Chapter 7
MP-SoC/NoC Architectures for Error Correction

Carlo Condo, Maurizio Martina, and Guido Masera

7.1 Introduction

In the last year several standards for both wired and wireless communications have been proposed. Indeed, modern terminals, such as smartphones and tablets, are equipped with different modules for ubiquitous access to internet. As argued in [38], flexibility has become a fundamental property of architectures for digital baseband processing, as it allows to support different operating modes more efficiently than simply placing several modules together and turn them on or off. Unfortunately, the high throughputs to be sustained make actual flexibility implementation a challenging task, especially in the context of channel code decoder architectures, where often decoding algorithms are both complex and iterative.

High throughput imposes to have parallel architectures made of several processing elements (PEs) connected via and appropriate communication backbone. The design of an optimized architecture for a single code is a well-established problem and has been largely investigated in the past. On the contrary, depending on the amount of required flexibility, the problem of designing efficient architectures becomes increasingly challenging. In this sense we can have two main classes of flexibility: (a) architectures that support only one family of codes (such as turbo codes or LDPC codes) or (b) architectures that support more families of codes (e.g., both turbo and LDPC codes). In order to achieve interoperability among different standards, the second class is the most interesting one. Indeed it contains three sub-classes of particular interest: (1) architectures that support different codes within a standard (e.g., both turbo and LDPC codes for the WiMAX standard), (2) architectures that support different codes within more standards (e.g., both turbo

C. Condo • M. Martina • G. Masera (✉)
Politecnico di Torino, Corso Duca degli Abruzzi, 24 Torino, Italy
e-mail: Carlo.Condo@polito.it; Maurizio.Martina@polito.it; Guido.Masera@polito.it

C. Chavet, P. Coussy (eds.), *Advanced Hardware Design for Error Correcting Codes*, DOI 10.1007/978-3-319-10569-7_7

and LDPC codes for WiFi, WiMAX and LTE standards), (3) architectures that support different codes and that are *future proof* or *fully flexible*, which means that these architectures can accommodate any code, provided that the amount of available resources is enough. This last case is the most challenging one as adding flexibility can lead to less optimized solutions with respect to cases (1) and (2). Nevertheless, all the aforementioned cases require to design both flexible PEs and flexible communication structures.

Flexibility in the PEs can be achieved adding some multiplexers to select among a fixed set of alternatives or via programmable architectures such as Application Specific Instruction-set Processors (ASIPs). On the other hand, flexibility in the communication structure can be obtained resorting to crossbars, shuffling-networks, or borrowing results from the general NoC paradigm.

7.2 Flexibility in the Communication Structure

Some works in the literature propose flexible and efficient communication structures either for turbo [16] or LDPC codes [4] relying on crossbars, or shuffling-networks. Unfortunately, these solutions are usually tailored around specific characteristics of some particular classes of codes, thus not being used or extended in the case of a general approach. Stemming from the NoC paradigm [12], Neeb et al. [36] proposed an interesting NoC-based approach to enable flexible and efficient interconnection among P processing elements (PE) in parallel turbo decoder architectures. According to [43] 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. The literature shows that the intra-IP NoC approach has been mainly studied in the context of (1) parallel turbo decoder architectures [25, 27, 30], (2) semiparallel LDPC code decoder architectures [14,31,43], (3) flexible turbo/LDPC code decoder architectures [6, 31]. In the following we will assume that each PE is made of a processing core and a memory (see Fig. 7.1a), where the processing core implements LLR (Logarithmic Likelihood Ratio) computation/updating operations and the memory is used both for the storage of data coming from the network and as an input buffer for the processing core.

IntraIP-NoCs tailored around iterative channel decoder architectures exhibit some specific features that are summarized in the following. Since the data block is partitioned into P subblocks and each PE is assigned one subblock, all nodes exchange nearly the same amount of data. Moreover, idle times in the PEs are reduced to maximize the throughput. As a consequence, the nodes exchange the data at a fairly constant rate. Besides, due to the random nature of interleavers for turbo codes and $\mathbf{H}$ matrices for LDPC codes, the pattern of connection among PE has little adjacency. One of the main consequences of these properties is that the injected traffic load tends to be uniform both in time and space. Furthermore, since decoding algorithms are iterative, minimizing the latency of the single iteration is of paramount importance to achieve high throughput. Thus, simple routing algorithms

and routing circuits have to be designed. To this purpose the designer can take advantage of the uniform traffic load and the homogenous nature of the nodes. It is worth noting that, as the amount of packets injected into the network is known and depends on P and on the data block size N_f, flow control is not required. Nevertheless, it is important to define the packet injection rate r, namely the number of packets injected in the network by a PE in a clock period. This parameter can simultaneously model two phenomena: (1) the use of different clock frequencies for the PEs and the network (usually the maximum clock frequency of PEs is lower than the one used for the network). (2) the fact that a PE may not produce a valid packet at each clock cycle. Finally, if the permutation laws of the turbo code interleavers and the **H** matrices of the LDPC codes considered by a decoder are known, then the traffic patterns can be derived by off-line analysis. This leads to the so called *zero-overhead* networks introduced in [43] and further developed in [25,30], where all the routing information is precalculated by the means of a simulator, as the one in [24], leading to significant simplification in the architecture of the nodes.

The structure of the packets is common to all the proposed architectures, namely each packet is made of a header and a payload. The header contains routing information, such as the identifier of the destination PE that, for P processing elements, requires $\log_2(P)$ bits. The payload contains both a refined LLR and the memory location where the LLR has to be stored. Even if the number of bits used to represent LLRs impacts on the bit-error-rate performance of a code, 8 bits is a typical value. On the other hand, the memory location is represented on $\lceil \log_2(N_f/P) \rceil$ bits.

The NoC-based approaches proposed in the literature for iterative channel decoder architectures can be divided into two main categories depending on the type of network they employ: (1) indirect networks [30,31] (2) direct networks [6,14,25]. In the following both approaches are described.

7.2.1 Indirect Networks

The most popular indirect network topologies proposed in the literature for iterative channel decoder architectures are Multistage Interconnection Networks (MINs) [10]. MINs rely on the cascade of several switch stages, each of which is often referred to as router in the literature of NoC-based iterative channel decoder architectures. On the other hand, PEs are not part of the routers, but they serve as both the input and the output of the network (see Fig. 7.1a). Examples of such networks are Clos, Benes, Omega, and Butterfly networks [10]. The number of stages and the number of switches per stage changes with the topology, as an example the Butterfly network is made of $\log_2(P)$ stages where each stage relies on $P/2$ switches.

It is worth noting that the degree of connectivity obtained using these networks is larger than that achievable with shared bus structures, but it is smaller than the one reachable with crossbars. As a consequence, when two or more paths connecting

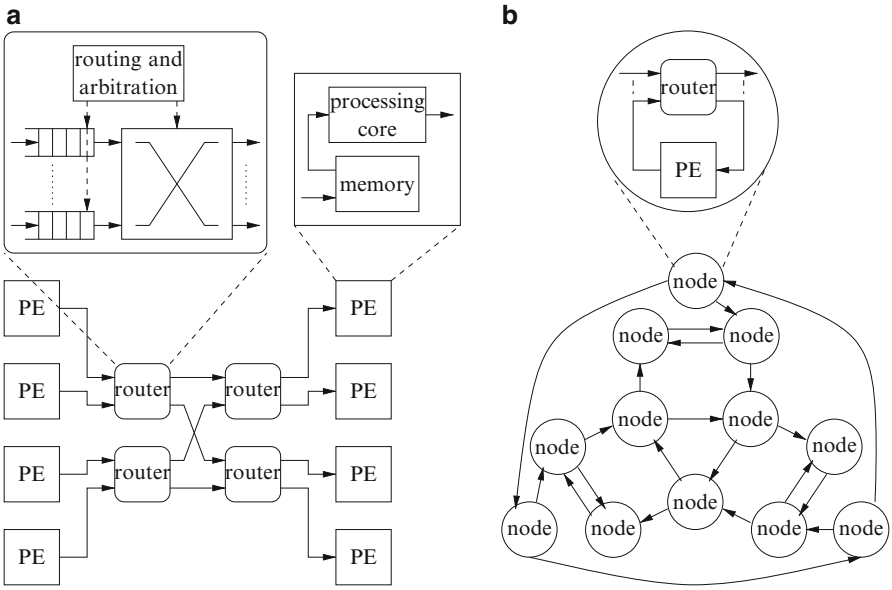

Fig. 7.1 Network and node architecture: (**a**) indirect network example, (**b**) direct network example

source/destination PEs cross, conflicts arise. A simple solution to handle conflicts is to include two FIFO queues in the router architecture. Thus, each router relies on a 2×2 switch, two input FIFOs for storing conflicting packets and a routing and arbitration block that serves the input queues with a round-robin policy and generates all the required control signals as depicted in the top part of Fig. 7.1a.

The most significant results in the literature concerning indirect networks relate to Butterfly and Benes networks [27,30]. The Butterfly network features logarithmic diameter and a highly scalable recursive structure. Moreover, routing in the Butterfly network is very simple as the bits of the destination PE are used to select the output port at each stage of the network. On the other hand, this network has no path diversity, namely there exist only one path connecting two PEs; as a consequence, conflicts can arise frequently.

An alternative solution to partially solve conflicts is the Benes network. The Benes network has a larger path diversity than the Butterfly network, even if its diameter is about two times the diameter of the Butterfly network. However, conflicts are avoided with the Benes network only if the source/destination pattern is a permutation of the PE identifiers. Unfortunately, this is not always the case of turbo and LDPC codes. An interesting solution to solve this problem is the one described in [30], where a time-division-multiple-access architecture is proposed. The idea is to exploit the deterministic characteristics of the traffic to assign a time slot to each packet, that is the cycle when the packet will be injected into the network. As a consequence, the network inputs are scheduled such that at each cycle there

is only one packet per output port. Then, the routing information is precalculated off-line and stored in the packet header, leading to an architecture belonging to the zero-overhead class.

The analysis presented in [30] refers to $r = 0.2$ to avoid network congestion for both Butterfly and Benes networks. Moreover, implementation results shown in [27] on a 130 nm standard cell technology for $P = 16$ show that the Benes network performs better than the Butterfly one both in terms of occupied area and maximum achievable frequency/throughput. The Butterfly network occupies 0.75 mm^2 and achieves a clock frequency of 345 MHz corresponding to a throughput of 138 Mb/s. On the other hand, the Benes network occupies only 0.48 mm^2 (−35 %) and achieves a clock frequency of 381 MHz (+10.4 %) and a throughput of 152 Mb/s (+10 %).

7.2.2 Direct Networks

Direct networks analyzed in the literature [14, 25, 31] rely on P nodes whose architecture can be seen as a generalization of the one proposed for indirect networks. Indeed, each node contains one PE, D input, and D output connections from/to the network, where D is the topology degree, and a routing and arbitration block (see Fig. 7.1b). Thus, each node relies on $D+1$ input FIFOs and a $(D+1) \times (D+1)$ switch.

In [31] binary de-Bruijn topologies are proposed to design a flexible interconnection structure for an LDPC and a turbo/LDPC code decoder respectively. The binary de-Bruijn network is indeed very appealing to support the communications of a multiprocessor turbo/LDPC decoder as it features good path diversity properties. In addition, de-Bruijn networks have logarithmic diameter that leads to small latencies, and a recursive structure that makes them highly scalable. The analysis presented in [31] highlights that flexibility comes at the expense of a larger area with respect to indirect networks. Indeed, scaling the results to a 130 nm standard cell technology for $P = 16$ to compare with indirect network results, we obtain that the binary de-Bruijn network requires 0.64 mm^2 for a 244 MHz clock frequency, that is about +33 % of area and −36 % of clock frequency.

The works in [14, 25] extend the analysis to other topologies. Both [14, 25] analyze 2D-mesh topologies, showing interesting speed-up and scalability results. Moreover, in [25] several topologies are compared for network degrees ranging from 2 to 4. The analysis corroborates the results presented in [31] by showing that logarithmic topologies as the (generalized) de-Bruijn and Kautz ones are the most suited to achieve both flexibility and high performance. Moreover, experimental results presented in [25] for a wide number of cases prove that zero-overhead architectures require less area with respect to the ones where hardware resources for the routing algorithm are employed. Considering $P = 16$, $r = 0.33$ and a clock frequency of 200 MHz, zero-overhead architectures for generalized Kautz topologies achieve in the best case a throughput of about 102 Mb/s with an area of 0.74 mm^2 for $D = 2$ and about 103 Mb/s with 0.92 mm^2 for $D = 4$.

7.3 Review of Flexible Decoding Architectures

In this section, the state of the art on flexible channel decoders is briefly reviewed for the two most interesting classes mentioned in Sect. 7.1, namely architectures supporting different codes within more standards (multi-standard architectures) and architectures that in principle accommodate any code (fully flexible architectures).

7.3.1 Multi-Standard Architectures

While the same type of code can be used in various standards, code size, rate, and construction method can vary greatly between one application and the other. A first step towards flexibility in channel decoders is then guaranteeing support for a single type of code, taking in account the code parameters employed in different standards.

A flexible turbo code decoder architecture is devised in [19], where a decoder targeting both 3GPP-LTE and Mobile WiMAX standards is proposed. The different nature of the considered turbo codes (single-binary in 3GPP-LTE and double-binary in Mobile WiMAX) is tackled by means of bit-to-symbol and symbol-to-bit conversions [17], addressed later in Sect. 7.4.1 of this chapter, that allow almost complete memory sharing. Moreover, a novel dual-mode interleaver is introduced that reduces the overhead relative to the implementation of the native ARP [15] and QPP [45] interleavers respectively. The resulting multi-standard decoder can reach up to 186 Mb/s with moderate complexity.

Multi-standard support is achieved in [1] by means of partially parallel turbo code decoder based on ASIPs. Support is given for 3GPP-LTE, WiMAX, and DVB-RCS standards, reaching a maximum throughput of 170 Mb/s and yielding good efficiency. Different ASIPs are used for the in-order and interleaved phases of the decoding process, and the complexity and latency are kept in check via smart information exchange networks and pipeline idle time minimization.

The flexible LDPC decoder described in [43] is one of the first work to consider a Network-on-Chip (NoC) as a possible interconnection structure. Together with the design of processing elements, the design of the application-specific NoC is carried out in detail: it is shown that many of the characteristics of general purpose NoCs are not necessary, thus reducing the overhead commonly associated with complex interconnections. The complete decoder is arranged on a toroidal mesh topology.

A decoder with extremely good error correction capabilities is shown in [39]. It is designed targeting LDPC convolutional codes that can be obtained from quasi-cyclic LDPC codes. Again, the regular structure of these codes allows for easy design of largely parallel structure. Up to 2 Gb/s are obtained with a frequency of 100 MHz.

7.3.2 *Fully Flexible Architectures*

Few recent works have focused on extending the concept of flexible decoders not only to multiple codes, but also to multiple code types, providing complete support for whole standards.

The work in [33] describes the design of a multi-standard turbo/LDPC decoder based on ASIPs and the sharing of memories between the two code types. Each ASIP has two separate datapaths, one for each decoding mode: eight ASIPs are instantiated and connected via a simple but flexible interconnection network that can be reconfigured when switching decoding mode to adapt to the different communication patterns. Also in [2] is presented an ASIP-based decoder that includes convolutional code decoding as well. This work is characterized by extremely small area occupation and high achievable frequency that helps to meet most throughput requirements for the considered standards.

The works presented in [11, 35, 42] exploit commonalities between turbo and LDPC decoding to design a unified architecture for multiple standards. By viewing an LDPC code as a series of turbo codes [23], the BCJR algorithm can be applied to both code types. The shared datapath and memories result in an overall area much lower than separate dedicated decoder implementations.

7.4 Improving the Efficiency of NoC-Based Decoders

It has been shown in the previous section that NoCs are extremely flexible interconnection structures, able to guarantee connectivity among all the nodes of the network. NoC-based decoders can in fact support up to numerous standards at the same time [2, 7, 11, 41]. However, flexibility comes at the cost of increased interconnection complexity and additional latencies that impact heavily on both throughput and energy consumption, reducing the overall decoder efficiency. NoC-induced latencies in particular can be a major obstacle in extending the support of a decoder to multiple standards, whereas excessive energy consumption and area occupation limit the set of applications in which the decoder can be employed. To reduce the impact of NoCs on the decoder efficiency, various techniques can be applied.

7.4.1 *Energy Reduction Techniques*

The area occupation of a NoC usually consists of 20–40 % of the total decoder area [9, 26] that results in a proportional impact on the power consumption. It is consequently desirable to reduce the complexity of the NoC as much as possible. A set of interesting techniques is considered in [26], aimed at limiting the width of

the channels between the nodes of NoC-based turbo code decoders by reducing the number of concurrently transmitted extrinsic metrics and the number of quantization bits of each metric.

Let us define $\lambda^{ext}[u]$ as the extrinsic information associated to symbol u, obtained at the output of the SISO decoder at each half-iteration. All $\lambda^{ext}[u]$ must be sent through the NoC according to the interleaving rule: the width of each NoC channel must accommodate, depending on the nature of the considered code, the transmission of one or more concurrent extrinsic metrics. In Double-Binary turbo codes like the ones used in WiMAX, where each symbol u is composed of two bits, the transmitted $\lambda^{ext}[u]$ is an array of three elements. However, following the architecture presented in [18], it is possible to switch between symbol-level $\lambda^{ext}[u]$ and bit-level $\lambda^{ext}[A]$ and $\lambda^{ext}[B]$. The width of the transmitted packet is thus reduced by 1/3, together with the width of the memories in which $\lambda^{ext}[u]$ is stored. The consequent area reduction is much more consistent than the increment brought by the bit-to-symbol and symbol-to-bit conversion units, necessary since the BCJR algorithm implemented in SISOs requires symbol-level metrics. The conversion operation, however, introduces an overall Bit Error Rate (BER) performance loss of about 0.2 dB. A further step towards the reduction of the area occupation of the NoC and of the decoder in general can be taken by addressing the quantization of $\lambda^{ext}[A]$ and $\lambda^{ext}[B]$. Applying the Pseudo-Floating-Point (PFP) representation suggested in [37], it is possible to reduce the quantization without incurring in significant performance degradation. The idea is based on the fact that bits within the representation of extrinsic metrics play a different role in the decoding process according to their weight, as highlighted also in [40, 44]. Analyzing the binary representation of $\lambda^{ext}[A]$ and $\lambda^{ext}[B]$ from the most significant bit to the least significant bit, it is possible to detect the first zero-to-one or one-to-zero transition. This signals the starting bit of the significant part of the extrinsic metric. Finally, an equal number of bits is assigned to the significant parts of both $\lambda^{ext}[A]$ and $\lambda^{ext}[B]$, alongside a common shift factor used at reception to reconstruct the extrinsic values. It is show in [26] that the joint application of both methods can reduce the total NoC channel width of more than 50 % that, depending on the router architecture and decoder structure, can save up to 40 % of the total NoC area.

In iteratively decoded codes like turbo and LDPC codes, the energy required for the decoding of a frame is directly proportional to the number of performed iterations: limiting their number is then an effective way of limiting the energy consumption. A few Early Stopping Criteria (ESCs) can be found in the literature for turbo codes [13, 28], but many more have been proposed for LDPC codes, e.g., [3, 22, 29]. Since LDPCs have a straightforward method of identification of correct decoding, existing ESCs focus on the early identification of situations in which the decoding is going to fail. This is usually achieved via observation of the evolution of metrics throughout different iterations. Most ESCs offer limited flexibility, and their performance can vary substantially when applied to different codes. The Multi-Standard-ESC (MSESC) proposed in [8] has been designed to adapt on-the-fly to code parameters and channel conditions, thus being particularly fit for flexible multi-standard decoders. During each iteration, by comparing the

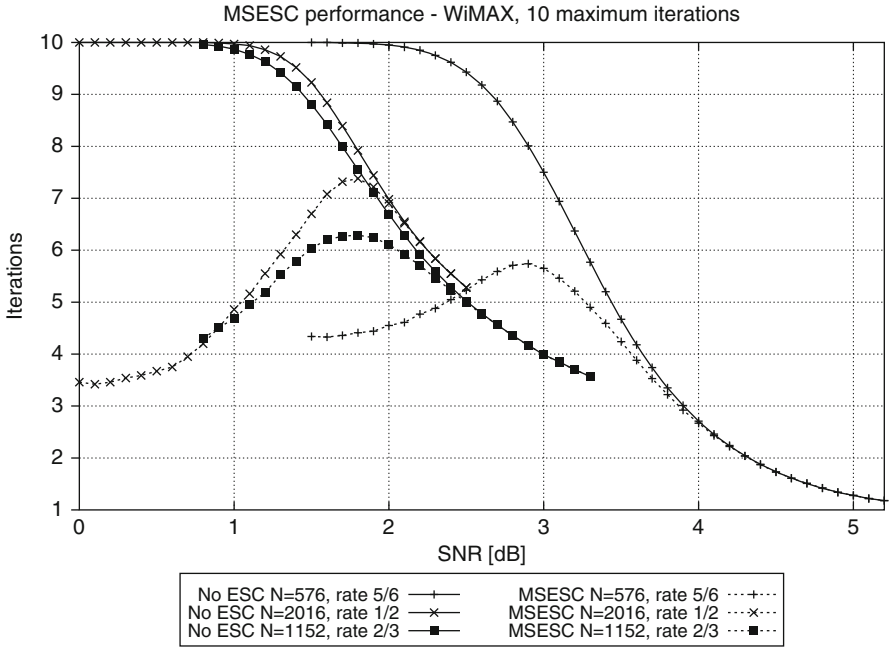

Fig. 7.2 MSESC performance for WiMAX codes

Check-Node-Mean-Magnitude metric [3] and the syndrome value against three thresholds $T1$, $T2$, and $T3$, MSESC decides on the stopping of the decoding process, effectively identifying both impossible decoding and insufficient available iterations for successful decoding. The thresholds can be computed as follows:

$$T1 = \mathrm{M} \cdot 2^{-6} \quad T2 = \mathrm{M} \cdot 2^{bits_f} \quad T3 = \mathrm{M} \cdot 2^{-5} = T1 \cdot 2 \tag{7.1}$$

where M is the number of rows in the LDPC parity check matrix and $bits_f$ is the number of fractional bits assigned to the representation of LLRs. The dependency of $T1$, $T2$, and $T3$ on code parameters and design choices allows MSESC to maximize the number of saved iterations without affecting the BER performance with a very wide range of codes. Figure 7.2 shows the number of performed iterations for the decoding of three WiMAX codes against the Signal-to-Noise Ratio (SNR), with and without MSESC. The codes are characterized by different block lengths and rates, but MSESC maintains its effectiveness with all of them. At low SNR successful decoding is improbable, and after very few iterations the decoding is stopped, once impossible decoding is identified. The performed iterations start to rise with the SNR, reaching a peak in the early waterfall region, where successful decoding is likely but still requires a high number of iterations. Finally, at high SNR, early impossible decoding is very rare, and MSESC is almost always deactivated. The implementation of MSESC reported in [8] shows that the simple logic involved

in the on-the-fly computation of threshold values guarantees a very small power consumption overhead that is compensated by the reduced number of iterations. Compared to decoders implementing no ESCs or alternative solutions, MSESC shows energy saving ranging from 10 to 90 %.

7.4.2 Latency Reduction Techniques

The latency introduced by NoCs with respect to less flexible interconnection structures is unsustainable for most decoders, due to the strict throughput requirements of many communication standards. In fact the required throughput imposes an upper bound on the duration of the decoding process and, consequently, on delivery time of each message injected in the NoC: high message injection rates, traffic, and collisions often result in late messages. These, even in small percentages, can be disastrous for the decoder BER performance.

A common choice to avoid late messages is to stall the decoder between decoding phases, waiting for the delivery of all information. Unfortunately, a straightforward implementation of this approach severely limits the achievable throughput. A possible solution has been studied in [26, 32] for turbo codes and [5] for LDPC codes, where the reliability of the exchanged information is evaluated through threshold-based measures. In [32], reliable information is characterized by a small enough difference between a-priori and extrinsic information, while in [5] reliable LLRs are those with a large enough magnitude. In both cases, if the information is deemed reliable it is not exchanged anymore, reducing the traffic on the NoC and possibly the total delivery time. This technique can be particularly effective in presence of large networks with light traffic patterns, where up to 20 % throughput gains have been observed.

Stalling the decoder, however, is often unfeasible, especially in decoders with heavy traffic loads, in which the transmission times are already long and additional latency cannot be sustained. A possible approach in these situations can be the artificial reduction of the rate of packet injection r in the NoC by setting the frequency of the network at a multiple of that of the processing elements, that is reducing the value of r. While effective, this solution is extremely expensive in terms of power consumption. A set of alternative methods have been studied in [9], aimed at the reduction and optimization of NoC traffic. The Hard Importance (HI) method is very similar to the threshold-based reliability measures devised in [32] and [5], while the Soft Importance (SI) allows to discard low-importance packets in case they collide once they have been injected in the NoC. Both techniques offer significant advantages in heavy traffic conditions, reducing the percentage of late messages and the NoC switching activity. The remaining late messages, however, still cause severe performance degradation in the majority of cases. Traffic optimization is based on the fact that an estimate of the number of clock cycles available for message delivery can be sent together with the information and can be updated in the NoC. This field is used as a priority indicator: most urgent messages

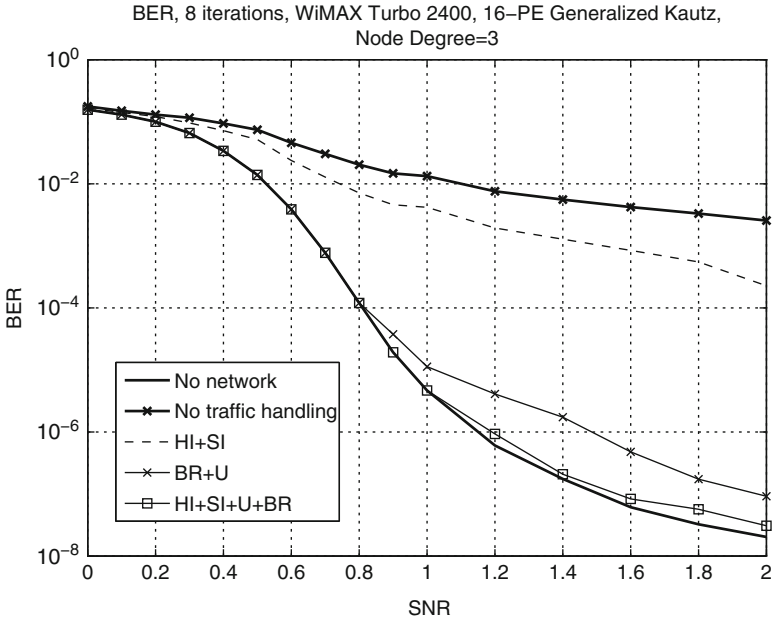

Fig. 7.3 BER performance of WiMAX code on NoC with combinations of traffic reduction and optimization

are favored in case of collisions. Exploiting this Urgency (U) priority field, NoC buffers can be changed from FIFOs to urgency-ordered buffers (Buffer Reordering, BR) so that urgent messages are always the first to be served in routers. These two traffic optimization methods are extremely effective in guaranteeing on-time message delivery. Figure 7.3 plots the BER performance of a WiMAX turbo code mapped on a 16 PE Kautz NoC, under the influence of different traffic handling techniques. The "No network" represents the achievable performance, with 0 % late updates, while "No traffic handling" has been obtained without the application of any of the described methods. It can be noticed how traffic reduction alone (HI+SI) is not able to bring the BER to acceptable levels. On the contrary, very good results are shown by traffic optimization (BR+U) and even more performance improvement is observed when all four techniques work together. Implementation of the four techniques on the decoder presented in [7] results in a 13 % area overhead and a 15 % power consumption reduction, thanks to the lower NoC frequency [9].

7.5 Dynamic Reconfiguration

Extensive research has been conducted in the last few years on flexible multi-mode and multi-standard decoding architectures, and large efforts have been spent to limit the impact of flexibility on achievable throughput and dissipated energy. However,

the issue of dynamic reconfiguration has been often neglected in this context. Surprisingly enough, most of flexible decoders available in the open literature [2, 11, 25, 27, 27, 31, 35, 42, 43] efficiently work on a wide range of codes, but do not support the run-time switch from one code to another.

Very few contributions have considered the fundamental requirement of rapidly reconfiguring the architecture for a new decoding task and the related overheads in terms of area and energy [7,20,21,34]. Change of decoding mode, standard, or code parameters requires not only hardware support, but also memory initialization and specific controls; moreover, since in many standards a code switch can be issued as early as one data frame ahead, reconfiguration time is also a major need.

In this section, a detailed analysis of the reconfiguration issue is carried out and the different kinds of reconfiguration are described, together with alternative solutions and trade-offs.

7.5.1 The Reconfiguration Task

We can distinguish between three levels of dynamic reconfiguration, associated to growing levels of flexibility and complexity:

1. intra-standard reconfiguration,
2. inter-standard reconfiguration,
3. reconfiguration to a new code.

The first case does not refer to a change of standard, but simply to a change of communication mode: the decoder has to switch between two codes of the same family, belonging to the same standard. New and old codes can have different length and code rate, but they typically share many other elements, such as, the decoding algorithm and the iteration control. The similarity between the two codes can be exploited to simplify the configuration process and reduce the amount of data to be updated. In the second case, the switch is between two codes with potentially large differences with respect to each other: in addition to code length and rate, other important features may be changed, such as, the decoding algorithm or the permutation law. The reconfiguration process tends to be more complex and it involves a larger amount of data. The last case is of interest for fully flexible or future-proof decoders, which are able to dynamically adapt to any code in the considered families (e.g., turbo and LDPC codes). This flexibility is also extended to codes not yet known at the time of decoder design, provided that the length is within limits that depend on the size of the internal memories allocated to store received frames.

The actual complexity of the dynamic reconfiguration and its impact on the overall decoder in terms of additional occupied area and dissipated energy depend on the amount of internal data to be updated when switching to a new code and on the available time to complete this operation.

In modern wireless communication systems, a code switch can be issued as early as one data block ahead and the decoder must be reconfigured for the new data block while it is still running on the previous one. Therefore, a time efficient reconfiguration technique is mandatory. The worst-case reconfiguration latency can be simply expressed as

$$L_{rec} = \frac{N_f R}{T} \tag{7.2}$$

where N_f is the bit length of the current block, R is the code rate, and T is the specified throughput. It can be easily seen from (7.2) that L_{rec} tends to become very short in the case of short block, low code rate, and high throughput. Considering current wireless communication standards, it has a typical value of a few µs. As an example, in the 3GPP LTE standard, 188 different information block sizes are specified, ranging from 40 to 6,144, the lowest code rate is $1/3$ and the throughput can be as high as 450 Mbit/s for the down-link; the peak throughput is specified for the largest block, while it scales for shorter blocks. Using the largest block, (7.2) gives a latency of 13.6 µs. The requirement on L_{rec} is expected to become even more severe in LTE Advanced, which specifies a throughput as high as 1 Gbit/s.

Estimating the amount of data to be updated at the reconfiguration of the decoder is rather difficult, as it is heavily dependent on the specific adopted architecture and on the kind of addressed reconfiguration.

An FPGA based decoder is intrinsically reconfigurable, as the running decoder can be stopped at any time and a new design can be loaded into the programmable device to support a completely different code. Current FPGA technology allows for run-time partial configuration, which means that the decoder structure can be modified or extended, while it is serving the current decoding task. These dynamic reconfiguration techniques offer a large potential flexibility, ranging from the simple switch between two similar codes, decoded by architectures that share a high percentage of hardware resources, up to the load of a complete new decoder, running a different decoding algorithm. However, FPGA configuration is still a slow and energy inefficient process, which involves the uploading of tens of Mbytes, therefore it is hardly acceptable for flexible decoding applications, with severe speed and power consumption constraints.

Large flexibility is also offered by ASIP-based decoders, which are basically customized processors with specialized datapath and instruction set. In this kind of decoder, the dynamic reconfiguration can be viewed as a context switch, where instruction memory and internal registers are dynamically loaded with new contents when the switch to a new code is requested. In [21], the amount of configuration data for a multi-ASIP decoder is evaluated as equal to about 1 kbits, due to both processor instructions (around $2/3$ of the total) and parameters ($1/3$). Proper hardware resources are necessary to support the context switch (e.g., data busses, shadow memory, and registers).

As a further alternative, which was explored in [21] and [7], the single processing element can be implemented in the form of a parameterized dedicated architecture,

which only needs a few configuration bits, as there is no instruction memory . If the decoder flexibility is limited to a small number of standards and both interleaving laws and parity check matrices are algorithmically generated by means of simple parameterized hardware components, then the whole configuration becomes very easy, fast, and involves a reduced amount of data.

Finally, the decoder can be based on an application specific NoC, with parameterized processing elements and routing elements [7]. This fully flexible solution supports the third level of configuration mentioned above and allows for dynamic switching between any couple of codes, including codes with a structure of either interleaver or parity check matrix that cannot be generated algorithmically, or is not known at the design time. In this case, the amount of data that need to be updated is significantly larger, because it includes the management of the routing. Basically, for each generated LLR, its destination node in the NoC must be available to properly forward the LLR. As a consequence, when the decoder is configured for a size N code, the number of bits to be uploaded tends to grow as $O(N\lceil log_2 N\rceil)$, where N is the codes size and $\lceil log_2 N\rceil$ bits are used to represent each location address. For example, to support the LTE codes, more than 150,000 reconfiguration bits are necessary to update the NoC routing.

From these rough evaluations, one can see that the reconfiguration process implies quite a large throughput of data moved to the decoder. Therefore, an efficient organization of the configuration process is of utmost importance to confine the related complexity and energy overheads.

7.5.2 Reconfiguration of ASIP Based Decoders

The problem of dynamically reconfiguring an ASIP based turbo decoder and the development of an efficient hardware implementation is addressed in [20, 21, 34]. It is assumed that each received block is associated to a specific configuration and that the loading of the configuration for a new block is performed during the processing of the current block.

An already available multi-mode and multi-standard turbo decoder [33] is initially considered and a set of modifications are applied in order to enable the dynamic switch between different codes. The original decoder architecture is organized around two sets of ASIPs interconnected via a Butterfly Network on Chip, where each set corresponds to a component decoder. Each ASIP unit includes ten pipeline stages and it is associated to several memories, used to store input channel LLR values, extrinsic information, state metric, instructions, and configuration data. The required interleaving/deinterleaving addresses are generated algorithmically and make use of a set of parameters which depend on the specific code to be supported and therefore are part of the configuration. From the complete list of configuration data, the information to be updated at each code switch is classified into four categories:

1. component decoder dependent parameters,
2. identical parameters for all ASIPs,
3. ASIP dependent parameters,
4. parameters required only by the last ASIP of a component decoder (for tail bits).

A low latency configuration process is then developed exploiting both multicast and broadcast mechanisms. Two multicast operations are used to update the first class of parameters, one multicast operation is necessary for parameters shared among all ASIPs, and multiple unicast transfers are exploited to upload the ASIP dependent parameters. Moreover, the ASIP configuration load has been significantly reduced by adopting a unified program, which works for all considered codes, including double-binary turbo codes. Finally, the internal configuration memory has been extended to store multiple configurations at the same time: this allows to save time in the switch between frequently used codes.

The decoder has been enriched with a dynamic reconfiguration infrastructure that supports unicast, multicast, and broadcast transfers. The adopted solution consists of a master-slave 26 bit bus-based structure. A master unit initially receives the configuration data, which are then moved to one or multiple slave units, based on the required type of transfer. A dedicated unit allows to select at run-time the target destination ASIPs. Finally, configuration data are moved from the slave units to their final destinations.

This configuration infrastructure and the related protocol controller have been implemented using both FPGA and ASIC technologies. The worst case latency is lower than 10 µs in the case of FPGA implementation, while 1 µs is reached with a 65 nm ASIC technology, allowing for a 500 MHz clock frequency. The corresponding area overhead is around 2 % of the decoder.

The described solution allows to handle the configuration process in two different ways. In the first approach, the configuration data is generated off-line for all possible codes and stored in a global memory. At every code switch, a configuration manager simply reads the new configuration data from the global memory and send it to the decoder, by means of the dedicated infrastructure. This static solution is functionally simple, but requires a large global memory to support a large set of codes and it is not compatible with any extension. The second approach is dynamic and allows for the run-time generation of new configurations. The single bus reconfiguration architecture, applied to a multi-ASIP decoder, is shown in Fig. 7.4a, where a reconfiguration interface (RI) connects each the shared bus to each ASIP and the reconfiguration data can be provided by either a memory or a reconfiguration manager.

Although the described reconfiguration technique is tailored around a specific ASIP based decoder that only supports turbo codes in WiMAX and LTE standards, it can be easily extended to include more codes. However, since the reconfiguration infrastructure uses a shared bus, this solution is expected to become less efficient if the number of ASIPs or PEs to be configured in the decoder is significantly larger than shown in [21].

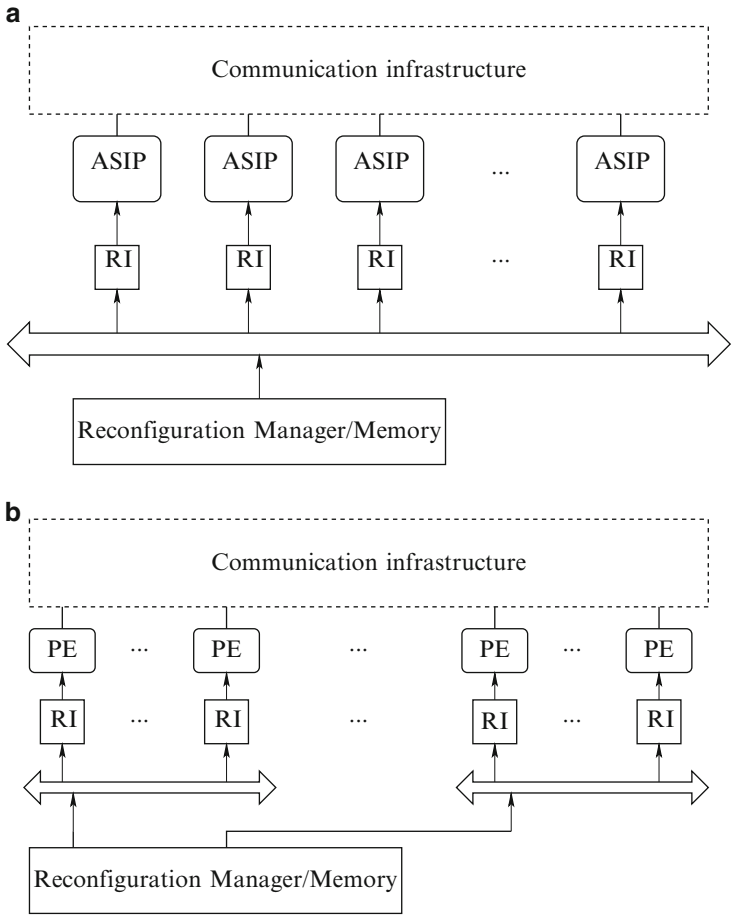

Fig. 7.4 Reconfiguration architectures: (**a**) single bus architecture, (**b**) multiple bus architecture

7.5.3 Reconfiguration of NoC Based Decoders

For the decoding architecture described in Sect. 7.3.2, the reconfiguration task affects the content of the location memory, which stores the destination addresses for each processed message, and a few parameters, such as check node degrees and the SISO window size. We assume that the whole set of reconfiguration data are saved in a distributed configuration memory (CM), allocated in each PE of the decoder. As reconfiguration must occur within the decoding of the previous block, the NoC interconnects cannot be exploited to update the CM in each processing element. Instead, N_b buses are dedicated to the configuration task, and each bus serves P/N_b PEs. The use of multiple buses allows to increase the bandwidth and to handle larger amount of reconfiguration data. This high level architecture is shown

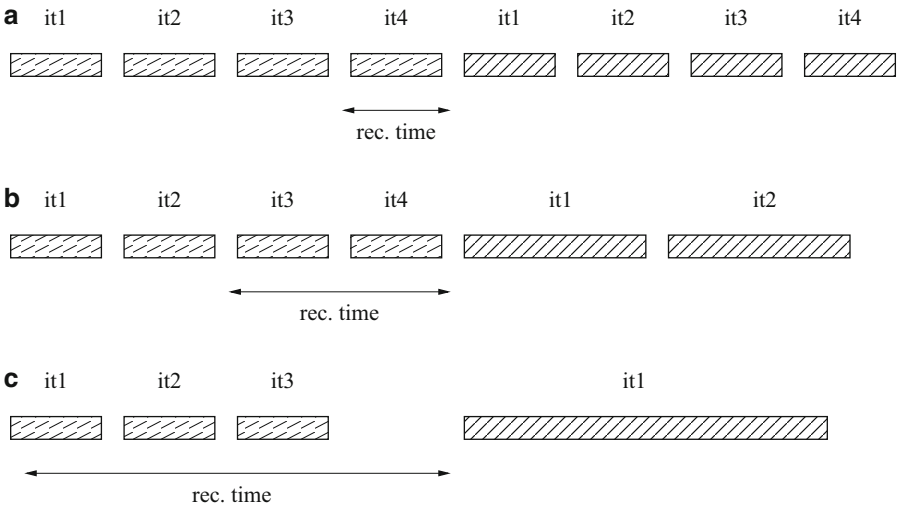

Fig. 7.5 Examples of reconfiguration. Reconfiguration can be overlapped with a single decoding iteration (**a**), with two iterations (**b**), or with a number of iterations (**c**)

in Fig. 7.4b, where multiple buses (N_b) concurrently update the decoding PEs and reconfiguration data are taken from a memory or from a manager, as in part (a) of the figure.

The CM is managed as a circular buffer, where two sets of configuration data can be present at the same time: the one necessary for the decoding of the current block, and a new one, to be used when the decoder will switch to a new code. Proper read and write pointers allow to separate old and new configurations. Limiting the size B of the CM is of interest, because it impacts on area and energy consumption. To this purpose, multiple strategies can be adopted. The first idea is to partially overlap the configuration process with the decoding of one or more blocks (N_o) of the current code. A second option is to overlap an additional part of the configuration process with the first decoding iteration of the newly loaded code. A final option is skipping one or multiple iterations (N_{sk}) while decoding the last received block: the saved time can be used to complete the loading of CM before starting the decoding of the new block. Differently from the other solutions, the last one has an impact on the error correction performance, because of the skipped iterations.

A few examples of reconfiguration are shown in Fig. 7.5, where distinct patterns are used to indicate blocks decoded with different codes. In part (a) of the Figure, the switch to a new code is considered, after four decoding iterations with the previous code: as new and old codes have similar size, the reconfiguration time is close to the time length of a single iteration. Therefore, in this case, $N_o = 1$ and $N_{sk} = 0$. In part (b), the destination code is longer and reconfiguration time needs to be extended to cover more than one iteration. Finally, in part (c), the destination code is much longer than the current code, therefore, one decoding iteration is skipped ($N_{sk} = 1$).

In general, given a set of standards and turbo or LDPC codes to be supported, the reconfigurable decoder can be designed by selecting a proper combination of the four mentioned parameters (the number of busses N_b, the buffer size B, the number of overlapping blocks N_o, and the number of skipped iterations N_{sk}), with the purpose of minimizing the overall complexity and matching the required latency at the same time, with minimum impact on the error correction performance.

Intuitively, the global latency decreases with larger N_b, because more PEs can be reconfigured at the same time. Moreover, increasing B reduces the need for skipped iterations and overlapping blocks. When the destination code in the reconfiguration process is larger than the current one, the uploading of CM may require either larger N_o or larger N_{sk}.

A complete analysis of both intra- and inter-standard reconfiguration is presented in [7], where several codes from a large variety of standards (WiFi, DVB-RCS, WiMAX, CMMB, DTMB, 3GPP-LTE, and HPAV) are considered. Starting from the architecture proposed in [6], an extended NoC based fully flexible decoder is obtained with shared memory and interconnect resources. Moreover configuration busses, memories, and control logic have been included to support dynamic switch between different codes. The decoder has been synthetized with a 90 nm CMOS standard cell technology, with 22 PEs running at 200 and 170 MHz when configured to support LDPC and turbo decoding respectively; 300 MHz is the target clock frequency for the NoC. The whole set of turbo and LDPC codes included in the mentioned standards are fully supported and achievable throughput meets the specifications until ten iterations for LDPC codes and eight for turbo codes. Post place & route estimated area is 3.42 mm^2 and peak dissipated power is 120 mW.

The reconfiguration process has been tested on every possible couple of codes in the seven considered standards. From this large set of experiments, the following conclusions can be drawn.

- The intra-standard reconfiguration is possible with no need for skipped iterations ($N_{sk} = 0$) in all standards, except LTE, which requires $N_{sk} > 0$ in 6.8 % of possible switches between the 188 specified codes.
- Inter-standard reconfiguration is also possible with no skipped iterations when the new code is not belonging to LTE, CMMB, or DTMB standards.
- Inter-standard reconfigurations towards LTE, CMMB, or DTMB requires $N_{sk} > 0$ in a percentage of possible cases which ranges between 5 and 77 %. Moreover, for all considered cases, $N_{sk} \leq 3$.

The impact of the reconfiguration process and particularly of the skipped iterations on the decoder performance has been assessed by means of BER simulations, taking into account the probability that a reconfiguration is required as a consequence of channel fading. Based on different fading scenarios, the channel conditions may change at a rate f_{ch} between 10 and 150 Hz [7]. The actual reconfiguration probability can be calculated as

$$P_{rec} = \frac{f_{ch} RN}{T} \tag{7.3}$$

where the ratio T/N is the number of coded frames received per unit time. P_{rec} is shown to range between 0.25 and 0.3 % in presence of a fast moving receiver, while it remains under 0.15 % in the other cases. The effect of $N_{sk} \leq 3$ on BER performance in such conditions is negligible and lower than 0.05 dB for all considered codes and standards.

Acknowledgements This work has been partially supported by the Newcom# project.

References

1. Al-Khayat R, Baghdadi A, Jezequel M (2012) Architecture efficiency of application-specific processors: a 170Mbit/s 0.644mm2 multi-standard turbo decoder. In: 2012 International symposium on system on chip (SoC), pp 1–7. doi:10.1109/ISSoC.2012.6376368
2. Alles M, Vogt T, Wehn N (2008) FlexiChaP: a reconfigurable ASIP for convolutional, turbo, and LDPC code decoding. In: 2008 5th International symposium on turbo codes and related topics, pp 84 –89
3. Cai Z, Hao J, Sethakaset U (2008) Efficient early stopping method for LDPC decoding based on check-node messages. In: Proceedings of Asilomar conference on signals, systems and computers, pp 466–469. doi:10.1109/ACSSC.2008.5074448
4. Chen X, Lin S, Akella V (2010) QSN-a simple circular-shift network for reconfigurable quasi-cyclic LDPC decoders. IEEE Trans Circuits Syst II 57(10):782–786
5. Condo C, Masera G (2011) A flexible NoC-based LDPC code decoder implementation and bandwidth reduction methods. In: 2011 Conference on design and architectures for signal and image processing (DASIP), pp 1 –8
6. Condo C, Martina M, Masera G (2012) A network-on-chip-based turbo/ldpc decoder architecture. In: Design, automation and test in Europe conference and exhibition, pp 1525–1530
7. Condo C, Martina M, Masera G (2013) VLSI implementation of a multi-mode turbo/LDPC decoder architecture. IEEE Trans Circuits Syst I 60(6):1441–1454
8. Condo C, Baghdadi A, Masera G (2014) Energy-efficient multi-standard early stopping criterion for low-density-parity-check iterative decoding, Communications, IET. 8(12):2171, 2180. doi:10.1049/iet-com.2013.0869
9. Condo C, Baghdadi A, Masera G (to appear) Reducing the dissipated energy in multi-standard turbo and LDPC decoders. Circuits Syst Signal Process
10. Duato J, Yalamanchili S, Ni L (2003) Interconnection networks: an engineering approach. Morgan Kaufmann, Los Altos
11. Gentile G, Rovini M, Fanucci L (2010) A multi-standard flexible turbo/LDPC decoder via ASIC design. In: International symposium on turbo codes & iterative information processing, pp 294–298
12. Guerrier P, Greiner A (2000) A generic architecture for on-chip packet-switched interconnections. In: Design, automation and test in Europe conference and exhibition, pp 250–256
13. Hagenauer J, Offer E, Papke L (1996) Iterative decoding of binary block and convolutional codes. IEEE Trans Inf Theory 42(2):429–445. doi:10.1109/18.485714
14. Hu W-H, Chen C-Y, Bahn JH, Bagherzadeh N (2012) Parallel low-density parity check decoding on a network-on-chip-based multiprocessor platform, Computers & Digital Techniques, IET. 6(2): 86, 94. doi:10.1049/iet-cdt.2010.0177
15. IEEE Std 802.16, part 16: air interface for fixed broadband wireless access systems (2004)
16. Kim B, Yoo I, Park IC (2013) Low-complexity parallel QPP interleaver based on permutation patterns. IEEE Trans Circuits Syst II 60(3):162–166

17. Kim JH, Park IC (2008) A 50Mbps double-binary turbo decoder for WiMAX based on bit-level extrinsic information exchange. In: IEEE Asian solid-state circuits conference, pp 305–308. doi:10.1109/ASSCC.2008.4708788
18. Kim JH, Park IC (2009) Bit-level extrinsic information exchange method for double-binary turbo codes. IEEE Trans Circuits Syst II 56(1):81–85
19. Kim JH, Park IC (2009) A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE. In: IEEE custom integrated circuits conference, 2009 (CICC '09), pp 487–490. doi:10.1109/CICC.2009.5280790
20. Lapotre V, Murugappa P, Gogniat G, Baghdadi A, Diguet JP, Bazin JN, Hubner M (2013) A reconfigurable multi-standard asip-based turbo decoder for an efficient dynamic reconfiguration in a multi-asip context. In: 2013 IEEE computer society annual symposium on VLSI (ISVLSI), pp 40–45. doi:10.1109/ISVLSI.2013.6654620
21. Lapotre V, Murugappa P, Gogniat G, Baghdadi A, Hubner M, Diguet JP (2013) Stopping-free dynamic configuration of a multi-asip turbo decoder. In: 2013 Euromicro conference on digital system design (DSD), pp 155–162. doi:10.1109/DSD.2013.24
22. Li J, You XH, Li J (2006) Early stopping for LDPC decoding: convergence of mean magnitude (CMM). IEEE Commun Lett 10(9):667–669. doi:10.1109/LCOMM.2006.1714539
23. Mansour M, Shanbhag N (2002) Turbo decoder architectures for low-density parity-check codes. In: IEEE Global Telecommunications Conference, 2002 (GLOBECOM '02), vol 2, pp 1383–1388. doi:10.1109/GLOCOM.2002.1188425
24. Martina M (2010) Turbo NOC: network on chip based turbo decoder architectures. Downloadable at www.vlsilab.polito.it/~martina
25. Martina M, Masera G (2010) Turbo NOC: a framework for the design of network-on-chip-based turbo decoder architectures. IEEE Trans Circuits Syst I 57(10):2776–2789
26. Martina M, Masera G (2013) Improving network-on-chip-based turbo decoder architectures. J Signal Process Syst 73(1):83–100
27. Martina M, Masera G, Moussa H, Baghdadi A (2011) On chip interconnects for multiprocessor turbo decoding architectures. Elsevier Microprocess Microsyst 35(2):167–181
28. Moher M (1993) Decoding via cross-entropy minimization. In: IEEE global telecommunications conference, 1993 (GLOBECOM '93), including a communications theory mini-conference. Technical program conference record, IEEE in Houston, vol 2, pp 809–813. doi:10.1109/GLOCOM.1993.318192
29. Mohsenin T, Shirani-Mehr H, Baas B (2011) Low power LDPC decoder with efficient stopping scheme for undecodable blocks. In: Proceedings of IEEE international symposium on circuits and systems
30. Moussa H, Muller O, Baghdadi A, Jezequel M (2007) Butterfly and Benes-based on-chip communication networks for multiprocessor turbo decoding. In: Design, automation and test in Europe conference and exhibition, pp 654–659
31. Moussa H, Baghdadi A, Jezequel M (2008) Binary de Bruijn on-chip network for a flexible multiprocessor LDPC decoder. In: Design automation conference, pp 429–434
32. Muller O, Baghdadi A, Jezequel M (2006) Bandwidth reduction of extrinsic information exchange in turbo decoding. Electron Lett 42(19):1104–1105. doi:10.1049/el:20062209
33. Murugappa P, Al-Khayat R, Baghdadi A, Jezequel M (2011) A flexible high throughput multi-ASIP architecture for LDPC and turbo decoding. In: Design, automation and test in Europe conference and exhibition, pp 1–6
34. Murugappa P, Lapotre V, Baghdadi A, Jezequel M (2013) Rapid design and prototyping of a reconfigurable decoder architecture for qc-ldpc codes. In: 2013 International symposium on rapid system prototyping (RSP), pp 87–93. doi:10.1109/RSP.2013.6683963
35. Naessens F, Bougard B, Bressinck S, Hollevoet L, Raghavan P, Van der Perre L, Catthoor F (2008) A unified instruction set programmable architecture for multi-standard advanced forward error correction. In: Proceedings of IEEE workshop on signal processing systems, pp 31–36. doi:10.1109/SIPS.2008.4671733

36. Neeb C, Thul MJ, Wehn N (2005) Network-on-chip-centric approach to interleaving in high throughput channel decoders. In: IEEE international symposium on circuits and systems, pp 1766–1769
37. Park SM, Kwak J, Lee K (2008) Extrinsic information memory reduced architecture for non-binary turbo decoder implementation. In: IEEE vehicular technology conference, 2008 (VTC Spring 2008), pp 539–543
38. Polydoros A (2008) Algorithmic aspects of radio flexibility. In: IEEE international symposium on personal, indoor and mobile communications, pp 1–5
39. Sham CW, Chen X, Lau F, Zhao Y, Tam W (2013) A 2.0 Gb/s throughput decoder for QC-LDPC convolutional codes. IEEE Trans Circuits Syst I Regul Pap 60(7):1857–1869. doi:10.1109/TCSI.2012.2230506
40. Singh A, Boutillon E, Masera G (2008) Bit-width optimization of extrinsic information in turbo decoder. In: 2008 5th International symposium on turbo codes and related topics, pp 134–138. doi:10.1109/TURBOCODING.2008.4658686
41. Studer C, Benkeser C, Belfanti S, Huang Q (2011) Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE. IEEE J Solid State Circuits 46(1):8–17
42. Sun Y, Cavallaro JR (2010) A flexible LDPC/Turbo decoder architecture. J Signal Process Syst 64(1):1–16
43. Vacca F, Moussa H, Baghdadi A, Masera G (2009) Flexible architectures for LDPC decoders based on network on chip paradigm. In: Euromicro conference on digital system design, pp 582–589
44. Vogt J, Ertel J, Finger A (2000) Reducing bit width of extrinsic memory in turbo decoder realisations. Electron Lett 36(20):1714–1716. doi:10.1049/el:20001177
45. 3gpp ts 36.211, evolved universal terrestrial radio access (e- utra): physical channels and modulation, version 8.4.0 (2008)