (T_p is the time overhead needed for filling the pipeline stages).
- A unique interleaver sequence of minimal size distributes addresses for all partitions. The memory size needed for the interleaver is also reduced by a factor of N_p.
- The parallel implementation our model is suitable for both regular and irregular LDPC codes.

This model could be used in conjunction with any of the BP-based decoding algorithms presented in the literature. More precisely, our architectural model is independent on the data and check update equations, since different message-passing algorithms (BP, logBP, Min-Sum, etc.) could be implemented only by changing the PUs. However, the use of our parallel architecture requires a specific scheduling (namely, the *flooding* schedule), and therefore, one can not directly apply a modified scheduling (as in [24]).

Fig. 4 illustrates the overall parallel architecture for a $N_p = 4$ partitions decoder. The four PUs are mainly similar and composed of two independent data paths: the data-node processor (DP) and check-node processor (CP). The two processors work alternatively, and sequentially compute data updates [(3)] and check updates [(4)]. Interpartition edges are realized through two identical interconnection networks configured by the tag part of the interleaver. A unique address generator is in charge of delivering U and V memory addresses (address part of the interleaver), network configurations, and control signals. For irregular codes, the DPs are working on partitions with different column degrees. Therefore, a dedicated first-in, first-out (FIFO) memory is inserted in the write paths for latency harmonization. All CP structures are exactly similar when row degrees are uniform.

Figs. 5 and 6 show, respectively, DP and CP structures. Computations are done in a two-step process:

1) a complete sum of incoming messages is first performed in the A accumulator $A = u_0 + \sum_{k=1}^{i} u_k$ (for DP), and $A = \sum_{k=1}^{j} f(|v_k|)$ (for CP). The last accumulation is done in the B register;
2) partial substractions within the second register give the outgoing messages $v_l = B - u_l$ (for DP) and $|u_l| = g(B - f(|v_l|))$ (for CP).

The two processors are designed to work in a pipeline fashion, thus leading to a message being written in memory at each clock cycle.

Although the two mathematical functions $f(x)$ and $g(x)$ are identical, the pipeline structure needs two distinct look-up tables (LUTs) for simultaneous calculation of incoming and outgoing messages. Signs are treated in a separate data path. Write latencies of a t_c-degree DP and a t_r-degree CP are, respectively, given by $L_{\text{DP}} = t_c + 1$ and $L_{\text{CP}} = t_r + 5$. All DP message-write cycles can be strictly synchronized by inserting a second FIFO memory of depth $(t_{c_{\max}} - t_c)$, where $t_{c_{\max}}$ is the maximum data-node degree in the code.

B. Interconnection Networks

In the construction of an HC-LDPC code, the moving of edges from local partitions to interpartition connections is

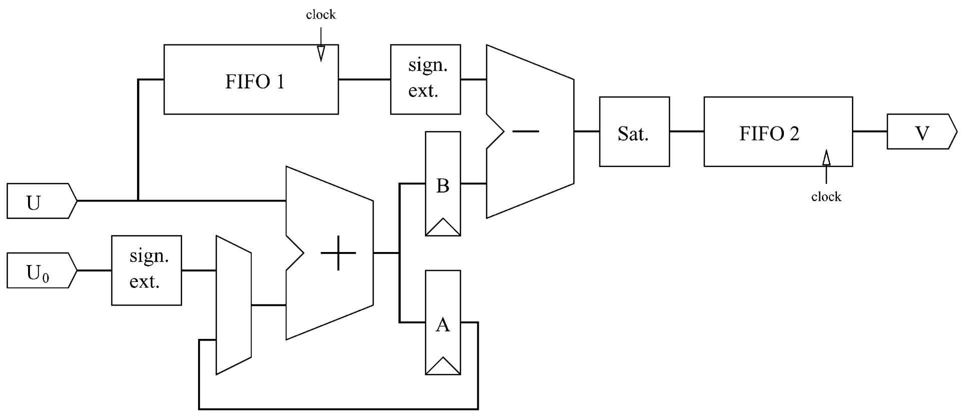

Fig. 5 Architecture of the DP unit. A first FIFO memory whose depth depends on the data-node degree is present for accumulation of incoming messages. A second FIFO memory is added for latency harmonization.

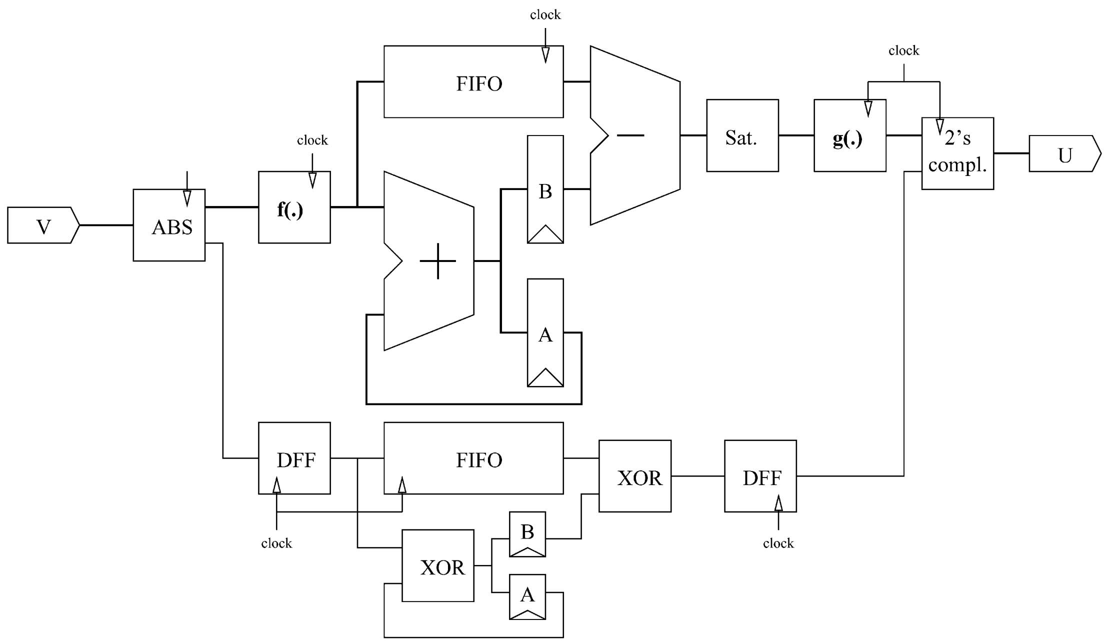

Fig. 6 Architecture of the CP unit. Absolute values and signs (produced by the ABS module) are treated separately according to (4). The FIFO memories in this CP unit allow only regular connections on the check nodes. Two D-type flip-flops (DFF) are used to balance latencies between the absolute values (upper) and signs (lower) data paths.

equivalent to finding a path between u memories to PUs or PUs to v memories. These paths are realized through two separate (and identical) interconnection networks. The networks are configured by the tag value in the interleaving sequence associated with each address. Unmoved edges are considered as *local* accesses between PUs and u and v memories. When a moved edge is concerned, the interconnection networks are configured, and all memory accesses are done in a similar way. Networks are used only during the check-update step and work as circular shift connections. When using dual-port memories with separate input and output data busses, the two networks may be easily implemented with tri-state buffers.

The address calculation unit is realized by a finite-state machine associated with a memory containing the interleaving sequence (and tags). During the first step of the algorithm (data update), adresses are delivered by a simple binary counter. Addresses are delivered through a memory during the second step (check update). The address-calculation unit also produces all control signals needed by the PUs. The size of the interleaving memory depends only on the number of edges in each partition, and on the number of partitions. Let an HC-LDPC decoding matrix with N_e ones be implemented on N_p partitions. Each partition contains (N_e/N_p) edges, and tags are coded on $\log_2(N_p)$ bits. Interleaving memory thus contains N_e/N_p words with $\log_2(N_e/N_p)$ bits for addresses and $\log_2(N_p)$ bits for tags. The memory size is then given by

$$\frac{N_e}{N_p} \times (\log_2(\frac{N_e}{N_p}) + \log_2(N_p)) = \frac{N_e}{N_p} \times (\log_2(N_e)).$$

That is, the size of the interleaver memory is exactly divided by a factor equal to the number of processors.

C. Finite Precision Computation

Some attention has been given to data representation in LDPC decoders [25], [26]. Among other results, it has been shown that the number of bits used in a fixed-point representation has a great impact on code performance. We have shown in [23] that both the error floors and decoding thresholds depend directly on the number of bits used for coding integer and fractional parts of messages when a log-BP algorithm is used. In particular, it has been shown that the number of bits used for coding the integer part of the messages has a direct consequence on the error-floor level, and the number of bits for the fractional part influences the code threshold. Actually, the more bits are used for the integer part of the messages, the lower the error floor is, and the more bits are used for the fractional part of the messages, the closer is the decoding threshold to the Shannon limit.

The same behavior has been observed for regular as well as for irregular HC-LDPC codes. This error-floor effect caused by quantization is similar for other types of LDPC codes, as shown in [20]. We also have theoreticaly studied by discrete DE the impact of a reduced-precision computation of the two math functions $f(x)$ and $g(x)$. A modification of the logarithm base compensates part of the performance loss due to the finite-precision coding. We have determined through DE the optimal log-bases for different coding precisions, by choosing the bases which exhibit the lowest decoding threshold [23]. Note that the optimal bases depend both on the code structure (irregularity profile) and the decoding precision.

Simulations and results presented in the next section are all obtained with optimal log-bases and fractionals coded in two's complement on 8 b (4 b for the fractional part, 4 b for the integer part). Of course, using more bits would yield better results, but an 8-b coding seems to be a good compromise between memory cost and decoding performance.

IV. Results

A. Simulations

We present here simulation results for two differents codes with log-BP decoding. The first is a (3,6) regular code with two different sizes ($N = 1056$ and $N = 4128$) implemented on 1 (single PU) and 2, 4, 8, and 16 PUs and logarithm base 1.27. The second is the irregular code "Code43" of size $N = 4096$ and of rate $R = 0.531$ with an irregular column degree and a regular row degree (see Table I). Due to its particular irregular profile, this code can be implemented on four partitions or any multiple of four. Only four and eight parallelization degrees are presented for this code. The logarithm base used for this irregular code is 1.28. All simulations have been done with fixed-precision messages coded with 8 b (4 b for the integer part, and 4 b for the fractional part).

Fig. 7 shows that the number of partitions has no significant impact on performance for these three codes. We have also made extensive simulations to compare the performance of HC-LDPC codes with a completely random matrix. The results reported in [21] show that the two types of codes perform equally, as long as a reasonable number of PUs are considered (typically less than [24]).

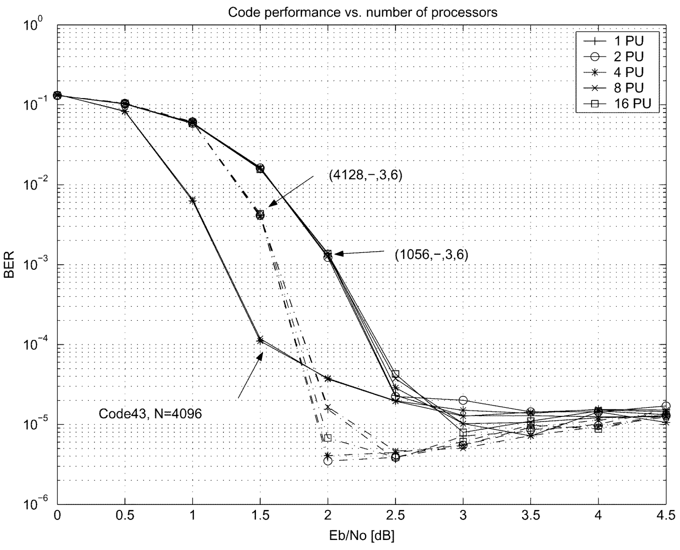

Fig. 7 Compared decoding performances for two regular codes and one irregular code. 50 decoding iterations have been applied, with fractionals coded in two's complement with 8 b (Q4.4).

An interesting remark can be made about the error-floor level. This floor level has been discussed in our previous work [21], [23], and proved to be dependent on the number of bits used for coding the fractional part of messages. Simulations show that this floor level is preserved, whatever the parallelization degree. Two important behaviors are preserved when parallelization is applied: 1) a lower decoding threshold when block length increases ($N = 1056$ versus $N = 4128$); and 2) better performances when irregular codes are used.

By using the fact that our parallelization model strictly divides the overall decoding time by the number of available PUs, we can see a significant performance gain when allocating more resources (see Fig. 8). In particular, one can gain 1.5 dB at a bit-error rate (BER) of 2.10^{-5} when using two PUs instead of one (and 0.5 dB more with four PUs). At the same time, BER at 2 dB is lowered by more than two decades when four PUs are allocated instead of a single one. However, the decoding algorithm reaches its performance limit near the fortieth iteration in the case of this particular regular code. In order to take into account this limit in our discussion, we chose to present the BER at a given signal-to-noise ratio (SNR) with respect to the actual decoding time. The decoding time can be defined as the total number of iterations divided by the number of PUs (see Fig. 9). We can see that although the performance of the code at SNR $= 2.0$ dB saturates, the limit is reached more rapidly when allocating more PUs. In particular, five iterations are needed with 16 PUs, while 15 iterations are necessary when two PUs are used.

B. FPGA Implementation Results

Based on the model described, several regular and irregular HC-LDPC decoders were implemented on an APEX 20 Ke ALTERA FPGA device (EPXA10). A high level of flexibility was possible by specifying the whole architecture in VHDL, and only minor modifications of the code were necessary for the exploration of different code lengths, rates, and parallelization factors.

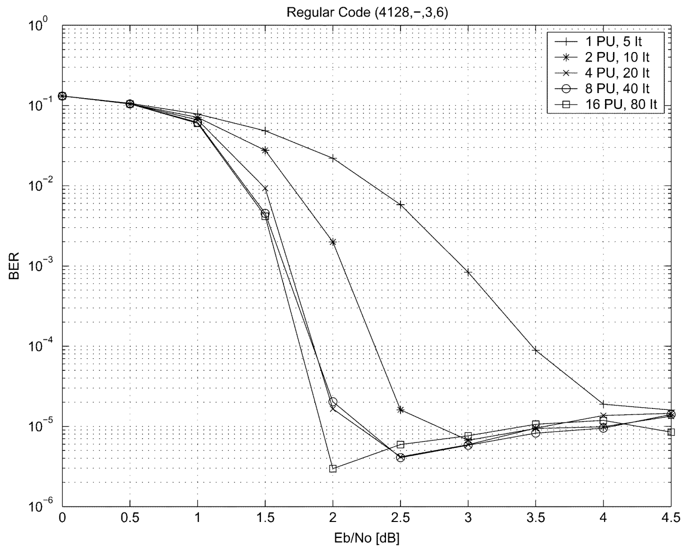

Fig. 8 Compared decoding performances of the (4128,-,3,6) regular decoder at constant decoding time.

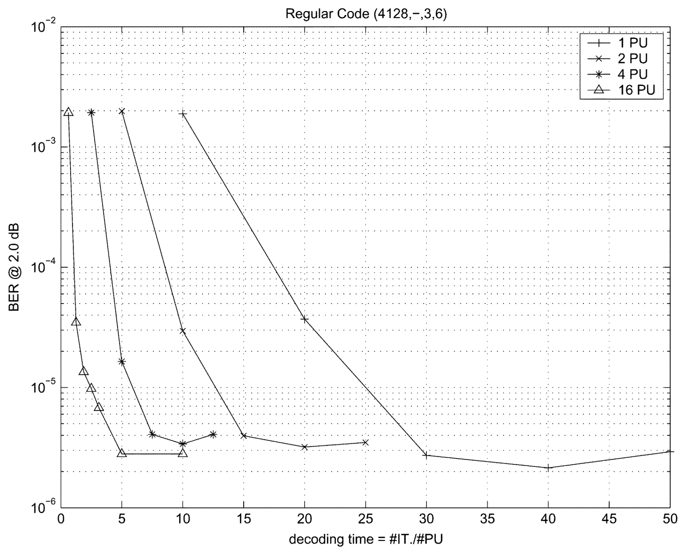

Fig. 9 BER at SNR $= 2.0$ dB as a function of decoding time. The x axis represents decoding time (number of iterations divided by number of available processors).

Implementation results of a rate-0.531, 2048-b irregular decoder with a parallelization factor of four PUs running log-BP are the following: 978 logic cells (LCs) are used for the implementation of the four PUs (2.5% occupation), 71 LCs for the two inteconnection networks (barrel shifters), 542 LCs for the sequential logic, and a total of 164 224 memory bits for LLR, messages, and interleaver. The negligible amount of LCs used by the routing networks and the 24 960 memory bits needed for storing the interleaver, thus the decoding matrix, confirm that a high parallelization degree can be reached without combinatorial or routing penalty.

A decoding iteration is completed in $2 \times (N_e/N_p) + N_{\mathrm{LAT}}$ clock cycles where N_{LAT} represents the accumulated CP and DP latencies. The maximum working frequency of this decoder is 37.89 MHz, thus allowing a maximum channel throughput of 1 Mb/s with only four PUs.

V. Conclusion

HC-LDPC codes are codes optimized under specific hardware considerations encompassing optimal bit coding of messages and decoding performances under the log-BP technique. Using this class of codes, we presented in this paper an architectural model for implementing parallel regular and irregular HC-LDPC decoders. Parallelization is done by splitting the sparse decoding matrix into several partitions treated simultaneously. In addition to an easy implementation model, the simulations of HC-LDPC decoders exhibit very good performances.

By applying parallelization on iterative LDPC decoding algorithms, the resulting hardware model avoids some critical drawbacks present in other LDPC decoders. Our architecture does not rely on a particular code; changing the interleaver memory content is equivalent to using a new code in the transmission. Moreover, only simple interconnect networks are needed, instead of very complex routing networks. Scalability is also achievable for either handling very long block-length codes or reaching better decoding speeds. This simple scalable parallel model of LDPC decoders is, in our opinion, the first solution for an easy integration of channel-decoding coprocessors in future low-cost communication devices (next-generation mobile phones, smart PDAs).

Although our architectural model is independent of a particular technology (VLSI or FPGA), it has been successfully implemented on an SoC platform (an ALTERA Excalibur-ARM FPGA device) for regular and irregular HC-LDPC codes. Benchmarks have been applied and give results comparable to the simulations.

Acknowledgment

The authors would like to thank F. Tulars and A. Segard for their valuable contributions to this work during the FPGA design, implementation, and test.

References

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

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

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

[4] E. Eleftheriou and S. Ölçer, "Low-density parity-check codes for digital subscriber lines," in *Proc. IEEE Int. Conf. Commun.*, Apr.–May 2002, vol. 3, pp. 1752–1757.

[5] H. Song, R. Todd, and J. Cruz, "Performance of LDPC codes on magnetic recording channels," in *Proc. Turbo-Codes Conf.*, Brest, France, 2000, pp. 395–398.

[6] *ETSI Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (DVB-S2)*, EN 302 307, ETSI, Apr. 2005.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Int. Conf. Commun.*, Geneva, Switzerland, 1993, pp. 1064–1070.

[8] E. Yeo, B. Nikolić, and V. Anantharam, "Architectures and implementations of low-density parity-check decoding algorithms," in *Proc. IEEE Int. Symp. Circuits Syst.*, Scottsdale, AZ, May 2002, vol. III, pp. 437–440.

[9] E. Boutillon, J. Castura, and F. Kschischang, "Decoder-first code design," in *Proc. Turbo Codes Conf.*, Brest, France, Sep. 2000, pp. 459–462.

[10] V. Sorokine, F. Kschischang, and S. Pasupathy, “Gallager codes for CDMA applications —Part II: Implementations, complexity, and system capacity,” *IEEE Trans. Commun.*, vol. 48, no. 11, pp. 1818–1828, Nov. 2000.

[11] T. Zhang and K. Parhi, “An FPGA implementation of (3,6)-regular low-density parity-check code decoder,” *EURASIP J. Appl. Signal Process.*, no. 6, pp. 530–542, May 2003.

[12] A. J. Blanksby and C. J. Howland, “A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder,” *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[13] A. Selvarathinam, G. Choi, K. Narayanan, A. Prabhakar, and E. Kim, “A massively scaleable decoder architecture for low-density parity-check codes,” in *Proc. IEEE Int. Symp. Circuits Syst.*, Bangkok, Thailand, May 2003, vol. II, pp. 61–64.

[14] S. Ölçcer, “Decoder architecture for array-code-based LDPC codes,” in *Proc. IEEE Global Commun. Conf.*, San Francisco, CA, Dec. 2003, pp. 2046–2050.

[15] H. Zhong and T. Zhang, “Design of VLSI implementation-oriented LDPC codes,” in *Proc. IEEE Veh. Technol. Conf.*, Orlando, FL, Oct. 2003, pp. 670–673.

[16] D. E. Hocevar, “LDPC code construction with flexible hardware implementation,” in *Proc. IEEE Int. Conf. Commun.*, Anchorage, AK, May 2003, pp. 2708–2712.

[17] Y. Chen and D. Hocevar, “An FPGA and ASIC implementation of rate 1/2 8088-b irregular low density parity check decoder,” in *Proc. IEEE Global Commun. Conf.*, San Francisco, CA, Dec. 2003, pp. 113–117.

[18] M. M. Mansour and N. R. Shanbhag, “A novel design methodology for high-performance programmable decoder cores for AA-LDPC codes,” in *Proc. IEEE Workshop Signal Process. Syst.*, 2003, pp. 29–34.

[19] ——, “High-thoughput LDPC decoders,” *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

[20] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, “Reduced-complexity decoding of LDPC codes,” *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[21] F. Verdier and D. Declercq, “An LDPC parity check matrix construction for parallel hardware decoding,” in *Proc. 3rd Int. Symp. Turbo Codes, Related Topics*, Brest, France, Sep. 2003, pp. 235–238.

[22] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, Improved low-density parity-check codes using irregular graphs and belief propagation Digital Equipment Corp. Sys. Res. Center, Berkeley, CA, Tech. Rep. TR-97-044, 1997 [Online]. Available: urlciteseer.nj.nec.com/luby99improved.html

[23] D. Declercq and F. Verdier, “Optimization of LDPC finite precision belief propagation decoding with discrete density evolution,” in *Proc. 3rd Int. Symp. Turbo Codes, Related Topics*, Brest, France, Sep. 2003, pp. 479–482.

[24] J. Zhang and M. Fossorier, “Shuffled belief propagation decoding,” in *Proc. Asilomar Conf.*, Pacific Grove, CA, Nov. 2002, pp. 8–15.

[25] L. Ping and W. Leung, “Decoding low density parity check codes with finite quantization bits,” *IEEE Commun. Lett.*, vol. 4, no. 2, pp. 62–64, Feb. 2000.

[26] T. Zhang, Z. Wang, and K. Parhi, “On finite precision implementation of low density parity check codes decoder,” in *Proc. IEEE Int. Conf. Circuits Syst.*, Sydney, Australia, May 2001, vol. 4, pp. 202–205.

François Verdier received the M.S. degree in 1990, and the Ph.D. degree with distinction in 1995, both in electrical engineering and computer science, from the University of Paris-Orsay, Paris, France.

During his Ph.D. study, he participated in the development of an automatic architecture synthesis environment dedicated to real-time image-processing applications. He is currently an Assistant Professor of Electrical Engineering and Computer Science with the University of Cergy-Pontoise, Cergy-Pontoise, France, where he also leads the Architecture group of the ETIS laboratory. His research interests are architectures for LDPC decoding, dynamically reconfigurable system-on-chip architectures, and embedded operating systems for real-time signal- and image-processing architectures.

David Declercq received the Ph.D. degree in electrical and computer engineering in 1998 from the University of Cergy-Pontoise, Cergy-Pontoise, France.

He is currently a full Professor with the ENSEA, a graduate school in electrical and computer engineering, and teaches digital signal processing and communication theory. He is associated with the ETIS Laboratory in Cergy-Pontoise, France, and is the Head of the signal processing group since 2003. His research interests are mainly statistical model estimation and coding theory for digital communication. In particular, he is interested in designing good binary and nonbinary LDPC codes for various types of realistic channels (OFDM, multiuser, etc.).