# A flexible NoC-based LDPC code decoder implementation and bandwidth reduction methods

# A Flexible NoC–based LDPC code decoder implementation and bandwidth reduction methods

Carlo Condo, Guido Masera, *Senior Member IEEE*

***Abstract*—The need for efficient and flexible LDPC (Low Density parity Check) code decoders is rising due to the growing number and variety of standards that adopt this kind of error correcting codes in wireless applications. From the implementation point of view, the decoding of LDPC codes implies intensive computation and communication among hardware components. These processing capabilities are usually obtained by allocating a sufficient number of processing elements (PEs) and proper interconnect structures. In this paper, Network on Chip (NoC) concepts are applied to the design of a fully flexible decoder, capable to support any LDPC code with no constraints on code structure. It is shown that NoC based decoders also achieve relevant throughput values, comparable to those obtained by several specialized decoders. Moreover, the paper explores the area and power overhead introduced by the NoC approach. In particular, two methods are proposed to reduce the traffic injected in the network during the decoding process, namely early stopping of iterations and message stopping. These methods are usually adopted to increase throughput. On the contrary, in this paper, we leverage iteration and message stopping to cut the area and power overhead of NoC based decoders. It is shown that, by reducing the traffic injected in the NoC and the number of iterations performed by the decoding algorithm, the decoder can be scaled to lower degrees of parallelism with small losses in terms of BER (Bit Error Rate) performance. VLSI synthesis results on a 130 nm technology show up to 50% area and energy reduction while maintaining an almost constant throughput.**

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

## I. Introduction

LDPC codes were first studied by Gallager in [1], and later rediscovered by MacKay and Neal [2]: the outstanding performance offered by these codes led to intensive research in both theory and implementations. LDPC codes are currently included in several standards such as IEEE 802.11n [3] and IEEE 802.16e [4]: the need for flexible decoders able to support multiple codes is rising, and so is the attention of the research community. Therefore flexible decoders capable of working for multiple codes are receiving a significant attention.

Flexibility issues must be tackled on different fronts: parametrized processing elements (PE) and specialized programmable processors are valid solutions at the processing level [5] [6]. On the other side, flexibility must be provided also at the inter–PE interconnect level. Communication structures optimized for single codes or classes of codes like [7] and [8] achieve great efficiency by statically mapping the communication needs on low–cost structures. This is the approach commonly used with quasi-cyclic LDPC codes [9], where the peculiar structure of the parity check matrix ($\mathbf{H}$) permits the usage of very simple interconnect devices like barrel–shifters.

Though efficient, this approach limits greatly the achievable level of flexibility: fully flexible decoders must be able to work with $\mathbf{H}$ matrices very different from one another. The intrinsically flexible Network on Chip paradigm has been proposed as a possible structure to interconnect both heterogeneous processors (*Inter–IP* NoCs [10] [11]) and homogeneous hardware components concurrently executing a single task (*Intra–IP* NoCs, [12]). NoC–based LDPC decoders [13] are composed of a set of ($P$) PEs connected by means of an NoC, which can accommodate the specific communication needs of any LDPC code.

Stemming from a previously proposed flexible and scalable NoC based decoder [14], in this work improved architectures are described with the purpose of showing the feasibility of the NoC approach. A first decoder implementation is introduced based on a $5 \times 5$ two dimensional mesh NoC: synthesis results for this decoder proves that, notwithstanding its large flexibility, it is capable to reach high throughput; in particular it is shown that the designed decoder is compliant with the WiMAX standard throughput requirements.
However the flexibility offered by the NoC approach comes at a power and area cost. Therefore modified versions of the original decoder are proposed to limit both area and power overhead. These new decoding architectures incorporate two algorithms able to reduce the amount of messages that PEs exchange across the NoC in the decoding of LDPC codes. The first algorithm is an already known method to implement early stopping of decoding iterations [15]: by limiting to the minimum the number of iterations sequentially performed by the decoder on a data frame, the global number of messages across the NoC is decreased with respect to the case of a decoder that always runs the same number of iterations. Various methods for early stopping of iterations have been proposed to save power during the decoding [16] [17] [18] [19] [20]. In this paper, iteration stopping is adopted to also reduce the number of PEs and thus the occupied area.

The second key modification applied to the NoC based decoder is the introduction of a method to dynamically stop the delivery of single messages across the NoC, when they are not strictly required. In particular, in the decoding of LDPC codes, generated inter–PE messages carry a twofold information: the sign of the message is a binary information on the value of a bit in the codeword, while the modulus is associated to the level of reliability of the binary information. Reliabilities of the codeword bits tend to grow from one iteration to the other, but this growth occurs at different rates for different bits. Therefore the proposed method compares each message to be delivered with a threshold and if the threshold is passed the

message is considered as reliable enough and it is not sent. The reduction in terms of traffic injected by the PEs into the NoC is exploited to decrease both power and occupied area, leading to improved NoC based decoders. Provided synthesis results show that the joint application of the two mentioned methods achieves relevant advantages with respect to the original NoC based decoder: occupied area can be reduced up to 43%, while 40% to 57% of dissipated energy is saved.

The paper structure is organized with Section II summarizing the adopted decoding algorithm, while Section III describes the NoC approach to LDPC code decoding. Section IV describes briefly the architecture of the single processing element, Sections V and VI detail two different methods for reducing the NoC traffic, with implementation issues and advantages. Section VII shows results in terms of achievable throughput, occupied area and power saving; comparison with state of the art solutions are also given. Conclusions are drawn in Section VIII.

## II. LDPC DECODING

An LDPC code is a linear block code characterized by a sparse parity check matrix $\mathbf{H}$. Columns (index $j$) of $\mathbf{H}$ are associated to received bits, while rows (index $m$) correspond to parity check constraints (PCC). In the layered decoding method [21], which provides approximately a $\times 2$ factor in convergence speed w.r.t the two phase decoding, PCCs are clustered in layers ([22]) and extrinsic probability values are updated from one layer to the other.

According to the notation adopted in [22]. $L(c)$ represents the logarithmic likelihood ratio (LLR) of symbol $c$ ($L(c) = log(P\{c=0\}/P\{c=1\})$). For each $\mathbf{H}$ column $j$, bit LLR $L(q_j)$ is initialized to the corresponding channel–estimated soft value. Then, for each PCC $m$ in a given layer, the following operations are executed:

$$L(q_{mj}) = L(q_j^{(\text{old})}) - R_{mj}^{(\text{old})} \tag{1}$$

$$A_{mj} = \sum_{n \in N(m), n \neq j} \Psi(L(q_{mn})) \tag{2}$$

$$s_{mj} = \prod_{n \in N(m), n \neq j} Sign(L(q_{mn})) \tag{3}$$

$$R_{mj}^{(\text{new})} = -s_{mj}\Psi(A_{mj}) \tag{4}$$

$$L(q_j^{(\text{new})}) = L(q_{mj}) + R_{mj}^{(\text{new})} \tag{5}$$

$L(q_j^{(\text{old})})$ is the extrinsic information received from the previous layer: it is updated in (5) and eventually passed to the following layer. $R_{mj}^{(\text{old})}$ is used in equation (1); the same amount is then updated in (4), $R_{mj}^{(\text{new})}$, used to compute (5) and stored to be used again in the following iteration. $N(m)$ is the set of codeword bits connected to parity constraint $m$. Finally, $\Psi(\cdot)$ is a non–linear non–limited function, often substituted by the normalized min–sum approximation [23]:

$$A_{mj}^1 = min_{n \in N(m)}(|L(q_{mn})|) \tag{6}$$

$$A_{mj}^2 = min_{n \in N(m), n \neq t}(|L(q_{mn})|) \tag{7}$$

where $t$ is the index related to first minimum $A_{mj}^1$, while $A_{mj}^2$ is the second minimum. Equation (4) is also changed to

$$R_{mj}^{(\text{new})} = \begin{cases} -s_{mj} \cdot A_{mj}^1/\alpha & \text{when } |L(q_{mj})| \neq A_{mj}^1 \\ -s_{mj} \cdot A_{mj}^2/\alpha & \text{otherwise} \end{cases} \tag{8}$$

where the normalization factor $\alpha$ is used to limit the approximation performance degradation due to the min–sum non optimality [23].

## III. NOC BASED DECODING

This work focuses on complete flexibility of the decoder and therefore no assumption is made on the structure of supported LDPC codes. To achieve such a large flexibility, the possible use of NoC based interconnect architectures has already been suggested and partially explored in [24] and [13]: however, a complete evaluation of the potential of the NoC based approach in terms of achievable performance and implementation complexity is not available.

Figure 1 shows the adopted NoC topology, a two–dimensional toroidal mesh. Each node includes a PE and a routing element (RE) with five inputs/outputs. The simple input queuing architecture is implemented with first-in first-out memory queues (FIFO). Every FIFO is connected to the output registers by means of a crossbar switch. Since the number of PCCs in a code is much higher than the number of PEs, several PCCs will be scheduled on each PE. A control unit ($\text{CU}_{RE}$) generates commands for the RE components, implementing a given routing strategy. In particular, control bits are necessary at the cross–bar to implement a given switching of incoming data, read signals must be applied to the FIFOs, and write signals are required by output registers. Thus, in general, the $\text{CU}_{RE}$ must receive destination addresses for received packets and implement a certain routing algorithm.

Alternatively, the $\text{CU}_{RE}$ can be reduced to a simple routing memory (RM), which statically apply pre–calculated control signals to the RE. Given a certain code, the inter–processor communication needs are known a priori, depending on the structure of the $\mathbf{H}$ matrix. To reduce as much as possible the implementation overhead due to routing information in the packet header and routing algorithm, the so–called Zero Overhead NoC (ZONoC) [13] concept can be exploited. For each supported code the best path followed by all messages during a decoding iteration is statically derived: a routing memory in each node stores the necessary controls, avoiding the routing algorithm implementation and reducing the packet size and depth of FIFOs.

Static information is derived via simulation on a cycle accurate Python simulation tool. This model receives a description of the NoC and of the parity constraints mapped on each PE, and by simulating the behavior of the NoC as PEs inject messages, produces the routing decisions across a complete iteration, together with the maximum size of input FIFOs. These informations can be easily coded into binary control signals and stored in the routing memory.

## IV. ARCHITECTURE OF THE PROCESSING ELEMENT

In this section, the general structure of the PE is summarized. Figure 3 shows a simplified block scheme. The PE

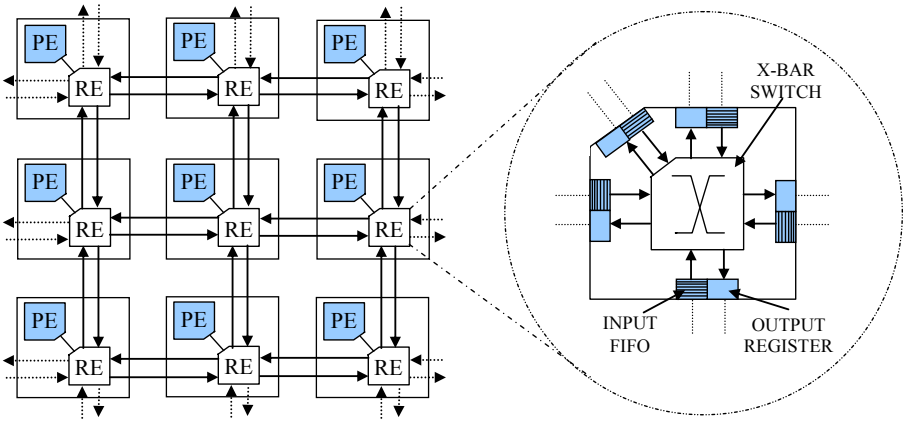


Figure 1. NoC torus mesh topology with detailed routing element

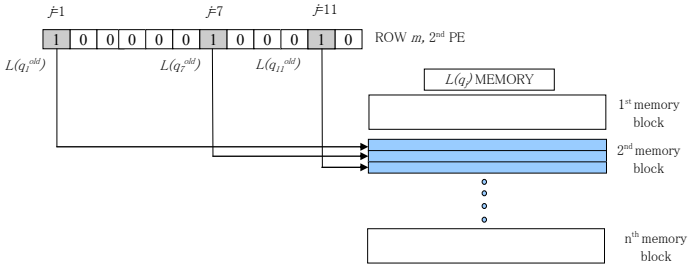


Figure 2. Example of memory organization for extrinsic values

executes equations (1) to (8) in a pipelined way, in order to achieve high throughput. Finite precision representation of data and number of decoding iterations are decided by means of extensive simulations of the considered LDPC codes.

The data flow begins when previous layer's extrinsic values $L(q_j^{(old)})$ are received by the PE and stored in $L(q_j)$ MEMORY. This is a two–port memory with $N_{pc} \times N_d$ locations, where $N_{pc}$ is the maximum number of PCCs mapped onto the PE and $N_d$ is the maximum degree of PCCs.

Similarly to RM, also the Write Address Generator memory (WAG) is initialized with data obtained via off–line simulation. Its purpose is to generate writing addresses for incoming messages: since at every PE the sequence of arrival of $L(q_j^{(old)})$ is the same through every iteration, it can be derived statically. WAG MEMORY contains pointers to $L(q_j)$ MEMORY, where extrinsic values are stored. Fig. 2 shows an example of $L(q_j)$ MEMORY organization with $N_d = 3$. The memory is divided into $N_{pc}$ blocks, each one containing 3 consecutive locations. In the example, the $2^{nd}$ scheduled PCC receives three $L(q_j^{(old)})$ values from previous layer: these extrinsic values are sequentially stored in the $2^{nd}$ block, starting from offset 3.

The CNT/CMP unit is a counter used to compute the read addresses for $L(q_j)$ MEMORY. It counts $N_d$ successive locations from an initial offset that points at the first $L(q_j)$ of a parity check. Upon recognizing the last read operation, a new offset and $N_d$ values are loaded. $R_{mj}$ MEMORY contains $R_{mj}$ amounts and it is sized exactly as $L(q_j)$ MEMORY.

The subtraction of $L(q_j^{(old)})$ and $R_{mj}^{(old)}$ is used to compute the $L(q_{mj})$ values, and the first and second minimum are derived by the MINIMUM EXTRACTION unit. A cumulative XOR keeps track of the overall sign $L(q_{mj})$.

The output of the COMPARE unit, that implements (8), is multiplied by $1/\alpha$, according to the NMS algorithm to obtain $R_{mj}^{(new)}$. At the end of the flow, $L(q_{mj})$ is retrieved by means of a short FIFO and added to $R_{mj}^{(new)}$, obtaining the new $L(q_{mj}^{(new)})$ (5), which is sent to the NoC via an output buffer.

The implementation results of a flexible decoder based on the NoC approach are reported in Section VII: a $5 \times 5$ NoC and the described PE architecture are used to design an highly versatile decoder, which reaches a worst–case throughput of more than 80 Mbps on WiMAX LDPC codes. It also supports any structured or unstructured LDPC code, up to the block size of the largest WiMAX code, including codes adopted in WiFi. This result actually proves that NoC based decoders are a feasible solution for multi-standard applications.

## V. Message stopping

Two improvements are proposed in this paper to reduce the amount of messages that have to be exchanged among PEs: (i) message stopping, that results in a reduction of the traffic injected into the NoC, and (ii) early stopping of iterations, that cuts the decoding time. This advantages can be exploited to reduce dissipated power and occupied area.

A C++/Python fixed point simulation model has been developed for the whole transmission chain, consisting of encoder, AWGN (additive white gaussian noise) channel and decoder. The model allows to statistically study the extrinsic values exchanged among PEs in the layered decoding of a set of LDPC codes. In particular, Figure 4 shows how extrinsic values change from one iteration to another: as expected, extrinsic values tend to increase with iterations and the number of messages that carry high reliability values for the corresponding bits also increases along the decoding process. Moreover, divergence from 0 of extrinsics occurs earlier for higher signal to noise ratios (SNR), while extrinsics tend to float around the zero at low SNRs, expressing uncertainty about the bit value.

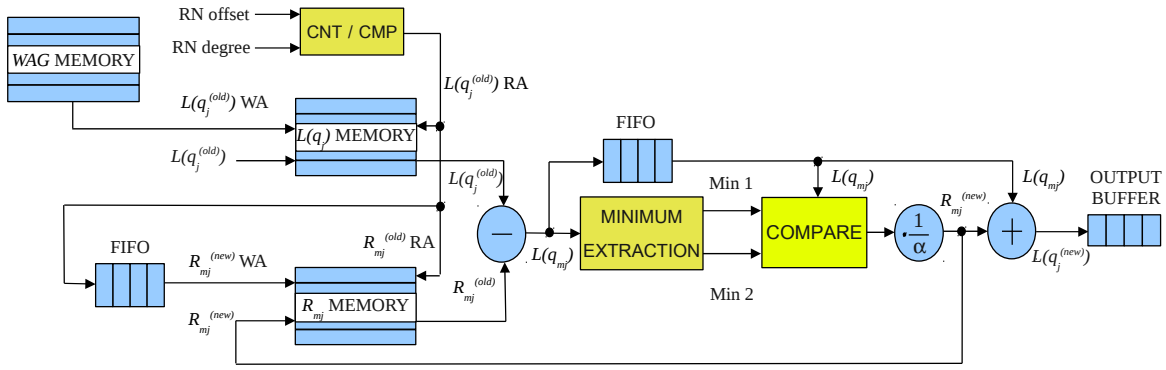


Figure 3. Simplified block scheme for the processing element

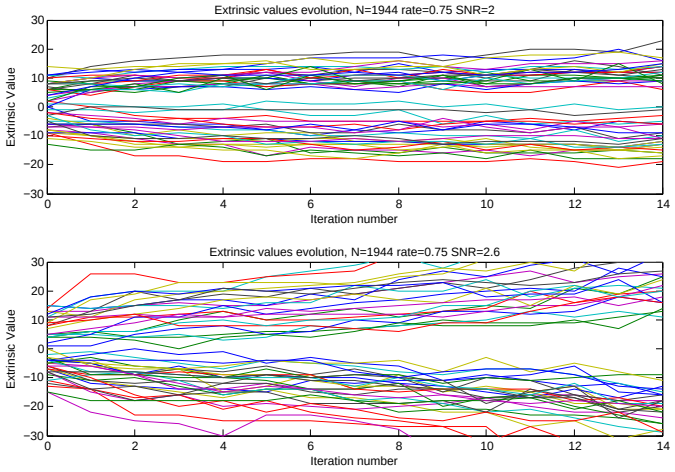


Figure 4. Extrinsic values evolution for WiFi code $(1944, 0.75)$, with SNR=2.0 (upper) and SNR=2.6 (lower)

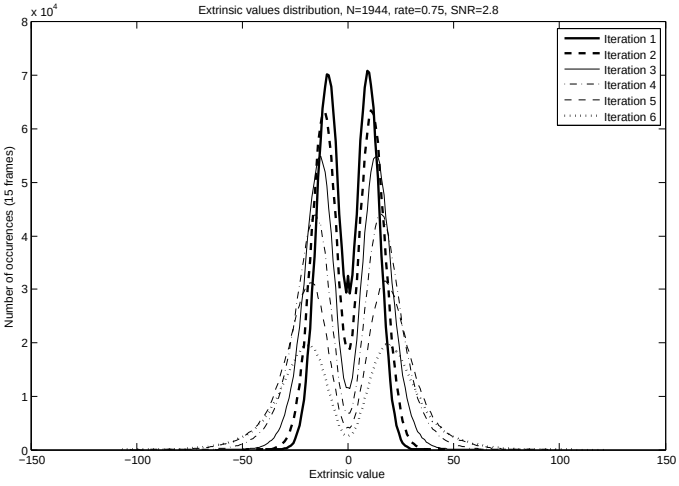


Figure 5. Extrinsic values distribution over 15 frames, WiFi $(1944, 0.75)$, SNR=2.6

Figure 5 shows the distribution of message values at different iterations. As expected, the number of uncertain extrinsics decreases as the decoding proceeds, meaning that most of errors introduced by the channel are corrected in the initial iterations; remaining errors after initial iterations are associated to low absolute values of extrinsics ($L(q_j)$). This behavior of extrinsics can be exploited to reduce the traffic injected in the NoC. The basic idea is very simple: once a given extrinsic has reached a high enough value, it is not updated anymore in the following iterations. This implies that extrinsics are saturated to a certain limit and saturated values are not sent through the NoC. We call this method of traffic reduction on the NoC "message stopping" (MS).

In order to apply MS, extrinsics must be compared against a threshold during the decoding process and the value of the threshold has to be decided by simulation. Several thresholds have been tried for different codes and the value that guarantees at the same time a high number of stopped extrinsics and a small effect on BER performance has been selected. The choice of the threshold, THR, is also affected by decoding algorithm, finite precision representation of data and SNR.

Table I (column 2) shows the average percentages of stopped messages ($S_{msg}$) for the decoders addressed in Section VII. The given percentages have been computed by considering two NoC based decoders: decoder *A* does not support MS and simply executes decoding iterations up to a given maximum number $It_{max}$; the decoding of a frame is actually stopped before reaching $It_{max}$ only if all PCCs are verified. Decoder *B* executes the same algorithm as *A*, but it also implements MS, meaning that extrinsics are compared against THR and when THR is passed they are no more updated. For both decoders, the global numbers of extrinsics that are updated in the decoding of a data frame are registered and averaged across several frames. Table I shows that $S_{msg}$ ranges between 19% and 40% for considered codes. these relevant percentages motivated us to further study the impact of MS on the decoder implementation.

Figure 6 gives the BER curves for different cases of MS applied to the (2304 , 1152) WiMAX code. Different SNR losses (0.1, 0.2 and 0.3 dB) are obtained at the $10^{-5}$ BER crossing point. Percentages in Table I have been obtained for the 0.3 dB case.

In an NoC based decoder, the time length of a decoding iteration has three components. The first component depends on the number of cycles taken to inject messages into the network: ideally, a PE needs $N_{pc} \times N_d$ cycles to generate and send out all messages corresponding to all assigned PCCs. The second component comes from the distance between source and destination PEs in the NoC: the physical delivery of a

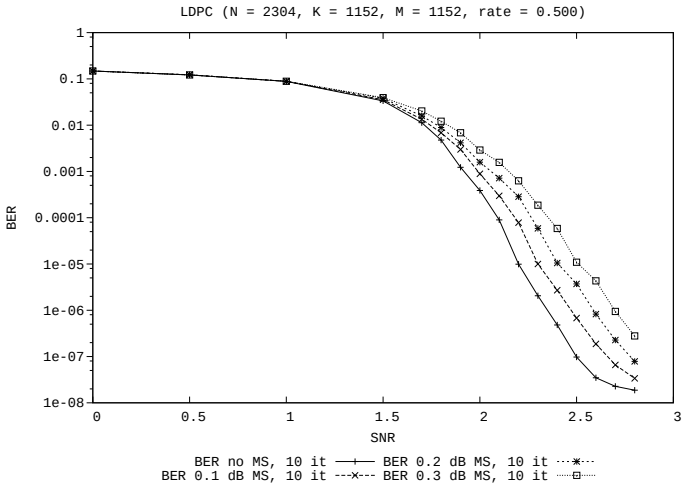


Figure 6. BER curves without and with message stopping

message corresponds to a number of hops depending on the size of the NoC and on the selected path. The last component derives from the conflicts that occur at NoC nodes when multiple messages have to be routed across the same port of the switch. Intuitively, in a small to medium size NoC decoder the third contribution to the iteration length tends to dominate, as sent messages generate several conflicts and spend several cycles in the FIFOs. In such a case, message stopping is a very efficient way to improve the throughput: if the injected number of messages is reduced by a certain percentage, the corresponding throughput is expected to improve by approximately the same percentage.
However, in a larger NoC, the iteration length is dominated by the first and second contributions, thus the actual gain in throughput tends to be smaller that the percentage reduction of injected messages.

The cycle–accurate NoC model has been used to simulate decoding iterations with and without message stopping. The throughput gain achieved by means of message stopping in the decoding of several LDPC codes is reported in Table I (columns 3 and 4). For each code, two NoC based decoders have been considered: a large decoder with $P_1^2$ PEs, and a smaller one with $P_2^2$ PEs. In both cases, the throughput $T_{gain}$ is provided as a percentage increase with respect to the same decoding executed with no message stopping. It can be seen from Table I that larger gains, between 10% and 15%, are obtained for the smaller NoCs

### A. Architecture

Additional architecture components are required to support the MS (Fig. 7). Each extrinsic $L(q_j)$ to be injected into the NoC has to be compared against $THR$. If $|L(q_j)| > THR$ the message is considered as reliable enough and must not be sent. To implement such behavior, the Check Block (CB) is inserted at the output buffer of each PE. The CB performs the threshold–message comparison: a subtraction generates a sign bit, which is appended to the message and used as a "stopped" flag (F) to inform the destination PE that the current message is received for the last time.

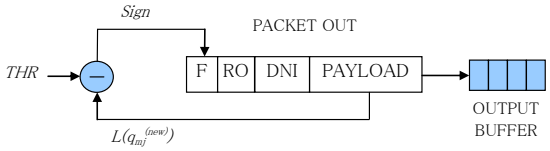


Figure 7. Architecture of the Check Block (CB)

Table I
EFFECT OF BANDWIDTH REDUCTION METHODS FOR DIFFERENT LDPC CODES. MESSAGE STOPPING WITH THRESHOLD $THR$ ON LDPC DECODER WITH $P_x$ PES, AND $S_{msg}$ STOPPED MESSAGES. $T_{gain}$ IS THE THROUGHPUT GAIN OVER THE AVERAGE.

| Code | Msg. stopping | | | | | | Early stop |
|---|---|---|---|---|---|---|---|
| | $THR$ | $S_{msg}$ | $P_1$ | $T_{gain}$ | $P_2$ | $T_{gain}$ | $T_{gain}$ |
| 802.16e (2304, 0.5) | 16 | 19% | 25 | 5.2% | 9 | 15.1% | 6.2% |
| 802.16e (2304, 0.83) | 17 | 27% | 25 | 5.5% | 9 | 12.9% | 9.2% |
| 802.16e (1632, 0.5) | 14 | 32% | 25 | 4.2% | 9 | 11.2% | 6.2% |
| 802.16e (1632, 0.83) | 16 | 36% | 25 | 5.6% | 9 | 19.9% | 32.8% |
| 802.16e (576, 0.5) | 10 | 39% | 25 | 7.9% | 9 | 10.6% | 11.3% |
| 802.16e (576, 0.83) | 12 | 40% | 25 | 6.3% | 9 | 11.5% | 43.9% |
| 802.11n (1944, 0.75) | 16 | 21% | 16 | 3.7% | 9 | 6.5% | 6,7% |

To support MS a dynamic routing is also required instead of a static routing. As the stopping of messages cannot be predicted, the off–line derivation of routing decisions is not possible. A packet header must be created, containing the Destination Node Identifier (DNI), which is used by the routing algorithm executed at each NoC node to properly deliver incoming messages. The so-called O1Turn routing method [25] is adopted in this work due to its reduced complexity. Finally, an additional field is required in the packet to compensate for the unpredictable arrival order of messages. This field (RO) contains the address for writing the corresponding extrinsic in the $L(q)$ memory. The whole structure of the packet is shown in Fig. 7, where the field PAYLOAD contains the extrinsic value.

## VI. EARLY STOPPING OF ITERATIONS

In the decoding of LDPC codes, the average number of iterations (ANI) is known to be much lower than $It_{max}$: for example, the first row in Table II shows that, in the decoding of WiMAX codes with $It_{max} = 10$, the ANI ranges between 2.9 and 6.1, depending on the SNR. These results have been obtained on a $5 \times 5$ topology by simply stopping the decoding of a frame as soon as a valid codeword is found. The introduction of an early stopping (ES) criterion can be of great benefit to reduce power dissipation and several ES methods have been proposed in the literature to this purpose. In this work we apply a recently proposed ES method [15] with the aim of reducing the occupied area. A lower ANI can be easily exploited to increase the decoding throughput. However a $5 \times 5$ NoC based decoder with no ES and no MS (first row

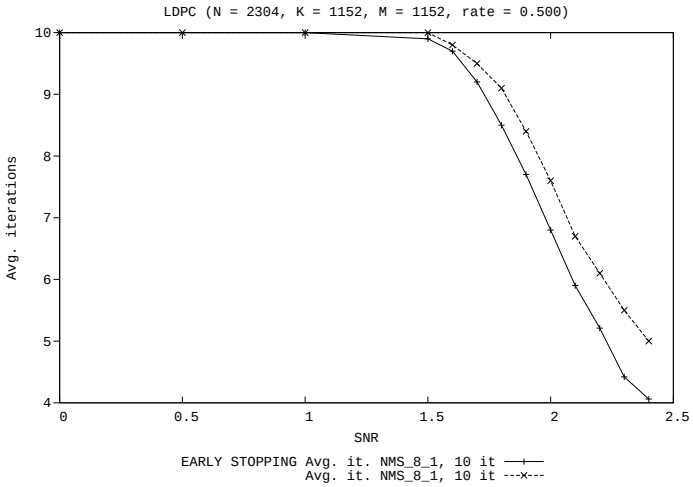


Figure 8. Average iterations curves with and without early stopping

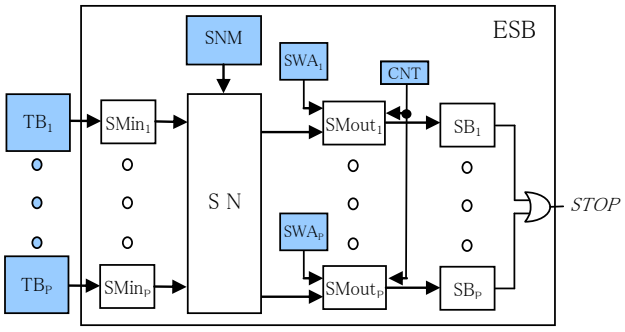


Figure 9. Iteration early stopping block scheme

in Table II) is already compliant with WiMAX standard in terms of achievable throughput. Therefore we exploit ES to reduce the degree of parallelism of the decoder, $P$, and thus the size of the NoC. In particular we show that a $3 \times 3$ NoC decoder with ES guarantees the same throughput offered by the $5 \times 5$ architecture with no ES at the cost of a small BER performance penalty.

In [15] an early stopping method is described with reference to WiMAX and WiFi LDPC codes. The proposed method basically detects iterations that are required to correct parity bits and skip them. To this purpose, incorrect codewords are divided into two types. Type I takes into account errors located either in the information part of the codeword or in the first $z$ positions of the parity part, where $z$ is the expansion factor of $\mathbf{H}$. Type II refers to errors located in the last M-$z$ positions of the codeword, where $M$ is the number of rows in $\mathbf{H}$. At high SNR values ($>$ 1.7 dB), Type II errors are much more frequent than Type I. Said $s_i$ the result of the $i^{th}$ parity check equation, the syndrome vector $\mathbf{s}$ is defined as $\mathbf{s} = [s_0, s_1, ..., s_{M-2}, s_{M-1}]^T$.

The syndrome accumulation vector (SAV) $\mathbf{a} = [a_0, a_1, ..., a_{z-2}, a_{z-1}]^T$ is defined so as

$$a_i = \sum_{k=0}^{c-1} (s_{i+kz}) \tag{9}$$

and $c = M/z$. It is shown in [15] that the SAV vector is entirely composed by even numbers for Type II errors: in this case, the decoding process can be stopped with no loss of information. On the contrary, if an odd number is present in the SAV, then the codeword is of Type I and the decoding must continue. Even though presented for WiMAX and WiFi standards, the ES method can easily be extended to less structured codes.

The effects of this ES criterion have been evaluated by means of the same C++/Python simulation model used for the MS method. Performed simulations show that BER performance is weakly affected by the selected ES method. On the other side, the ANI is greatly reduced, as shown in Figure 8 for the WiMAX $(2304, 0.5)$ code. It can be seen that the curves corresponding to the decoding with and without ES criterion are almost overlapped at low SNR, meaning that the number of Type II codewords is limited in this region. At high SNRs, the ES method offers a percentage reduction of ANI close to 20%

### A. Architecture

Additional hardware resources are necessary to support the described ES method (Figure 9): $P$ Transmission Blocks (TB), one for each PE, and a unique Early Stopping Block (ESB). The ES processing can be divided in three steps.

1) In step 1, $s_i$ are computed for each PCC $i$ ($i = 0, 1, \ldots, M-1$) and delivered from PEs to ESB. The computation is locally performed by the TB of each PE: this simply requires XOR–ing the sign bits of extrinsic values in PCCs. The delivery requires one dedicated connection from every PE to the ESB. At receiving, the $s_i$ are sequentially stored in the $P$ input memories $\text{SMin}_h$, $h = 1, \ldots, P$.
2) In step 2, $s_i$ are reordered by means of a shuffling network (SN) to enable the SAV calculation (9). The whole set $S = \{s_i | i = 0, 1, \ldots, M-1\}$ of syndromes stored in SMin memories is actually partitioned into $P$ sub–sets: $S = \bigcup_{h=1}^{P} S_{in}^h$, where $S_{in}^h$ contains all $s_i$ evaluated from PCCs mapped to the $h^{th}$ PE. The shuffling network generates a new partitioning of $S$, where $s_i$ are divided according to the SAV vector: $S = \bigcup_{h=0}^{z-1} S_{out}^h$, where sub–set $S_{out}^h$ includes all $s_i$ evaluated from PCCs belonging to SAV element $h$. After shuffling, syndromes are stored in the $P$ output memories $\text{SMout}_h$ ($h = 1, \ldots, P$).
3) In step 3, $a_j$ elements of SAV $\mathbf{a}$ ($j = 0, 1, \ldots, z-1$) are computed in parallel by $P$ xor gates and $P$ SAV blocks (SB). Since usually $z > P$, each SB computes in sequence multiple items of $\mathbf{a}$. A final OR gate generates the binary output STOP, which is the final decision on stopping.

The TBs operate concurrently with each decoding iteration and do not introduce latency. On the contrary, the ESB processing introduces additional cycles of latency: $M/P$ cycles are necessary to move $s_i$ syndromes from SMI to SMO memories; the same number of cycles are required to read $P$ $s_i$ syndromes at the time from SMO memories and evaluate the

Table II
LDPC ARCHITECTURES COMPARISON: CMOS TECHNOLOGY PROCESS (TP), AREA OCCUPATION ($A$), NORMALIZED AREA OCCUPATION FOR 65NM TECHNOLOGY ($An$), CLOCK FREQUENCY ($f_{clk}$), PRECISION BITS ($b$), AVERAGE ENERGY PER FRAME DECODING ($E_f$), MAXIMUM ($It_{max}$) AND AVERAGE (ANI) NUMBER OF ITERATIONS, MINIMUM THROUGHPUT ($T$) AND SNR TO ACHIEVE BER=$10^{-5}$ ($SNR$)

| Decoder | | TP [nm] | $A$ [mm$^2$] | $An$ [mm$^2$] | $f_{clk}$ [MHz] | $b$ [bits] | $E_f$ [$\mu$J] | $It_{max}$ | ANI | Code length - rate | $T$ [Mb/s] | SNR [dB] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $5\times5$ **NoC** | No MS No ES | 130 | 4.72 | 1.18 | 300 | 8 | 1.14 | 10 | 2.9 | 576 - 0.5 | 71 | 2.9 |
| | | | | | | | 4.98 | 10 | 4.9 | 1632 - 0.5 | 78 | 2.4 |
| | | | | | | | 8.28 | 10 | 6.1 | 2304 - 0.5 | 82 | 2.2 |
| $5\times5$ **NoC** | No MS ES | 130 | 5.49 | 1.37 | 300 | 8 | 0.93 | 10 | 2.9 | 576 - 0.5 | 70 | 2.9 |
| | | | | | | | 4.44 | 10 | 4.9 | 1632 - 0.5 | 76 | 2.4 |
| | | | | | | | 7.59 | 10 | 6.1 | 2304 - 0.5 | 81 | 2.2 |
| $6\times3$ **NoC** | MS ES | 130 | 4.20 | 1.05 | 300 | 8 | 0.56 | 10 | 2.9 | 576 - 0.5 | 70 | 3.1 |
| | | | | | | | 3.26 | 10 | 4.9 | 1632 - 0.5 | 72 | 2.6 |
| | | | | | | | 5.32 | 10 | 6.1 | 2304 - 0.5 | 74 | 2.4 |
| $4\times4$ **NoC** | MS ES | 130 | 3.61 | 0.90 | 300 | 8 | 0.68 | 10 | 2.9 | 576 - 0.5 | 61 | 3.0 |
| | | | | | | | 3.57 | 10 | 4.9 | 1632 - 0.5 | 67 | 2.5 |
| | | | | | | | 6.69 | 10 | 6.1 | 2304 - 0.5 | 64 | 2.3 |
| $3\times3$ **NoC** | MS ES | 130 | 2.68 | 0.67 | 300 | 8 | 0.49 | 10 | 2.9 | 576 - 0.5 | 74 | 3.2 |
| | | | | | | | 2.84 | 10 | 4.9 | 1632 - 0.5 | 71 | 2.7 |
| | | | | | | | 4.81 | 10 | 6.1 | 2304 - 0.5 | 72 | 2.5 |
| [6] | | 65 | 0.62 | 0.62 | 400 | N/A | | 20 | N/A | WiMAX | 27.7 | N/A |
| [26] | | 180 | 3.39 | 0.442 | 100 | N/A | | 10 | N/A | WiMAX | 68 | N/A |
| [27] | | 65 | 1.337 | 1.337 | 400 | 6 | | 20 | N/A | WiMAX | 48 (min) | N/A |
| [13] | | 130 | 3.7 | 0.93 | 300 | 6 | | 10 | N/A | 2304 - 0.5 | 56 | N/A |
| [28] | | 90 | 0.679 | 0.354 | 400 | 7 | | 12 | 6.64 | 2304 - 05 | 66.7 | 2.15 |
| [7] | | 90 | 6.22 | 3.24 | 300 | 6 | | 20 | N/A | WiMAX | 212 (max) | 2.2 (min) |

final decision (STOP). The whole latency, $2M/P$, corresponds to several cycles, depending on code length and NoC size, however it can be easily accomodated within a decoding iteration. For example, the $5 \times 5$ NoC based decoder with no MS and no ES in Table II needs 421 cycles to complete a single iteration when decoding the (2304,1152) WiMAX code; for this example, $M = 1152$ and $P = 25$, thus the additional latency to implement ES is 92 cycles, equal to 22% of the length of one iteration. For the same example code, the overhead due to the ESB latency can be evaluated in Fig. 8, where the third curve shows the effective ANI obtained with the implemented ES method: at 2.2 dB, ES method should ideally reduce the ANI from 6 to 4.8; the ESB latency causes a delay in the stopping decision and this changes the ANI to 5.2, which is still a relevant advantage with respect to the original value.

The results in terms of throughput gain can be seen in the last column of Table I for several codes. The given percentages also take in account the ESB latency.

The ESB contains four types of memories:

1) $P$ SMin memories receive the $M$ syndromes, therefore each of them has size $M/P \times 1$.
2) $P$ SMout memories receive the reordered syndromes (size $M/P \times 1$).
3) SNM memory contains controls for the shuffling network. For each syndrome to be moved from SMin$_i$ to SMout$_j$, SNM must enable the right path between ports $i$ and $j$. As the network has $P$ input and output ports, $P \cdot \lceil log_2 P \rceil$ control bits are required. $M/P$ syndromes are received at each input port. thus the total size for SNM is $M \cdot \lceil log_2 P \rceil$ bits.
4) Finally, $P$ SWA memories are allocated to store write addresses for SMout components. For each of them, $M/P$ words are needed and every word contains $\lceil log_2(M/P) \rceil$ address bits, plus one additional bit to be used as write command for SMout memories. The total size of a SWA memory is therefore $M \cdot (1 + \lceil log_2(M/P) \rceil)$ bits.

The content of SNM and SWA memories depends on the specific scheduling of PCCs on decoder PEs. The global amout of memory can be expressed as $M \cdot (3 + \lceil log_2(M/P) \rceil + \lceil log_2(P) \rceil)$. For example, the ESB memory required to support WiMAX codes on a $4 \times 4$ NoC based decoder, is obtained with $P = 16$ and $M = 1152$ and is equal to 16K bits.

## VII. ACHIEVED RESULTS

The first row in Table II refers to a 25-PEs NoC sized to support all WiMAX LDPC codes. The decoder is fully flexible and able to support any other LDPC code with size lower than the largest WiMAX code. However, in this paper, we limit presented results to the case of WiMAX codes; obtained performance on other LDPC codes are available in [14]. Even if the decoder does not include MS and ES methods, the number of iterations allowed and the degree of parallelism of the NoC guarantee a throughput of at least 70 Mbits/s for all code lengths and rates in WiMAX standard. Comparing this decoder with implementations reported in the last row of the Table, it can be seen that the worst–case throughput is quite high, while larger area is required due to the high degree of

flexibility provided by the NoC approach. This overhead can be significantly reduced by introducing ES and MS methods.

Row 2 in Table II is related to a $5 \times 5$ NoC based decoder implementing the ES method as described in Section VI. Comparing this implementation with the decoder in the first row, which has the same size but does not supports ES, it can be seen that the worst–case throughput is not dramatically changed, while occupied area increases by almost 20%, due to the additional hardware components required to support ES. The only advantage provided by ES is related to the average energy dissipated to decode a data frame , $E_f$, which is reduced by roughly 20%.

In row 3, results are given for an 18 PEs NoC ($6 \times 3$) supporting both ES and MS. The potential offered by MS method is exploited to reduce the number of PEs, so saving both area (-11%) and energy (-37%). The throughput still reaches 70 Mbps for the WiMAX codes, while a 0.2 dB penalty is paid in terms of BER performance.

Two further solutions are explored with decoders in rows 4 and 5. The $4 \times 4$ NoC based decoder in row 4 provides reduction of 24% on area and 19% to 40% on $E_f$, with minor penalties in terms of throughput (15%) and BER performance (0.1 dB). The $3 \times 3$ case achieves the lowest occupied area, which is comparable with the best implementations reported in the last row of the Table, and the lowest $E_f$. Moreover the throughput is compliant with WiMAX standard. The BER penalty in this case if 0.3 dB.

## VIII. Conclusions

The design of a fully flexible NoC based LDPC decoder is presented, together with two complementary methods for reducing the traffic injected into the network. These methods provide relevant area and power saving.

The first proposed decoder implementation offers an unparalleled degree of flexibility and throughput higher than 70 Mbps on WiMAX codes. A penalty in terms of additional area and power is paid for this decoder with respect to state of the art dedicated or partially flexible decoders. The other presented NoC based decoders exploit early stopping of iterations and message stopping to scale the whole NoC to lower degrees of parallelism: the scaled architectures still achieve high enough worst–case throughput at a much lower area and power cost.

## References


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

[2] D. MacKay, "Good error-correcting codes based on very sparse matrices," in *Information Theory. 1997. Proceedings., 1997 IEEE International Symposium on*, 1997.

[3] J. Lorincz and D. Begusic, "Physical layer analysis of emerging IEEE 802.11n WLAN standard," in *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, vol. 1, 2006, pp. 6 pp. –194.

[4] M. Khan and S. Ghauri, "The WiMAX 802.16e physical layer model," in *Wireless, Mobile and Multimedia Networks, 2008. IET International Conference on*, 2008, pp. 117 –120.

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

[6] 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.

[7] C.-H. Liu, C.-C. Lin, S.-W. Yen, C.-L. Chen, H.-C. Chang, C.-Y. Lee, Y.-S. Hsu, and S.-J. Jou, "Design of a multimode QC-LDPC decoder based on shift-routing network," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 56, no. 9, pp. 734 –738, 2009.

[8] X. Chen, S. Lin, and V. Akella, "QSN:a simple circular-shift network for reconfigurable quasi-cyclic LDPC decoders," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 57, no. 10, pp. 782 –786, 2010.

[9] M. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *Information Theory, IEEE Transactions on*, vol. 50, no. 8, pp. 1788 – 1793, 2004.

[10] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70 –78, Jan. 2002.

[11] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *Design Test of Computers, IEEE*, vol. 22, no. 5, pp. 414 – 421, 2005.

[12] L. Benini, "Application specific NoC design," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, 2006, pp. 1 –5.

[13] 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.

[14] C. Condo and G. Masera, "Omitted for blind review," *IEEE Trans. VLSI Syst.*, submitted for publication, available on *arxiv.org*.

[15] Z. Chen, X. Zhao, X. Peng, D. Zhou, and S. Goto, "An early stopping criterion for decoding ldpc codes in wimax and wifi standards," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 30 2010-june 2 2010, pp. 473 –476.

[16] G. Gilikiotis and V. Paliouras, "A low–power termination criterion for iterative LDPC code decoders," *Signal Processing Systems Design and Implementation, 2005. IEEE workshop on*, 2005.

[17] Y. Sun and J. R. Cavallaro, "High throughput VLSI architecture for soft-output MIMO detection based on a greedy graph algorithm," in *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI'09*, New York, USA, Mar. 2009, pp. 445 – 450.

[18] W. Wang and G. Choi, "Minimum–energy ldpc decoder for real–time mobile application," *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, 2007.

[19] ——, "Speculative energy scheduling for ldpc decoding," *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, 2007.

[20] W. Wang, G. Choi, and K. Gunnam, "Low–power vlsi design of ldpc decoder using dvfs for awgn channels," *VLSI Design, 2009 22nd International Conference on*, 2009.

[21] F. Guilloud, E. Boutillon, J. Tousch, and J.-L. Danger, "Generic description and synthesis of LDPC decoders," *Communications, IEEE Transactions on*, vol. 55, no. 11, pp. 2084 –2091, 2007.

[22] 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.

[23] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *Communications, IEEE Transactions on*, vol. 47, no. 5, pp. 673 –680, May 1999.

[24] T. Theocharides, G. Link, N. Vijaykrishnan, and M. Irwin, "Implementing LDPC decoding on network-on-chip," in *VLSI Design, 2005. 18th International Conference on*, 2005, pp. 134 – 137.

[25] D. Seo, A. Ali, W.-T. Lim, and N. Rafique, "Near-optimal worst-case throughput routing for two-dimensional mesh networks," in *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, june 2005, pp. 432 – 443.

[26] T.-C. Kuo and A. Willson, "A flexible decoder IC for WiMAX QC-LDPC codes," in *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, 2008, pp. 527 –530.

[27] T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, N. Wehn, N. L'Insalata, F. Rossi, M. Rovini, and L. Fanucci, "Low complexity LDPC code decoders for next generation standards," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, 2007, pp. 1 –6.

[28] Y.-L. Wang, Y.-L. Ueng, C.-L. Peng, and C.-J. Yang, "Processing-task arrangement for a low-complexity full-mode WiMAX LDPC codec," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 2010.