VLSI implementation of a multi-mode turbo/LDPC decoder architecture

VLSI implementation of a multi-mode turbo/LDPC decoder architecture

Carlo Condo, Maurizio Martina, *Member IEEE*, Guido Masera, *Senior Member IEEE*
Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino, Italy

***Abstract*—Flexible and reconfigurable architectures have gained wide popularity in the communications field. In particular, reconfigurable architectures for the physical layer are an attractive solution not only to switch among different coding modes but also to achieve interoperability. This work concentrates on the design of a reconfigurable architecture for both turbo and LDPC codes decoding. The novel contributions of this paper are: i) tackling the reconfiguration issue introducing a formal and systematic treatment that, to the best of our knowledge, was not previously addressed; ii) proposing a reconfigurable NoC-based turbo/LDPC decoder architecture and showing that wide flexibility can be achieved with a small complexity overhead. Obtained results show that dynamic switching between most of considered communication standards is possible without pausing the decoding activity. Moreover, post-layout results show that tailoring the proposed architecture to the WiMAX standard leads to an area occupation of 2.75 mm^2 and a power consumption of 101.5 mW in the worst case.**

***Index Terms*—VLSI, LDPC/Turbo Codes Decoder, NoC, Flexibility, Wireless communications**

I. Introduction

In the last years several efforts were spent to develop systems able to give ubiquitous access to telecommunication networks. These efforts were spent mainly in three directions: i) improving the transmission rate and reliability; ii) developing bandwidth efficient technologies; iii) designing low cost receivers. The most relevant results produced by such a vivid research were included in the last standards for both wireless and wired communications [1]–[7]. Besides, several standards provide multiple modes and functionalities. However, sharing common features is a challenging task to achieve flexibility and interoperability.

Several recent works, including [8], have shown that flexibility is an important property in the implementation of communication systems. Some works investigated this direction facing the challenge of implementing flexible architectures for the decoding of channel codes. In particular, flexible turbo/Low-Density-Parity-Check (LDPC) decoder architectures have been proposed not only to support different coding modes within a specific standard but also to enable interoperability among different standards. In [9]–[11] flexibility is achieved through the design of Processing Elements (PEs) based on Application-Specific-Instruction-set-Processor (ASIP) architectures, whereas in [12]–[14] PEs rely on Application-Specific-Integrated-Circuit (ASIC) solutions. In both approaches, flexible and efficient interconnection structures are required to connect PEs to each other.

Unfortunately, the communication patterns of turbo and LDPC codes suffer from collisions, namely two or more PEs require concurrent access to the same memory resource. To break the collision a Network-on-Chip (NoC) like approach was proposed in [15] for turbo codes. This idea has been further developed in other works. In particular, in [16] the NoC approach is used as a viable solution to implement flexible and high throughput interconnection structures for turbo/LDPC decoders.

An *intra-IP* NoC [17] is an application specific NoC [18] where the interconnection structure is tailored to the characteristics of the Intellectual Property (IP). The use of an intra-IP NoC as the interconnection framework for both turbo and LDPC code decoders has been demonstrated in several works [16], [19]–[21]. This choice enables larger flexibility with respect to other interconnection schemes [16], [22], [23], but introduces penalties in terms of additional occupied area and latency in the communication among PEs.

Stemming from the work presented in [14], [19], [20], where an ASIC implementation of an NoC-based turbo/LDPC decoder architecture is proposed, this paper aims to further investigate and optimize it. In particular, this work features the following novel contributions: i) management of dynamic reconfiguration to switch between a code to another one without pausing the decoding, ii) description of a new PE architecture with an improved shared memory solution which provides relevant saving of occupied area for min-sum decoding algorithm, iii) evaluation of a wide set of standards for both wireless and wired applications: IEEE 802.16e (WiMAX) [5], IEEE 802.11n (WiFi) [6], China Mulitimedia Mobile Broadcasting (CMMB) [3], Digital Terrestrial Multimedia Broadcast (DTMB) [4], HomePlug AV (HPAV) [2], 3GPP Long Term Evolution (LTE) [7], Digital Video Broadcasting - Return Channel via Satellite (DVB-RCS) [1], iv) complete VLSI implementation of the decoder up to layout level and accurate evaluation of dissipated power.

It is worth noting that, to the best of our knowledge, this is the first work addressing dynamic reconfiguration of flexible channel decoders with an analytical approach, and showing the actual impact of reconfiguration on both performance and complexity. The paper is structured as follows. In Section II decoding algorithms are briefly discussed, whereas section III deals with the basics of NoC-based turbo/LDPC decoder architectures and summarizes the main results this work starts from. The decoder reconfiguration techniques are detailed in Section IV and V, while Section VI deals with the description of LDPC and turbo decoding cores, along with their

respective memory organization. In Section VII evaluations of the architecture performance on various existing standards are provided. Implementation results are portrayed and discussed in Section VIII, and conclusions are drawn in section IX.

II. Decoding algorithms

Turbo and LDPC decoding algorithms are characterized by strong resemblances: they are iterative, work on graph-based representations, are routinely implemented in logarithmic form, process data expressed as Logarithmic-Likelihood-Ratios (LLRs) and require high level of both processing and storage parallelism. Both algorithms receive intrinsic information from the channel and produce extrinsic information that is exchanged across iterations to obtain the *a priori* information of uncoded bits, in the case of binary codes, or symbols, in the case of non binary codes. Moreover, their arithmetical functions are so similar that joint or derived algorithms for both LDPC and turbo decoding exist [24]. In the following for both codes we will refer to K, N and $r = K/N$ as the number of uncoded bits, the number of coded bits and the code rate respectively.

A. LDPC codes decoding algorithm

Every LDPC code is completely described by its $M \times N$ parity check matrix $\mathbf{H}$ ($M = N - K$) which is very sparse [25]. Each valid LDPC codeword x satisfies $\mathbf{H} \cdot x' = 0$, where $(\cdot)'$ is the transposition operator. The decoding of LDPC codes stems from the Tanner graph representation of $\mathbf{H}$ where two sets of nodes are identified: Variable Nodes (VNs) and Check Nodes (CNs). VNs are associated to the N bits of the codeword, whereas CNs correspond to the M parity-check constraints. The most common algorithm to decode LDPC codes is the *Belief Propagation* (BP) algorithm. There are two main scheduling schemes for the BP: two-phase scheduling and layered scheduling [26]. The latter nearly doubles the converge speed as compared to two-phase scheduling. In a layered decoder, parity-check constraints are grouped in layers each of which is associated to a component code. Then, layers are decoded in sequence by propagating extrinsic information from one layer to the following one [26]. This process is iterated up to the desired level of reliability.

Let $\lambda[c]$ represent the LLR of symbol c and, for column k in $\mathbf{H}$, bit LLR $\lambda_k[c]$ is initialized to the corresponding received soft value. Then, for all parity constraints l in a given layer, the following operations are executed:

$$Q_{lk}[c] = \lambda_k^{old}[c] - R_{lk}^{old} \tag{1}$$

$$A_{lk} = \sum_{n \in N(l), n \neq k} \Psi(Q_{ln}[c]) \tag{2}$$

$$\delta_{lk} = \prod_{n \in N(l), n \neq k} \mathrm{sgn}(Q_{ln}[c]) \tag{3}$$

$$R_{lk}^{new} = -\delta_{lk} \cdot \Psi^{-1}(A_{lk}) \tag{4}$$

$$\lambda_k^{new}[c] = Q_{lk}[c] + R_{lk}^{new} \tag{5}$$

$\lambda_k^{old}[c]$ is the extrinsic information received from the previous layer and updated in (5) to be propagated to the succeeding layer. Term R_{lk}^{old}, pertaining to element (l,k) of $\mathbf{H}$ and initialized to 0, is used to compute (1); the same amount is then updated in (4), R_{lk}^{new}, and stored to be used again in the following iteration. In (2) and (3) $N(l)$ is the set of all bit indexes that are connected to parity constraint l.

According to [27], the $\Psi(\cdot)$ function in (2) and (4) can be simplified with a limited BER performance loss as

$$R_{lk}^{new} \approx -\delta'_{lk} \cdot \min_{n \in N(l), n \neq k} \{|Q_{nk}|\}, \tag{6}$$

usually referred to as *normalized-min-sum* approximation, where $\delta'_{lk} = \sigma \cdot \delta_{lk}$ and $\sigma \leq 1$.

B. Turbo codes decoding algorithm

Turbo codes are obtained as the parallel concatenation of two constituent Convolutional Code (CC) encoders connected by the means of an interleaver (Π). Thus, the decoder is made of two constituent decoders, referred to as Soft-In-Soft-Out (SISO) or Maximum-A-Posteriori (MAP) decoders [28] connected in an iterative loop by the means of the interleaver Π and the de-interleaver Π^{-1}. Each constituent decoder performs the so called BCJR algorithm [29] that starting from the intrinsic and *a priori* information produces the extrinsic information. Let k be a step in the trellis representation of the constituent CC, and u an uncoded symbol. Each constituent decoder computes $\lambda_k[u] = \sigma \cdot (\lambda_k^{apo}[u] - \lambda_k^{apr}[u] - \lambda_k[\mathbf{c}^u])$ where $\sigma \leq 1$ [30], $\lambda_k^{apo}[u]$ is the a-posteriori information, $\lambda_k^{apr}[u]$ is the *a priori* information and $\lambda_k[\mathbf{c}^u]$ is the systematic component of the intrinsic information. According to [29] a-posteriori information is computed as

$$\lambda_k^{apo}[u] = \overset{*}{\max}_{e:u(e)=u} \{b(e)\} - \overset{*}{\max}_{e:u(e)=\tilde{u}} \{b(e)\} \tag{7}$$

where $\tilde{u} \in \mathcal{U}$ is an uncoded symbol taken as a reference (usually $\tilde{u} = \mathbf{0}$) and $u \in \mathcal{U} \setminus \{\tilde{u}\}$ with $\mathcal{U}$ the set of uncoded symbols; e is a trellis transition and $u(e)$ is the corresponding uncoded symbol. Several exact and approximated expressions are available for the $\overset{*}{\max}\{x_i\}$ function [31]: for example, it can be implemented as $\max\{x_i\}$ followed by a correction term, often stored in a small Look-Up-Table (LUT). The correction term, usually adopted when decoding binary codes (Log-MAP), can be omitted with minor Bit-Error-Rate (BER) performance degradation (Max-Log-MAP). The term $b(e)$ in (7) is defined as:

$$b(e) = \alpha_{k-1}[s^S(e)] + \gamma_k[e] + \beta_k[s^E(e)] \tag{8}$$

$$\alpha_k[s] = \overset{*}{\max}_{e:s^E(e)=s} \left\{\alpha_{k-1}[s^S(e)] + \gamma_k[e]\right\} \tag{9}$$

$$\beta_k[s] = \overset{*}{\max}_{e:s^S(e)=s} \left\{\beta_{k+1}[s^E(e)] + \gamma_k[e]\right\} \tag{10}$$

$$\gamma_k[e] = \lambda_k^{apr}[u(e)] + \lambda_k[\mathbf{c}(e)] \tag{11}$$

where $s^S(e)$ and $s^E(e)$ are the starting and the ending states of e, $\alpha_k[s^S(e)]$ and $\beta_k[s^E(e)]$ are the forward and backward metrics associated to $s^S(e)$ and $s^E(e)$ respectively. The term $\lambda_k[\mathbf{c}(e)]$ represents the intrinsic information received from the

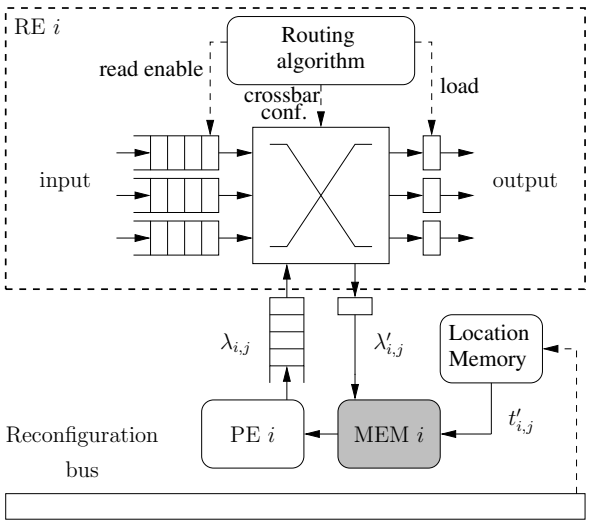

Figure 1. Node structure

channel. For further details on the decoding algorithm the reader can refer to [32].

In a parallel decoder, the decoding operations summarized in previous paragraphs are partitioned among P PEs. When configured in turbo code mode, these PEs operate as concurrent SISOs. On the other hand, they execute (1) to (5) in parallel for P slices of parity check constraints when configured in LDPC code mode. In both cases, messages are exchanged among PEs to propagate $\lambda_k[u]$ and $\lambda_k^{new}[c]$ amounts in accordance with the code structure. In the following, we indicate the j-th message received and generated by PE i as $\lambda'_{i,j}$ and $\lambda_{i,j}$ respectively.

III. NoC-based Decoder

The goal of this work is to design a highly flexible LDPC and turbo decoder, able to support a very wide set of different communication standards. The proposed multi-mode/multi-standard decoder architecture relies on an NoC-based structure, where each node contains a PE and a routing element (RE). Each PE implements the BCJR and layered normalized min-sum algorithms. On the other hand, REs are devoted to deliver $\lambda_{i,j}$ values to the correct destination.

The node architecture employed in this work for node i is represented in Fig. 1. Each RE is constituted by a 4×4 crossbar switch with 4 input FIFOs and 4 output registers. The routing algorithm is the one proposed in [19] as Single-Shortest-Path-FIFO-Length (SSP-FL). SSP-FL relies on a distributed table-based routing algorithm where each table contains the information for shortest path routing. The routing information is precalculated by running off-line the Floyd-Warshall algorithm. Moreover, in SSP-FL shortest path routing is coupled with an input serving policy based on the current status to the FIFOs, namely in case two messages must be routed to the same output port, priority is given to the message coming from the longer FIFO. It is worth noting that the destination of each $\lambda_{i,j}$ is imposed by the interleaver and the $\mathbf{H}$ matrix respectively. As a consequence, the routing is deterministic.

The PE includes both LDPC and turbo decoding cores: their architectures are structured to be as independent as possible of the supported codes. The LDPC decoding core is completely serial and able to decode any LDPC code, provided that enough memory is available. The SISO core for turbo decoding is tailored around 8-state turbo codes, and no other constraints are present: the two cores share the memories where the incoming data $\lambda'_{i,j}$ are stored and the location memory containing the pre-computed $t'_{i,j}$ values, i.e. the memory addresses to store $\lambda'_{i,j}$. Also the interconnection structure depends only on the location memory size, that sets an upper bound to the number of messages each PE can handle.

The decoding task is divided uniformly among the different nodes. The process is straightforward in turbo mode, with each node being assigned a portion of the trellis that is processed in a sliding-window fashion [33], [34]. Extrinsic and window-initialization information are carried through the network according to the code interleaving and deinterleaving rules [19]. On the contrary, in LDPC mode the partitioning of the decoding task on the PEs is obtained as follows. Using a proprietary tool based on the METIS graph coloring library [35], the $\mathbf{H}$ matrix is partitioned on the chosen network topology. At this point the destination of every message coming out of each decoding core is known. Thus, in both turbo and LDPC modes each outgoing message is made of a payload $\lambda_{i,j}$ and a header containing the destination node.

Performance of meshes, toroidal meshes, spidergon, honeycomb, De Bruijn and Kautz graphs were compared, along with a number of other design choices, as routing algorithm and collision management policies. This analysis shows that the Kautz topology yields the best results in terms of area occupation and obtainable throughput. In particular, in [14] a 22-nodes Kautz NoC was used to fully support IEEE 802.16e standard, each node being connected to a decoding PE and to three other nodes via a 4-way router.

IV. Decoder Reconfiguration

Flexible decoders available in the literature [9]–[13], [16], [17], [19], [20], though supporting a wide range of codes, do not address the reconfiguration issue. Change of decoding mode, standard or code parameters requires not only hardware support, but also memory initialization and specific controls: since in many standards a code switch can be issued as early as one data frame ahead [5], a time efficient reconfiguration technique must be developed.

For the proposed decoder the reconfiguration task consists of i) rewriting the location memory containing $t'_{i,j}$ values; ii) reloading the CN degree (*deg*) parameters and the window size in the control unit of LDPC decoding cores and SISOs respectively. In the following, the whole set of storage locations to be updated at reconfiguration time will be indicated as "reconfiguration memory". When possible, the decoder must be reconfigured while the decoding process is still running on the previous data frame. This means that the reconfiguration data can be distributed by means of the NoC interconnections only at the cost of severe performance penalties. Consequently, we suppose that the reconfiguration data are moved directly to the PEs via a set of N_b dedicated buses, each one linked to $\frac{P}{N_b}$ PEs.

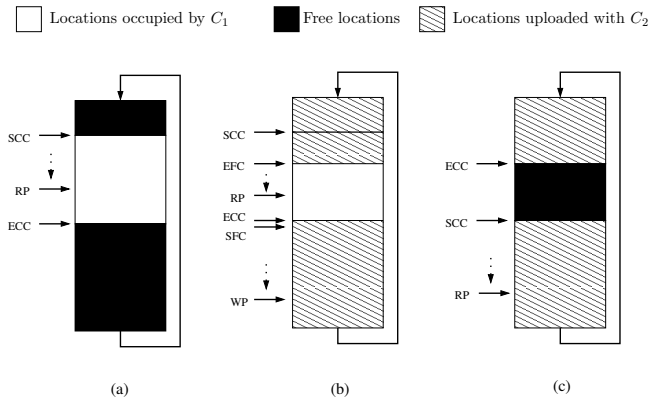

Figure 2. Memory reconfiguration process: (a) Decoding of C_1; (b) Upload of reconfiguration data required for C_2 (phases Φ_1 to Φ_3 and Φ_5); (c) First iteration of C_2 and concurrent upload of reconfiguration data (Φ_4)

In the following we estimate reconfiguration occurrence assuming mobile receivers moving at different speeds and the carrier frequency $f_c = 2.4$ GHz. This frequency is included in most standards' operation range, and used in a variety of applications. In this scenario the communication channel is affected by fading phenomena, namely slow fading, whose effects have very long time constants, and fast fading. Fast fading can be modeled assuming a change of channel conditions every time the receiver is moved by a distance similar to the wavelength λ of the carrier. Being $\lambda = 0.125$ m, at a speed $v = 70$ km/h the channel changes with a frequency $f_{chng} = 155$ Hz (WiMAX, WiFi, 3GPP-LTE), whereas, at $v = 10$ km/h (DVB-RCS, HPAV, CMMB, DTMB) changes occurs at $f_{chng} = 22$ Hz. These scenarios result in different reconfiguration probabilities, whose impact on BER performance is addressed in Section V.

The reconfiguration memory is organized as a circular buffer: two sets of pointers are used to manage reading and writing operations. The Start of Current Configuration (SCC) pointer and the End of Current Configuration (ECC) pointer delimit the memory blocks that are currently being used. A Read Pointer (RP) is used to retrieve the data during the decoding process, as shown in Fig. 2.(a). The Start of Future Configuration (SFC) and End of Future Configuration (EFC) pointers are instead used concurrently with the Write Pointer (WP) to delimit the locations that are going to be used to store the new configuration data.

The reconfiguration of the considered decoder to switch from the code currently processed (C_1) to a new one (C_2) can be overlapped with the decoding of both current and new code, provided that enough locations are free in the configuration memories. In particular, part of the configuration process can be concurrent with the decoding of one or more frames of C_1; if necessary, another portion of the configuration can be scheduled during the first iteration of the new code C_2. Finally, in case the overlap with decoding activity is not sufficient to complete the whole configuration, a further option is pausing the decoder by skipping one or more iterations on the last received frame for C_1 and using the available time, before starting the decoding of the new frame encoded with C_2.

Let us define B as the size of the location buffer available at each PE to store configuration data, t_{it1} and t_{it2} as the duration in clock cycles of a single decoding iteration for codes C_1 and C_2. Moreover, l_{c1} and l_{c2} express the number of locations required to store configurations of codes C_1 and C_2 at each PE, and n_{it1} and n_{it2} their iteration numbers.

In the considered architecture, the duration of one decoding iteration t_{it} expressed in clock cycles is directly proportional to the number of memory locations a PE has to read throughout the decoding process, and consequently to the number of used locations in the reconfiguration memory (l_c). Though the actual relationship between t_{it} and l_c is affected by memory scheduling and ratio between PE and NoC clock frequencies, this analysis is carried out with the worst-case assumption that the reconfiguration memory is read at every clock cycle of each iteration, setting $l_c = t_{it}$ for both C_1 and C_2 codes.

We define five phases Φ_i, $i = 1, 2, 3, 4, 5$ in the configuration process and for each phase we identify i) $t_a^{\Phi_i}$ as the number of clock cycles available during phase Φ_i, and ii) $l_a^{\Phi_i} = N_b \cdot t_a^{\Phi_i}/P$ as the number of locations in each reconfiguration memory that can be written in $t_a^{\Phi_i}$ clock cycles.

- Φ_1 In the reconfiguration from code C_1 to code C_2, l_{c1} words must be replaced with l_{c2} new words. The first part of the configuration can be scheduled during the initial $n_{it1} - 1$ decoding iterations on C_1 and therefore the available time is $t_a^{\Phi_1} = (n_{it1} - 1) \cdot t_{it1}$; in this range of time a maximum of $l_a^{\Phi_1} = \frac{N_b}{P} \cdot (n_{it1} - 1) \cdot t_{it1}$ words can be loaded into each buffer. However, assuming that the buffer size is larger than l_{c1}, we define $B - l_{c1}$ as the number of unused memory blocks in current configuration for code C_1. Therefore, the actual number of locations written in Φ_1 is the minimum between $B - l_{c1}$ and $l_a^{\Phi_1}$. The SFC pointer is thus initialized as ECC (Fig. 2.(b)).
- Φ_2 During the last iteration on C_1, every memory location between SCC and the current position of RP is available for reconfiguration. This means that up to l_{c1} locations are available for receiving configuration words for C_2. However, this has to be done during a single iteration, and therefore $t_a^{\Phi_2} = t_{it1}$ cycles are available. During these cycles, up to $l_a^{\Phi_2} = \frac{N_b}{P} \cdot t_{it1}$ words can be loaded.
- Φ_3 As mentioned before, part of the configuration can be overlapped with the first decoding iteration on C_2 code. SCC is initialized as SFC, and RP will take the duration of a full iteration to arrive to ECC (Fig. 2.(c)). The available time is $t_a^{\Phi_3} = t_{it2}$ and the maximum number of words that can be loaded in this phase is $l_a^{\Phi_3} = \frac{N_b}{P} \cdot t_{it2}$.
- Φ_4 In the event that previously listed phases are not sufficient to complete the configuration, an early stopping in the decoding of code C_1 can be scheduled to make available additional cycles to be used for loading the remaining part of the configuration words. We indicate the number of cycles available in this phase as $t_a^{\Phi_4} = t_{stop}$. The number of words that can be loaded in Φ_4 is $l_a^{\Phi_4} = \frac{N_b}{P} \cdot t_{stop}$. As one or more complete iterations are dropped in Φ_4, t_{stop} is a multiple of t_{it1}, which can be formalized as

$$t_{stop} = n_{stop} \cdot t_{it1}, \qquad n_{stop} = 0, 1, 2, 3, \cdots \quad (12)$$

Differently from the other four phases, Φ_4 affects the

Table I
RECONFIGURATION PHASES Φ_i: $t_a^{\Phi_i}$, AVAILABLE CLOCK CYCLES DURING Φ_i AND $l_a^{\Phi_i}$ NUMBER OF LOCATIONS THAT CAN BE WRITTEN IN $t_a^{\Phi_i}$

	$t_a^{\Phi_i}$	$l_a^{\Phi_i}$
Φ_1	$(n_{it1}-1)\cdot t_{it1}$	$\frac{N_b}{P}\cdot(n_{it1}-1)\cdot t_{it1}$
Φ_2	t_{it1}	$\frac{N_b}{P}\cdot t_{it1}$
Φ_3	t_{it2}	$\frac{N_b}{P}\cdot t_{it2}$
Φ_4	t_{stop}	$\frac{N_b}{P}\cdot t_{stop}$
Φ_5	$n_{it1}\cdot t_{it1}\cdot N_f$	$\frac{N_b}{P}\cdot n_{it1}\cdot t_{it1}\cdot N_f$

decoder performance, as if $n_{stop} > 0$ the number of decoding iterations is reduced for code C_1. Evaluating the actual effect on BER and FER curves is necessary to understand the feasibility of this approach.

Φ_5 If necessary, the reconfiguration process can be overlapped with the decoding of a number N_f of data frames encoded with C_1, in addition to the last frame, which was already considered in Φ_1. The available time depends on the chosen N_f:

$$t_a^{\Phi_5} = n_{it1}\cdot t_{it1}\cdot N_f, \qquad l_a^{\Phi_5} = \frac{N_b}{P}\cdot n_{it1}\cdot t_{it1}\cdot N_f \quad (13)$$

The five described phases are reported in Table I, together with the corresponding $t_a^{\Phi_i}$ and $l_a^{\Phi_i}$. Thus, B, N_b, n_{stop} and N_f are design parameters, and their values must be decided based on decoder parallelism (P) and supported codes, which determine l_{c1} and l_{c2}.

Two alternative cases can arise during Φ_1: either this phase is limited by the available time, or it is limited by the number of free locations in the reconfiguration memory:

$$(n_{it1}-1)\cdot t_{it1} \gtrless \frac{P}{N_b}\cdot(B-l_{c1}) \quad (14)$$

Then, assuming $t_{it1} = l_{c1}$ we define the threshold

$$l_{th} = \frac{P\cdot B}{P+(n_{it1}-1)\cdot N_b} \quad (15)$$

and distinguish between two cases:

1) $l_{c1} < l_{th}$ (*small* C_1 codes),
2) $l_{c1} \geq l_{th}$ (*large* C_1 codes).

Let us study the two cases separately.

A. $l_{c1} < l_{th}$: small C_1 codes

When $l_{c1} < l_{th}$, phase Φ_4 is not useful at all, as dropping n_{stop} decoding iterations has the effect of reducing the time of Φ_1 by the same amount that is gained in Φ_4. Therefore, the following constraint can be set:

$$\frac{P}{N_b} l_{c2} < t_a^{\Phi_1} + t_a^{\Phi_2} + t_a^{\Phi_3} + t_a^{\Phi_5} \quad (16)$$

This constraint simply means that the overall available time through Φ_1, Φ_2, Φ_3 and Φ_5 must be long enough to update l_{c2} locations in the reconfiguration memories. From the values in the second column of Table I the constraint in (16) becomes

$$l_{c2} < \frac{N_b}{P}\cdot(n_{it1}\cdot t_{it1} + t_{it2}) + \frac{N_b}{P}\cdot n_{it1}\cdot t_{it1}\cdot N_f \quad (17)$$

Then, if $t_{it2} = l_{c2}$, we have

$$l_{c2} < \frac{N_b\cdot n_{it1}\cdot(1+N_f)}{P-N_b}\cdot l_{c1} \quad (18)$$

A number N_f of preceding frames can be exploited only if enough locations are unused in the buffers during Φ_1 and Φ_5. This condition can be expressed as

$$t_a^{\Phi_1} + t_a^{\Phi_5} \leq \frac{P}{N_b}\cdot(B-l_{c1}) \quad (19)$$

namely

$$(n_{it1}-1)\cdot t_{it1} + n_{it1}\cdot t_{it1}\cdot N_f \leq \frac{P}{N_b}\cdot(B-l_{c1}). \quad (20)$$

Thus, given that $t_{it1} = l_{c1}$, the maximum useful value of N_f depends on l_{c1} as

$$N_f(l_{c1}) \leq \frac{\frac{P}{N_b}\cdot(B-l_{c1}) - (n_{it1}-1)\cdot l_{c1}}{n_{it1}\cdot l_{c1}} \triangleq Nf_{max} \quad (21)$$

Thus, (18) can be better written as

$$l_{c2} < \frac{N_b\cdot n_{it1}\cdot[1+N_f(l_{c1})]}{P-N_b}\cdot l_{c1} \quad (22)$$

This means that the size of code C_2 has an upper bound and this bound is proportional to the size of code C_1. Therefore, the most critical reconfiguration cases are those involving "small" C_1 codes: in such cases, there could be many C_2 codes that violate condition (22). The bound is also proportional to N_b, and can be consequently increased by rising the number of reconfiguration buses.

B. $l_{c1} \geq l_{th}$: large C_1 codes

In this case, $l_{c1} \geq l_{th}$. Now the use of phase Φ_4 makes sense as the duration of Φ_1 does not depend on the number of iterations, because it is limited by the number of free locations in the reconfiguration memory. As a consequence, additional reconfiguration time can be gained if n_{it1} is reduced. On the contrary, Φ_5 is not useful for large C_1, because Φ_1 is limited by the available memory, whereas the number of available cycles is sufficient. Thus, in this case Φ_1 is completed in $P/N_b\cdot(B-l_{c1})$ cycles (when all the available locations are written) and the constraints on l_{c2} is now written as

$$l_{c2} < B - l_{c1} + \frac{N_b}{P}\cdot(t_{it1}+t_{it2}+t_{stop}) \quad (23)$$

If $t_{it1} = l_{c1}$ and $t_{it2} = l_{c2}$, we have

$$l_{c2} < \frac{P\cdot B}{P-N_b} - \left(1 - \frac{N_b\cdot n_{stop}}{P-N_b}\right)\cdot l_{c1} \quad (24)$$

Also for this case, there is a limit to the size of code C_2 that can replace C_1 during phases from Φ_1 to Φ_4. However, this limit can be increased by increasing n_{stop} or B.

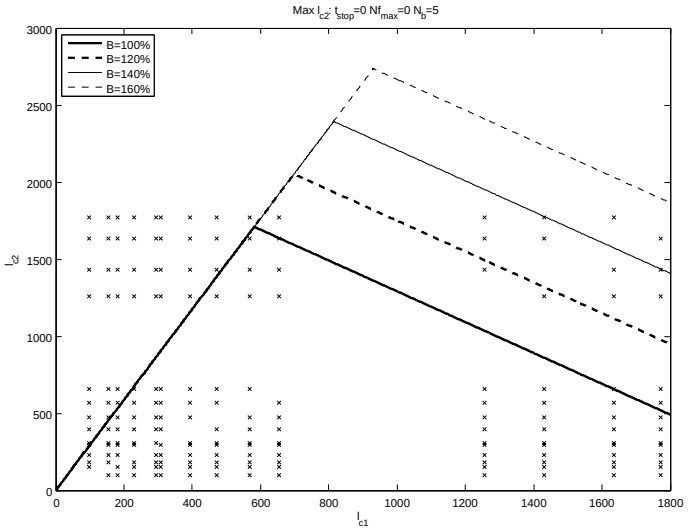

Figure 3. Maximum l_{c2} plot with varying B

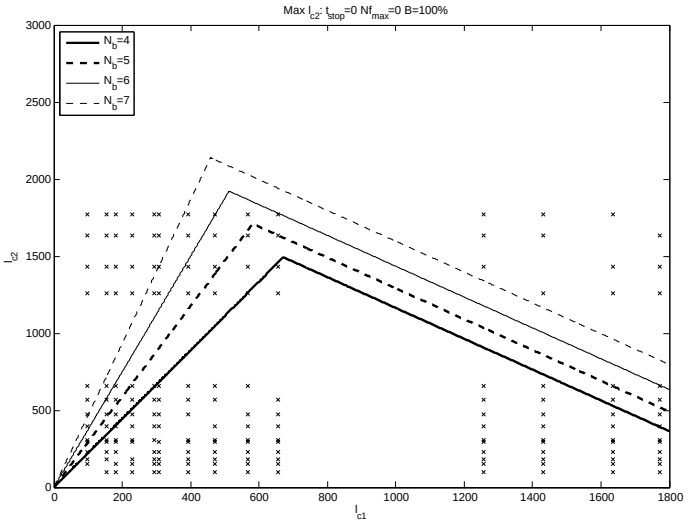

Figure 4. Maximum l_{c2} plot with varying N_b

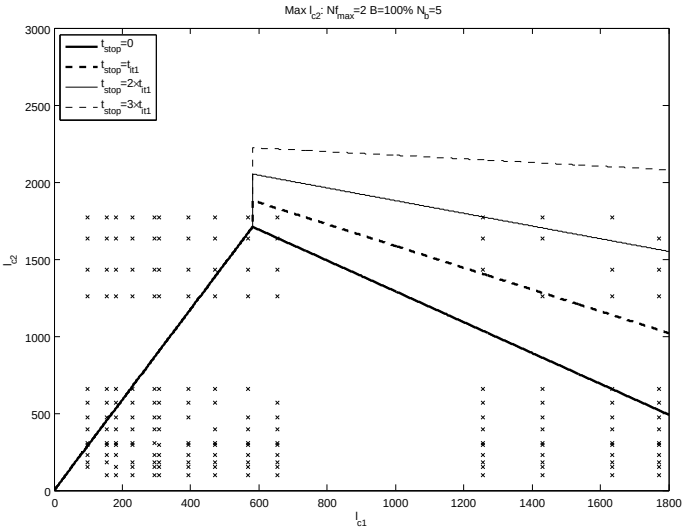

Figure 5. Maximum l_{c2} plot with varying t_{stop}

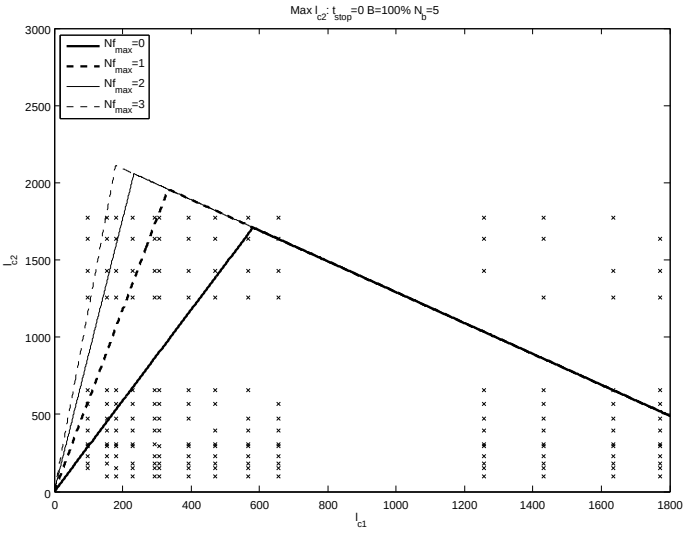

Figure 6. Maximum l_{c2} plot with varying Nf_{max}

V. Reconfiguration: cases and examples

The reconfiguration method detailed in Section IV has been applied to a set of target standards, in order to identify suitable design parameters (i.e. N_b, B, n_{stop}, Nf_{max}) that enable reconfiguration without pausing the decoder for most of code sizes. The following analysis has been performed with $n_{it1} = 10$.

Figures 3 to 7 plot the maximum l_{c2}, as defined by (22) and (24), for a continuous set of l_{c1} values. The $\times$ markers represent a subset of the considered *intra*- and *inter*-standard code changes: markers below the curve identify reconfigurations that can be performed without pausing the decoder.

Figure 3 shows the maximum l_{c2} for different values of B: in this plot, $B = 100\%$ corresponds to $B = 1771$, which is the size of the largest considered l_{c1}, while 160% means $B = 1.6 \cdot 1771$. It can be seen that in the cases of small C_1 codes, increasing the buffer size does not affect the positive slope portion of the curve.

On the contrary, in Fig. 4, the maximum l_{c2} is shown for different values of N_b: in this case, an increase of N_b is reflected in all areas of the plot. A higher number of buses means a shorter reconfiguration time, and a larger maximum l_{c2}.

Variation in the maximum allowed t_{stop} (Fig. 5) only affects the maximum l_{c2} in case of large C_1 codes (negative-slope portion of the curve), as shown in (24). It can be noticed that with $n_{stop} = 3$ all the large codes are below the right side of the curve: later in this section it will be demonstrated how these skipped iterations are negligible in terms of BER performance.

In Fig. 6, the effect of different choices of N_f is shown: from the plot it can be seen that $N_f > 0$ actually increases the maximum l_{c2} only for small C_1 codes.

Finally, Fig. 7 plots some combinations of the analyzed parameters in order to allow dynamic reconfiguration among most of considered codes. The represented combinations of N_b, B, t_{stop} and Nf_{max} all yield very similar performance: the cost underlying every parameter choice consequently becomes the decision metric. A 20% increase in memory, even if backed up by a smaller number of buses, heavily

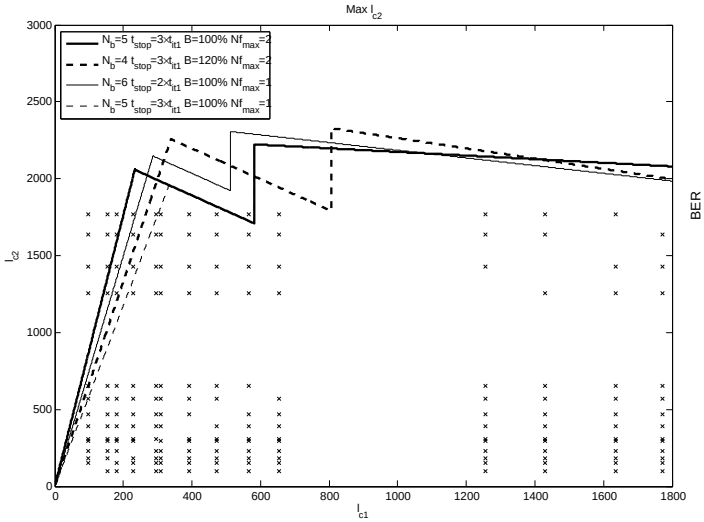

Figure 7. Maximum l_{c2} plot, different solutions

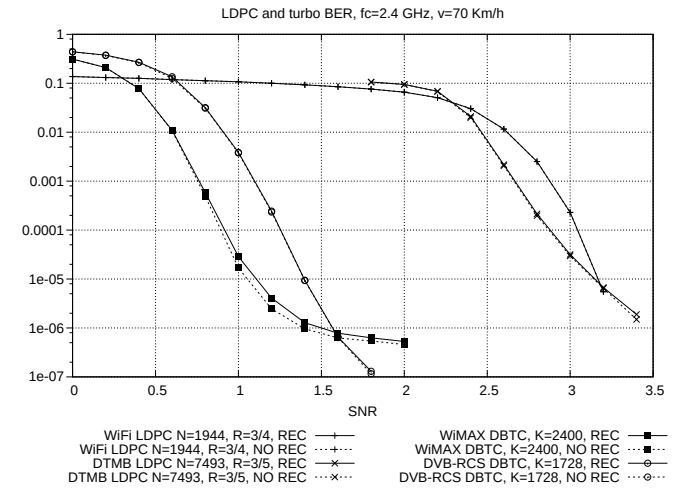

Figure 8. LDPC and turbo BER with and without reconfiguration loss, AWGN channel, BPSK modulation

affects the decoder area occupation, ruling out the solution represented by the thick dashed line. Among the remaining three combinations, the one that makes use of 6 buses yields a higher area occupation than the others. Since with $Nf_{max} = 1$ the thin dashed curve crosses one of the lower $\times$ markers, the final choice falls on $N_b = 5$, $B = 100\%$, $t_{stop} = 3 \cdot t_{it1}$ and $Nf_{max} = 2$. Given that $P = 22$, and consequently $\frac{P}{N_b} = \frac{22}{5}$ is not an integer number, every bus will be exclusively connected to four nodes, while the reconfiguration of the remaining two nodes will be shared among all 5 buses.

The impact of the reconfiguration process on the decoder area is addressed in Section VIII, whereas a set of BER simulations has been performed to evaluate the impact of different t_{stop}, on WiFi, DVB-RCS, WiMAX, CMMB, DTMB, 3GPP-LTE and HPAV codes. Considering the worst case for each tested standard (i.e. the largest block length, the most unfavorable throughput/code rate ratios), the reconfiguration probability can be expressed as the probability for each incoming frame to request a code change, computed as the channel changing frequency f_{chng} over the number of coded frames received in a second:

$$P_R = \frac{f_{chng} \cdot r \cdot N}{T_{max}} \tag{25}$$

where T_{max} is the maximum throughput required by the standard for the N and r code choices. The reconfiguration probability ranges between 0.25% and 0.3% in presence of the fast moving receiver, while it remains under 0.15% in the other case. Simulation results show how the BER penalty is negligible as long as $n_{it} - n_{stop} \geq \lceil n_{it}^{avg} \rceil$, with n_{it}^{avg} the average number of iterations performed before a correct codeword is obtained and $\lceil \cdot \rceil$ is the next highest integer value.

Figure 8 shows the BER curves obtained with $n_{it} = 10$ and $n_{stop} = 3$, in the pessimistic assumption that a reconfiguration requiring always $n_{stop} = 3$ occurs with P_R. As it can be observed, the difference between the case when reconfiguration occurs (solid lines) and the no-reconfiguration case (dashed lines) is completely negligible.

VI. Decoding Cores

The design of the decoding cores must yield the same degree of flexibility of the NoC, being as independent as possible of the set of supported codes. In [14] a completely serial LDPC decoding core has been designed, mostly independent of block length and code rate: an arbitrary number of CN operations can be scheduled on it. The same holds true for the serial SISO, where different windows can be scheduled, regardless of the size of the interleaver.

This work stems from the results presented in [14], improving the architectures through novel memory scheduling and addressing methods, reduced latency and simpler control. As shown in [36], sharing the datapath of a min-sum based decoder architecture with a log-MAP SISO does not provide significant advantages. As a consequence, in this work logic sharing is not addressed. Experimental results show that the area of the architecture is dominated by memories indeed.

A. Quantization and Memory Organization

Memory organization evolves from the idea presented in [14], in which in every decoding core two memories are instantiated: a 7-bit memory and a 5-bit memory. Their usage is shown in the left part of Fig. 9: LDPC VN-to-CN values are stored in the 7-bit memory, together with turbo extrinsic information and state metrics. The 5-bit memory is instead used for CN-to-VN values in LDPC decoding, while storing the intrinsic channel information in turbo decoding. The memories are sized to the largest WiMAX codes ($N = 2034$, $M = 576$ for LDPC and $K = 2400$ for turbo). However, according to post-layout synthesis results, memory access multiplexers suffer from excessive area overhead for these particular cuts. To reduce this problem and at to reduce at the same time the overall memory area occupation, a novel memory organization technique is proposed, as shown in the rightmost part of Fig. 9. Different colors highlight different metrics, while black-striped parts are unused.

Extensive simulations of WiFi, WiMAX, CMMB and DTMB have shown how, in LDPC decoding, $\lambda_k[c]$ and channel

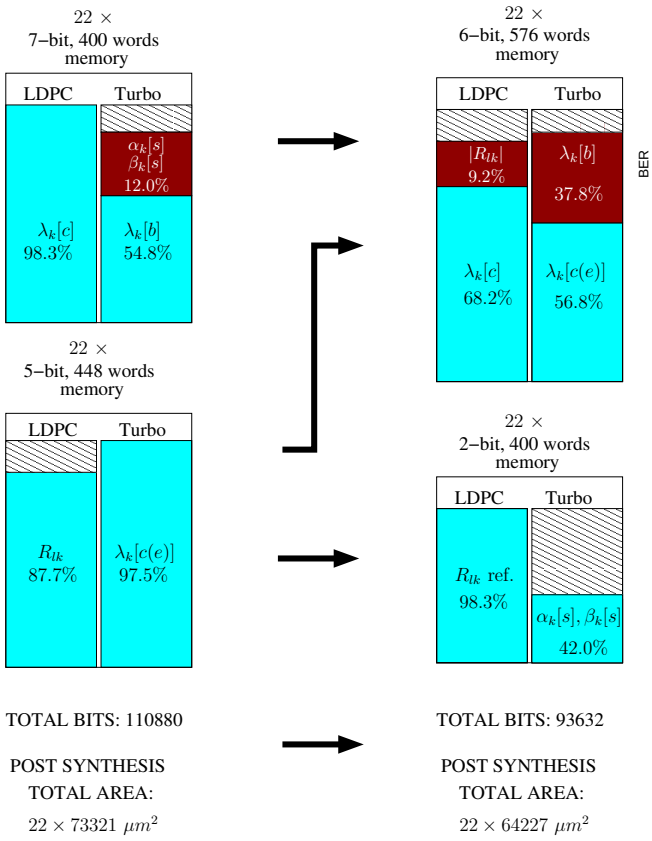

Figure 9. Memory organization, WiMAX maximum usage percentages

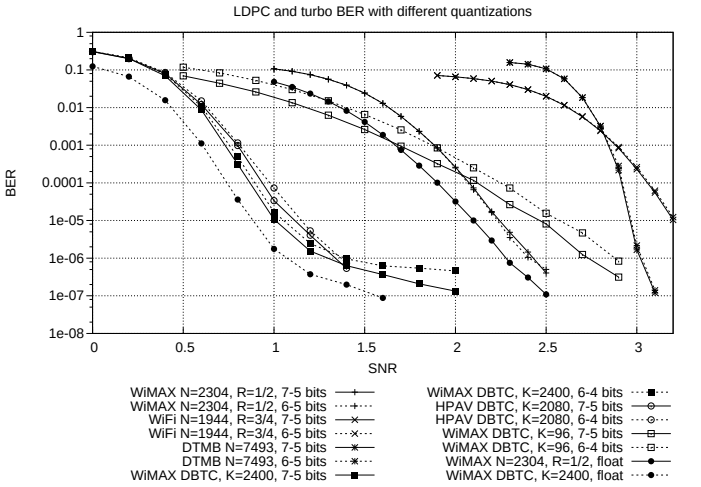

Figure 10. LDPC and turbo BER with quantization change, AWGN channel, BPSK modulation

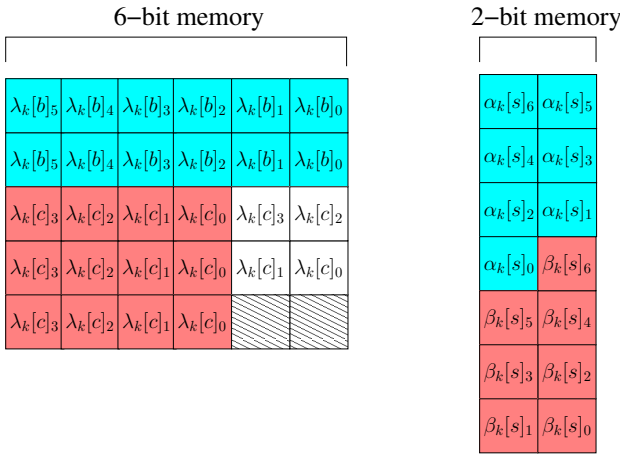

Figure 11. Turbo mode in-depth memory organization

LLR quantization can be reduced from 7 to 6 bits without consistent performance degradation. Figure 10 shows the BER curves for some WiMAX, DTMB and WiFi LDPC codes with the two quantization choices: the difference is smaller than 0.05 dB for all rates of medium and large code sizes. On the same graph, yielding similar results, a few turbo codes examples (WiMAX and HPAV) are plotted, in which $\lambda_k[b]$ and the channel LLR representation changes from 7 to 6 bits, and $\lambda_k[c(e)]$ from 5 to 4 bits (the meaning of $\lambda_k[b]$ will be detailed in Section VI-C1). Also for turbo codes, the performance loss introduced by the proposed quantization change is almost negligible. Very small codes, as the ones that can be found in 3GPP-LTE and WiMAX, suffer more from the quantization reduction (Fig. 10). Curves obtained with floating point precision show improvements between 0.1 and 0.2 dB w.r.t. the selected precisions. Thanks to these changes, a single 6-bit wide memory is instantiated, in which both $\lambda_k[c]$ and R_{lk} values are saved. Storing all the R_{lk} values requires $\frac{M \cdot row_{deg}}{P} = \frac{576 \cdot 15}{22}$ locations in each decoding core. However, with the normalized min-sum algorithm the number of necessary bits can be reduced by 21.2% by changing the addressing mode as follows. For every CN in the **H** matrix *deg* metrics R_{lk} are updated. Since R_{lk} can take only two possible values, for each CN we can memorize $576 \cdot 2$ magnitudes, and $576 \cdot 15$ 2-bit indexes that identify the correct R_{lk} magnitude and its sign.

The sizing of the 6-bit memory is determined by the Double Binary Turbo Code (DBTC) decoding mode, since it must store $\lambda_k[b]$ and $\lambda_k[c(e)]$ values. To limit the area overhead and speed up the loading process, four $\lambda_k[c(e)]$ values are stored in three memory locations, as portrayed in the left part of Fig. 11. Three 4-bit $\lambda_k[c(e)]$ are stored in three 6-bit locations: the remaining metric can be divided in two pairs of bits, and stored in the leftover locations. Three clock cycles are used to read the four $\lambda_k[c(e)]$ values for a trellis step with minimal logic overhead. In case of Single Binary Turbo Codes (SBTC), like those used in 3GPP-LTE, only two $\lambda_k[c(e)]$ and one $\lambda_k[b]$ are necessary for a trellis step, and they can be read in two clock cycles without impairing the throughput.

With a similar method the 2-bit memory is used in turbo decoding mode to store $\beta_k[s]$ and $\alpha_k[s]$ between iterations, as suggested in [34]. Six locations are used to store 2 $\beta_k[s]$ or $\alpha_k[s]$ (Fig 11): since at most three 8-state windows initialization metrics, i.e. 24 $\beta_k[s]$ and 24 $\alpha_k[s]$, are stored at the same time, only 144 out of 400 locations are used. Multiple memory accesses are necessary to read a single value: the issue is handled with appropriate scheduling (see Section VI-C2) and does not affect the throughput.

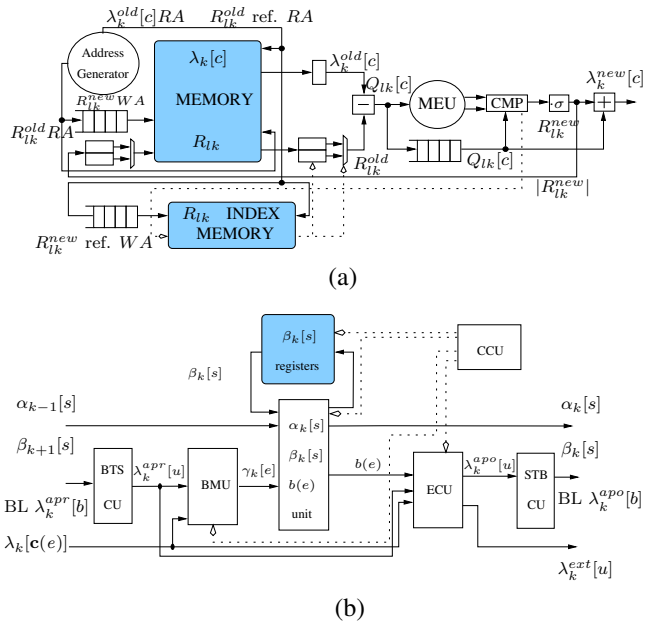

Figure 12. (a) LDPC decoding core, (b) turbo decoding core (SISO)

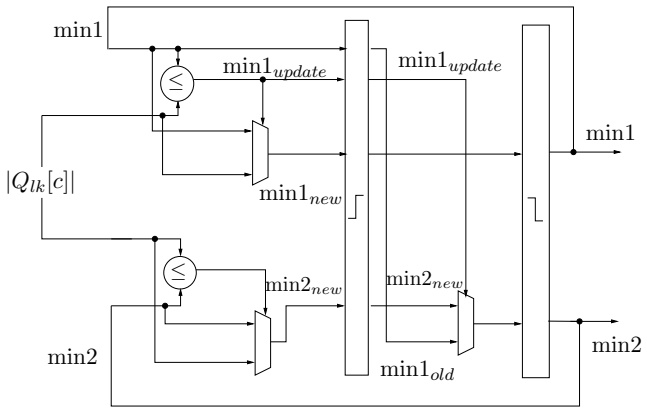

Figure 13. Minimum Extraction Unit

B. LDPC Decoding Core

The LDPC decoding core used in the decoder described in [14] relies on a serial architecture suited for exclusive memory usage. The main drawback of this solution is the variable number of cycles to produce the output. The average number of cycles–per–data varies between one and two. To overcome this limitation and to share the memory with the SISO a novel architecture with limited area overhead is proposed.

1) Architecture: The LDPC decoding core is detailed in Fig. 12.(a): this architecture supports all kind of LDPC codes, as long as the memory requirements are met.

A $Q_{lk}[c]$ value (1) is produced at every clock cycle and fed to the Minimum Extraction Unit (MEU) depicted in Fig. 13. Then, $|Q_{lk}[c]|$ is compared to the current first and second minimum (min1 and min2), that are initialized as the maximum allowed value at the beginning of each CN phase. The minimum of both comparisons (min1_{new} and min2_{new}) is passed on and sampled on the rising edge of the clock signal, together with the previous first minimum (min1_{old}) and a flag signaling if $\text{min1} \neq \text{min1}_{new}$ (min1_{update}). If $\text{min1}_{update} = 0$, min2_{new} is substituted with min1_{old}: min1 and min2 are finally updated on the falling edge of the clock, ready and stable for the next $|Q_{lk}[c]|$. Differently from the MEU used in [14], that could halt the pipeline in case both min1 and min2 had to be updated, the negative–edge triggered registers allow both updates in a single clock cycle, leading to a constant cycles–per–data rate close to one, after the initial *deg*+2 cycles latency. Concurrently, $Q_{lk}[c]$ signs are XORed as in (3).

Once min1 and min2 have been successfully extracted, they are compared to all the $Q_{lk}[c]$ of the CN, that are delayed by a number of clock cycles equal to the degree of the CN (*deg*), to compute R_{lk}^{new} as in (4). The CMP unit handles the comparison and produces the two flags (sign and identification) to be stored in the R_{lk} index memory. The correction factor σ in (6) is applied before the final addition in (5) and $\lambda_k^{new}[c]$ are sent to the output buffer.

The length of the delay lines used for $Q_{lk}[c]$, R_{lk}^{new} magnitudes and indexes is initialized by the control unit to *deg*.

2) Memory Scheduling: Both 6-bit and 2-bit memories are implemented as dual port RAMs, allowing two concurrent operations. At iteration n, for the j-th CN, two clock cycles are devoted to write on port 1 of the 6-bit memory. This allows the storage of the two R_{lk}^{new} magnitudes of CN j and computed during current iteration. On the contrary, port 2 is set to read mode, loading the two R_{lk}^{old} magnitudes of CN $j+1$ stored during previous iteration. In the 2-bit memory port 1 is always in write mode, storing R_{lk}^{new} indexes as soon as they are computed, while port 2 is constantly in read mode. During this first phase, though, no data is loaded.

The second phase of the scheduling lasts for *deg* cycles. The ports on the 6-bit memory switch functionality: port 1 is used to store $\lambda_k[c]$ incoming from the network, while port 2 is used to read *deg* $\lambda_k^{old}[c]$ values of CN $j+1$. The 2-bit memory is enabled, loading the R_{lk}^{old} indexes of CN $j+1$ and storing R_{lk}^{new} indexes of CN j.

C. Turbo Decoding Core

As for the LDPC decoding core, also the SISO core yields a very high degree of flexibility, limited only by the size of the memories: any double-binary turbo code can be decoded as long as the memory capacity is sufficient.

1) Architecture: Figure 12.(b) portraits the designed architecture. The SISO interfaces with the NoC via two dedicated input and output blocks, respectively called Bit-To-Symbol Conversion Unit (BTS CU) and Symbol–To–Bit Conversion Unit (STB CU). According to [37], symbol-level (SL) information in double-binary codes can be approximated from bit-level (BL) extrinsics, with a limited BER loss and an average NoC complexity reduction of 1/3. The BTS-CU changes the received *a priori* BL $\lambda_k^{apr}[b]$ to SL $\lambda_k^{apr}[u]$, as required by the algorithm, while the STB-CU reduces the number of messages to be sent on the NoC by converting $\lambda_k[u]$ *extrinsic* values into BL $\lambda_k[b]$.

For every trellis step in a window, the Branch Metric Unit (BMU) and the Extrinsic Computation Unit combine the two $\lambda_k^{apr}[b]$ converted by the BTS-CU with four $\lambda_k[c(e)]$ values read from the 6-bit memory to calculate $\gamma_k[e]$ (11) and to

update $\lambda_k[u]$ respectively. The output of the BMU is used by the main computation unit, that tackles the calculation of $b(e)$, $\alpha_k[s]$ and $\beta_k[s]$ (8), (9) and (10). These metrics are computed in this exact order, thus storing $\beta_k[s]$ values in a dedicated set of registers while $\alpha_k[s]$ are being processed: the $b(e)$ metric, that needs both $\beta_k[s]$ and $\alpha_k[s]$, is calculated last.

2) Memory Scheduling: In turbo mode, each trellis step requires three clock cycles to be completed. However, up to five cycles are needed to read all the necessary $\lambda_k^{apr}[b]$ and $\lambda_k[c(e)]$. Early simulation results presented in [14] show that the SISO working frequency (f_{core}^{turbo}) can be lower than the NoC's one (f_{NoC}^{turbo}). By timing the memories with the faster clock signal, six values (five memory locations) can be read from port 1 of the 6-bit memory in three SISO-cycles. Port 2 is kept in write mode for the duration of the decoding, and used to store values coming from the network.

The 2-bit memory is used in the same way, with port 1 in read mode and port 2 in write mode. At the beginning of every new window 16 values are needed (8 $\alpha_k[s]$ and 8 $\beta_k[s]$) from this memory to initialize the trellis. Due to the memory organization used for state metrics, see Fig. 11, one $\alpha_k[s]$ and one $\beta_k[s]$ are spread over 7 memory locations. Consequently, $7 \cdot 8$ clock cycles are necessary to load the 16 values. The values must be loaded from the memory before the window is processed. Thus, they are loaded during the processing of the previous window. Since every window is composed of at least 20 trellis steps, requiring $3 \cdot 20$ clock cycles to be executed, there is enough time to load $\beta_k[s]$ and $\alpha_k[s]$ values to initialize the next window.

VII. Supported Standards

The 22-node architecture presented in this work has been tested on a large set of communication standards. In particular, the whole set of turbo and LDPC codes included in [1]–[6] have been tested.

As explained in the previous section, if f_{core} is smaller enough than f_{NoC}, communication time between PEs is negligible. Taking in account the presented 22-node architecture, the maximum ratio f_{core}/f_{NoC} for which this assumption stands is 2/3 for LDPC codes and SBTC, while 3/5 is necessary for DBTC. The maximum number of iterations $n_{it}^{(max)}$ has been set to 10 for LDPC codes, and to 8 for turbo codes.

Every standard has different throughput requirements: both f_{core} and f_{NoC} can be adjusted consequently.

- *IEEE 802.16e*: the WiMAX standard [5], is fully supported. A high enough throughput (> 70 Mb/s) can be obtained with $f_{core}^{LDPC} = 200$ MHz and $f_{core}^{turbo} = 80$ MHz. Table II summarizes the results.
- *IEEE 802.11n*: IEEE 802.11n standard [6] requires a higher throughput than WiMAX, demanding for the $N = 1944$, $r = 5/6$ code up to 450 Mb/s. The 22–node architecture can guarantee it with $f_{core} = 820$ MHz. Taking in account the f_{core}/f_{NoC} ratio constraint, this would mean $f_{NoC} = 1.23$ GHz. Both frequencies are over the decoder maximum working frequency, and two alternatives have been devised. By increasing the size of the NoC to 35 nodes, the f_{core}/f_{NoC} still holds at 2/3, but f_{core} can be lowered to 520 MHz, and $f_{NoC} = 780$ MHz. By choosing $f_{core} = 350$ MHz, support still can be given for other WiFi transmission modes, requiring up to 300 Mb/s. The results are shown in Table III.

Table II
IEEE 802.16e standard throughput (T), 10 iterations for LDPC, 8 for Turbo

CODE	f_{core} [MHz]	T [Mb/s]
DBTC	80	73
LDPC r=1/2	200	70
LDPC r=2/3	200	88
LDPC r=3/4	200	88
LDPC r=5/6	200	110

Table III
IEEE 802.11n standard throughput (T) with different f_{core} and NoC sizes, 10 iterations

CODE	T [Mb/s]		
	@520 MHz 35 nodes	@350 MHz 35 nodes	@200 MHz 22 nodes
LDPC r=1/2	248	167	60
LDPC r=2/3	364	245	88
LDPC r=3/4	406	273	94
LDPC r=5/6	455	306	110

- *DVB-RCS*: the return channel for DVB satellite communications [1], thought for multimedia applications, employs 12 different payloads and 7 coding rates. The throughput required by this standard is very small, and can go up to 2.05 Mb/s in case of corporate-driven applications. This throughput is easily sustained by the 22-node architecture with $f_{core} = 3$ MHz.
- *HomePlug AV*: the HPAV standard [2] makes use of a small set of DBTC, with interleaver sizes of 64, 544 and 2080. The throughput requirements of HPAV demand at least 150 Mb/s: on the 22-node architecture, with $f_{core} = 170$ MHz, achieved throughput is 156 Mb/s.
- *CMMB and DTMB*: the CMMB [3] and DTMB [4] Chinese broadcast standards, though serving the same purposes as DVB, work with smaller LDPC codes. Like in DVB, also in CMMB codes feature double diagonal submatrices, slightly limiting the concurrent number of row nodes that can be instantiated on the proposed decoder. Both CMMB and DTMB codes demand an increased memory capacity with respect to the aforementioned standards, requiring PE memories to be enlarged by 55% to support CMMB, and by 68% for DTMB. A working frequency $f_{core} = 60$ MHz is sufficient to guarantee the 20.22 Mb/s throughput required by CMMB standard, while to comply with DTMB 40.6 Mb/s, frequency must be risen to $f_{core} = 200$ MHz, as shown in Table V.
- *3GPP-LTE*: the LTE version of 3GPP [7] uses a set of 188 SBTC with coding rate 1/3, thus being characterized by a range of widely spaced block lengths. The required 150 Mb/s throughput can be obtained on the 22-node architecture with $f_{core} = 330$ MHz; however, if we

Table IV
RECONFIGURATION CASES IN INTRA- AND INTER-STANDARD COMBINATIONS. DARK GRAY: PERCENTAGE OF CODE COMBINATIONS REQUIRING DECODER PAUSING. LIGHT GRAY: PERCENTAGE OF CODE COMBINATIONS REQUIRING $0 < N_f \leq 2$. WHITE: ALL CODE COMBINATIONS RECONFIGURABLE WITH $N_f = 0$. THROUGHPUT OBTAINED WITH 10 ITERATIONS FOR LDPC, 8 FOR TURBO

C_1 \ C_2	WiMAX LDPC	WiMAX turbo	WiFi	DVB-RCS	HPAV	CMMB	DTMB	3GPP-LTE
WiMAX LDPC	3.5%	8.2%	1.8%	2.4%	14.0%	21.0%	36.4%	5.2%
WiMAX turbo	10.1%	15.8%	4.4%	5.9%	17.6%	47.0%	54.9%	10.0%
WiFi	8.2%	13.2%	6.8%	4.0%	19.4%	33.3%	44.4%	6.8%
DVB-RCS	18.5%	0.3%	13.5%	8.5%	22.2%	55.9%	67.0%	13.5%
HPAV	14.4%	17.6%	11.1%	17.8%	33.3%	33.3%	77.7%	12.3%
CMMB								
DTMB								
3GPP-LTE	7.9%	10.0%	5.3%	4.2%	12.3%	31.6%	38.0%	6.8%

Table V
CMMB AND DTMB STANDARD THROUGHPUT (T), 10 ITERATIONS

CODE	f_{core} [MHz]	T [Mb/s]
CMMB LDPC r=1/2	60	22
CMMB LDPC r=3/4	60	33
DTMB LDPC r=2/5	200	42
DTMB LDPC r=3/5	200	55
DTMB LDPC r=4/5	200	68

consider the extended 35-node architecture mentioned for the WiFi standard, compliance with the throughput requirement is met at $f_{core} = 200$ MHz. This standard requires additional 41% memory capacity w. r. t. WiMAX, WiFi, DVB-RCS and HPAV standards, but can be fully supported by the CMMB and DTMB memory sizing.

Table IV summarizes possible switching among the selected standards, taking in account all possible code combinations. The dark gray cells represent the percentages of C_1, C_2 combinations between two standards whose reconfiguration requires pausing of the decoder. A few cases arise between DVB-RCS and WiMAX turbo codes and within 3GPP-LTE (due to its wide variety of codes), while when C_2 belongs to the CMMB, DTMB and LTE standards, it is more likely to encounter a critical combination. On the contrary, if C_1 belongs to CMMB or DTMB standards, any reconfiguration can be completed with $N_f = 0$: this is also the most common situation among the other standards. The choice of maximum $N_f = 2$ allows to handle all the other reconfiguration cases: the light gray cells show the percentages of code combinations in which $0 < N_f \leq 2$ is necessary.

VIII. IMPLEMENTATION RESULTS

The results presented in Section VII show a broad range of possibilities for implementation, and the designed decoder can be scaled with very low effort. Three different complete decoders have been synthesized with TSMC 90 nm CMOS technology: post-layout results have been obtained for all of them, with accurate functional verification, area and power estimation. Synthesis has been carried on with Synopsys Design Compiler, functional simulation with Mentor Graphics ModelSIM, and place and route with CADence SoC Encounter [38].

A. Implementation A

The first decoder implementation has been devised to fully support WiMAX, HPAV and DVB-RCS standards. The memory sizing and organization described in Section VI-A is able to handle the addressed standards with 22 PEs. To comply with each standard throughput requirements, a single $f_{NoC} = 300$ MHz is sufficient in both LDPC and turbo mode, consequently identifying $f_{core}^{LDPC} = 200$ MHz and $f_{core}^{turbo} = 170$ MHz, both under the f_{core}/f_{NoC} constraint. Obtained throughput is presented in Table VI.

Each reconfiguration bus is 18 bits wide: 3 bits are the node identifier, used to address one of the connected decoding cores, 5 bits are assigned to the node degree or window size information, and the remaining 10 bits carry the $t'_{i,j}$.

These design choices have led to an overall area of 2.75 mm^2 after place and route, taking in account the reconfiguration additional hardware as well. The logic of the SISO cores occupies 15% of the overall area, while the LDPC cores 11%. Core memories account for another 53%, while the NoC, together with the reconfiguration buses and additional logic, constitute the remaining 21%. This area overhead is due to two specific functionalities that have been introduced in the proposed decoder: (i) full flexibility in terms of supported turbo and LDPC codes, and (ii) dynamic reconfiguration between different standards.

Estimated power consumption, based on the switching activity in case of WiMAX LDPC code $N = 2304$, $r = 1/2$ (for ease of comparison with the state of the art) is 87.8 mW; for WiMAX turbo code with $K = 2400$ estimated power is 101.5 mW.

A screenshot of the final layout is portrayed in Fig. 14: the irregularity of the placement is due to the large number of memories and their complex interconnections. However, two different areas can be easily identified: a central zone in which most of the logic is found (black contour), and a border area where the majority of memories have been placed, some of which are highlighted with a white line.

B. Implementation B

The second implementation presented extends the set of standards supported by implementation A to WiFi LDPC codes and 3GPP-LTE turbo codes. To limit the complexity

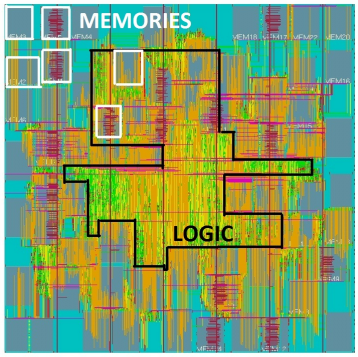

Figure 14. Implementation A layout screenshot

of off-chip clock generators, also in this case a single NoC working frequency has been chosen, $f_{NoC} = 780$ MHz, while $f_{core}^{LDPC} = 520$ MHz is necessary to provide high enough throughput, f_{core}^{turbo} can remain set to 200 MHz. The parallelism of the NoC is increased from 22 nodes to 35 nodes, the reconfiguration buses rise from 5 to 8, and the support of LTE requires an increase in the size of 6-bit memories. Throughput results are reported in Table VI. The post place & route estimated area is 4.87 mm^2, with 331.6 mW of power consumption in LDPC mode, and 183.2 mW in turbo mode.

C. Implementation C

This third implementation extends implementation A's support to CMMB and DTMB. Neither frequency nor NoC parallelism modification are necessary, but the core and reconfiguration memories must be enlarged. Consequently, an extra bit is added to the reconfiguration bus data width. The new post place & route estimated area is 3.42 mm^2, while power reaches 120 mW for both tested turbo and LDPC codes. This is because the LDPC consumption is calculated on a DTMB code, that makes full use of the extended memories, while the memory usage percentages for DBTC remains low. The enlarged memories allow also LTE codes to be decoded, but the SBTC f_{core} would need to rise up to 333 MHz to meet the throughput requirements. Throughput results for CMMB and DTMB are shown in the Implementation C column of Table VI.

D. Comparisons

Table VIII shows the detailed implementation results in comparison with the state of the art flexible turbo/LDPC decoders. Even though A, B and C are the only decoders capable of dynamic switching, area, power and efficiency figures prove the effectiveness of this approach.

In order to make a fair comparison, normalized area occupation has been included in the Table, $\text{An}_{tot} = \text{A}_{tot} \cdot (65/\text{Tp})^2$, where A_{tot} is the total area and Tp is the technology process, together with throughput and power consumption. Moreover, two further metrics have been introduced: the energy efficiency $E_{eff} = Pow/(T \cdot n_{it}^{(max)})$, where Pow is the peak power consumption, expressing the energy spent for decoded bit, and the area efficiency $\text{A}_{eff} = (T \cdot n_{it}^{(max)}/f_{clk}) \cdot (1000/\text{An}_{tot})$, reported in Table VII, an efficiency figure that considers both throughput and area occupation.

Table VI
THROUGHPUT (T) RESULTS FOR EACH STANDARD, WITH EVERY IMPLEMENTATION

	T [Mb/s]					
CODE STD, r	Impl. A f_{NoC} 300 MHz		Impl. B f_{NoC} 780 MHz		Impl. C f_{NoC} 300 MHz	
	f_{core} [MHz]	T [Mb/s]	f_{core} [MHz]	T [Mb/s]	f_{core} [MHz]	T [Mb/s]
DBTC						
WiMAX	170	156	200	292	170	156
HPAV	170	156	200	292	170	156
DVB-RCS	170	156	200	292	170	156
SBTC						
3GPP-LTE	N/A	N/A	200	150	170	78
LDPC						
WiMAX 1/2	200	70	520	289	200	70
WiMAX 2/3		88		364		88
WiMAX 3/4		88		364		88
WiMAX 5/6		110		455		110
WiFi 1/2	200	60	520	248	200	60
WiFi 2/3		88		364		88
WiFi 3/4		94		406		94
WiFi 5/6		110		455		110
CMMB 1/2	N/A	N/A	N/A	N/A	200	73
CMMB 3/4						110
DTMB 2/5	N/A	N/A	N/A	N/A	200	42
DTMB 3/5						55
DTMB 4/5						68

Baghdadhi *et al.* in [11] propose an ASIP decoder architecture supporting WiMAX and WiFi LDPC codes, and WiMAX, 3GPP-LTE and DVB-RCS turbo codes. The A, B and C implementations are designed such that the minimum throughput is sufficient to comply with the supported standards. On the contrary, worst case throughput in [11] is not high enough for WiMAX. Comparison reveals similar area occupations, but very different frequencies. This leads to a better area efficiency in all three proposed implementations for most of the codes: particularly evident is the difference for DBTC (second last row of Table VII).

The work presented in [9] supports convolutional, LDPC and turbo codes, giving results for WiMAX LDPC, WiFi and general binary and double-binary turbo codes. It yields a very small area occupation with low power consumption and good maximum throughput for LDPC decoding. On the contrary, it features less interesting figures in turbo mode. This situation is reflected both on E_{eff} and A_{eff}, with Implementation A, B and C having, when comparing the same codes, better efficiencies in turbo mode (last row of Table VII), and worse in LDPC mode. However, under the worst case conditions (N=672, $r = 1/2$, 20 iterations), A and B outperform [9] also in LDPC mode.

The multi-standard decoder designed in [12] supports 3GPP-HSDPA, WiFi, WiMAX and DVB-SH. No specific information on the codes used is given, only minimum guaranteed throughput: for this reason, results in Table VII refer to the minimum throughput of each standard. Implementation A and B have comparable minimum A_{eff} when working with WiMAX LDPC codes, and A, B and C yield much better results in turbo mode. When comparing WiFi results [12] guarantees a higher A_{eff} than A, B and C, even though aiming for a lower throughput than B. All three proposed

Table VII
AREA EFFICIENCY (A_{eff}) FOR DIFFERENT CODES AND IMPLEMENTATIONS. N/A: CODE NOT SUPPORTED. DASH: RESULTS NOT AVAILABLE.

	A_{eff} $[\frac{bits}{mm^2 \cdot kcycles}]$						
CODE	A	B	C	[11]	[9]	[12]	[13]
LDPC						(min)	
2304, 1/2	2447	2188	1966	882	–	2013	–
2304, 5/6	3846	3445	3090	4412	9589		10779
1944, 5/6	3846	3445	3090	3719	10363	3484	8982
7493, 4/5	N/A	N/A	1910	N/A	N/A	N/A	N/A
Turbo						(min)	
DB 2400	5134	4598	4124	1468	750	2084	N/A
SB 6144	N/A	2362	2062	1468	375	N/A	3233

implementations yield better E_{eff}, and both A and C have a smaller area occupation.

Sun and Cavallaro describe in [13] a decoder working with 3GPP-LTE turbo codes and WiMAX and WiFi LDPC codes. They obtain very high maximum throughput efficiency in both LDPC and turbo mode: the range of supported codes is however quite limited w.r.t. all considered implementations, and the area occupation is larger than A. Since no power analysis is given, comparison based on E_{eff} is impossible, although the difference in working frequencies would suggest a smaller power consumption for at least A and C.

IX. CONCLUSIONS

This work describes a flexible turbo/LDPC decoder architecture able to fully support a wide range of modern communication standards. A complete analysis of the never previously addressed inter- and intra-standard reconfiguration issue is presented, together with a dedicated reconfiguration technique that limits the complexity overhead and performance loss. Three different implementations are proposed to cover different sets of standards. Full layout design has been completed to provide accurate area and power figures. Comparison of the proposed architectures with the state of the art show very good efficiency, competitive area occupation and an unmatched degree of flexibility.

REFERENCES

[1] *Digital Video Broadcasting (DVB); Interaction channel for Satellite Distribution Systems*, ETSI Std. TR 101 790 V1.I., 2005.

[2] *Homeplug AV Specification*, Homeplug Alliance Std. ., 2005.

[3] *Mobile Multimedia Broadcasting (P. R. China) Part 1: Framing Structure, Channel Coding and Modulation for Broadcasting Channels*, Std.

[4] *Quality Supervision and Quarantine, GB 20600-2006, digital terrestrial television broadcasting transmission system frame structure, channel coding and modulation*, Beijing: China Standard Press Std., 2006.

[5] *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless*, IEEE Std 802.16e-2005 Std., 2006.

[6] *IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks*, IEEE Std 802.11n-2009 Std., 2009.

[7] *Multiplexing and Channel Coding*, 3GPP Std. TS36.212, 2012.

[8] A. Polydoros, "Algorithmic aspects of radio flexibility," in *IEEE International Symposium on Personal, Indoor and Mobile Communications*, 2008, pp. 1–5.

[9] M. Alles, T. Vogt, and N. Wehn, "FlexiChaP: A reconfigurable ASIP for convolutional, turbo, and LDPC code decoding," in *Turbo Codes and Related Topics, 2008 5th International Symposium on*, 2008, pp. 84 –89.

[10] F. Naessens, B. Bougard, S. Bressinck, L. Hollevoet, P. Raghavan, L. V. D. Perre, and F. Catthoor, "A unified instruction set programmable architecture for multi-standard advanced forward error correction," in *IEEE Workshop on Signal Processing Systems*, 2008, pp. 31–36.

[11] P. Murugappa, R. Al-Khayat, A. Baghdadi, and M. Jezequel, "A flexible high throughput multi-ASIP architecture for LDPC and turbo decoding," in *Design, Automation and Test in Europe Conference and Exhibition*, 2011, pp. 1–6.

[12] G. Gentile, M. Rovini, and L. Fanucci, "A multi-standard flexible turbo/LDPC decoder via ASIC design," in *International Symposium on Turbo Codes & Iterative Information Processing*, 2010, pp. 294–298.

[13] Y. Sun and J. R. Cavallaro, "A flexible LDPC/Turbo decoder architecture," *Jour. of Signal Processing Systems*, vol. 64, no. 1, pp. 1–16, 2010.

[14] C. Condo, M. Martina, and G. Masera, "A network-on-chip-based turbo/LDPC decoder architecture," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 1525 –1530.

[15] C. Neeb, M. J. Thul, and N. Wehn, "Network-on-chip-centric approach to interleaving in high throughput channel decoders," in *IEEE International Symposium on Circuits and Systems*, 2005, pp. 1766–1769.

[16] H. Moussa, A. Baghdadi, and M. Jezequel, "Binary de Bruijn interconnection network for a flexible LDPC/turbo decoder," in *IEEE International Symposium on Circuits and Systems*, 2008, pp. 97–100.

[17] F. Vacca, H. Moussa, A. Baghdadi, and G. Masera, "Flexible architectures for LDPC decoders based on network on chip paradigm," in *Euromicro Conference on Digital System Design*, 2009, pp. 582–589.

[18] L. Benini, "Application specific NoC design," in *Design, Automation and Test in Europe Conference and Exhibition*, 2006, pp. 1330–1335.

[19] M. Martina and G. Masera, "Turbo NOC: A framework for the design of network–on–chip–based turbo decoder architectures," *IEEE Trans. on Circuits and Sistems I*, vol. 57, no. 10, pp. 2776 – 2789, 2010.

[20] M. Martina, G. Masera, H. Moussa, and A. Baghdadi, "On chip interconnects for multiprocessor turbo decoding architectures," *Elsevier Microprocessors and Microsystems*, vol. 35, no. 2, pp. 167–181, Mar 2011.

[21] H. Moussa, A. Baghdadi, and M. Jezequel, "Binary De Bruijn onchip network for a flexible multiprocessor LDPC decoder," in *ACM/IEEE Design Automation Conference*, 2008, pp. 429–434.

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

[23] M. Martina, M. Nicola, and G. Masera, "A flexible UMTS-WiMax Turbo decoder architecture," *Circuits and Systems II, IEEE Transactions on*, vol. 55, no. 4, pp. 369 –373, april 2008.

[24] M. Mansour and N. Shanbhag, "Memory-efficient turbo decoder architectures for LDPC codes," in *Signal Processing Systems, IEEE Workshop on*, oct. 2002, pp. 159 – 164.

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

[26] D. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Signal Processing Systems, IEEE Workshop on*, 2004, pp. 107 – 112.

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

[28] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo codes," in *IEEE International Conference on Comm.*, 1993, pp. 1064–1070.

[29] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, no. 3, pp. 284–287, Mar 1974.

[30] M. Martina, G. Masera, S. Papaharalabos, P. Mathiopoulos, and F. Gioulekas, "On practical implementation and generalizations of $\max^*$ operator for turbo and LDPC decoders," *Instrumentation and Measurement, IEEE Transactions on*, vol. 61, no. 4, pp. 888 –895, april 2012.

[31] S. Papaharalabos, P. T. Mathiopoulos, G. Masera, and M. Martina, "On optimal and near-optimal turbo decoding using generalized $\max^*$ operator," *IEEE Comm. Letters*, vol. 13, no. 7, pp. 522–524, Jul 2009.

[32] E. Boutillon, C. Douillard, and G. Montorsi, "Iterative decoding of concatenated convolutional codes: Implementation issues," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1201–1227, Jun 2007.

[33] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Algorithm for continuous decoding of turbo codes," *IET Electronics Letters*, vol. 32, no. 4, pp. 314–315, Feb 1996.

[34] A. Abbasfar and K. Yao, "An efficient and practical architecture for high speed turbo decoders," in *IEEE Vehicular Technology Conference*, 2003, pp. 337–341.

Table VIII
LDPC/TURBO ARCHITECTURES COMPARISON: DECODER PARALLELISM P, CMOS TECHNOLOGY PROCESS (TP), PROCESSING AREA OCCUPATION (A_{core}), TOTAL AREA OCCUPATION (A_{tot}) NORMALIZED AREA OCCUPATION FOR 65NM TECHNOLOGY (An_{tot}), CLOCK FREQUENCY (f_{clk}), PEAK POWER CONSUMPTION (POW), ENERGY EFFICIENCY (E_{eff}), DATA WIDTH (DW), MAXIMUM NUMBER OF ITERATIONS ($n_{it}^{(max)}$), CODE LENGTH (N) AND RATE (r), INTERLEAVER SIZE (K) AND THROUGHPUT (T)

Decoder		[2] **A**	[2] **B**	[2] **C**	[11]	[9]	[12]	[13]
P	LDPC DBTC	22	35	22	8	1	12	12
Tp $[nm]$	LDPC DBTC	90	90	90	90	65	45	90
A_{core} [mm^2]	LDPC DBTC	2.19	3.83	2.56	2.44	N/A	N/A	1.18
A_{tot} [mm^2]	LDPC DBTC	2.75	4.87	3.42	2.6	0.62	0.9	3.20
An_{tot} [mm^2]	LDPC DBTC	1.43	2.54	1.78	1.36	0.62	1.88	1.67
f_{clk} [MHz]	LDPC DBTC	200[1] 170[1]	520[1] 200[1]	200[1] 170[1]	520	400	150	500
Pow [mW]	LDPC DBTC	87.8 101.5	331.6 183.2	118.6 121.6	N/A	76.8	86.1	N/A
E_{eff} $[\frac{nJ}{bits}]$	LDPC DBTC	0.125 0.081	0.073 0.078	0.174 0.097	N/A	0.032 0.826	0.151 0.147	N/A
DW [bits]	LDPC DBTC	6 - 5 6 - 4	6 - 5 6 - 4	6 - 5 6 - 4	7 - 5 8 - 6	7 - 5 8	7 - 5 7 - 5	9 - 6 9 - 6
$n_{it}^{(max)}$	LDPC DBTC	10 8	10 8	10 8	10 6	10 5	8 8	15 6
N, r K	LDPC DBTC	2304, 1/2 2400	1944, 5/6 2400	7493, 4/5 2400	2304, 1/2 2400	2304, 5/6 2400	N/A N/A	2304, 5/6 SBTC 6144
T [Mb/s]	LDPC DBTC	70 156	455 292	68 156	62.5 173	237.8 37.2	71.05 73.46	600 450

[1] f_{core}
[2] post–layout results

[35] Family of graph and hypergraph partitioning software. [Online]. Available: http://www.cs.umn.edu/m̃etis

[36] J. Dielissen, N. Engin, S. Sawitzki, and K. van Berkel, "Multistandard FEC decoders for wireless devices," *Circuits and Systems II, IEEE Transactions on*, vol. 55, no. 3, pp. 284 –288, march 2008.

[37] J.-H. Kim and I.-C. Park, "A 50Mbps double-binary turbo decoder for WiMAX based on bit-level extrinsic information exchange," in *Solid-State Circuits Conference, IEEE Asian*, nov. 2008, pp. 305 –308.

[38] A. Pulimeno, M. Graziano, and G. Piccinini, "UDSM trends comparison: From technology roadmap to UltraSparc Niagara2," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 7, pp. 1341 –1346, july 2012.

ASSOCIATE EDITOR AND REVIEW NUMBER 1

The authors would like to thank the associate editor and the reviewers for their fast and constructive work. Answers to the comments reported follow.

AE **When evaluating the fixed-point performance, the authors should show the performance degradation compared to a floating-point decoder. Also, the initial channel LLR quantization is not mentioned in the simulation.**

AU To better clarify the degradation introduced by the quantization, the authors have included in Figure 10 two new curves, obtained with floating point precision. They are compared to fixed point precision in Section VI.A; here, also, the channel LLR's quantization has been specified as requested.

AE **The proposed decoder's efficiency in Table VII is relatively lower than the ASIC solutions, which is understandable. When compared with ASIP solution of [9], the proposed decoder is significantly worse in LDPC mode, but much better in Turbo mode. Are those numbers in Table VII normalized to the same number of iterations?**

AU The previous numbers did not take in account the iterations, and Table VII has been modified to consider them. The Area efficiency metric has been redefined as $A_{eff} = (T \cdot n_{it}^{(max)}/f_{clk}) \cdot (1000/An_{tot})$, where T is the throughput obtained with $n_{it}^{(max)}$ iterations. This kind of

normalization means that A_{eff} of each decoder will be evaluated on its performance once it has been detached from the number of iterations, being T= $\frac{bits \cdot f}{cycles_{it} \cdot n_{it}^{(max)}}$. Still the difference underlined by the reviewer is present, and even accentuated. The particularly unsatisfying A_{eff} of [9] in turbo mode is mainly due to the low throughput obtained: 37.2 Mb/s are achieved with 5 iterations only, making for a far smaller numerator than the compared solutions, for example 156 Mb/s with 8 iterations.