A Network-on-Chip-based turbo/LDPC decoder architecture

A Network-on-Chip-based turbo/LDPC decoder architecture

[†]Carlo Condo, [†]Maurizio Martina, [†]Guido Masera

***Abstract*—The current convergence process in wireless technologies demands for strong efforts in the conceiving of highly flexible and interoperable equipments. This contribution focuses on one of the most important baseband processing units in wireless receivers, the forward error correction unit, and proposes a Network-on-Chip (NoC) based approach to the design of multi-standard decoders. High level modeling is exploited to drive the NoC optimization for a given set of both turbo and Low-Density-Parity-Check (LDPC) codes to be supported. Moreover, synthesis results prove that the proposed approach can offer a fully compliant WiMAX decoder, supporting the whole set of turbo and LDPC codes with higher throughput and an occupied area comparable or lower than previously reported flexible implementations. In particular, the mentioned design case achieves a worst-case throughput higher than 70 Mb/s at the area cost of 3.17 mm^2 on a 90 nm CMOS technology.**

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

I. Introduction

Wireless communications employ high-performance forward error correction codes as turbo [1] and Low-Density-Parity-Check (LDPC) [2] codes to achieve reliable transmission. Excellent error correction performance of turbo and LDPC codes are obtained at the expense of significant complexity at the decoder side. Even if the implementation of turbo and LDPC code decoders is a well studied problem in the literature, two critical needs emerged in the last years: i) achieving high throughput, ii) granting flexibility and interoperability. The intrinsic differences between the turbo and LDPC decoding algorithms and their iterative nature make the design of high throughput, flexible turbo/LDPC decoder architectures a challenging task.

In both turbo and LDPC decoders high throughput is routinely achieved by employing parallel architectures [3], [4], where several processing elements (PEs) perform the decoding algorithm concurrently on different portions of the received frame. However, PEs require a large bandwidth and an efficient interconnection structure to concurrently read/write data from/to the memory. High throughput PEs able to support both turbo and LDPC decoding [5]–[9] can be implemented as Application-Specific-Integrated-Circuits (ASICs) or Application-Specific-Instruction-set-Processors (ASIPs). In general, ASIC solutions achieve higher throughput with lower complexity as compared to ASIP implementations. However, ASIP architectures are usually more flexible than ASIC ones.

Stemming from the general Network-on-Chip (NoC) paradigm [10], Neeb et alii [11] proposed an interesting NoC-based approach to enable flexible and efficient interconnection among the processing elements in parallel turbo decoder architectures. According to [12] this approach, where the network structure is used to connect PEs belonging to the same Intellectual Property (IP), is referred to as intra-IP NoC. In [13] the intra-IP NoC approach is studied in the context of parallel turbo decoder architectures investigating a number of direct and indirect networks. A similar approach has been employed for LDPC decoder architectures [12], [14]. Few recent works [7], [9], [15] tried to exploit the intra-IP NoC approach to design flexible turbo/LDPC decoder architectures. However, from these works it is not clear how the design of the PEs and the design of the network influence each other. Moreover most of these implementations are not fully compliant with the high throughput requirements of modern wireless standards.

978-3-9810801-8-6/DATE12/©2012 EDAA

This paper exploits the Turbo NoC cycle-accurate simulation tool [16] described in [17], where an extensive analysis of the performance achieved by various NoC topologies in the context of turbo decoder architectures is shown. The contribution of this paper is twofold: i) to propose a competitive intra-IP-NoC-based ASIC architecture for flexible turbo/LDPC decoding with a clear design flow; ii) to prove that the flexibility achieved via the intra-IP-NoC-based approach has a limited impact on the area of the decoder architecture. As a case of study the WiMAX standard is considered. The architecture has complexity comparable to the latest state-of-the-art proposed flexible turbo/LDPC decoders, together with a higher worst-case throughput, and guarantees multi-standard compliance and low power consumption.

The paper is organized as follows: in Section II turbo and LDPC decoding algorithms are summarized, whereas Sections III and IV deal with the architectures of the NoC interconnection structure and the PEs respectively. Experimental results for the WiMAX standard are shown in Section V and conclusions are drawn in Section VI.

II. Decoding Algorithms

Algorithms used to decode turbo and LDPC codes are both iterative and based on data processing and message passing phases. The two phases can be partially overlapped employing pipeline architectures to increase the throughput. In the following paragraphs these algorithms are summarized.

A. Turbo code decoding algorithm

Convolutional turbo codes rely on the parallel concatenation of two constituent convolutional codes (CC) by the means of an interleaver. The decoding algorithm is based on the iterative exchange of extrinsic information between the two constituent decoders, usually referred to as Soft-In-Soft-Out (SISO) units. Extrinsic information is exchanged according to the order imposed by the permutation law defined by the interleaver. A SISO produces, at each step k, the extrinsic information of the corresponding uncoded symbol u, that is $\lambda_k[u] = \sigma \cdot (\lambda_k^{apo}[u] - \lambda_k^{apr}[u])$ in the Log-Likelihood-Ratio (LLR) domain where $\sigma \leq 1$ [18]. The *a-priori* information $\lambda_k^{apr}[u]$ is the extrinsic information produced during the previous half iteration, whereas the *a-posteriori* information $\lambda_k^{apo}[u]$ is obtained by the means of the BCJR algorithm [1]

$$\lambda_k^{apo}[u] = \mathop{\overset{*}{\max}}_{e:u(e)=u}\{b(e)\} - \mathop{\overset{*}{\max}}_{e:u(e)=\tilde{u}}\{b(e)\} - \lambda_k[\mathbf{c}^u] \tag{1}$$

where $\lambda_k[\mathbf{c}^u]$ is the systematic component of the *intrinsic* information, $\tilde{u} \in \mathcal{U}$ 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. The $\overset{*}{\max}\{x_i\}$ function is implemented as $\max\{x_i\}$ followed by a correction term often stored in a small Look-Up-Table (LUT) [19]. The correction term, usually adopted when decoding binary codes (Log-MAP), can be omitted for double-binary turbo codes with minor Bit-Error-Rate (BER) performance degradation (Max-Log-MAP). The term $b(e)$ in (1) is defined as:

$$b(e) = \alpha_{k-1}[s^S(e)] + \gamma_k[e] + \beta_k[s^E(e)] \tag{2}$$

$$\alpha_k[s] = \max_{e:s^E(e)=s} \left\{\alpha_{k-1}[s^S(e)] + \gamma_k[e]\right\} \tag{3}$$

$$\beta_k[s] = \max_{e:s^S(e)=s} \left\{\beta_{k+1}[s^E(e)] + \gamma_k[e]\right\} \tag{4}$$

$$\gamma_k[e] = \lambda'_k[u(e)] + \lambda_k[\mathbf{c}(e)] \tag{5}$$

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 $\lambda_k[\mathbf{c}(e)]$ is the *intrinsic* information received from the channel.

B. LDPC code decoding algorithm

LDPC codes are characterized by a very sparse $M \times N$ parity-check matrix $\mathbf{H}$ and valid codewords x satisfy $\mathbf{H} \cdot x^T = 0$. Each code can be represented as a bipartite graph, known as Tanner Graph, containing two sets of nodes: 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 [20]. 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 [20]. 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:

$$Q_{lk}[c] = \lambda_k^{old}[c] - R_{lk}^{old} \tag{6}$$

$$A_{lk} = \sum_{n \in N(l), n \neq k} \Psi(Q_{ln}[c]) \tag{7}$$

$$\delta_{lk} = \prod_{n \in N(l), n \neq k} \operatorname{sgn}(Q_{ln}[c]) \tag{8}$$

$$R_{lk}^{new} = -\delta_{lk} \cdot \Psi^{-1}(A_{lk}) \tag{9}$$

$$\lambda_k^{new}[c] = Q_{lk}[c] + R_{lk}^{new} \tag{10}$$

$\lambda_k^{old}[c]$ is the extrinsic information received from the previous layer and updated in (10) 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 (6); the same amount is then updated in (9), R_{lk}^{new}, and stored to be used again in the following iteration. In (7) and (8) $N(l)$ is the set of all bit indexes that are connected to parity constraint l.

According to [21], the $\Psi(\cdot)$ function in (7) and (9) can be simplified with a limited BER performance loss as

$$R_{lk}^{new} \approx -\delta'_{lk} \cdot \min_{n \in N(l), n \neq k} \{|Q_{nk}|\}\,, \tag{11}$$

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

In a parallel decoder, the decoding algorithms summarized in previous paragraphs are partitioned among P PEs. When configured in turbo code mode, these PEs operate as concurrent SISOs, while they execute (6) to (10) in parallel for P parity check constraints when configured in LDPC code mode. In both cases, messages are exchanged among PEs to propagate $\lambda_k^{apo}[u]$ 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. Design of the NoC architecture

Turbo and LDPC decoding have in common a complex message passing phase, which varies in terms of duration and intertwining with the parallelism of the decoder. In this section we study the characteristics of NoC architectures requested to support both turbo and LDPC decoding. To this purpose, we start from results presented in [17] for NoC–based decoding of turbo codes alone and extend them towards inclusion of LDPC decoding. We also assume the same node architecture detailed in [17] and shown in Fig. 1: each node in the network is made of a Routing Element (RE), a Processing Element (PE) and a memory (MEM) to store the incoming messages. The RE is based on an $F \times F$ crossbar switch with F input FIFOs and F output registers. $t'_{i,j}$ represents the memory location where $\lambda'_{i,j}$ will be stored. In this work we concentrate on the two most promising node architectures proposed in [17]: the All-Precalculated (AP) and the Partially-Precalculated (PP) architectures. The AP architecture makes use of off-line simulations to compute the routing information of each node and to store it in a routing memory. Since the routing

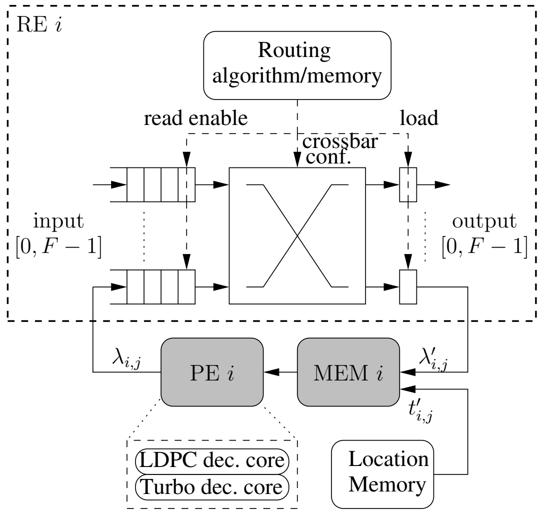

Figure 1. Node structure

information is precalculated, very complex routing algorithms can be employed to compute the routing information and this allows to reduce the depth of input FIFOs. Moreover, this solution does not require any kind of header in the packet structure, reducing the width of the input FIFOs. However, as pointed out in [13], the AP architecture requires additional memories to store the routing information of all supported codes. In PP architecture the routing is performed on-line by a routing algorithm: only $t'_{i,j}$ sequences are precalculated, while destination node identifiers are included in the packet header. Details on PE architectures will be given in Section IV.

A. NoC analysis and simulation tool

The SystemC simulator developed in [17] is here used to extensively analyze the performance of NoC-based LDPC decoder architectures, in terms of throughput and memory requirements. A set of parameters is defined to take into account a large number of possible design choices including routing algorithms, node architectures and packet structures. The simulator requires the description of the NoC topology, i.e. the number of nodes and their links, then, it derives the communication pattern among the nodes of the network.

In order to evaluate the performance of an NoC-based LDPC decoder, a pre-processing tool has been developed to produce equivalent interleavers. Indeed $\mathbf{H}$, the parity check matrix of the LDPC code, can be transformed in a turbo-like interleaver once the decoding scheduling and the topology are chosen. The following flow has been employed to analyze the performance of NoC-based LDPC decoder architectures.

- The first step is the definition of the graph representation of the $\mathbf{H}$ matrix. Size and structure of this graph depends on the chosen scheduling. With the layered decoding approach, the resulting graph has M nodes, and an arc between row–nodes i and j is defined when a non-zero entry is present on the same column of both i and j.
- The second step is the choice of the NoC topology and its degree of parallelism. To this purpose, a set $\mathcal{T}$ of various topologies, including mesh, toroidal mesh, spidergon, honeycomb, generalized De-Bruijn and generalized Kautz has been considered.
- The problem of mapping the LDPC codes on a specific NoC is then formulated in terms of graph partitioning and solved using the Metis bundle of graph–coloring algorithms [22]. Once graph nodes are assigned to NoC-nodes, the equivalent interleaver is constructed. The framework built around the Metis package checks the produced interleavers for minimum length and uniform message distribution, selecting the optimal one for each code–topology couple.

As a result of these analysis steps, LDPC check nodes are partitioned among the nodes of each NoC: then, simulation is used to evaluate the number of cycles required to perform a decoding iteration with each NoC in $\mathcal{T}$. Simulations are repeated for several values of the following parameters:

- *Processing element output rate* (R): is the number of messages produced by a PE in a clock cycle.
- *Routing algorithm*: three different routing policies are embedded in [16]. They rely on the off-line computation of the shortest paths between nodes. This information is stored in one or more routing tables. When only one shortest path is used (one routing table) the routing algorithm is referred to as Single-Shortest-Path (SSP), whereas when more shortest paths are computed (multiple routing tables) the algorithm will be named All-local-Shortest-Paths (ASP). The first approach described in [17] is the SSP-Round-Robin (SSP-RR): it is based on a circling serving policy. Similarly the SSP-FIFO-Length (SSP-FL) routing algorithm is based on the current status of input FIFOs. The third approach, named ASP–FIFO-length-with-Traffic-spreading (ASP-FT), takes in account all the possible different shortest paths. The serving policy is a modified version of FL: it keeps a statistic of sent messages to spread the traffic on the network [17].
- *Delay/Send Colliding Message* (DCM/SCM): this parameter activates a collision management technique. A collision arises when two or more messages require to be routed to the same output port. In this case, if the DCM strategy is employed, the first message is routed according to the selected routing algorithm, whereas the colliding messages are kept in their FIFOs. On the contrary, if SCM is used, colliding messages will be randomly routed to one of the available output ports. Namely, the configuration of the crossbar switch is chosen to route non-colliding messages, whereas colliding messages are treated as “don’t-care”.
- *Route Local* (RL): this flag allows to choose if local messages, i.e. messages sent and received by the same PE, are routed on the network ($RL = 1$) or are stored in an internal queue, bypassing the routing ($RL = 0$).

B. Analysis of NoCs for LDPC codes

In order to show the potential of the NoC approach in the design of LDPC code decoders, the whole set of WiMAX

Table I

THROUGHPUT [MB/S]/AREA[MM^2] FOR WIMAX LDPC $N = 2304$, $r = 0.5$ CODE, FOR DIFFERENT TOPOLOGIES, PARALLELISM P, NODE DEGREE D, ROUTING ALGORITHMS AND NODE ARCHITECTURES. FREQUENCY IS 300 MHZ, TECHNOLOGY IS CMOS 90 NM. RESULTS ARE OBTAINED FOR PARAMETERS $RL = 0$, SCM, $R = 0.5$

	$D = 2$, generalized De Bruijn				$D = 2$, generalized Kautz			
	$P = 16$	$P = 24$	$P = 32$	$P = 36$	$P = 16$	$P = 24$	$P = 32$	$P = 36$
SSP-RR (PP)	37.77/2.02	41.19/3.16	50.16/3.68	50.31/4.02	38.10/2.05	49.23/2.79	48.20/3.67	55.47/3.84
SSP-FL (PP)	42.15/1.82	45.47/3.27	55.12/0.65	56.20/4.18	41.69/1.84	53.09/2.68	55.74/3.61	61.71/0.68
ASP-FT (AP)	42.15/0.40	45.47/0.59	55.12/0.65	56.84/0.71	41.69/0.40	53.09/0.51	55.74/0.64	61.71/0.68

	$D = 3$, spidergon				$D = 3$, generalized Kautz			
	$P = 16$	$P = 24$	$P = 32$	$P = 36$	$P = 16$	$P = 24$	$P = 32$	$P = 36$
SSP-RR (PP)	55.74/0.35	67.11/1.34	70.67/2.69	71.11/3.14	55.74/0.29	78.37/0.47	93.66/0.96	92.65/1.22
SSP-FL (PP)	55.47/0.30	69.82/1.11	75.62/2.59	75.79/3.20	55.74/0.28	77.49/0.43	97.63/0.69	101.05/0.86
ASP-FT (AP)	55.31/0.30	72.45/0.42	76.63/0.64	78.37/0.73	55.74/0.29	77.49/0.35	97.08/0.42	101.05/0.46

	$D = 4$, rectangular honeycomb				$D = 4$, generalized Kautz			
	$P = 16$	$P = 24$	$P = 32$	$P = 36$	$P = 16$	$P = 24$	$P = 32$	$P = 36$
SSP-RR (PP)	55.12/0.42	77.49/0.61	98.46/0.72	97.90/1.03	55.74/0.31	72.45/0.60	70.10/1.06	104.73/0.76
SSP-FL (PP)	55.47/0.39	78.01/0.53	98.18/0.63	106.67/0.87	55.74/0.29	77.84/0.49	72.00/0.98	109.37/0.72
ASP-FT (AP)	55.65/0.40	78.01/0.48	99.03/0.55	109.37/0.58	55.74/0.39	78.01/0.47	100.47/0.54	108.68/0.58

codes has been used as a design case. In Table I the most relevant results for the WiMAX LDPC code with $N = 2304$ and rate $r = 0.5$ are shown. This code is the most demanding one within WiMAX specification in terms of PE resources. Given that $D = F - 1$ is the degree of the topology, for each topology in $\mathcal{T}$ all routing algorithms have been tested for $D = 2, 3, 4$ and $P = 16, 24, 32, 36$. The throughput has been computed as

$$T = \frac{(N - M) \cdot f_{clk}}{(lat_{core} + n_{cycles}) \cdot It_{max}} \tag{12}$$

where f_{clk} is the clock frequency, It_{max} is the maximum number of iterations, lat_{core} is the maximum latency of the decoding core and n_{cycles} is the duration of the message passing phase. Results in Table I have been obtained with (12), imposing $f_{clk} = 300$ MHz, $It_{max} = 10$ and $lat_{core} = 15$ cycles. The value of n_{cycles} is measured by the means of the simulator [26].

The area occupation is the post synthesis result obtained with Synopsys Design Compiler on a 90 nm CMOS technology. These area results do not take in account the PE and the incoming message memories.

Generalized Kautz topologies outperform all the other ones in terms of both throughput and complexity in the case of LDPC codes. Moreover, $D = 3$ solutions give higher throughputs than $D = 2$ ones, whereas they are comparable to $D = 4$ topologies but with lower area occupation.

C. Analysis of NoCs for turbo and LDPC codes

To find the most suited NoC for both turbo and LDPC decoding according to the WiMAX standard we analyzed the results shown in [17] for turbo codes and the ones presented in Table I for LDPC codes. Generalized Kautz topologies show the best average throughput-to-area ratio both for turbo and LDPC codes and $D = 3$ is a good throughput/complexity trade-off. For LDPC codes the minimum value of P to achieve the 70 Mbit/s throughput required by the IEEE 802.16e standard, with a 300 MHz clock frequency, is 22. On the contrary, turbo codes yield a higher than required throughput with a 22-nodes NoC. Thus, the working frequency for turbo codes can be lowered to 75 MHz and throughput is still above limit. For both codes throughput and area show a weak dependence on the routing algorithm. However, the SSP-FL routing algorithm guarantees the best average performances also with different topologies and non-WiMAX codes, thus being the best choice in terms of flexibility. A choice of the obtained results are given in Table II for $N = 2400$ turbo code and $N = 2304$, $r = 0.5$ LDPC code.

Table II

THROUGHPUT [MBITS/S]/AREA[MM^2] FOR NOC BASED ARCHITECTURES SUPPORTING ALL WIMAX TURBO AND LDPC CODES

	$P = 22$, $D = 3$, generalized Kautz, $R = 0.5$	
	turbo @ 75 MHz $N = 2400$, $r = 0.5$	LDPC @ 300 MHz $N = 2304$, $r = 0.5$
SSP-RR (PP)	74.25/0.63	72.45/0.46
SSP-FL (PP)	74.26/0.60	72.30/0.39
ASP-FT (AP)	73.29/0.69	72.91/0.34

IV. DESIGN OF PROCESSING ELEMENT ARCHITECTURES

Each PE includes two distinct decoding cores.

A. LDPC decoding core

For LDPC decoding, the PE must be structured so that all the block lengths and code rates imposed by the standard are supported. A simple and effective architecture based on a sequential processing has been designed. Fig. 2 shows the architecture of the LDPC decoding core: $\lambda_k^{old}[c]$ values are read from the $\lambda_k[c]$ memory, used to store the incoming messages received from the network. Similarly, R_{lk}^{old}, required to compute $Q_{lk}[c]$, is stored in a dedicated memory. Then, $Q_{lk}[c]$ values are compared sequentially in the Minimum Extraction Unit (MEU) to find the first two minimum values (11) [21]. A further comparison selects which of the two

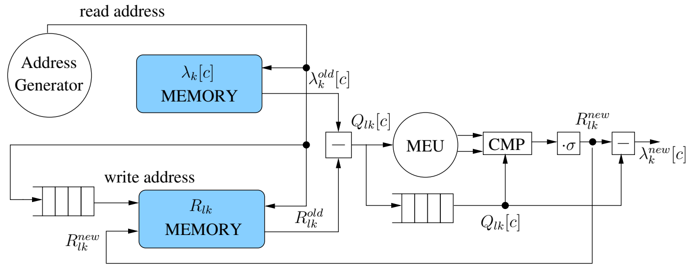

Figure 2. LDPC decoding core

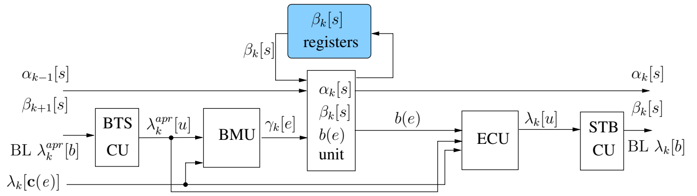

Figure 3. Turbo decoding core (SISO)

minimum values is required to update $\lambda_k^{new}[c]$ as in (10). Concurrently, R_{lk}^{new} values are stored in the R_{lk} memory for use during the next iteration. This architecture is completely independent of the code, but it is limited by the size of memories. Both $\lambda_k[c]$ and R_{lk} memories must have enough storage capability to cope with the heaviest possible workload among the supported codes. Simulations show that the worst case among WiMAX codes is given by the 1152 parity checks of degree 6/7 in the $N = 2304$, $r = 0.5$ code. Given this sizing, not only all the other WiMAX codes, but any QC LDPC code with no superposition of vertexes between nodes and code with smaller size can be decoded.

B. Turbo decoding core

The proposed solution for the turbo decoding core is the SISO architecture in Fig. 3. Since the turbo code used in the WiMAX standard is double-binary each message $\lambda_{i,j}$ is a vector of three elements. According to [23], sending bit-level (BL) instead of symbol-level extrinsic information reduces the NoC complexity of roughly 1/3. Resorting to the solution proposed in [24] this complexity reduction comes at the expense of a 0.2 dB BER loss. A dedicated unit, the Bit-To-Symbol Conversion Unit (BTS CU) converts the incoming *a priori* values from bit (BL $\lambda_k^{apr}[b]$) to symbol ($\lambda_k^{apr}[u]$) information, whereas, the Symbol–To–Bit Conversion Unit (STB CU), converts the processed *extrinsic* values ($\lambda_k[u]$) before sending them on the network (BL $\lambda_k[b]$). The BMU, i.e. Branch Metric Unit, is entitled the task of computing the $\gamma_k[e]$ (5). Another unit handles, sequentially, $\beta_k[s]$, $\alpha_k[s]$ and $b(e)$ as in (1). $\beta_k[s]$ values are stored in a set of registers for use during the calculation of $b(e)$. The Extrinsic Computation Unit (ECU) produces the updated LLR $\lambda_k[u]$. As in [7], the number of bits to represent $\lambda_k[c]$, $\alpha_k[s]$, $\beta_k[s]$ and $\lambda_k[u]$ is set to 7, whereas 5 bits are sufficient for R_{lk} and $\lambda_k[\mathbf{c}(e)]$.

The SISO and the LDPC core share their internal memories. The number of 7-bit memory locations is determined by the $N = 2304$, $r = 0.5$ LDPC code, that needs to store all the $\lambda_k^{old}[c]$ values (1152 × 7). On the same $\lambda_k[c]$ memory are also mapped the $22 \times 3 \times 16$ blocks for $\alpha_k[s]$ and $\beta_k[s]$, $8 + 8$ for each of the 3 windows assigned to every one of the 22 SISOs. The size of the 5-bit R_{lk} memory is instead fixed by the 2400×4 blocks needed by the SISO for $\lambda_k[\mathbf{c}(e)]$, while the LDPC core only needs 1152×7 for R_{lk}.

V. Results

As a case of study, a complete turbo/LDPC decoder for the WiMAX standard has been implemented. Table III shows the pre–layout synthesis results (2^{nd} row), obtained with Synopsys Design Compiler on a 90 nm deep sub-micron CMOS technology [25], together with recent state–of–the–art dual code decoders. Where possible, worst–case throughput and the relative code are reported. For the sake of fairness it is worth noting that [5] and [9] support both WiMAX and LTE modes. In this work the SISO/LDPC core shared memories account for 61.8% of the processing core area: SISO-exclusive logic contributes for 18.6%, while LDPC core-exclusive logic occupies the remaining 19.6%. Comparison with [9] shows similar core area occupation, whereas our NoC contributes for 0.61 mm^2, about the 20% of the total area occupation. Its larger complexity is mainly due to the more distributed topology and complex node architecture. Both LDPC and turbo cases in this work show compliance with the WiMAX standard throughput requirements. LDPC codes are considered with $R = 0.5$, and a clock frequency of 300 MHz for both the NoC and the LDPC core. The worst case values, obtained for the $N = 2304$, $r = 0.5$ code, are still above 70 Mb/s: in [9], according to the provided formula, for the same code throughput is below the standard threshold. The best working conditions for turbo decoding are with $R = 0.33$. Since the designed SISO architecture produces two $\lambda_k[u]$ every three clock cycles, it must run at half the clock frequency of the NoC: $f_{clk}^{SISO} = 0.5 f_{clk}^{NoC}$. Moreover sufficient throughput is obtained with $f_{clk}^{NoC} = 75$ MHz, allowing the SISO to run at 37.5 MHz. If f_{clk}^{NoC} is rescaled to 200 MHz and f_{clk}^{SISO} to 100 MHz, our worst–case throughput overperforms the best–case value of [9] (198 vs. 173 Mb/s), despite the higher number of iterations and the much lower f_{clk}.

This characteristic, together with the lower memory accesses rate of turbo decoding, results in a large power reduction w.r.t. LDPC decoding. It is worth noting that, in the turbo decoding mode the proposed architecture achieves the lowest power consumption as compared with [5]–[9].

The architecture in [5] not only supports both WiMAX and LTE modes but it also features a very small area occupation. However, it does not reach a high enough throughput for the WiMAX standard, while our decoder has both smaller area and higher throughput than [7]. Area occupation is smaller than [6], but throughput analysis is difficult, since standard compliance is stated but no minimum values are reported. The architecture for WiMAX/WiFi LDPC codes and 3GPP-LTE turbo code presented in [8] runs at 500 MHz and achieves the highest throughput among compared architectures with the same complexity as our architecture. A fair comparison

Table III
LDPC/Turbo architectures comparison: CMOS technology process (TP), processing area occupation (A$_{core}$), total area occupation (A$_{tot}$) normalized area occupation for 65nm technology (An), clock frequency (f_{clk})(^{1} is f_{clk}^{NoC}, ^{2} is f_{clk}^{SISO}), peak power consumption (Pow), data width (DW), maximum number of iterations (It_{max}), code length (N) and rate (r) and throughput (T)

Decoder	P	Tp	A$_{core}$	A$_{tot}$	An$_{tot}$	f_{clk}	Pow	DW	It_{max}	Code	N, r	T
		[nm]	[mm^2]		[mm^2]	[MHz]	[mW]	[bits]				[Mb/s]
This Work	22	90	2.56	3.17	1.65	300	415	7 – 5	10	LDPC	2304, 0.5	72.00 (min.)
						75^1 – 37.5^2	59	7 – 5	8	DBTC	4800, 0.5	74.26 (min.)
[9]	8	90	2.44	2.6	1.36	520	N/A	7 – 5	10	LDPC	2304, 0.5	62.5 (min.)
								8 – 6	6	DBTC	N/A	173 (max.)
[5]	1	65	N/A	0.62	0.62	400	76.8	7 – 5	20	LDPC	N/A	27.7 (min.)
								8	5	DBTC	N/A	18.6 (min.)
[7]	12	45	N/A	0.9	1.88	150	86.1	7 – 5	8	LDPC	N/A	71.05 (min.)
								7 – 5	8	DBTC	N/A	73.46 (min.)
[6]	384	45	N/A	0.94	1.96	333	1000	N/A	25	LDPC	N/A	333 (avg.)
[8]	12	90	1.18	3.20	1.67	500	N/A	9 – 6	15	LDPC	2304, 0.5	600 (max.)
								9 – 6	6	BTC	6144, 0.3	450 (max.)

is not possible as WiMAX turbo code is not addressed. The proposed decoder guarantees compliance with WiMAX, but is not limited to its codes: the SISO can work with any 8 state Double-Binary-Turbo-Code (DBTC), whereas the LDPC core can sustain any code smaller than 802.16e ones (e.g. WiFi).

VI. Conclusions

The design of a fully flexible NoC based turbo/LDPC decoder is presented, together with custom simulation software models for a thorough analysis of the NoC architecture. The proposed decoder implementation offers an unmatched degree of flexibility and full compliance with the WiMAX standard, guaranteeing the highest worst–case throughput and small area occupation compared to the latest state–of–the–art solutions, together with particularly low power consumption in turbo mode.

References

[1] 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.

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

[3] O. Muller, A. Baghdadi, and M. Jezequel, "Exploring parallel processing levels for convolutional turbo decoding," in *IEEE International Conference on Information and Communication Technologies: from Theory to Applications*, 2006, pp. 2353–2358.

[4] T. Brack, F. Kienle, and N. Wehn, "Disclosing the LDPC code decoder design space," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, 2006, pp. 1–6.

[5] 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.

[6] F. Naessens, B. Bougard, S. Bressinck, L. Hollevoet, P. Raghavan, L. V. der 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.

[7] 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.

[8] 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.

[9] 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.

[10] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Design, Automation and Test in Europe Conference and Exhibition*, 2000, pp. 250–256.

[11] 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.

[12] F. Vacca, G. Masera, H. Moussa, A. Baghdadi, and M. Jezequel, "Flexible architectures for LDPC decoders based on network on chip paradigm," in *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, 2009, pp. 582 –589.

[13] 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.

[14] 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.

[15] ——, "Binary de Bruijn interconnection network for a flexible LDPC/turbo decoder," in *IEEE International Symposium on Circuits and Systems*, 2008, pp. 97–100.

[16] M. Martina, "Turbo NOC: Network On Chip based turbo decoder architectures," downloadable at http://personal.delen.polito.it/maurizio.martina/turbo.html.

[17] 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.

[18] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *IEE Electronics Letters*, vol. 36, no. 23, pp. 1937–1939, Nov 2000.

[19] 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.

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

[21] 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.

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

[23] M. Martina and G. Masera, "Improving network-on-chip-based turbo decoder architectures," submitted to Jour. of Parallel and Distributed Computing, available at: http://arxiv.org/abs/1105.1014.

[24] J. H. Kim and I. C. Park, "Bit-level extrinsic information exchange method for double-binary turbo codes," *IEEE Trans. on Circuits and Systems II*, vol. 56, no. 1, pp. 81–85, Jan 2009.

[25] A. Pulimeno, M. Graziano, and G. Piccinini, "UDSM trends comparison: From technology roadmap to UltraSparc Niagara2," *IEEE Trans. on VLSI*, 10.1109/TVLSI.2011.2148183, to appear.