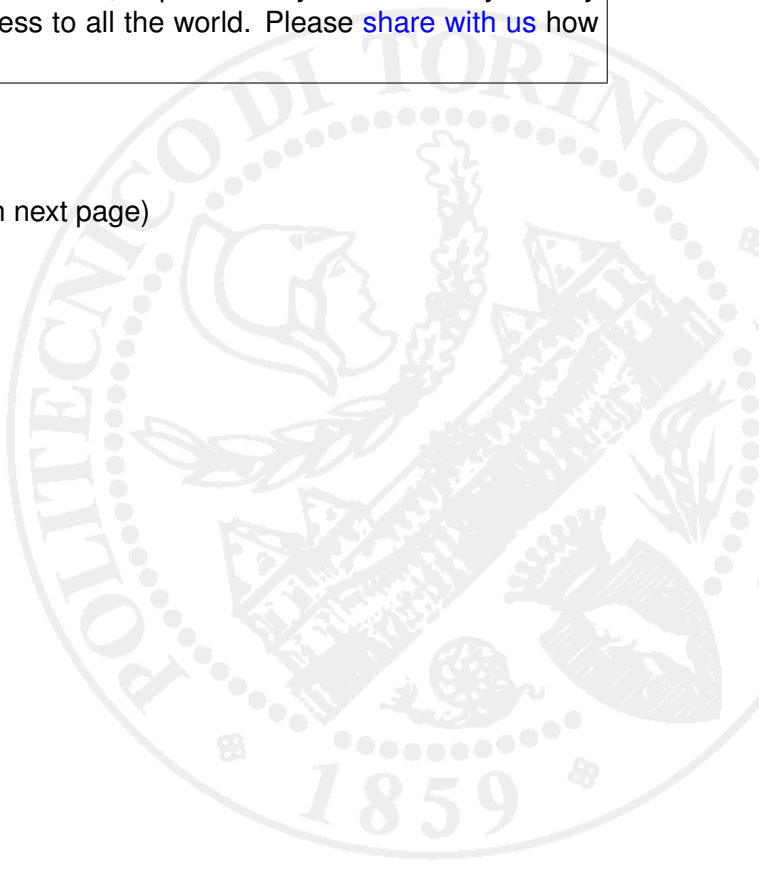

VLSI decoding architectures: flexibility, robustness and performance

Carlo Condo

Advisors: Guido Masera, Amer Baghdadi

February 18, 2014

Contents

List of Figures

List of Tables

Acronyms

AFPI	Average Failures Per Iteration
AP	All Precalculated
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processors
ASNoC	Application-Specific NoC
ASP	All local Shortest Paths
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BL	Bit-Level
BMU	Branch Metric Unit
BP	Belief Propagation
BR	Buffer Reordering
BTC	Block Turbo Code
BTS CU	Bit-To-Symbol Conversion Unit
CC	Convolutional Code
CMOS	Complementary Metal Oxide Semiconductor
CN	Check Node
CNMC	Check Node Mean Checksum
CNMM	Check Node Mean Magnitude
CM	Collision Management
CTC	Convolutional Turbo Code
DB	Duo-Binary or Double-Binary
DBTC	Duo-Binary Turbo Code

DCM	Delay Colliding Message
ECC	End of Current Configuration (Chapter 3.4) or Error Correcting Code (Chapter 4)
ECU	Extrinsic Computation Unit
EFC	End of Future Configuration
EMESC	Erased Massages Early Stopping Criterion
ESC	Early Stopping Criterion
FEC	Forward Error Correction
FER	Frame Error Rate
FIFO	First In First Out
FIT	Failures In Time
FL	FIFO Length
FT	FIFO length Traffic spreading
HI	Hard Importance
IC	Inner Code
IDD	Impossible Decoding Detection
IP	Intellectual Property
JANoCS	Joint Application and Network-on-Chip Simulator
LDPC	Low Density Parity Check
LLR	Logarithmic Likelihood Ratio
LSB	Least Significant Bit
LT	Luby Transform

MAP	Maximum-A-Posteriori
MEU	Minimum Extraction Unit
MP-SoC	Multi-Processor System-on-Chip
MSB	Most Significant Bit
MSESC	Multi-Standard Early Stopping Criterion
MTBF	Mean Time Between Failures
NB-LDPC	Non-Binary LDPC
NDT	Normalized Delivery Time
NED	NEtwork Description
NI	Network Interface
NMS	Normalized Min-Sum
NoC	Network on Chip
OC	Outer Code
PE	Processing Element
PP	Partially Precalculated
QC-LDPC	Quasi-Cyclic LDPC
QoS	Quality-of-Service
RE	Routing Element
RL	Route Local
RP	Read Pointer
RR	Round Robin
RSC	Recursive Systematic Convolutional
SISO	Soft-Input Soft-Output

SB	Single-Binary
SBTC	Single-Binary Turbo Code
SCC	Start of Current Configuration
SCM	Send Colliding Message
SCMS	Self-Corrected Min-Sum
SFC	Start of Future Configuration
SI	Soft Importance
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SRD	Symbol Reliability Difference
SSP	Single Shortest Path
STB CU	Symbol-To-Bit Conversion Unit
SW	Sliding Window
U	Urgency
UEP	Unequal Error Protection
VN	Variable Node
WP	Write Pointer

Chapter 1

Introduction

1.1 Motivation

The field of communications, both wireless and wired, has seen in the last decades an unprecedented development. Every day, we use tens of different devices to store, receive and exchange information. Regardless of their specific function, they are all characterized by a common component: the channel decoder. Forward Error Correction (FEC) or channel coding is a technique present in all kinds of communication standards: by encoding the transmitted information, it is possible to detect and correct errors introduced during the transmission due to a noisy channel. Diverse FEC schemes are currently used and both literature and industry continuously propose new codes and code combinations.

The design of an efficient channel decoder is a challenging task, since many variables come into play. First of all, although the quest for better performance, simple hardware and small overhead is continuous, decoders are still complex components that implement computationally intensive algorithms. In fact, they often account for a large part of the device Silicon footprint and power consumption, and they are a potential bottleneck in increasing the system throughput. Secondly, the most recent nanometric fabrication technologies are not able to guarantee degrees of reliability as high as before. Finally, the rate at which new standards are introduced and the wide number of codes and related parameters considered by each one of them require decoders to be adaptable and flexible. New design approaches and new solutions at architectural level are consequently necessary to face these continuously evolving needs.

Covolutional Turbo Codes (CTCs) [1] ()and Low-Density Parity-Check (LDPC) codes [2] yield very good error correction capabilities and rely on iterative decoding algorithms. They are considered in a large number of standards, both alone and concatenated with other codes, for a variety of applications: wireless communications (WiMAX [3], WiFi [4], 3GPP-LTE [5]), wired communications (HpAV [6]), broadcasting (DVB-S2 [7], CMMB [8], DTMB [9]) and space communications (CCSDS [10]). Every standard can foresee different code types, which in turn can support hundreds of different code parameter combinations: even a channel decoder targeting a single standard must be extremely flexible. Obviously, flexibility comes at cost in speed, area occupation and power consumption.

The design of a channel decoder can be a critical task, due to the complexity of the system and the wide number of constraints usually involved. In particular, technological constraints (e.g. nanotechnologies) and application bounds (standard specifications) result in problematic design of mainly four types of channel decoders:

- multi-standard decoders, flexible with respect to existing standards (legacy-proof) and to future standards (future-proof);

- low-power consumption decoders;
- reliable decoders, able to correct errors within the memories;
- high-performance decoders, with extremely high error correction capabilities.

Stemming from previous studies on flexible LDPC decoders, this thesis work has been mainly focused on the development of flexible turbo and LDPC decoder designs, and on the narrowing of the power, area and speed gap they might present with respect to dedicated solutions. Additional studies have been carried out within the field of increased code performance and of decoder resiliency to hardware errors.

1.2 Outline and contributions

In the above described context, several contributions have been proposed and will be presented in the rest of this manuscript which is organized as follows:

1. Chapter 2 gives a brief overview on turbo and LDPC codes. The theory base behind these codes is reviewed, and the most common decoding algorithms are described. The state of the art on current decoder implementations is finally presented.

2. Chapter 3 regroups several main contributions in the design and implementation of flexible channel decoders. The first part concerns the design of a Network-on-Chip (NoC) serving as an interconnection network for a partially parallel LDPC decoder. This study follows a former analysis that identified the best combination of design choices for an application specific NoC used in a turbo decoder. The NoC simulator used in [11] is modified and extended to LDPC codes as well, while additional topologies and design choices are taken in account. Joining the new results with those in [11], a best-fit NoC architecture is selected and a complete multi-standard turbo/LDPC decoder is designed and implemented. This decoder serves as the starting point for a study on the reconfiguration of NoC-based channel decoder. Every time the code is changed, the decoder must be reconfigured: this usually translates on the overwriting of one or more memories, with the new data brought by dedicated buses. A number of variables influence the duration of the reconfiguration process, starting from the involved codes down to decoder design choices. Their effect is evaluated in order to achieve a reconfiguration architecture capable to sustain as many codes as possible. The flexible decoder designed before is subsequently modified, including also the proposed reconfiguration technique: three implementations are presented, characterized by different decoder parallelism degrees, supported codes and achieved throughput. NoC-based decoders, while granting flexible connectivity among all their nodes, can be burdened with long transmission times. To meet with often high throughput requirements, it is not always possible to wait for the transmitted information to reach its destination: to evaluate the impact of the interconnection network on the performance of a decoder, and vice versa, a Joint Application and NoC Simulator (JANoCS) is proposed and presented. It is a generic tool for concurrent simulation of processing elements and interconnection for Multi-Processor Systems-on-Chip (MP-SoCs), and it is useful for design space exploration in cases where processing and communication need to be jointly optimized. JANoCS is then applied to the turbo/LDPC decoder case to analyze the impact of late information delivery: correct functionality can be ensured with a high NoC clock frequency, but to avoid the power consumption increase traffic

reduction and optimization methods are devised, tested and implemented.

The main contributions of the work presented in this chapter are:

- a NoC simulator used to obtain the achievable throughput and an estimation of the NoC area occupation, given the network details and the communication needs of the processing elements;
- a best-fit NoC design targeting both turbo and LDPC code decoding;
- a multi-standard partially parallel turbo/LDPC decoder based on the designed NoC;
- an analysis of the reconfiguration problem in NoC-based decoders;
- the design of a multi-standard reconfigurable turbo/LDPC decoder, with three different implementations targeting different sets of standards;
- the JANoCS simulator, for concurrent application-communication simulation of generic MP-SoCs;
- the identification and analysis of the late message delivery issue in partially parallel turbo/LDPC decoders, with the evaluation of its impact on the decoder performance;
- the design of four power reduction techniques that tackle the late message delivery issue and avoid power-hungry solutions: the proposed methods are implemented within the reconfigurable decoder;
- the validations of the designed solutions, carried out with appropriate tools on synthesized, placed and routed circuits;

The obtained results have been presented at, published in or submitted to

- DASIP'11 conference [12]
- DATE'12 conference [13]
- IEEE Transactions on Circuits and Systems I [14]
- DSD'13 conference [15]
- Springer Circuits, Systems & Signal Processing [16]

3. In Chapter 4 a study on the early stopping of iterations for LDPC decoders is presented. Iterative decoders typically fix a maximum number of available iterations for the decoding of a frame, but a codeword might be corrected in less iterations, or the maximum number might not be enough. For this reason, many decoders employ Early Stopping Criteria (ESCs) that try to identify situations in which additional iterations are useless. In this chapter we propose a new

criterion that observes the evolution of simple metrics and via on-the-fly threshold computation outperforms the state of the art in terms of both iteration and energy saving.

The main contributions of the work presented in this chapter are:

- an early stopping criteria for LDPC decoding that is adaptive to both code parameters and channel conditions, and identifies efficiently both correct codewords and impossible decoding;
- the hardware implementation of the proposed criterion and evaluation of its impact on performance and energy consumption under realistic channel conditions;

The obtained results have been presented at, published in or submitted to

- IET Communications [17]

4. Chapter 5 portrays a study on the resilience of LDPC decoders under the effect of memory errors. Due to the soft nature of the stored values, LDPC decoders will be affected differently according to the meaning of the wrong bits: ad-hoc error protection techniques can consequently be applied to different bits according to their significance. The proposed Unequal Error Protection (UEP) allows to recover from errors on most significant bits, limits the impact of errors on least significant bits, and ensures the decoder performance also in presence of large numbers of errors.

   The main contributions of the work presented in this chapter are:

   - an extensive analysis of the sensitivity to errors of LDPC decoders, depending on decoding algorithm, quantization, iterations, code size and rate;
   - the proposal of the UEP method and the evaluation of its effectiveness;
   - the design and implementation of the hardware structure of proposed method in a case of study;

   The obtained results are going to be submitted to IEEE Transactions on Computers.

5. In Chapter 6 the serial concatenation of LDPC and turbo codes is presented, set within the framework of deep space communications. These are characterized by critical spacecraft-to-Earth transmissions: the long distances and low power available require very good FEC schemes. The proposed FEC joins the performance of turbo codes at low Signal-to-Noise Ratio (SNR) with that of LDPC codes at high SNR, and outperforms both current deep-space FEC schemes and concatenation-based FECs. A decoder for the proposed FEC scheme is designed and implemented.

The main contributions of the work presented in this chapter are:

- a novel FEC scheme based on the serial concatenation of LDPC and turbo codes, and its comparison against current solutions;
- a low complexity, low power decoder design targeting the proposed FEC: it relies on a shared datapaths and on a smart memory sharing structure that avoids the need for memory interleaving when changing decoding mode;

The obtained results have been presented at, published in or submitted to

- SPACOMM'13 conference [18]
- IEEE Transactions on Aerospace and Electronic Systems [19]

6. Finally, Chapter 7 draws the conclusions. Additional contributions by the author are briefly addressed [20–22], together with ongoing work and future perspectives.

Chapter 2

LDPC and turbo code decoding

2.1 LDPC and turbo codes

Channel coding is a technique applied to most communication systems: the encoding of transmitted information allows, at reception, to detect and correct errors introduced by a noisy channel. In this section we introduce two powerful types of codes, employed in a variety of standards targeting different applications: turbo codes and Low-Density Parity-Check (LDPC) codes.

2.1.1 Turbo codes

Turbo codes have been proposed for the first time by Claude Berrou in [1] in the early Nineties: they are obtained from the parallel concatenation of two Convolutional Codes (CC), as shown in Fig. 2.1. Information bits are fed to both encoders, but not in the same order, since an interleaver Π is placed before the second convolutional encoder.

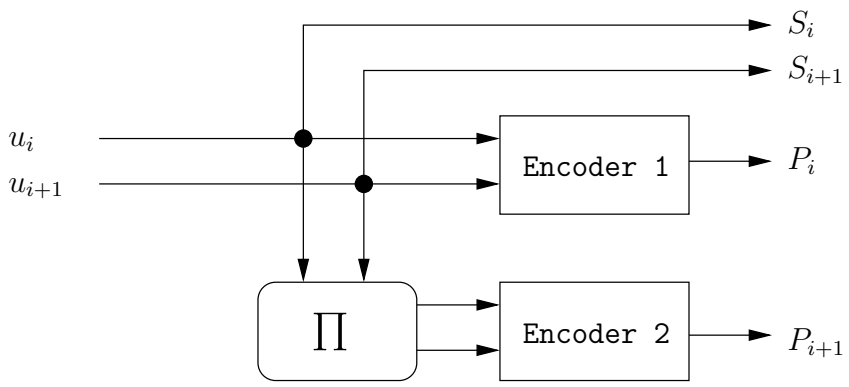

Figure 2.1: Turbo code encoder

The ratio between the number of information bits fed to the encoder and the number of encoder output bits determines the code rate r. The number of concurrent bits injected in the turbo encoder, i.e. the size of input symbols, determines if the turbo code is Single Binary (SB, one-bit symbols) or Duo-Binary (DB, two-bit symbols). While turbo encoders can work with infinite streams of bits, for practical purposes standards work with finite frame sizes, that effectively consider turbo codes as block codes. The number of input symbols is identified by K, while the frame is constituted of N symbols.

As the first practical codes with performance close to the Shannon limit, turbo codes have immediately known widespread study and usage. Most of the standards that use turbo codes rely on two Recursive Systematic Convlutional (RSC) encoders. This means that the information bits are

explicitly present in the encoded sequence (S_i and S_{i+1} in Fig. 2.1).

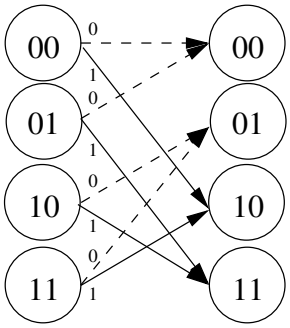

Figure 2.2: Turbo code trellis

According to the structure of the turbo encoder, a code can be pictured as a sequence of trellises, a representation exploited during the decoding process. In Fig. 2.2, each node of the trellis graph is a possible state of the encoder: the left ones are starting states, and the right ones are ending states. Edges are linked to input symbols: so, it is possible to move from the starting state to the ending state according to the received bits.

2.1.2 LDPC codes

LDPC codes are block error correcting codes explored for the first time by Robert Gallager in the early Sixties [2]. LDPC codes were shown impractical to use until the late nineties, when they were rediscovered by MacKay and Neal [23]. LDPC codes took hold thanks to the powerful computational hardware available and to the advent of approximated decoding algorithms.

The key characteristic of an LDPC code is its binary parity check $\mathbf{H}$ matrix, constituted of N columns and M rows: $\mathbf{H}$ specifies all the valid codewords of the code. These codewords are identified by the set of vectors x of N bits for which $\mathbf{H} \cdot x' = 0$, where $(\cdot)'$ is the transposition operator, meaning that the M parity check constraints defined by the matrix rows must be all verified. The $\mathbf{H}$ matrix is sparse: this means that, with respect to the dimensions of the matrix, very few non-zero entries are present. Their placement is the main task in the construction of the code, since the achievable error correcting performance is heavily influenced by the relative positioning of the ones among the zeros. While $\mathbf{H}$ is used in the decoding process, the information bits u are encoded as $x = u \cdot \mathbf{G}$, where $\mathbf{G}$ is a $K \times N$ generator matrix that can be obtained from $\mathbf{H}$ and $K = N - M$.

Many representations of the $\mathbf{H}$ matrix are possible, but the Tanner graph representation is commonly used in the decoding process. The Tanner graph identifies a bipartition of the $\mathbf{H}$ matrix, with the Variable Nodes (VNs) representing the columns and the Check Nodes (CNs) representing the

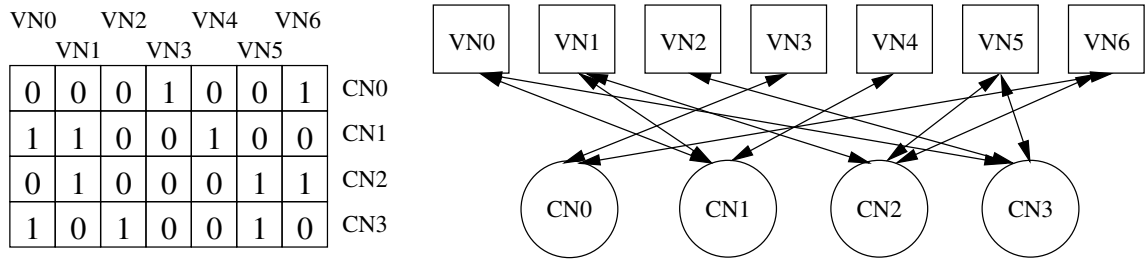

Figure 2.3: LDPC code **H** matrix and Tanner graph

rows. Consequently, VNs are associated to the N bits of the codeword, whereas CNs correspond to the M parity-check constraints. An edge between CN l and VN k is present if a non-zero entry is found in row l, column k of **H**. Fig. 2.3 shows an example **H** matrix, with seven VNs and four CNs, with the relative Tanner graph.

2.2 LDPC and turbo code decoding algorithms

Based on the aforementioned code representations, this section presents turbo and LDPC code decoding algorithms.

2.2.1 Turbo codes decoding algorithm

The turbo decoder is made of two constituent decoders, referred to as Soft-In-Soft-Out (SISO) or Maximum-A-Posteriori (MAP) decoders connected in an iterative loop by the means of the interleaver Π and the de-interleaver Π^{-1}. Each constituent decoder performs the so called BCJR algorithm [24] that starting from the intrinsic and *a priori* information produces the extrinsic information. Let k be a step in the trellis representation of the constituent CC, and u an uncoded symbol. Each constituent decoder computes $\lambda_k[u] = \sigma \cdot (\lambda_k^{apo}[u] - \lambda_k^{apr}[u] - \lambda_k[\mathbf{c}^u])$ where $\sigma \leq 1$ [25], $\lambda_k^{apo}[u]$ is the a-posteriori information, $\lambda_k^{apr}[u]$ is the *a priori* information and $\lambda_k[\mathbf{c}^u]$ is the systematic component of the intrinsic information. According to [24] a-posteriori information is computed as

$$\lambda_k^{apo}[u] = \overset{*}{\max_{e:u(e)=u}} \{b(e)\} - \overset{*}{\max_{e:u(e)=\tilde{u}}} \{b(e)\} \tag{2.1}$$

where $\tilde{u} \in \mathcal{U}$ is an uncoded symbol taken as a reference (usually $\tilde{u} = \mathbf{0}$) and $u \in \mathcal{U} \setminus \{\tilde{u}\}$ with $\mathcal{U}$ the set of uncoded symbols; e is a trellis transition and $u(e)$ is the corresponding uncoded symbol. Several exact and approximated expressions are available for the $\overset{*}{\max}\{x_i\}$ function [26]: for example, it can be implemented as $\max\{x_i\}$ followed by a correction term (Log-MAP), often stored in a small Look-Up-Table (LUT). The correction term, usually adopted when decoding binary codes, can be omitted with minor Bit-Error-Rate (BER) performance degradation (Max-Log-MAP). The term $b(e)$ in (2.1) is defined as:

$$b(e) = \alpha_{k-1}[s^S(e)] + \gamma_k[e] + \beta_k[s^E(e)] \tag{2.2}$$

$$\alpha_k[s] = \overset{*}{\max_{e:s^E(e)=s}} \left\{\alpha_{k-1}[s^S(e)] + \gamma_k[e]\right\} \tag{2.3}$$

$$\beta_k[s] = \overset{*}{\max_{e:s^S(e)=s}} \left\{\beta_{k+1}[s^E(e)] + \gamma_k[e]\right\} \tag{2.4}$$

$$\gamma_k[e] = \lambda_k^{apr}[u(e)] + \lambda_k[\mathbf{c}(e)] \tag{2.5}$$

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

For practical implementations, the size of a turbo codeword is fixed to limited values, but it can

be quite large. The BCJR algorithm requires a forward (2.3) and a backward (2.4) recursion over the codeword symbols to compute (2.1) and it can be unacceptable both in terms of latency and of complexity. For this reason, decoding is usually constrained to a portion of the codeword, called *window*, long enough not to cause performance degradation, that allows to meet the demanding throughput requirements of most standards: it is the Sliding Window (SW) technique [27], that has evolved in diverse methods to improve metric initialization and exchange. Bordering windows, in fact, exchange information on the edge symbols in order to continue the decoding.

2.2.2 LDPC codes decoding algorithm

The most common algorithm to decode LDPC codes is the *Belief Propagation* (BP) algorithm. The BP algorithm, also known as probability propagation and sum-product algorithm, uses iterative message passing between the nodes to update bit error probabilities. It was invented by Gallager in 1963, and reinvented by MacKay and Neal [28]. It is most commonly implemented in its logarithmic form: in this version, the metrics on which is taken the decision on bits are Logarithmic Likelihood Ratios (LLRs).

There are two main scheduling schemes for the BP: two-phase scheduling and layered scheduling [29]. The two-phase scheduling relies on the Tanner graph representation of $\mathbf{H}$: every iteration is composed of a CN phase, in which the parity check constraints defined by $\mathbf{H}$ are computed, and a subsequent VN phase, that takes in account the results of the CN phase to update bit LLRs. In the layered scheduling, instead, a single type of node is present, and parity-check constraints are grouped in layers so that there are no edges between nodes belonging to the same layer. Each layer is associated to a component code: they are decoded in sequence by propagating LLRs from one layer to the following one [29].

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

$$Q_{lk}[c] = \lambda_k^{old}[c] - R_{lk}^{old} \tag{2.6}$$

$$A_{lk} = \sum_{n \in N(l), n \neq k} \Psi(Q_{ln}[c]) \tag{2.7}$$

$$\delta_{lk} = \prod_{n \in N(l), n \neq k} \mathrm{sgn}(Q_{ln}[c]) \tag{2.8}$$

$$R_{lk}^{new} = -\delta_{lk} \cdot \Psi^{-1}(A_{lk}) \tag{2.9}$$

$$\lambda_k^{new}[c] = Q_{lk}[c] + R_{lk}^{new} \tag{2.10}$$

$\lambda_k^{old}[c]$ is the extrinsic information received from the previous layer and updated in (2.10) to be propagated to the succeeding layer. Term R_{lk}^{old}, pertaining to element (l,k) of $\mathbf{H}$ and initialized to 0, is used to compute (2.6); the same amount is then updated in (2.9), R_{lk}^{new} (the CN-to-VN message), and stored to be used again in the following iteration. In (2.7) and (2.8) $N(l)$ is the set of all bit indexes that are connected to parity constraint l. The decoded vector y is obtained by observing the sign of all $\lambda_k[c]$ at the end of each iteration, and the syndrome SYN is consequently obtained as

$$\text{SYN} = \sum_{l=1}^{M} \mathbf{H}_l \cdot y' \tag{2.11}$$

The BP algorithm involves some complex computations, in particular $\Psi(\cdot)$ in (2.7) and (2.9) that requires the calculation of a hyperbolic tangent. In [30] a possible approximation of the $\Psi(\cdot)$ function causing limited performance loss is presented:

$$R_{lk}^{new} \approx -\delta'_{lk} \cdot \min_{n \in N(l), n \neq k} \{|Q_{nk}|\}, \tag{2.12}$$

where $\delta'_{lk} = \sigma \cdot \delta_{lk}$ and $\sigma \leq 1$. It is usually referred to as Normalized-Min-Sum (NMS) approximation. The Self-Corrected-Min-Sum (SCMS) approximation [31], while based on the same concept, combines (2.12) with a dynamic correction. When $Q_{lk}^i[c]$ is computed at the i^{th} iteration, it is compared with $Q_{lk}^{i-1}[c]$. If their signs are different, $Q_{lk}^i[c]$ is substituted with zero for the current iteration, inducing a conservative behavior in presence of the uncertainty represented by a sign change.

A particularly interesting exploitation of the similarities between turbo and LDPC codes has been proposed in [32], that allows to use the same decoding algorithm for both types of code. Every row of $\mathbf{H}$ is seen as a turbo code with trellis length equal to the row weight: a direct link between turbo and LDPC codes is drawn, and turbo decoding algorithms like BCJR can be applied to LDPC codes with minor adjustments. The BCJR-based LDPC decoding relies on the fact that binary LDPC codes have a 2-state trellis: state metrics can consequently be expressed as differences $\Delta\alpha[c]$ and $\Delta\beta[c]$, reducing the quantization noise. Considering the Max-Log-MAP approximation [25], the CN-to-VN message update becomes:

$$\Phi(x, y) = \max(x, y) - \max(x + y, 0) \tag{2.13}$$

$$R_{lk}^{new} = \Phi(\Delta\alpha_k[c], \Delta\beta_k[c]) \tag{2.14}$$

$$\Delta\alpha_k = \Phi(\Delta\alpha_{k-1}[c], Q_{lk}[c]) \tag{2.15}$$

$$\Delta\beta_k = \Phi(\Delta\beta_{k+1}[c], Q_{lk}[c]) \tag{2.16}$$

where $\Delta\alpha[c]$ and $\Delta\beta[c]$ at the edge of the trellis are initialized as the minimum value of the dynamic range.

2.3 LDPC and turbo code decoders

This section gives an overview on current turbo and LDPC decoder solutions, focusing on both separate and joint implementations.

2.3.1 Turbo code decoders

Since turbo codes are a well-established technology used in a wide variety of applications, the literature is ripe with many practical implementations of turbo decoders: successful solutions for turbo decoding have been proposed in the past years [33,34]. The latest trends in the community have seen an increased interest towards differentiation in high-performance decoders, low power decoders and flexible decoders.

The parallel decoder presented in [35] targets the 3GPP-LTE standard, characterized by high throughput requirements. They are met through the usage of 8 radix-4 SISOs, a common enough choice for high-throughput designs. Communication among cores is enforced by means of shared memory banks. The proposed architecture manages to avoid memory access contentions, and allows to reduce the bottleneck usually associated with turbo code interleaving. The resulting decoder yields a small area, and moderate to high throughputs, with very low power consumption when targeting a 100 MB/s throughput. A similar approach is employed in [36], where even higher performance peaks are reached. The decoder is again tailored for the 3GPP-LTE standard. The described architecture relies on 64 processing cores, internally structured as radix-4 SISOs: the decoder yields 1.28 Gb/s at 400 MHz and 6 iterations. Obviously, the price for such outstanding results is payed in terms of complexity (8.3 mm^2 in 65 nm Complementary Metal Oxide Semiconductor, CMOS, technology) and power consumption (845 mW).

A more flexible architecture is devised in [37], where a decoder targeting both 3GPP-LTE and Mobile WiMAX standards is proposed. The different natures of the considered turbo codes (single-binary in 3GPP-LTE and duo-binary in Mobile WiMAX) is tackled by means of bit-to-symbol and symbol-to-bit conversions [38], addressed later in Section 3.2 and 3.4 of this manuscript, 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 [3] and QPP [5] respective interleavers. The resulting multi-standard decoder can reach up to 186 Mb/s with moderate complexity.

The sliding window technique [27] is one of the most widely employed in turbo decoders. The SISO proposed in [39] implements this technique in its standard form, with dummy state metric initialization and sliding step of one trellis. Deeply pipelined architectures have been proposed in order to reduce the memory requirements [40]. These have been improved thanks to the tailbiting technique: the

WiMAX decoder proposed in [41] avoids the cumbersome initialization by considering the turbo code as a circular structure, and allows for consistent throughput enhancement. The improvement of the dummy recursion and an ad-hoc contention-free memory architecture allow the decoder presented in [42] to yield low complexity and low power consumption. Moreover, smart initialization of the dummy recursion results in faster decoding algorithm convergence, and consequently fewer necessary iterations for correct decoding.

Flexibility and multi-standard support is achieved in [43] by means of partially parallel decoder based on Application Specific Instruction-set Processors (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.

2.3.2 LDPC code decoders

The literature on LDPC code decoders, much like in the turbo case, has shown a trend towards design specialization targeting a particular performance aspect.

The flexible LDPC decoder described in [44] is one 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.

An extremely low-power and high performance decoder is designed in [45], targeting the WiMAX standard. Low power consumption is obtained through very low working frequency: a very high throughput is achieved via a high internal parallelism, adaptable to the code thanks to the regular structure of the WiMAX codes. Implemented in 65 nm CMOS technology, at 40 MHz, more that 1 Gb/s throughput is obtained, with a total power consumption of 29.5 mW. A similar design approach is carried out in [46]: the decoder, however, targets high throughputs and low complexity. A higher frequency of 110 MHz is used to obtain 1 Gb/s throughput, consuming 115 mW and occupying half of the area of [45].

A decoder with extremely good error correction capabilities is shown in [47]. 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.

2.3.3 Turbo/LDPC decoders

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 [48] 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 [49] 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 [50–52] exploit commonalities between turbo and LDPC decoding to design a unified architecture for multiple standards. By interpreting LDPC codes as a series of turbo codes, as introduced in Section 2.2, 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.

Chapter 3

Flexible architectures for channel decoding

With the term flexibility regarding channel decoding it is intended the ability of a decoder to support different codes and types of codes, enabling its usage in a wide variety of situations. The degree of flexibility of a decoder can be limited to a single standard, or even multiple ones. Much research has been done in this sense after the great increase in number of standards, standard complexity and code variety witnessed during the last years. Next generation wireless standards such as DVB-S2 [7], IEEE 802.11n (WiFi) [53], IEEE 802.3an (10GBASE-T) [54]. 3GPP-LTE [5] and IEEE 802.16e (WiMAX) [3] often feature multiple codes (e.g. LDPC, Turbo) where each code comes with various code lengths and rates. The necessity for flexible channel decoder intellectual properties (IPs) is evident: however, the design of flexible decoders can be challenging due to the often unforgiving throughput requirements and narrow constraints of decoder latency, power and area. Following these challenges, this chapter presents extensive research in the field of flexible channel decoder architectures: the aim of this study is to produce a flexible, reconfigurable architecture and to tackle its power consumption issues. A study on NoCs and their application to a decoder design is carried out in Section 3.1 and 3.2, while Section 3.3 and 3.4 address the decoder reconfiguration and improve on the previous design. An ad-hoc simulation tool is presented in Section 3.5, and used in Section 3.6 to devise decoder power reduction techniques.

3.1 Topologies for NoC-based channel decoding

Even if the implementation of turbo and LDPC code decoders is a well studied problem in the literature, two critical needs emerged in the last years: i) achieving high throughput, ii) granting flexibility and interoperability. The intrinsic differences between the turbo and LDPC decoding algorithms and their iterative nature make the design of high throughput, flexible turbo/LDPC decoder architectures a challenging task.

As said before, in both turbo and LDPC decoders high throughput is generally achieved by employing parallel architectures [55,56], where several processing elements (PEs) perform the decoding algorithm concurrently on different portions of the received frame. However, PEs require a large communication bandwidth and an efficient interconnection structure to concurrently read/write data from/to the memory. High throughput PEs able to support both turbo and LDPC decoding [48–52] can be implemented as Application-Specific-Integrated-Circuits (ASICs) or ASIPs. In general, ASIC solutions achieve higher throughput with lower complexity as compared to ASIP implementations. However, ASIP architectures are usually more flexible than ASIC ones.

Stemming from the general NoC paradigm [57], Neeb et alii [58] proposed an interesting NoC-based approach to enable flexible and efficient interconnection among the processing elements in parallel turbo decoder architectures. According to [44] this approach, where the network structure is used to connect PEs belonging to the same Intellectual Property (IP), is referred to as intra-IP NoC. In [59] the intra-IP NoC approach is studied in the context of parallel turbo decoder architectures investigating a number of direct and indirect networks. A similar approach has been employed for LDPC decoder architectures [44, 60]. Few recent works [48, 51, 61] tried to exploit the intra-IP NoC approach to design flexible turbo/LDPC decoder architectures. However, from these works it is not clear how the design of the PEs and the design of the network influence each other. Moreover most of these implementations are not fully compliant with the high throughput requirements of modern wireless standards.

The work described in this section and published in [13] exploits the Turbo NoC cycle-accurate simulation tool [62] described in [11], where an extensive analysis of the performance achieved by various NoC topologies in the context of turbo decoder architectures is shown. The aim is to propose a competitive intra-IP NoC-based ASIC architecture for flexible turbo/LDPC decoding with a clear design flow.

3.1.1 NoC architecture for channel decoding

Turbo and LDPC decoding have in common a complex message passing phase, which varies in terms of duration and intertwining with the parallelism of the decoder. To achieve a high degree of flexibility, ranging from multiple code to component reuse and scalability, the need of a suitable interconnection structure is paramount. NoC-based interconnect architectures have been suggested and partially explored in [63], [44] and [11]: NoC–based decoding is an emerging paradigm for partially parallel solutions, due to the adaptive abilities of these structures. NoCs guarantee virtual connectivity among all nodes, a key feature for tasks such as LDPC and turbo decoding, where complex graph are deeply involved. The work in [11] provides an extensive analysis of performances of various NoC topologies related to turbo decoding. The analysis is carried out via custom cycle–accurate simulation tool [62]. This section analyzes the characteristics of NoC architectures requested to support both turbo and LDPC decoding. To this purpose, the results presented in [11] act as the starting point towards the inclusion of LDPC decoding. The assumed node architecture is detailed in [11] and shown in Fig. 3.1: each node in the network is made of a Routing Element (RE), a Processing Element (PE) and a memory (MEM) to store the incoming messages. In the following, we indicate the j-th message received and generated by PE i as $\lambda'_{i,j}$ and $\lambda_{i,j}$ respectively. The RE is based on an $F \times F$ crossbar switch with F input First-In-First-Out (FIFO) buffers and F output registers. $t'_{i,j}$ represents the memory location where $\lambda'_{i,j}$ will be stored. This section focuses on the two most promising node architectures proposed [11]: the All-Precalculated (AP) and the Partially-Precalculated (PP) architectures. The AP architecture makes use of off-line simulations to compute the routing information of each node and to store it in a routing memory. Since the routing information is precalculated, very complex routing algorithms can be employed to compute the routing information and this allows to reduce the depth of input FIFOs. Moreover, this solution does not require any kind of header in the packet structure, reducing the width of the input FIFOs. However, as pointed out in [59], the AP architecture requires additional memories to store the routing information of all supported codes. In PP architecture the routing is performed on-line by a routing algorithm: only $t'_{i,j}$ sequences are precalculated, while destination node identifiers are included in the packet header.

The SystemC simulator developed in [11] is here used to extensively analyze the performance of NoC-based LDPC decoder architectures, in terms of throughput and memory requirements. A set of parameters is defined to take into account a large number of possible design choices including routing algorithms, node architectures and packet structures. The simulator requires the description of the NoC topology, i.e. the number of nodes and their links, then, it derives the communication pattern among the nodes of the network.

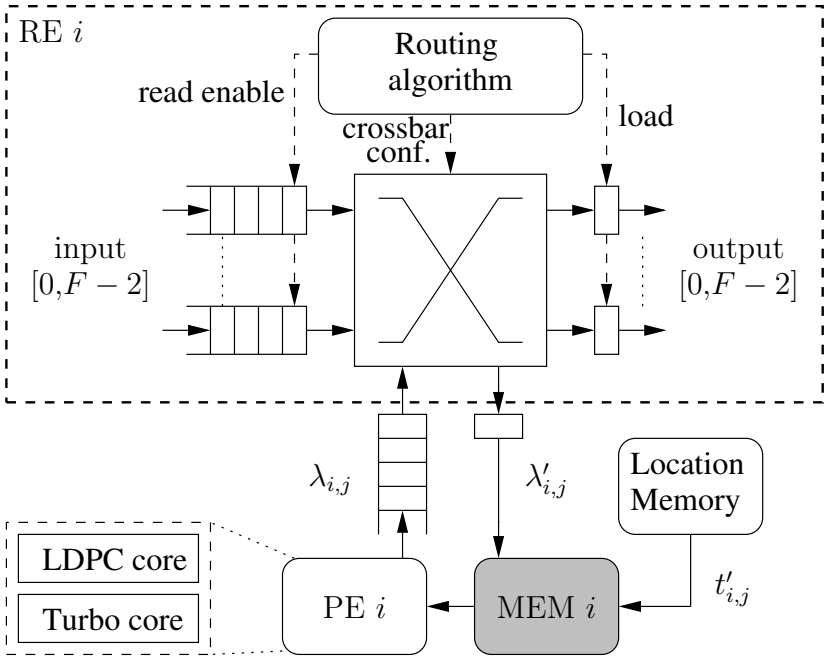

Figure 3.1: Node structure

In order to evaluate the performance of an NoC-based LDPC decoder, a pre-processing tool has been developed to produce the complete communication pattern. Indeed **H**, the parity check matrix of the LDPC code, can be expressed as a list of communication needs similar to a turbo code interleaver once the decoding scheduling and the topology are chosen. The following flow has been employed to analyze the performance of NoC-based LDPC decoder architectures.

- The first step is the definition of the graph representation of the **H** matrix. Size and structure of this graph depends on the chosen scheduling. With the layered decoding approach, the resulting graph has M nodes, and an arc between row–nodes i and j is defined when a non-zero entry is present on the same column of both i and j.

- The second step is the choice of the NoC topology and its size, which corresponds to the parallelism degree of the LDPC decoder. To this purpose, a set $\mathcal{T}$ of various topologies, including mesh, toroidal mesh, spidergon, honeycomb, generalized De-Bruijn and generalized Kautz has been considered.

- The problem of mapping the LDPC codes on a specific NoC is then formulated in terms of graph partitioning and solved using the Metis bundle of graph–coloring algorithms [64]. Once graph nodes are assigned to NoC-nodes, the communication pattern is constructed. The framework built around the Metis package checks the produced patterns for minimum length and uniform

message distribution, selecting the optimal one for each code–topology couple.

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

- *Processing element output rate* (R): is the number of messages produced by a PE in a clock cycle.

- *Routing algorithm*: three different routing policies are embedded in the available simulator [62]. They rely on the off-line computation of the shortest paths between nodes. This information is stored in one or more routing tables. When only one shortest path is used (one routing table) the routing algorithm is referred to as Single-Shortest-Path (SSP), whereas when more shortest paths are computed (multiple routing tables) the algorithm will be named All-local-Shortest-Paths (ASP). The first approach described in [11] is the SSP-Round-Robin (SSP-RR): it is based on a circling serving policy. Similarly the SSP-FIFO-Length (SSP-FL) routing algorithm is based on the current status of input FIFOs. The third approach, named ASP–FIFO-length-with-Traffic-spreading (ASP-FT), takes in account all the possible different shortest paths. The serving policy is a modified version of FL: it keeps a statistic of sent messages to spread the traffic on the network [11].

- *Delay/Send Colliding Message* (DCM/SCM): this parameter activates a collision management technique. A collision arises when two or more messages require to be routed to the same router output port. In this case, if the DCM strategy is employed, the first message is routed according to the selected routing algorithm, whereas the colliding messages are kept in their FIFOs. On the contrary, if SCM is used, colliding messages will be randomly routed to one of the available output ports. Namely, the configuration of the crossbar switch is chosen to route non-colliding messages, whereas colliding messages are treated as "don't-care".

- *Route Local* (RL): this flag allows to choose if local messages, i.e. messages sent and received by the same PE, are routed on the network ($RL = 1$) or are stored in an internal queue, bypassing the routing ($RL = 0$).

Table 3.1: Throughput [Mb/s]/Area[mm^2] for WiMAX LDPC N = 2304, r = 1/2 code, for different topologies, parallelism P, node degree D, routing algorithms and node architectures. Frequency is 300 MHz, technology is CMOS 90 nm. Results are obtained for parameters RL = 0, SCM, R = 0.5

	D = 2, generalized De Bruijn			
	P = 16	P = 24	P = 32	P = 36
SSP-RR (PP)	37.77/2.02	41.19/3.16	50.16/3.68	50.31/4.02
SSP-FL (PP)	42.15/1.82	45.47/3.27	55.12/0.65	56.20/4.18
ASP-FT (AP)	42.15/0.40	45.47/0.59	55.12/0.65	56.84/0.71
	D = 3, spidergon			
	P = 16	P = 24	P = 32	P = 36
SSP-RR (PP)	55.74/0.35	67.11/1.34	70.67/2.69	71.11/3.14
SSP-FL (PP)	55.47/0.30	69.82/1.11	75.62/2.59	75.79/3.20
ASP-FT (AP)	55.31/0.30	72.45/0.42	76.63/0.64	78.37/0.73
	D = 4, rectangular honeycomb			
	P = 16	P = 24	P = 32	P = 36
SSP-RR (PP)	55.12/0.42	77.49/0.61	98.46/0.72	97.90/1.03
SSP-FL (PP)	55.47/0.39	78.01/0.53	98.18/0.63	106.67/0.87
ASP-FT (AP)	55.65/0.40	78.01/0.48	99.03/0.55	109.37/0.58

Table 3.2: Throughput [Mb/s]/Area[mm^2] for WiMAX LDPC N = 2304, r = 1/2 code, for generalized Kautz topology, parallelism P, node degree D, routing algorithms and node architectures. Frequency is 300 MHz, technology is CMOS 90 nm. Results are obtained for parameters RL = 0, SCM, R = 0.5

	D = 2, generalized Kautz			
	P = 16	P = 24	P = 32	P = 36
SSP-RR (PP)	38.10/2.05	49.23/2.79	48.20/3.67	55.47/3.84
SSP-FL (PP)	41.69/1.84	53.09/2.68	55.74/3.61	61.71/0.68
ASP-FT (AP)	41.69/0.40	53.09/0.51	55.74/0.64	61.71/0.68
	D = 3, generalized Kautz			
	P = 16	P = 24	P = 32	P = 36
SSP-RR (PP)	55.74/0.29	78.37/0.47	93.66/0.96	92.65/1.22
SSP-FL (PP)	55.74/0.28	77.49/0.43	97.63/0.69	101.05/0.86
ASP-FT (AP)	55.74/0.29	77.49/0.35	97.08/0.42	101.05/0.46
	D = 4, generalized Kautz			
	P = 16	P = 24	P = 32	P = 36
SSP-RR (PP)	55.74/0.31	72.45/0.60	70.10/1.06	104.73/0.76
SSP-FL (PP)	55.74/0.29	77.84/0.49	72.00/0.98	109.37/0.72
ASP-FT (AP)	55.74/0.39	78.01/0.47	100.47/0.54	108.68/0.58

3.1.2 Simulation tool for NoC analysis

3.1.3 Analysis of NoCs for LDPC code decoding

In order to show the potential of the NoC approach in the design of LDPC code decoders, the whole set of WiMAX codes has been used as a design case. In Table 3.1 and 3.2 the most relevant results for the WiMAX LDPC code with N = 2304 and rate r = 1/2 are shown. This code is the most demanding

Table 3.3: Throughput [Mbits/s]/Area[mm^2] for NoC based architectures supporting all WiMAX turbo and LDPC codes

	$P = 22$, $D = 3$, generalized Kautz, $R = 0.5$	
	turbo @ 75 MHz $N = 2400$, $r = 1/2$	LDPC @ 300 MHz $N = 2304$, $r = 1/2$
SSP-RR (PP)	74.25/0.63	72.45/0.46
SSP-FL (PP)	74.26/0.60	72.30/0.39
ASP-FT (AP)	73.29/0.69	72.91/0.34

one within WiMAX specification in terms of PE resources. Given that $D = F - 1$ is the topology node degree, i.e. the number of router output ports connected to other routers, for each topology in $\mathcal{T}$ all routing algorithms have been tested for $D = 2, 3, 4$ and $P = 16, 24, 32, 36$. The throughput has been computed as

$$T = \frac{(N - M) \cdot f_{clk}}{(lat_{core} + n_{cycles}) \cdot \mathrm{It}_{max}} \tag{3.1}$$

where f_{clk} is the clock frequency, It_{max} is the maximum number of iterations, lat_{core} is the maximum latency of the decoding core and n_{cycles} is the number of cycles required to exchange all information through the NoC according to the communication pattern. Results in Table 3.3 have been obtained with (3.5), imposing $f_{clk} = 300$ MHz, $\mathrm{It}_{max} = 10$ and $lat_{core} = 15$ cycles.

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

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

3.1.4 Analysis of NoCs for turbo and LDPC code joint decoding

To find the most suitable NoC for both turbo and LDPC decoding according to the WiMAX standard the results shown in [11] for turbo codes and the ones presented in Table 3.1 and 3.2 for LDPC codes have been analyzed. Generalized Kautz topologies show the best average throughput-to-area ratio both for turbo and LDPC codes and $D = 3$ is a good throughput/complexity trade-off. For LDPC codes the minimum value of P to achieve the 70 Mbit/s throughput required by the IEEE 802.16e standard, with a 300 MHz clock frequency, is 22. On the contrary, turbo codes yield a higher than required throughput with a 22-nodes NoC. Thus, the working frequency in turbo codes mode can be lowered to 75 MHz and throughput is still above limit. For both codes, throughput and area show a weak dependence on the routing algorithm. However, the SSP-FL routing algorithm guarantees the

best average performances also with different topologies and non-WiMAX codes, thus being the best choice in terms of flexibility. A choice of the obtained results are given in Table 3.3 for $N = 2400$ turbo code and $N = 2304$, $r = 1/2$ LDPC code.

3.2 NoC-based turbo/LDPC decoder architecture

Based on the NoC design shown in Section 3.1, a complete flexible turbo/LDPC decoder is presented in this section to prove that the flexibility achieved via the intra-IP NoC-based approach has a limited impact on the area of the decoder architecture. As a case of study the WiMAX standard is considered. The architecture has complexity comparable to the latest state-of-the-art proposed flexible turbo/LDPC decoders, together with a higher worst-case throughput, and guarantees multi-standard compliance and low power consumption. The presented work has been published in [13].

3.2.1 LDPC decoding core

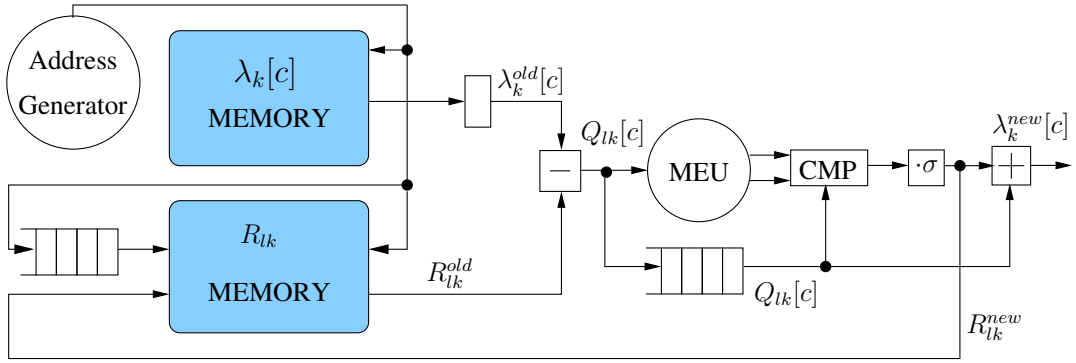

Figure 3.2: LDPC decoding core

For LDPC decoding, the PE must be structured so that all the block lengths and code rates imposed by the standard are supported. A simple and effective architecture based on a sequential processing has been designed. Fig. 3.2 shows the architecture of the LDPC decoding core, employing the NMS approximation: $\lambda_k^{old}[c]$ values are read from the $\lambda_k[c]$ memory, used to store the incoming messages received from the network. Similarly, R_{lk}^{old}, required to compute $Q_{lk}[c]$, is stored in a dedicated memory. Then, $Q_{lk}[c]$ values are compared sequentially in the Minimum Extraction Unit (MEU) to find the first two minimum values (2.12). A further comparison selects which of the two minimum values is required to update $\lambda_k^{new}[c]$ as in (2.10). Concurrently, R_{lk}^{new} values are stored in the R_{lk} memory for use during the next iteration. This architecture is completely independent of the code, but it is limited by the size of memories. Both $\lambda_k[c]$ and R_{lk} memories must have enough storage capability to cope with the heaviest possible workload among the supported codes. Simulations show that the worst case among WiMAX codes is given by the $N = 2304$, $r = 3/4$ code. Given this sizing, not only all the other WiMAX codes, but any Quasi-Cyclic LDPC (QC-LDPC) code with no superposition of vertexes between nodes and code with smaller size can be decoded.

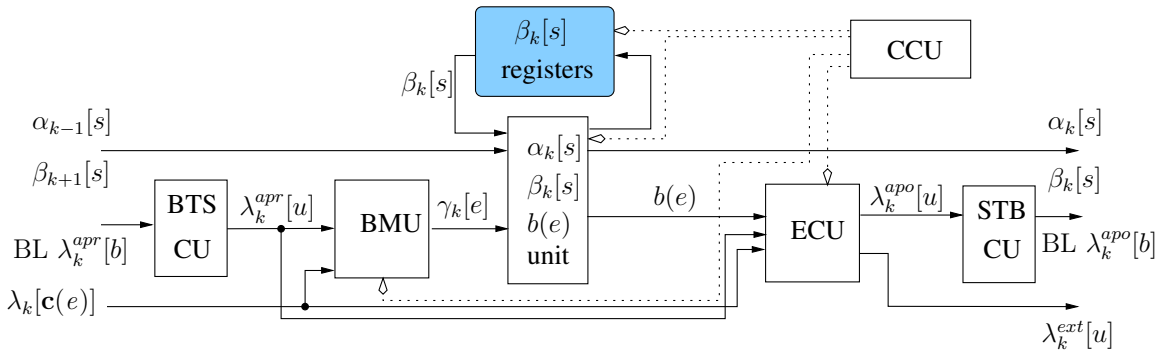

Figure 3.3: Turbo decoding core (SISO)

3.2.2 Turbo decoding core

The proposed solution for the turbo decoding core is the SISO architecture in Fig. 3.3.

Since the turbo code used in the WiMAX standard is double-binary each message $\lambda_{i,j}$ is a vector of three elements. According to [38], sending bit-level (BL) instead of symbol-level extrinsic information reduces the NoC complexity of roughly 1/3. Resorting to the solution proposed in [65] this complexity reduction comes at the expense of a 0.2 dB BER loss. A dedicated unit, the Bit-To-Symbol Conversion Unit (BTS CU) converts the incoming *a priori* values from bit (BL $\lambda_k^{apr}[b]$) to symbol ($\lambda_k^{apr}[u]$) information, whereas, the Symbol–To–Bit Conversion Unit (STB CU), converts the processed *extrinsic* values ($\lambda_k[u]$) before sending them on the network (BL $\lambda_k[b]$). The BMU, i.e. Branch Metric Unit, is entitled the task of computing the $\gamma_k[e]$ (2.5). Another unit handles, sequentially, $\beta_k[s]$, $\alpha_k[s]$ and $b(e)$ as in (2.1). $\beta_k[s]$ values are stored in a set of registers for use during the calculation of $b(e)$. The Extrinsic Computation Unit (ECU) produces the updated LLR $\lambda_k[u]$. As in [51], the number of bits to represent $\lambda_k[c]$, $\alpha_k[s]$, $\beta_k[s]$ and $\lambda_k[u]$ is set to 7, whereas 5 bits are sufficient for R_{lk} and $\lambda_k[\mathbf{c}(e)]$.

3.2.3 Decoder synthesis and comparison

As a case of study, a complete turbo/LDPC decoder for the WiMAX standard has been implemented. Table 3.4 shows the pre–layout synthesis results (2^{nd} row), obtained with Synopsys Design Compiler on a 90 nm deep sub-micron CMOS technology, together with recent state–of–the–art dual code decoders. Where possible, worst–case throughput and the relative code are reported. For the sake of fairness it is worth noting that [49] and [48] support both WiMAX and LTE modes. Comparison with [48] shows similar core area occupation, whereas our NoC contributes for 0.61 mm^2, about the 20% of the total area occupation. The proposed NoC is larger than the interconnection used in [48], mainly due to the more distributed topology and more complex node architecture. Both LDPC

Table 3.4: LDPC/Turbo architectures comparison: CMOS technology process (TP), processing area occupation (A_{core}), total area occupation (A_{tot}) normalized area occupation for 65nm technology (An), clock frequency (f_{clk}), peak power consumption (Pow), data width (DW), maximum number of iterations (It_{max}), code length (N) and rate (r) and throughput (T)

Decoder	P	Tp [nm]	A_{tot} [mm^2]	An_{tot} [mm^2]	f_{clk} [MHz]	Pow [mW]	DW [bits]	It_{max}	Code	N, r	T [Mb/s]
Proposed	22	90	3.17	1.65	300	415	7 – 5	10	LDPC	2304, 1/2	72.00 (min.)
					75	59	7 – 5	8	DBTC	2400, 1/3	74.26 (min.)
[48]	8	90	2.6	1.36	520	N/A	7 – 5	10	LDPC	2304,1/2	62.5 (min.)
							8 – 6	6	DBTC	N/A	173 (max.)
[49]	1	65	0.62	0.62	400	76.8	7 – 5	20	LDPC	N/A	27.7 (min.)
							8	5	DBTC	N/A	18.6 (min.)
[51]	12	45	0.9	1.88	150	86.1	7 – 5	8	LDPC	N/A	71.05 (min.)
							7 – 5	8	DBTC	N/A	73.46 (min.)
[50]	384	45	N/A	0.94	1.96	333	1000	25	LDPC	N/A	333 (avg.)
[52]	12	90	3.20	1.67	500	N/A	9 – 6	15	LDPC	2304, 1/2	600 (max.)
							9 – 6	6	BTC	6144, 1/3	450 (max.)

and turbo cases in this work show compliance with the WiMAX standard throughput requirements. LDPC codes are considered with R = 0.5, and a clock frequency of 300 MHz for both the NoC and the LDPC core. The worst case values, obtained for the N = 2304, r = 3/4 code, are still above 70 Mb/s: in [48], according to the provided formula, for the same code throughput is below the standard requirement. The optimal working conditions for turbo decoding are with R = 0.33, and sufficient throughput is obtained with f_{clk} = 75 MHz: however, if the frequency is rescaled to 200 MHz, our worst–case throughput overperforms the best–case value of [48] (198 vs. 173 Mb/s), despite the higher number of iterations and the much lower f_{clk}.

This characteristic, together with the lower memory accesses rate of turbo decoding, results in a large power reduction w.r.t. LDPC decoding. It is worth noting that, in the turbo decoding mode the proposed architecture achieves the lowest power consumption as compared with [48–52].

The architecture in [49] not only supports both WiMAX and LTE modes but it also features a very small area occupation. However, it does not reach a high enough throughput for the WiMAX standard, while our decoder has both smaller area and higher throughput than [51]. Area occupation is smaller than [50], but throughput analysis is difficult, since standard compliance is stated but no minimum values are reported. The architecture for WiMAX/WiFi LDPC codes and 3GPP-LTE turbo code presented in [52] runs at 500 MHz and achieves the highest throughput among compared architectures with the same complexity as the presented architecture. A fair comparison is not possible as WiMAX turbo code is not addressed. The proposed decoder guarantees compliance with WiMAX, but is not limited to its codes: the SISO can work with any 8 state Double-Binary-Turbo-Code (DBTC), whereas the LDPC core can sustain any code smaller than 802.16e ones (e.g. WiFi).

3.3 Reconfiguration of NoC-based channel decoders

A common trend in the implementation-centered works present in the state of the art is to neglect corollary issues that, while not performing the core operations, are absolutely fundamental for the correct functionality of the system. The reconfiguration in case of code or standard change is one of those, and it is particularly critical in a decoder targeting flexibility. Most flexible decoders available in the literature [11, 44, 48–52, 59, 61], though supporting a wide range of codes, do not address the reconfiguration issue. A few recent works [66–68] have considered reconfiguration issues for ASIP-based turbo decoders: however, given the much larger parallelism of NoC-based decoders and to the differences between ASIPs and ad-hoc PEs, the proposed techniques can not be extended to our case. Change of decoding mode, standard or code parameters requires not only hardware support, but also memory initialization and specific controls: since in many standards a code switch can be issued as early as one data frame ahead [3], a time efficient reconfiguration technique must be developed. The work presented in this section, and published in [14], a detailed analysis of the reconfiguration issue is carried out for a case of study, with different solutions and trade-offs.

3.3.1 Decoder Reconfiguration

The decoder described in Section 3.2 has been taken in account for the following reconfiguration analysis. The LDPC PE and the SISO core share the memories where the incoming data $\lambda'_{i,j}$ (representing both $\lambda_k[u]$ and $\lambda_k[c]$) are stored and the location memory containing the pre-computed $t'_{i,j}$ values, i.e. the memory addresses to store $\lambda'_{i,j}$. For such a decoder the reconfiguration task consists of i) rewriting the location memory containing $t'_{i,j}$ values; ii) reloading the CN degree (deg) parameters and the window size in the control unit of LDPC decoding cores and SISOs respectively. In the following, the whole set of storage locations to be updated at reconfiguration time will be indicated as "reconfiguration memory". When possible, the decoder must be reconfigured while the decoding process is still running on the previous data frame. This means that the reconfiguration data can be distributed by means of the NoC interconnections only at the cost of severe performance penalties. Consequently, we suppose that the reconfiguration data are moved directly to the P PEs via a set of N_b dedicated buses, each one linked to $\frac{P}{N_b}$ PEs.

In the following the reconfiguration occurrence is estimated assuming mobile receivers moving at different speeds and the carrier frequency f_c = 2.4 GHz. This frequency is included in most standards' operation range, and used in a variety of applications. In this scenario the communication channel is affected by fading phenomena, namely slow fading, whose effects have very long time constants, and fast fading. Fast fading can be modeled assuming a change of channel conditions every time the

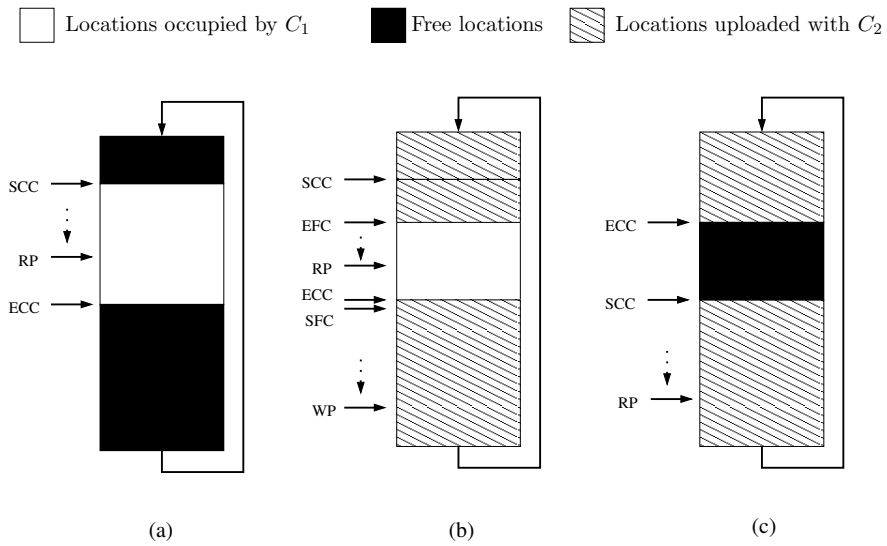

Figure 3.4: Memory reconfiguration process: (a) Decoding of C_1; (b) Upload of reconfiguration data required for C_2 (phases Φ_1 to Φ_3 and Φ_5); (c) First iteration of C_2 and concurrent upload of reconfiguration data (Φ_4)

receiver is moved by a distance similar to the wavelength λ of the carrier. Being λ = 0.125 m, at a speed v = 70 km/h the channel changes with a frequency f_{chng} = 155 Hz (WiMAX, WiFi, 3GPP-LTE), whereas, at v = 10 km/h (DVB-RCS, HPAV, CMMB, DTMB) changes occurs at f_{chng} = 22 Hz. These scenarios result in different reconfiguration probabilities, whose impact on BER performance is addressed in Section 3.3.2.

The proposed reconfiguration memory is organized as a circular buffer: two sets of pointers are used to manage reading and writing operations. The Start of Current Configuration (SCC) pointer and the End of Current Configuration (ECC) pointer delimit the memory blocks that are currently being used. A Read Pointer (RP) is used to retrieve the data during the decoding process, as shown in Fig. 3.4.(a). The Start of Future Configuration (SFC) and End of Future Configuration (EFC) pointers are instead used concurrently with the Write Pointer (WP) to delimit the locations that are going to be used to store the new configuration data.

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

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

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

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

Φ_1 In the reconfiguration from code C_1 to code C_2, l_{c1} words must be replaced with l_{c2} new words. The first part of the configuration can be scheduled during the initial $n_{it1} - 1$ decoding iterations on C_1 and therefore the available time is $t_a^{\Phi_1} = (n_{it1} - 1) \cdot t_{it1}$; in this range of time a maximum of $l_a^{\Phi_1} = \frac{N_b}{P} \cdot (n_{it1} - 1) \cdot t_{it1}$ words can be loaded into each buffer. However, assuming that the buffer size is larger than l_{c1}, we define $B - l_{c1}$ as the number of unused memory blocks in current configuration for code C_1. Therefore, the actual number of locations written in Φ_1 is the minimum between $B - l_{c1}$ and $l_a^{\Phi_1}$. The SFC pointer is thus initialized as ECC (Fig. 3.4.(b)).

Φ_2 During the last iteration on C_1, every memory location between SCC and the current position of RP is available for reconfiguration. This means that up to l_{c1} locations are available for receiving configuration words for C_2. However, this has to be done during a single iteration, and therefore $t_a^{\Phi_2} = t_{it1}$ cycles are available. During these cycles, up to $l_a^{\Phi_2} = \frac{N_b}{P} \cdot t_{it1}$ words can be loaded.

Φ_3 As mentioned before, part of the configuration can be overlapped with the first decoding iteration on C_2 code. SCC is initialized as SFC, and RP will take the duration of a full iteration to arrive to ECC (Fig. 3.4.(c)). The available time is $t_a^{\Phi_3} = t_{it2}$ and the maximum number of words that can be loaded in this phase is $l_a^{\Phi_3} = \frac{N_b}{P} \cdot t_{it2}$.

Φ_4 In the event that previously listed phases are not sufficient to complete the configuration, an early stopping in the decoding of code C_1 can be scheduled to make available additional cycles

Table 3.5: Reconfiguration phases Φ_i: $t_a^{\Phi_i}$, available clock cycles during Φ_i and $l_a^{\Phi_i}$ number of locations that can be written in $t_a^{\Phi_i}$

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

to be used for loading the remaining part of the configuration words. We indicate the number of cycles available in this phase as $t_a^{\Phi_4} = t_{stop}$. The number of words that can be loaded in Φ_4 is $l_a^{\Phi_4} = \frac{N_b}{P}\cdot t_{stop}$. As one or more complete iterations are dropped in Φ_4, t_{stop} is a multiple of t_{it1}, which can be formalized as

$$t_{stop} = n_{stop}\cdot t_{it1}, \qquad n_{stop} = 0,1,2,3,\cdots \tag{3.2}$$

Differently from the other four phases, Φ_4 affects the decoder performance, as if $n_{stop} > 0$ the number of decoding iterations is reduced for code C_1. Evaluating the actual effect on BER and Frame-Error-Rate (FER) curves is necessary to understand the feasibility of this approach.

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

$$t_a^{\Phi_5} = n_{it1}\cdot t_{it1}\cdot N_f, \qquad l_a^{\Phi_5} = \frac{N_b}{P}\cdot n_{it1}\cdot t_{it1}\cdot N_f \tag{3.3}$$

The five described phases are reported in Table 3.5, together with the corresponding $t_a^{\Phi_i}$ and $l_a^{\Phi_i}$. Thus, B, N_b, n_{stop} and N_f are design parameters, and their values must be decided based on decoder parallelism (P) and supported codes, which determine l_{c1} and l_{c2}.

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

$$(n_{it1}-1)\cdot t_{it1} \gtrless \frac{P}{N_b}\cdot(B-l_{c1}) \tag{3.4}$$

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

$$l_{th} = \frac{P\cdot B}{P+(n_{it1}-1)\cdot N_b} \tag{3.5}$$

and distinguish between two cases:

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

Let us study the two cases separately.

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

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

$$\frac{P}{N_b} l_{c2} < t_a^{\Phi_1} + t_a^{\Phi_2} + t_a^{\Phi_3} + t_a^{\Phi_5} \tag{3.6}$$

This constraint simply means that the overall available time through Φ_1, Φ_2, Φ_3 and Φ_5 must be long enough to update l_{c2} locations in the reconfiguration memories. From the values in the second column of Table 3.5 the constraint in (3.6) becomes

$$l_{c2} < \frac{N_b}{P} \cdot (n_{it1} \cdot t_{it1} + t_{it2}) + \frac{N_b}{P} \cdot n_{it1} \cdot t_{it1} \cdot N_f \tag{3.7}$$

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

$$l_{c2} < \frac{N_b \cdot n_{it1} \cdot (1 + N_f)}{P - N_b} \cdot l_{c1} \tag{3.8}$$

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

$$t_a^{\Phi_1} + t_a^{\Phi_5} \leq \frac{P}{N_b} \cdot (B - l_{c1}) \tag{3.9}$$

namely

$$(n_{it1} - 1) \cdot t_{it1} + n_{it1} \cdot t_{it1} \cdot N_f \leq \frac{P}{N_b} \cdot (B - l_{c1}). \tag{3.10}$$

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

$$N_f(l_{c1}) \leq \frac{\frac{P}{N_b} \cdot (B - l_{c1}) - (n_{it1} - 1) \cdot l_{c1}}{n_{it1} \cdot l_{c1}} \triangleq Nf_{max} \tag{3.11}$$

Thus, (3.8) can be better written as

$$l_{c2} < \frac{N_b \cdot n_{it1} \cdot \left[1 + N_f(l_{c1})\right]}{P - N_b} \cdot l_{c1} \tag{3.12}$$

This means that the size of code C_2 has an upper bound and this bound is proportional to the size of code C_1. Therefore, the most critical reconfiguration cases are those involving "small" C_1 codes: in such cases, there could be many C_2 codes that violate condition (3.12). The bound is also proportional to N_b, and can be consequently increased by rising the number of reconfiguration buses.

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

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

$$l_{c2} < B - l_{c1} + \frac{N_b}{P} \cdot (t_{it1} + t_{it2} + t_{stop}) \tag{3.13}$$

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

$$l_{c2} < \frac{P \cdot B}{P - N_b} - \left(1 - \frac{N_b \cdot n_{stop}}{P - N_b}\right) \cdot l_{c1} \tag{3.14}$$

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

3.3.2 Reconfiguration: cases and examples

The reconfiguration method detailed in Section 3.3.1 has been applied to a set of target standards, in order to identify suitable design parameters (i.e. N_b, B, n_{stop}, Nf_{max}) that enable reconfiguration without pausing the decoder for most of code sizes. The following analysis has been performed with $n_{it1} = 10$. As a first step, we considered all the *intra*-standard code combinations and we derived the duration of each reconfiguration phase when switching from a code to another one.

As an example, Figure 3.5 shows the duration of the defined configuration phases (measured in number of C_1 iterations) when switching between WiMAX $N = 2304, r = 3/4$ and $N = 2208, r = 3/4$ LDPC codes, for different values of N_b and B. It is possible to see how, increasing B, Φ_1 assumes a leading role, while at the same time the necessary t_{stop} is reduced. Increasing N_b reduces the overall duration of the reconfiguration. It is worth noting that in this case, Φ_4 is always necessary, except for the large combinations of N_b and B.

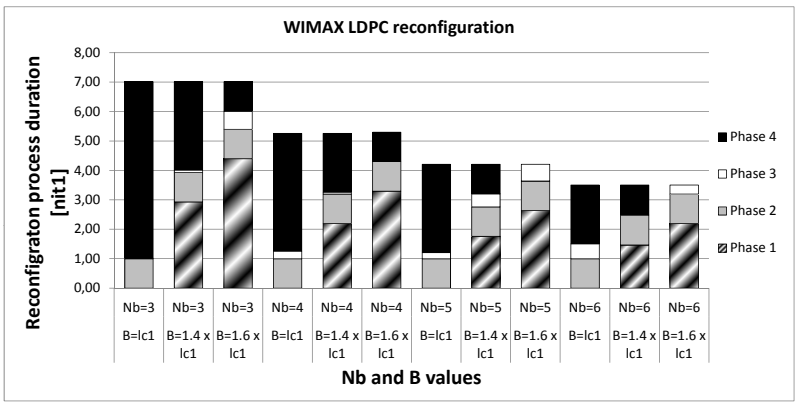

Figure 3.5: WiMAX LDPC reconfiguration case

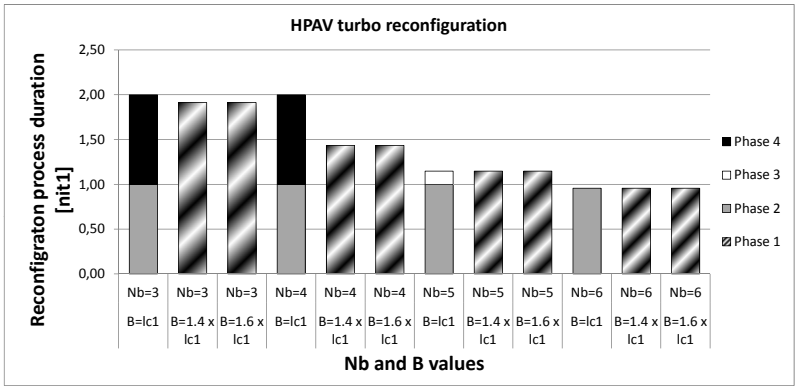

Figure 3.6: HPAV turbo reconfiguration case

A different trend can be noticed in the HPAV case (Fig. 3.6), due to the difference in memory requirements between the two largest codes of HPAV standard. The reconfiguration can be completed in Φ_1 if $B \geq 1.4 \cdot l_{c1}$, otherwise also Φ_2 is necessary.

In most cases, *intra*-standard reconfiguration is possible using phases Φ_1 to Φ_4, with no need for exploiting additional frames (Φ_5). Only in a few cases the switch from a very small code to a very large one requires $N_f \geq 1$ (see diagonal of Table 3.8). This occurrence is much more frequent when *inter*-standard reconfiguration is considered, due to the presence of small codes alongside very large ones, as DTMB and CMMB LDPCs (see out of diagonal entries in Table 3.8).

Figures 3.7 to 3.11 plot the maximum l_{c2}, as defined by (3.12) and (3.14), for a continuous set of l_{c1} values. The × markers represent a subset of the considered *intra*- and *inter*-standard code changes: markers below the curve identify reconfigurations that can be performed without pausing the decoder.

Figure 3.7 shows the maximum l_{c2} for different values of B: in this plot, B = 100% corresponds to B = 1771, which is the size of the largest considered l_{c1}, while 160% means $B = 1.6 \cdot 1771$. It can be

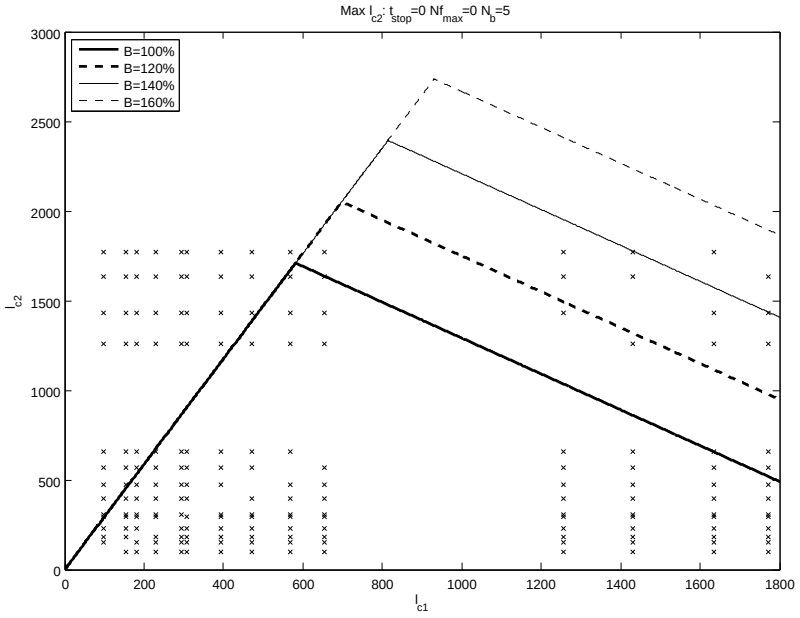

Figure 3.7: Maximum l_{c2} plot with varying B

seen that in the cases of small C_1 codes, increasing the buffer size does not affect the positive slope portion of the curve.

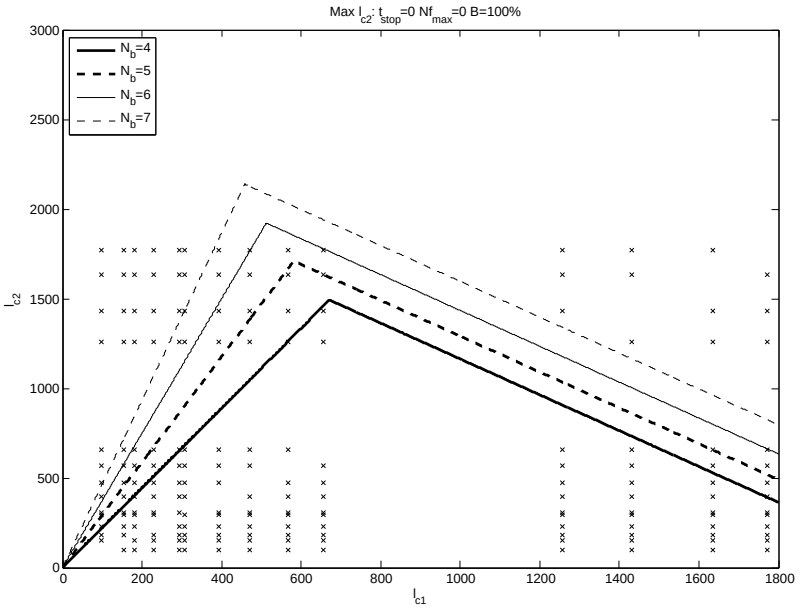

Figure 3.8: Maximum l_{c2} plot with varying N_b

On the contrary, in Fig. 3.8, the maximum l_{c2} is shown for different values of N_b: in this case, an increase of N_b is reflected in all areas of the plot. A higher number of buses means a shorter reconfiguration time, and a larger maximum l_{c2}.

Variation in the maximum allowed t_{stop} (Fig. 3.9) only affects the maximum l_{c2} in case of large C_1 codes (negative-slope portion of the curve), as shown in (3.14). It can be noticed that with $n_{stop} = 3$

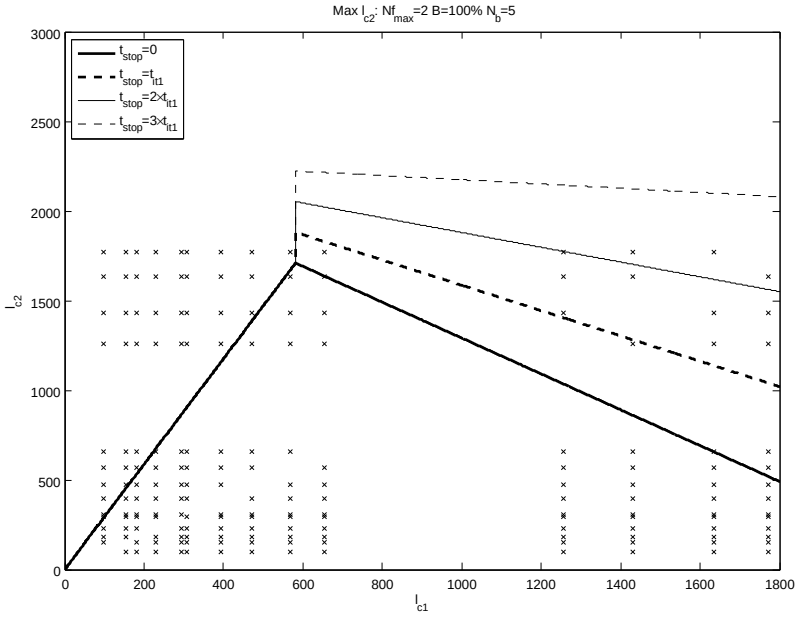

Figure 3.9: Maximum l_{c2} plot with varying t_{stop}

all the large codes are below the right side of the curve: later in this section it will be demonstrated how these skipped iterations are negligible in terms of BER performance.

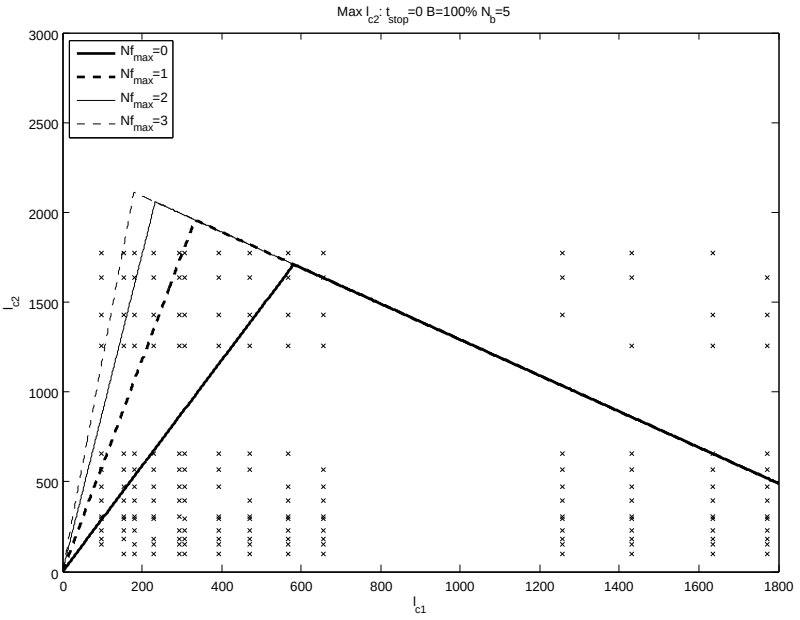

Figure 3.10: Maximum l_{c2} plot with varying Nf_{max}

In Fig. 3.10, the effect of different choices of N_f is shown: from the plot it can be seen that $N_f > 0$ actually increases the maximum l_{c2} only for small C_1 codes.

Finally, Fig. 3.11 plots some combinations of the analyzed parameters in order to allow dynamic reconfiguration among most of considered codes. The represented combinations of N_b, B, t_{stop} and

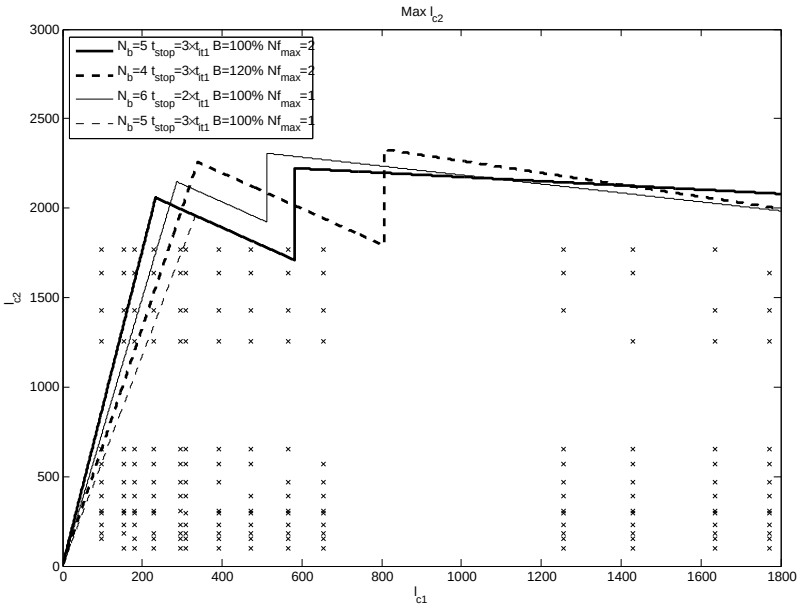

Figure 3.11: Maximum l_{c2} plot, different solutions

Nf_{max} all yield very similar performance: the cost underlying every parameter choice consequently becomes the decision metric. A 20% increase in memory, even if backed up by a smaller number of buses, heavily affects the decoder area occupation, ruling out the solution represented by the thick dashed line. Among the remaining three combinations, the one that makes use of 6 buses yields a higher area occupation than the others. Since with $Nf_{max} = 1$ the thin dashed curve crosses one of the lower × markers, the final choice falls on $N_b = 5$, $B = 100\%$, $t_{stop} = 3 \cdot t_{it1}$ and $Nf_{max} = 2$. Given that $P = 22$, and consequently $\frac{P}{N_b} = \frac{22}{5}$ is not an integer number, every bus will be exclusively connected to four nodes, while the reconfiguration of the remaining two nodes will be shared among all 5 buses. The impact of the reconfiguration process on the decoder area is addressed in Section 3.4.

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

$$P_R = \frac{f_{chng} \cdot r \cdot N}{T_{max}} \tag{3.15}$$

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

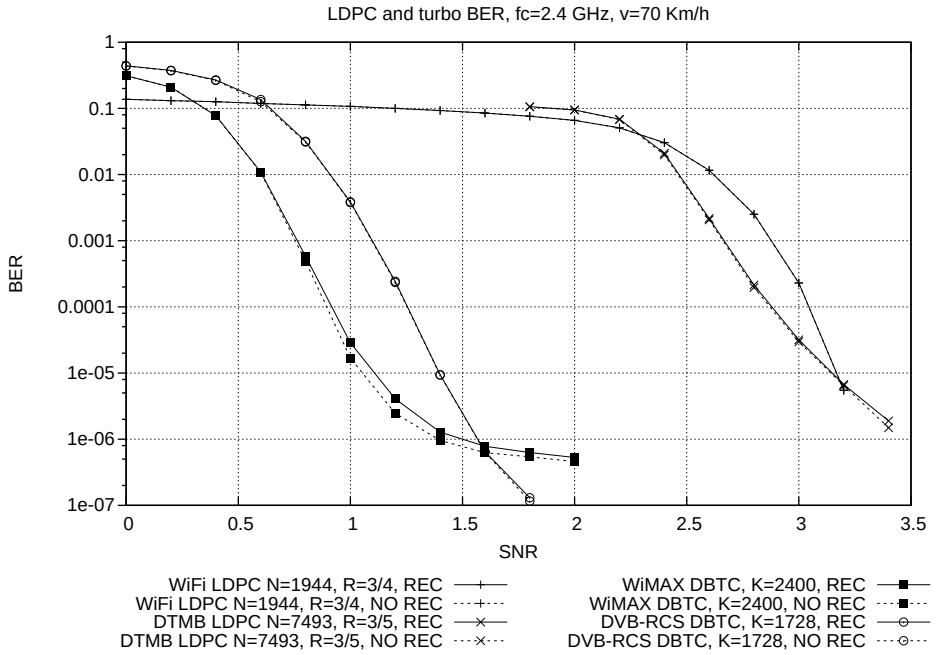

Figure 3.12: LDPC and turbo BER with and without reconfiguration loss, AWGN channel, BPSK modulation

a correct codeword is obtained and $\lceil \cdot \rceil$ is the next highest integer value.

Figure 3.12 shows the BER curves obtained with n_{it} = 10 and n_{stop} = 3, in the pessimistic assumption that a reconfiguration requiring always n_{stop} = 3 occurs with P_R. As it can be observed, the difference between the case when reconfiguration occurs (solid lines) and the no-reconfiguration case (dashed lines) is completely negligible.

3.4 VLSI implementation of a reconfigurable turbo/LDPC decoder

The work presented in this section and published in [14] stems from the results presented in Section 3.2, where a 22-PE NoC-based turbo/LDPC decoder is presented. With its 3.17 mm^2 area and multi-code support, it represents an important step towards flexibility: the design is improved in this section through novel memory scheduling and addressing methods, reduced latency and simpler control. As shown in [69], sharing the datapath of a min-sum based decoder architecture with a log-MAP SISO does not provide significant advantages. As a consequence, in this work logic sharing is not addressed. Experimental results show that the area of the architecture is dominated by memories indeed. Deriving from the reconfiguration analysis of Section 3.3 three reconfigurable turbo/LDPC decoder implementations are presented and compared to the state of the art.

3.4.1 Decoding Cores

The design of the decoding cores must yield the same degree of flexibility of the NoC, being as independent as possible of the set of supported codes. In [13] a completely serial LDPC decoding core has been designed, mostly independent of block length and code rate: an arbitrary number of CN operations can be scheduled on it. The same holds true for the serial SISO decoder, where different windows can be scheduled, regardless of the size of the interleaver.

Quantization and Memory Organization

Memory organization evolves from the idea presented in Section 3.2, in which in every decoding core two memories are instantiated: a 7-bit memory and a 5-bit memory. Their usage is shown in the left part of Fig. 3.13: LDPC VN-to-CN values are stored in the 7-bit memory, together with turbo extrinsic information and state metrics. The 5-bit memory is instead used for CN-to-VN values in LDPC decoding, while storing the intrinsic channel information in turbo decoding. The memories are sized to the largest WiMAX codes (N = 2034, M = 576 for LDPC and K = 2400 for turbo). However, according to post-layout synthesis results, memory access multiplexers suffer from excessive area overhead for these particular cuts. To reduce this problem and at the same time the overall memory area occupation, a novel memory organization technique is proposed, as shown in the rightmost part of Fig. 3.13. Different colors highlight different metrics, while black-striped parts are unused.

Extensive simulations of WiFi, WiMAX, CMMB and DTMB have shown how, in LDPC decoding, $\lambda_k[c]$ and channel LLR quantization can be reduced from 7 to 6 bits without consistent performance degradation. Fig. 3.14 shows the BER curves for some WiMAX, DTMB and WiFi LDPC codes with the two quantization choices: the difference is smaller than 0.05 dB for all rates of medium and large

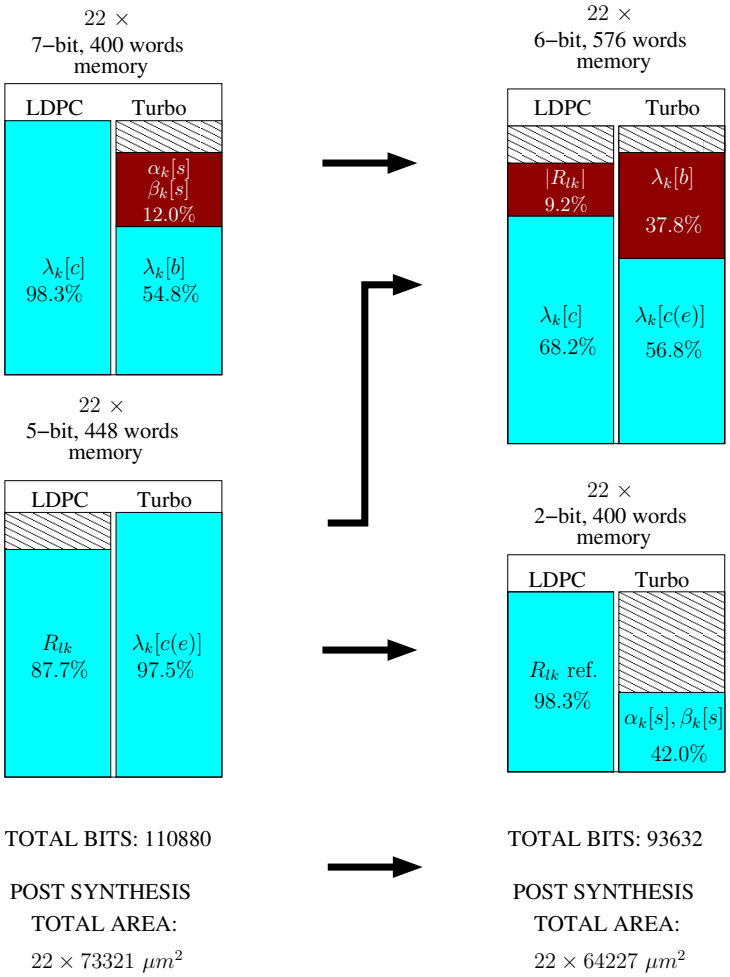

Figure 3.13: Memory organization, WiMAX maximum usage percentages

code sizes. On the same graph, yielding similar results, a few turbo codes examples (WiMAX and HPAV) are plotted, in which $\lambda_k[b]$ and the channel LLR representation changes from 7 to 6 bits, and $\lambda_k[c(e)]$ from 5 to 4 bits (the meaning of $\lambda_k[b]$ will be detailed later in this section). Also for turbo codes, the performance loss introduced by the proposed quantization change is almost negligible. Very small codes, like some of those included in 3GPP-LTE and WiMAX, suffer more from the quantization reduction (Fig. 3.14). Curves obtained with floating point precision show improvements between 0.1 and 0.2 dB w.r.t. the selected precisions. Thanks to these changes, a single 6-bit wide memory is instantiated, in which both $\lambda_k[c]$ and R_{lk} values are saved. Storing all the R_{lk} values requires $\frac{M \cdot row_{deg}}{P} = \frac{576 \cdot 15}{22}$ locations in each decoding core. However, with the normalized min-sum algorithm the number of necessary bits can be reduced by 21.2% by changing the addressing mode as follows. For every CN in the **H** matrix *deg* metrics R_{lk} are updated. Since R_{lk} can take only two possible values, for each CN we can memorize $576 \cdot 2$ magnitudes, and $576 \cdot 15$ 2-bit indexes that identify the correct R_{lk} magnitude and its sign.

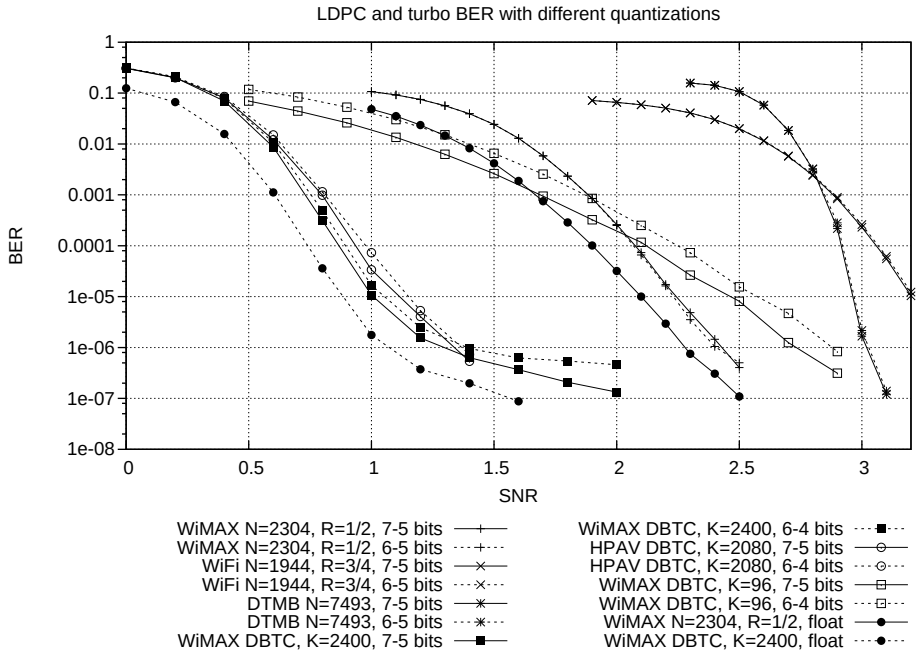

Figure 3.14: LDPC and turbo BER with quantization change, AWGN channel, BPSK modulation

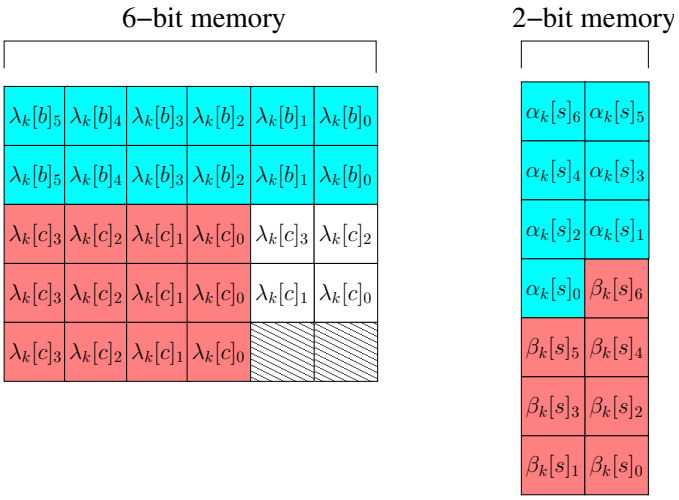

Figure 3.15: Turbo mode in-depth memory organization

The sizing of the 6-bit memory is determined by the DBTC decoding mode, since it must store $\lambda_k[b]$ and $\lambda_k[c(e)]$ values. To limit the area overhead and speed up the loading process, four $\lambda_k[c(e)]$ values are stored in three memory locations, as portrayed in the left part of Fig. 3.15. Three 4-bit $\lambda_k[c(e)]$ are stored in three 6-bit locations: the remaining metric can be divided in two pairs of bits, and stored in the leftover locations. Three clock cycles are used to read the four $\lambda_k[c(e)]$ values for a trellis step with minimal logic overhead. In case of SBTC, like those used in 3GPP-LTE, only two $\lambda_k[c(e)]$ and one $\lambda_k[b]$ are necessary for a trellis step, and they can be read in two clock cycles without impairing the throughput.

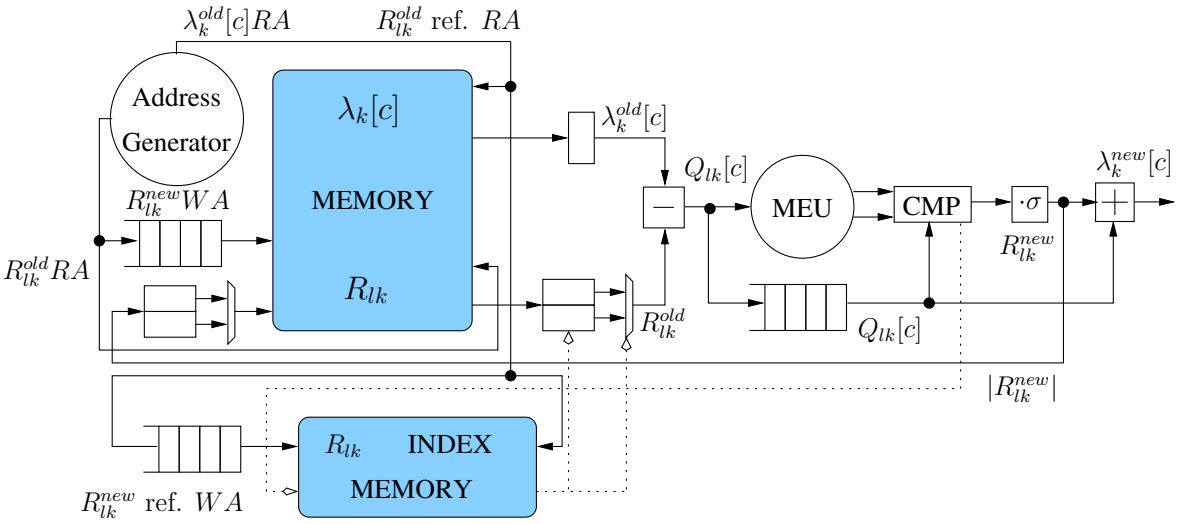

Figure 3.16: LDPC decoding core

With a similar method the 2-bit memory is used in turbo decoding mode to store $\beta_k[s]$ and $\alpha_k[s]$ between iterations, as suggested in [70]. Six locations are used to store 2 $\beta_k[s]$ or $\alpha_k[s]$ (Fig. 3.15): since at most three 8-state windows initialization metrics, i.e. 24 $\beta_k[s]$ and 24 $\alpha_k[s]$, are stored at the same time, only 144 out of 400 locations are used. Multiple memory accesses are necessary to read a single value: the issue is handled with appropriate scheduling (see Section 3.4.1) and does not affect the throughput.

LDPC Decoding Core

The LDPC decoding core used in the decoder described in [13] relies on a serial architecture suited for exclusive memory usage. The main drawback of this solution is the variable number of cycles to produce the output. The average number of cycles–per–data varies between one and two. To overcome this limitation and to share the memory with the SISO decoder a novel architecture with limited area overhead is proposed.

The LDPC decoding core is detailed in Fig. 3.16.: this architecture supports all kind of LDPC codes, as long as the memory requirements are met.

A $Q_{lk}[c]$ value (2.6) is produced at every clock cycle and fed to the Minimum Extraction Unit (MEU) depicted in Fig. 3.17. Then, $|Q_{lk}[c]|$ is compared to the current first and second minimum (min1 and min2), that are initialized as the maximum allowed value at the beginning of each CN phase. The minimum of both comparisons (min1_{new} and min2_{new}) is passed on and sampled on the rising edge of the clock signal, together with the previous first minimum (min1_{old}) and a flag signaling if $\text{min1} \neq \text{min1}_{new}$ (min1_{update}). If $\text{min1}_{update} = 0$, min2_{new} is substituted with min1_{old}: min1 and min2 are finally updated on the falling edge of the clock, ready and stable for the next $|Q_{lk}[c]|$. Differently

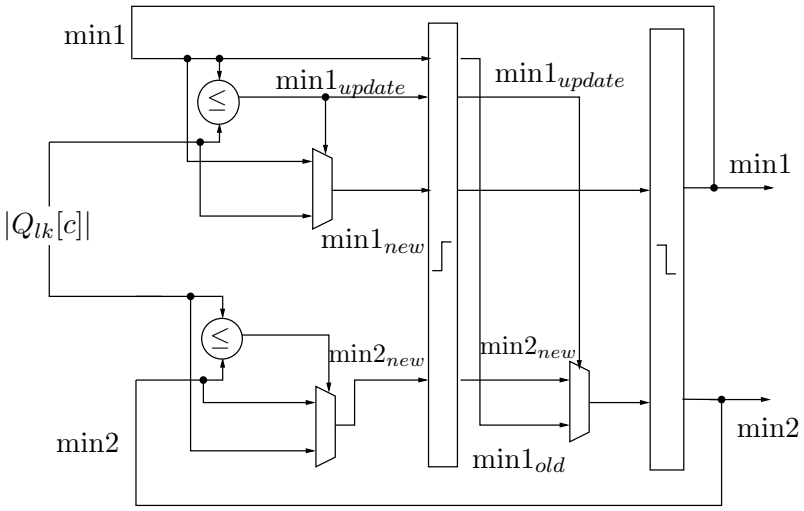

Figure 3.17: Minimum Extraction Unit

from the MEU used in [13], that could halt the pipeline in case both min1 and min2 had to be updated, the negative–edge triggered registers allow both updates in a single clock cycle, leading to a constant cycles–per–data rate close to one, after the initial deg+2 cycles latency. Concurrently, $Q_{lk}[c]$ signs are XORed as in (2.8).

Once min1 and min2 have been successfully extracted, they are compared to all the $Q_{lk}[c]$ of the CN, that are delayed by a number of clock cycles equal to the degree of the CN (deg), to compute R_{lk}^{new} as in (2.9). The CMP unit handles the comparison and produces the two flags (sign and identification) to be stored in the R_{lk} index memory. The correction factor σ in (2.12) is applied before the final addition in (2.10) and $\lambda_k^{new}[c]$ are sent to the output buffer.

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

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

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

storing R_{lk}^{new} indexes of CN j.

Turbo Decoding Core

As for the LDPC decoding core, also the SISO decoder yields a very high degree of flexibility, limited only by the size of the memories: any 8-state double-binary turbo code can be decoded as long as the memory capacity is sufficient.

The overall structure of the core architecture is the same of Fig. 3.3. The SISO interfaces with the NoC via two dedicated input and output blocks, i.e. the BTS CU and STB CU [38] already mentioned in Section 3.2. The BTS-CU changes the received *a priori* BL $\lambda_k^{apr}[b]$ to SL $\lambda_k^{apr}[u]$, as required by the algorithm, while the STB-CU reduces the number of messages to be sent on the NoC by converting $\lambda_k[u]$ *extrinsic* values into BL $\lambda_k[b]$.

For every trellis step in a window, the Branch Metric Unit (BMU) and the Extrinsic Computation Unit combine the two $\lambda_k^{apr}[b]$ converted by the BTS-CU with four $\lambda_k[c(e)]$ values read from the 6-bit memory to calculate $\gamma_k[e]$ (2.5) and to update $\lambda_k[u]$ respectively. The output of the BMU is used by the main computation unit, that tackles the calculation of $b(e)$, $\alpha_k[s]$ and $\beta_k[s]$ (2.2), (2.3) and (2.4). These metrics are computed in this exact order, thus storing $\beta_k[s]$ values in a dedicated set of registers while $\alpha_k[s]$ are being processed: the $b(e)$ metric, that needs both $\beta_k[s]$ and $\alpha_k[s]$, is calculated last.

In turbo mode, each trellis step requires three clock cycles to be completed. However, up to five cycles are needed to read all the necessary $\lambda_k^{apr}[b]$ and $\lambda_k[c(e)]$. Early simulation results presented in [13] show that the SISO working frequency (f_{core}^{turbo}) can be lower than the NoC's one (f_{NoC}^{turbo}). By timing the memories and their relative address generation circuitry with the faster clock signal, six values (five memory locations) can be read from port 1 of the 6-bit memory in three SISO-cycles. Port 2 is kept in write mode for the duration of the decoding, and used to store values coming from the network.

The 2-bit memory is used in the same way, with port 1 in read mode and port 2 in write mode. At the beginning of every new window 16 values are needed (8 $\alpha_k[s]$ and 8 $\beta_k[s]$) from this memory to initialize the trellis. Due to the memory organization used for state metrics, see Fig. 3.15, one $\alpha_k[s]$ and one $\beta_k[s]$ are spread over 7 memory locations. Consequently, $7 \cdot 8$ clock cycles are necessary to load the 16 values. The values must be loaded from the memory before the window is processed. Thus, they are loaded during the processing of the previous window. Setting the minimum window size to 20 trellis steps, the computations require $3 \cdot 20$ clock cycles, and there is enough time to load $\beta_k[s]$ and $\alpha_k[s]$ values to initialize the next window.

3.4.2 Supported Standards

The 22-node architecture developed in the previous sections has been tested on a large set of communication standards. In particular, the whole set of turbo and LDPC codes included in [3–6, 8, 9, 71] have been tested.

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

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

- *IEEE 802.16e*: the WiMAX standard [3], is fully supported. A high enough throughput (> 70 Mb/s) can be obtained with f_{core}^{LDPC} = 200 MHz and f_{core}^{turbo} = 80 MHz. Table 3.6 summarizes the results.

Table 3.6: IEEE 802.16e standard throughput (T), 10 iterations for LDPC, 8 for Turbo

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

- *IEEE 802.11n*: IEEE 802.11n standard [4] requires a higher throughput than WiMAX, demanding for the N = 1944, r = 5/6 code up to 450 Mb/s. The 22–node architecture can guarantee it with f_{core} = 820 MHz. Taking in account the f_{core}/f_{NoC} ratio constraint, this would mean f_{NoC} = 1.23 GHz. Both frequencies are over the decoder maximum working frequency, and two alternatives have been devised. By increasing the size of the NoC to 35 nodes, the f_{core}/f_{NoC} still holds at 2/3, but f_{core} can be lowered to 520 MHz, and f_{NoC} = 780 MHz. By choosing f_{core} = 350 MHz, support still can be given for other WiFi transmission modes, requiring up to 300 Mb/s. The results are shown in Table 3.7.

- *DVB-RCS*: the return channel for DVB satellite communications [71], thought for multimedia applications, employs 12 different payloads and 7 coding rates. The throughput required by this standard is very small, and can go up to 2.05 Mb/s in case of corporate-driven applications.

Table 3.7: IEEE 802.11n standard throughput (T) with different f_{core} and NoC sizes, 10 iterations

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

Table 3.8: Reconfiguration cases in intra- and inter-standard combinations. Dark gray: percentage of code combinations requiring decoder pausing. Light gray: percentage of code combinations requiring $0 < N_f \leq 2$. White: all code combinations reconfigurable with $N_f = 0$. Throughput obtained with 10 iterations for LDPC, 8 for Turbo

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

This throughput is easily sustained by the 22-node architecture with f_{core} = 3 MHz.

- *HomePlug AV*: the HPAV standard [6] makes use of a small set of DBTC, with interleaver sizes of 64, 544 and 2080. The throughput requirements of HPAV demand at least 150 Mb/s: on the 22-node architecture, with f_{core} = 170 MHz, achieved throughput is 156 Mb/s.

- *CMMB and DTMB*: the CMMB [8] and DTMB [9] Chinese broadcast standards, though serving the same purposes as DVB, work with smaller LDPC codes. Like in DVB, also in CMMB codes feature double diagonal submatrices, slightly limiting the concurrent number of row nodes that can be instantiated on the proposed decoder. Both CMMB and DTMB codes demand an increased memory capacity with respect to the aforementioned standards, requiring PE memories to be enlarged by 55% to support CMMB, and by 68% for DTMB. A working frequency f_{core} = 60 MHz is sufficient to guarantee the 20.22 Mb/s throughput required by CMMB standard, while to comply with DTMB 40.6 Mb/s, frequency must be risen to f_{core} = 200 MHz, as shown in Table 3.9.

- *3GPP-LTE*: the LTE version of 3GPP [72] uses a set of 188 SBTC with coding rate 1/3, thus being characterized by a range of widely spaced block lengths. The required 150 Mb/s throughput can be obtained on the 22-node architecture with f_{core} = 330 MHz; however, if

Table 3.9: CMMB and DTMB standard throughput (T), 10 iterations

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

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

Table 3.8 summarizes possible switching among the selected standards, taking in account all possible code combinations. The dark gray cells represent the percentages of C_1, C_2 combinations between two standards whose reconfiguration requires pausing of the decoder. A few cases arise between DVB-RCS and WiMAX turbo codes and within 3GPP-LTE (due to its wide variety of codes), while when C_2 belongs to the CMMB, DTMB and LTE standards, it is more likely to encounter a critical combination. On the contrary, if C_1 belongs to CMMB or DTMB standards, any reconfiguration can be completed with N_f = 0 (with N_f the number of additional frames on which to spread the reconfiguration process): this is also the most common situation among the other standards. The choice of maximum N_f = 2 allows to handle all the other reconfiguration cases: the light gray cells show the percentages of code combinations in which $0 < N_f \leq 2$ is necessary.

3.4.3 Implementation Results

The results presented in Section 3.4.2 show a broad range of possibilities for implementation, and the designed decoder can be scaled with very low effort. Three different complete decoders have been synthesized with TSMC 90 nm CMOS technology: post-layout results have been obtained for all of them, with accurate functional verification, area and power estimation. Synthesis has been carried on with Synopsys Design Compiler, functional simulation with Mentor Graphics ModelSIM, and place and route with CADence SoC Encounter.

Implementation A

The first decoder implementation has been devised to fully support WiMAX, HPAV and DVB-RCS standards. The memory sizing and organization described in Section 3.4.1 is able to handle the

addressed standards with 22 PEs. To comply with each standard throughput requirements, a single f_{NoC} = 300 MHz is sufficient in both LDPC and turbo mode, consequently identifying f_{core}^{LDPC} = 200 MHz and f_{core}^{turbo} = 170 MHz, both under the f_{core}/f_{NoC} constraint. Obtained throughput is presented in Table 3.10.

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

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

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

A screenshot of the final layout is portrayed in Fig. 3.18: the irregularity of the placement is due to the large number of memories and their complex interconnections. However, two different areas can be easily identified: a central zone in which most of the logic is found (black contour), and a border area where the majority of memories have been placed, some of which are highlighted with a white line.

Implementation B

The second implementation presented extends the set of standards supported by implementation A to WiFi LDPC codes and 3GPP-LTE turbo codes. To limit the complexity of off-chip clock generators, also in this case a single NoC working frequency has been chosen, f_{NoC} = 780 MHz, while f_{core}^{LDPC} = 520 MHz is necessary to provide high enough throughput, f_{core}^{turbo} can remain set to 200 MHz. The parallelism of the NoC is increased from 22 nodes to 35 nodes, the reconfiguration buses rise from 5 to 8, and the support of LTE requires an increase in the size of 6-bit memories. Throughput results are reported in Table 3.10. The post place & route estimated area is 4.87 mm^2, with 331.6 mW of power consumption in LDPC mode, and 183.2 mW in turbo mode.

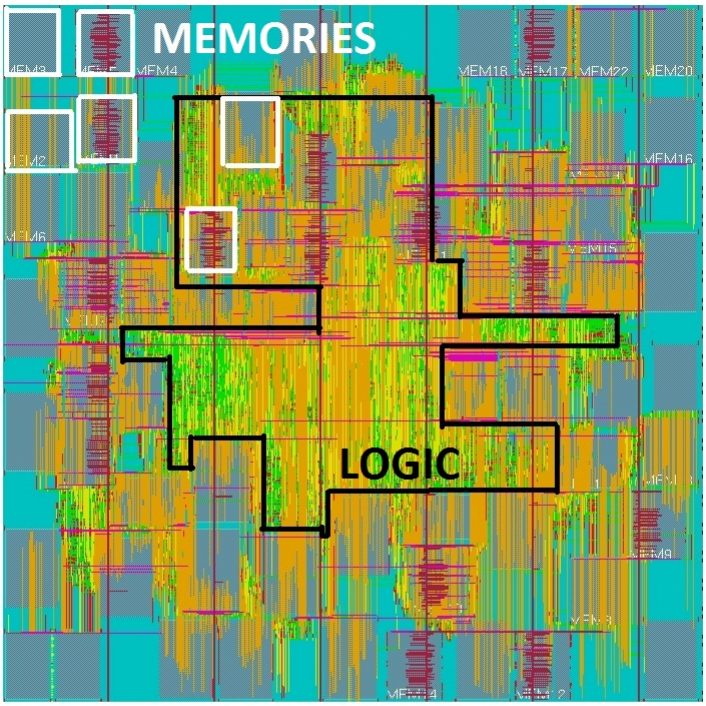

Figure 3.18: Implementation A layout screenshot

Implementation C

This third implementation extends implementation A's support to CMMB and DTMB. Neither frequency nor NoC parallelism modification are necessary, but the core and reconfiguration memories must be enlarged. Consequently, an extra bit is added to the reconfiguration bus data width. The new post place & route estimated area is 3.42 mm^2, while power reaches 120 mW for both tested turbo and LDPC codes. This is because the LDPC consumption is calculated on a DTMB code, that makes full use of the extended memories, while the memory usage percentages for DBTC remains low. The enlarged memories allow also LTE codes to be decoded, but the SBTC f_{core} would need to rise up to 333 MHz to meet the throughput requirements. Throughput results for CMMB and DTMB are shown in the Implementation C column of Table 3.10.

Comparisons

Table 3.12 shows the detailed implementation results in comparison with recent related state of the art flexible turbo/LDPC decoders. Even though A, B and C are the only decoders capable of dynamic switching, area, power and efficiency figures prove the effectiveness of this approach.

In order to make a fair comparison, normalized area occupation has been included in the table, $An_{tot} = A_{tot} \cdot (65/Tp)^2$, where A_{tot} is the total area and Tp is the technology process. Throughput and power consumption are also compared. Moreover, two further metrics have been introduced: the

Table 3.10: Throughput (T) results for each standard, with every implementation

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

Table 3.11: Area efficiency (A_{eff}) for different codes and implementations. N/A: code not supported. Dash: results not available.

	A_{eff} $[\frac{bits}{mm^2 \cdot kcycles}]$						
CODE	A	B	C	[48]	[49]	[51]	[52]
LDPC						(min)	
2304, 1/2	2447	2188	1966	882	–	2013	–
2304, 5/6	3846	3445	3090	4412	9589		10779
1944, 5/6	3846	3445	3090	3719	10363	3484	8982
7493, 4/5	N/A	N/A	1910	N/A	N/A	N/A	N/A
Turbo						(min)	
DB 2400	5134	4598	4124	1468	750	2084	N/A
SB 6144	N/A	2362	2062	1468	375	N/A	3233

energy efficiency $E_{eff} = Pow/(T \cdot \mathrm{It}_{max})$, where Pow is the peak power consumption, expressing the energy spent for decoded bit, and the area efficiency $\mathrm{A}_{eff} = (T \cdot \mathrm{It}_{max}/f_{clk}) \cdot (1000/\mathrm{An}_{tot})$, reported in Table 3.11, an efficiency figure that considers both throughput and area occupation.

Baghdadi *et al.* in [48] propose an ASIP decoder architecture supporting WiMAX and WiFi LDPC

Table 3.12: LDPC/Turbo architectures comparison: Decoder parallelism P, CMOS technology process (TP), processing area occupation (A_{core}), total area occupation (A_{tot}) normalized area occupation for 65nm technology (An_{tot}), clock frequency (f_{clk}), peak power consumption (Pow), energy efficiency (E_{eff}), data width (DW), maximum number of iterations (It_{max}), code length (N) and rate (r), interleaver size (K) and throughput (T)

Decoder		[2] **A**	[2] **B**	[2] **C**	[48]	[49]	[51]	[52]
P	LDPC DBTC	22	35	22	8	1	12	12
Tp $[nm]$	LDPC DBTC	90	90	90	90	65	45	90
A_{core} [mm^2]	LDPC DBTC	2.19	3.83	2.56	2.44	N/A	N/A	1.18
A_{tot} [mm^2]	LDPC DBTC	2.75	4.87	3.42	2.6	0.62	0.9	3.20
An_{tot} [mm^2]	LDPC DBTC	1.43	2.54	1.78	1.36	0.62	1.88	1.67
f_{clk} [MHz]	LDPC DBTC	200[1] 170[1]	520[1] 200[1]	200[1] 170[1]	520	400	150	500
Pow [mW]	LDPC DBTC	87.8 101.5	331.6 183.2	118.6 121.6	N/A	76.8	86.1	N/A
E_{eff} $[\frac{nJ}{bit}]$	LDPC DBTC	0.125 0.081	0.073 0.078	0.174 0.097	N/A	0.032 0.826	0.151 0.147	N/A
DW [bits]	LDPC DBTC	6 - 5 6 - 4	6 - 5 6 - 4	6 - 5 6 - 4	7 - 5 8 - 6	7 - 5 8	7 - 5 7 - 5	9 - 6 9 - 6
It_{max}	LDPC DBTC	10 8	10 8	10 8	10 6	10 5	8 8	15 6
N, r K	LDPC DBTC	2304, 1/2 2400	1944, 5/6 2400	7493, 4/5 2400	2304, 1/2 2400	2304, 5/6 2400	N/A N/A	2304, 5/6 SBTC 6144
T [Mb/s]	LDPC DBTC	70 156	455 292	68 156	62.5 173	237.8 37.2	71.05 73.46	600 450

[1] f_{core}

[2] post–layout results

codes, and WiMAX, 3GPP-LTE and DVB-RCS turbo codes. The A, B and C implementations are designed such that the minimum throughput is sufficient to comply with the supported standards. On the contrary, worst case throughput in [48] is not high enough for WiMAX. Comparison reveals similar area occupations, but very different frequencies. This leads to a better area efficiency in all three proposed implementations for most of the codes: particularly evident is the difference for DBTC (second last row of Table 3.11).

The work presented in [49] supports convolutional, LDPC and turbo codes, giving results for WiMAX LDPC, WiFi and general binary and double-binary turbo codes. It yields a very small area occupation with low power consumption and good maximum throughput for LDPC decoding. On

the contrary, it features less interesting figures in turbo mode. This situation is reflected both on E_{eff} and A_{eff}, with Implementation A, B and C having, when comparing the same codes, better efficiencies in turbo mode (last row of Table 3.11), and worse in LDPC mode. However, under the worst case conditions (N=672, r = 1/2, 20 iterations), A and B outperform [49] also in LDPC mode.

The multi-standard decoder designed in [51] supports 3GPP-HSDPA, WiFi, WiMAX and DVB-SH. No specific information on the codes used is given, only minimum guaranteed throughput: for this reason, results in Table 3.11 refer to the minimum throughput of each standard. Implementation A and B have comparable minimum A_{eff} when working with WiMAX LDPC codes, and A, B and C yield much better results in turbo mode. When comparing WiFi results [51] guarantees a higher A_{eff} than A, B and C, even though aiming for a lower throughput than B. All three proposed implementations yield better E_{eff}, and both A and C have a smaller area occupation.

Sun and Cavallaro describe in [52] a decoder working with 3GPP-LTE turbo codes and WiMAX and WiFi LDPC codes. They obtain very high maximum throughput efficiency in both LDPC and turbo mode: the range of supported codes is however quite limited w.r.t. all considered implementations, and the area occupation is larger than A. Since no power analysis is given, comparison based on E_{eff} is impossible, although the difference in working frequencies would suggest a smaller power consumption for at least A and C.

3.5 Joint application and communication NoC simulator

In a general purpose System-On-Chip (SoC), one or multiple applications are mapped on a number of PEs, and communication among them is handled by interconnects. Advanced interconnects use the NoC concept, which potentially guarantees very high flexibility and better scalability with respect to traditional bus based solutions. In the NoC design paradigm, processing is separated from communication, meaning that supported applications can be developed and optimized independently of the underlying interconnect structure. On the other hand, NoC design choices are made with the purpose of matching the communication requirements (e.g. latency, aggregate throughput, FIFO sizes, etc.) of a large variety of applications. While this separation is still important to facilitate development of new applications and reduce the time to market of new products, the original idea of general purpose NoCs has evolved and new kinds of NoCs, like Application-Specific NoCs (ASNoCs) targeting a single application, are today proposed with features fully optimized for a single or reduced number of processing needs. In this context, the separation paradigm looses part of its meaning and proper design methods and tools, involving both communication and processing sides must be proposed to support the development of efficient ASNoCs.

In fact, some cases arise where cycle accurate joint simulation of PEs and NoC is necessary to achieve efficient application specific NoC-based VLSI architectures. We distinguish three kinds of developments, to clarify the possible use of such a joint simulation environment:

1. new NoC designed for an already existing set of PEs associated with a given application
2. new application with new PEs to be interconnected by means of an already available NoC
3. new NoC and a new application, supported by new PEs.

In the first case, the inter PE communication needs are extracted from the application and used to drive the NoC design. Available simulation tools already support this kind of design flow, where effects of application on NoC performance must be verified. We can symbolically represent this dependency as: PEs → NoC. As an example, in [73] a complex digital TV SoC is extended by replacing the original interconnect structure with a programmable NoC, which provides improved flexibility and extends the lifecycle of the product. On the contrary, in case two, choices on the processing side depend on the capabilities of the available NoC: in this case we have a NoC → PEs dependency. Finally, the third case refers to the design of a new ASNoC, which is usually addressed exploiting the separation principle: first PEs are developed to be able to run in a distributed way the application, then a proper NoC is derived, starting from the specific communication needs coming from the PEs. In other words,

the case three development is usually organized around a PEs → NoC dependency. We believe that, in some cases, better results can be achieved if PEs and NoC are jointly designed, taking in account both the effects of generated inter PE traffic on NoC performance [74], and the effects of NoC design choices on the overall application performance. We can represent this inter dependency as PEs ↔ NoC. A key tool to support joint PE and NoC design is a high level simulation environment including proper modeling capabilities of both PEs and NoC.

In this section, starting from the OMNET++ library [75], a new simulation environment, namely JANoCS, is proposed, which allows for cycle-accurate joint simulations of both application and communication in NoC-based SoCs. Simulations are run at high level and do not require RTL modeling. Its usage is fit for early evaluation of design choices for all kinds of NoC-based SoCs, in particular those with custom PEs, complete design of multiprocessor ASICs, and evaluation of feasibility in case a new application is mapped on existing hardware. While an example design case is portrayed in this section, the proposed simulator is applied to the turbo and LDPC cases in Section 3.6 in order to explore and evaluate new power reduction methods for NoC-based channel decoders. JANoCS has been presented in [15].

3.5.1 Proposed simulation environment

The proposed simulator has been built on the concept that the simulation of PEs and interconnects must be handled at the same time, to be able to observe the impact of the network dynamic behavior on the application mapped over the PEs. Regardless of the wide and diversified range of possible applications, common characteristics allow to draw a list of features that the simulator should have:

- a high-level language must be used to simplify the modeling of the system and to improve design productivity;
- the overall structure should be as modular as possible, allowing fast and easy changes to the model, and guaranteeing a high degree of reusability and flexibility;
- hardware characteristics should be modeled with functional behavior and timing details only, to keep simulations fast without sacrificing precision.

An effective starting point for these purposes is the OMNET++ simulation library [75], an open-source C++ based environment targeting network simulation. It is composed of two different parts:

1. a set of C++ classes used to represent network elements (both computational hardware and data structures) and useful methods, which can be used to model the behavior of the network itself;

2. a description of the structure through a NEtwork Description (NED) language; here modules are interconnected, specifying channel types, data rates, topologies.

An initialization file serves as a parameter definition method.

The OMNET++ information exchange method is one of its key features: thanks to the `cMessage` class, modules can trigger behavioral responses when different kinds of messages are received. These are moved between modules through NED-defined channels: statistical data can be easily obtained and plotted by tweaking a set of customizable parameters. Cycle-accurate behavioral simulations and detailed analysis of the status of the network can easily be obtained by joining the inset message-triggered responses to models of the channels' timing details. It is consequently possible to evaluate latencies, congestion, data loss and many other dynamics.

With OMNET++ as a foundation, a Joint Application and Network-on-Chip Simulator (JANoCS) has been proposed, targeting the above listed characteristics. A completely parameterized NED structure of the NoC has been developed, with the definition of the topology, the number of nodes, the direction of the channels and their delays occurring in the initialization file. An example of JANoCS parametrized network definition is shown in Figure 3.19. The topology of the NoC is passed as first parameter: each construction file is associated to a different supported topology, defining the connection among nodes accordingly. Once the topology has been specified, variables common to all topologies are given to the construction file, like the number of nodes, the number of connections (degree) and the type of channels to be used for each connection.

The nodes of the NoC are instantiated as generic JANoCS blocks, *i.e.* compound modules constituted of a Routing Element (RE), a PE and a network interface (NI). Another set of user-defined parameters is passed to these compounds, in which the modules to be used are specified. In Fig. 3.19 a router implementing XY routing is chosen, while a further set of parameters clarifies characteristics that can either be common to all routers or particular to the selected module, like collision management policies or the definition of the size of the input buffers. The same happens for the NI block. The simulated application is defined in the PE module: according to the type of processing performed, each possible PE requires a dedicated set of parameters.

A clock generator module with user-defined period has been created: it relies on the `cMessage` class, and triggers responses in all the clocked modules. One or more latency parameters are defined in each instantiated module, and are expressed in number of clock cycles. These latencies are used to simulate the hardware structure within the models and to take in account packet communication latencies, with no need for a more detailed datapath description.

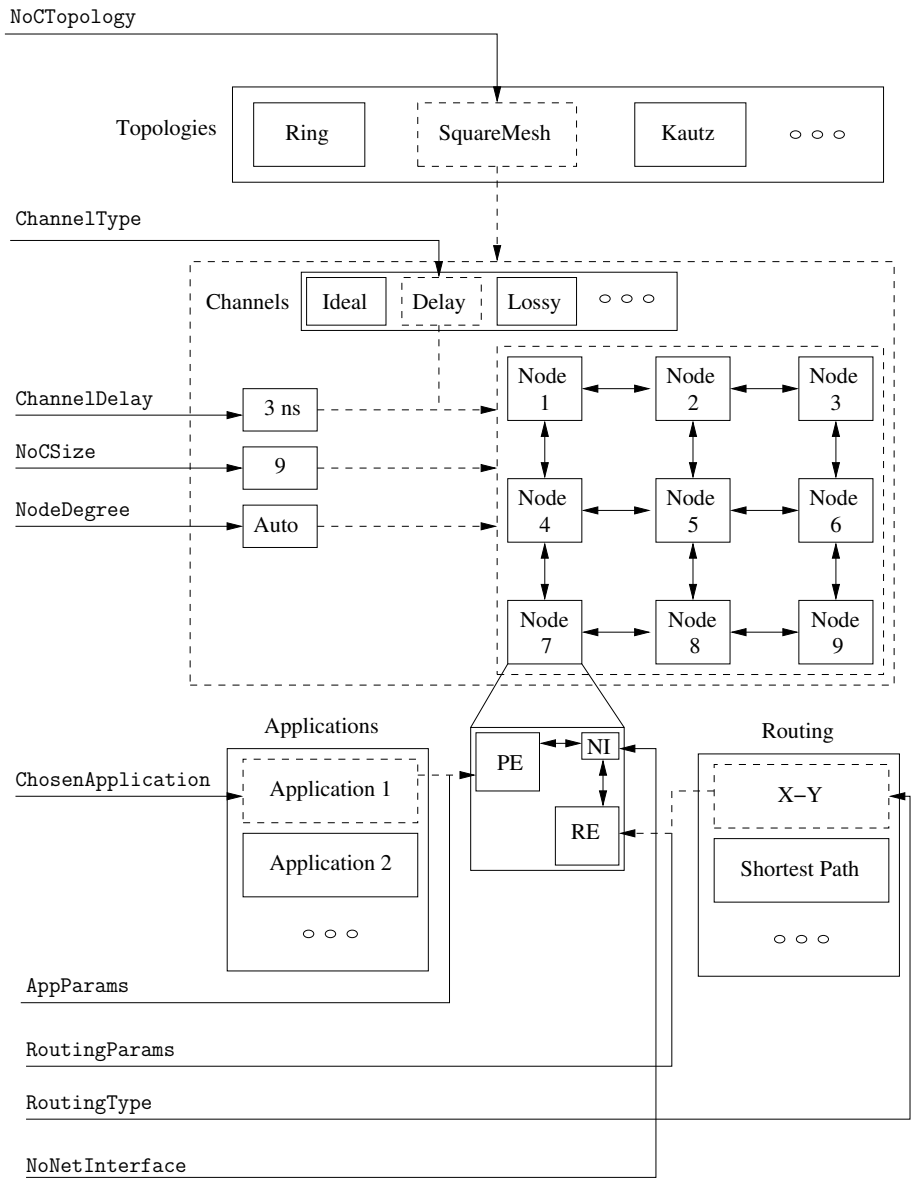

Figure 3.19: JANoCS network definition via parameters

This extremely modular structure of the system makes very simple the introduction of new modules and their customization. Each block within a compound module is instantiated as a generic class, and consequently identified by the user as a subclass of the instance. For example, both `XYRouter` and `ShortPathRouter` classes are subclasses of `RoutingElement`, with inherited parameters common to all REs, and new ones specific of their subclass.

The potential of JANoCS can be better explained through an example. For this purpose, an optimization task based on a genetic algorithm has been mapped on a NoC with `NoCSize`=16. This example situation, named *barrel construction*, is very simple, but does not alter the generality of the concept.

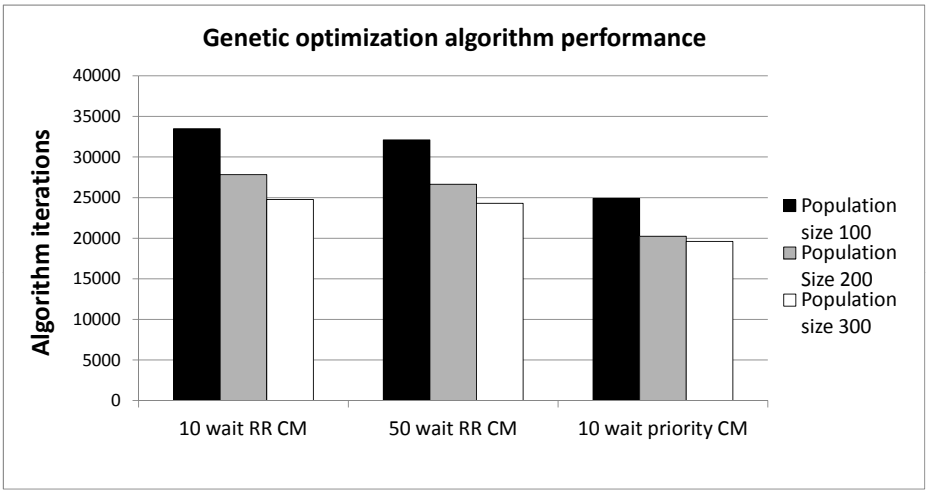

Figure 3.20: Performance of genetic algorithm based barrel construction

The goal of the optimization task is building the largest barrel possible, using a fixed number of planks and wasting as little material as possible: having very long planks when a short plank is present is considered a waste of resources. In genetic algorithms the fitting function expresses how good each member of the current population is: in this particular problem it has been defined as

$$f_{fit} = \frac{\sum_{i=0}^{p} l_i}{l_{max} - l_{min}} \tag{3.16}$$

where l_i is the length of the i^{th} plank in the barrel, the numerator is the whole size, while the denominator expresses the wasted wood.

Each PE is modeled to hold a small randomly generated initial population of `PopulationSize` barrels: new barrels are created through crossover and mutation (with probabilities `CrossoverProbability` and `MutationProbability`). Then the PEs broadcast a few of the best barrels to all the others PEs, speeding up the improvement process. The PEs are coded to wait `StallingTime` cycles after the last barrel has been sent, to allow the delivery of the barrels: when they have expired, another iteration (crossover - mutation - broadcast) starts. Iterations are stopped when a solution with fitting value higher than `FittingThreshold` is found: for this example, `FittingThreshold`=1000. A manager module can conveniently control the status of the application and order the start of iterations: dedicated `cMessage` subclasses can be used as control messages.

Communication is enforced via a 4×4 square mesh NoC with XY routing: `NodeDegree` is automatically set to 5, while `ChannelType` is set to Ideal. Routers can easily be built from basic `cQueue` blocks representing FIFOs, with functions implementing the actual routing decisions, timed by latency

parameters. Change of routing algorithm or arbitration method implies minimal changes to the code.

Joint simulations allow to evaluate the importance of inter-PE communication and possibly to improve the design towards a faster convergence to a good solution. In Figure 3.20 the number of algorithm iterations required to reach the fitting threshold is shown for different design choices. The first three histogram bars refer to a NoC implementing a Round-Robin Collision Management policy (RRCM), barrels built out of 12 planks, `StallingTime`=10 clock cycles, `CrossoverProbability`=30%, and `MutationProbability`=0.5%. The black bars refer to a population size of 100 elements, while the gray and white bars are for 200 and 300 elements respectively. Larger populations usually lead to larger room for improvement through crossover and mutation, thus explaining the faster convergence towards the desired fitting threshold: on the downside, they require additional PE resources and longer iterations. The second block of solutions considers an extended `StallingTime` of 50 cycles: it can be seen that although small improvements can be noticed, the congestion of the network is too severe, and would require a much longer time to be drained. In the third block, a 10-cycle `StallingTime` is assumed again, but priority is introduced in the handling of packets (priority CM), particularly in the management of collisions at the routers. The convergence is faster of averagely a 25% factor thanks to the higher number of best barrels reaching destination, and improving the population of each PE.

These results are meaningful for both the interconnection and the processing parts of the MP-SoC: the interdependence between the application and the network is evident, and joint simulation is necessary for both validating the design and reducing its overhead.

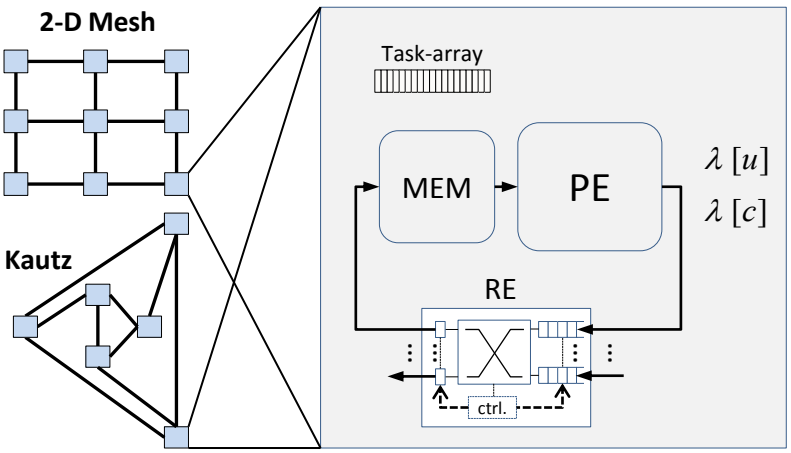

Figure 3.21: NoC-based parallel decoder structure

3.6 Power reduction methods for NoC-based channel decoders

General power reduction techniques, like clock gating and dynamic voltage scaling, can be applied to channel decoders. Additionally, specific features of LDPC and turbo decoding can be exploited to reduce power dissipation. As an example, since LDPC and turbo decoding are iterative processes, early stopping of iterations criteria have been proposed over the years [76, 77]: these techniques rely on the observation of a metric to decide if it is worth or not to perform additional iterations, avoiding unnecessary energy consumption. The power consumption of NoC-based decoders is usually much higher than that of dedicated channel decoders: moreover, the solutions presented in this chapter make use of high NoC frequencies, that increase the power consumption gap even more.

This section describes new power reduction techniques for flexible channel decoders. These techniques reduce and optimize the traffic due to messages exchanged among PEs, which account for a significant percentage of the consumed energy. Therefore the proposed methods are particularly effective with NoC-based decoders. Preliminary contributions in this direction have been made in [78] for turbo codes, and in [12] for LDPC codes, both dealing with the evaluation of the usefulness of exchanged information. The performance of the proposed techniques has been extensively evaluated and compared to alternative methods, while the most promising one has been implemented as an application example on one of the decoder implementations presented in Section 3.4. The presented work has been submitted to Springer Circuits, Systems & Signal Processing [16].

3.6.1 NoC-based decoding: practical issues

In parallel decoders, the decoding process of the received frame is typically partitioned among P PEs. Messages are exchanged among PEs by means of an interconnection structure, that is usually

deterministic, and guarantees fixed and uniform latency. To increase the degree of flexibility of the decoder, recent works have proposed NoCs as interconnection structures [11, 14, 60, 79]. Fig. 3.21 shows the basic structure of a NoC-based decoder. While in turbo decoders the PEs are concurrent SISOs executing (2.1) on different sub-blocks, in LDPC decoders each PE updates the LLRs involved in a certain set of parity check constraints. Consequently, the task array of each PE, i.e. the sequence of processes to be performed within an iteration, is a continuous set of trellis steps in the turbo case, and a uniform selection of parity checks from all $\mathbf{H}$ layers in the LDPC case. Each PE is connected to a Routing Element (RE), which in turn is linked to a number of other routers, with input buffers at every port. If every RE has an attached PE the NoC has a direct topology (like the 2-D mesh and the generalized Kautz [80] shown in Fig.3.21), while in indirect NoCs (e.g. Benes [81]) some REs are only used as intermediate communication nodes. The NoC traffic is constituted of $\lambda_k[u]$ (2.1) and $\lambda_k[c]$ (2.10) values for turbo and LDPC codes respectively, as shown in Fig. 3.21: messages are injected in the NoC directly by the PEs, while information received from the channel is stored in a memory for further use. The communication pattern with a NoC is deterministic, but the delays introduced are nonuniform and can vary consistently from code to code and with time. This nonuniform distribution of delays is basically due to the uneven distance among PEs. Choices like number of PEs, NoC topology and routing algorithm have a significant impact on achievable throughput and implementation complexity, as extensively studied in [11, 14]. In general, a NoC used to support turbo and LDPC decoding is a particular type of NoC, characterized by very low complexity and reduced functionalities in comparison to usually considered NoCs. For example, since exchanged messages are usually six to ten bit values, NoC packets are single phits, sent using source routing. Moreover, the inter-PE communication patterns exhibit a pseudo-random nature, because of the limited adjacency of both interleavers used in turbo codes and parity check matrices associated to LDPC codes. Thus, the optimal mapping of processing tasks onto PEs does not provide any relevant benefit in terms of NoC traffic [82].

Additional techniques to optimize the NoC traffic are available. In particular, Quality-of-Service (QoS) oriented networks can be considered for turbo and LDPC decoding. In [83], an area- and energy-wise breakdown of different router architectures is presented, providing a comprehensive overview of NoC designs. The CS network and the GuarVC network [83] both target QoS improvement: the first one is a circuit-switched network that relies on simplicity and static allocation of resources, while the second makes use of multiple virtual channels and flow control: priorities are assigned to packets and the traffic flow is optimized. However, CS networks are not effective for channel decoding, because the traffic pattern between PEs is not regular enough. Moreover, both circuit-switched and virtual

channel-based NoCs tend to introduce large area and power consumption overheads: based on the results in [83], the GuarVC and CS network have been applied to our case. A 22-PE NoC (the same choice made in Section 3.6.5) implemented as a CS network would be slightly larger than the whole flexible decoder in [51] and its power consumption would be higher by a 2.4 factor. The same NoC, implemented as a GuarVC network, leads to 1.3 times larger area and 2.9 times higher power consumption. Therefore, usual techniques to reduce traffic in NoCs are not effective in the case of NoC-based decoders and dedicated solutions are required.

The complex interactions between the topology of the NoC, the number of PEs, the code and the performance of the decoder have been analyzed with JANoCS 3.5, for which LDPC and turbo PEs were modeled along with a custom NoC.

The LDPC PE is constituted of a processing module and two memory modules: the data memory is used to store the incoming messages, while the interleaving memory is initialized at startup, and contains the destination of the outgoing messages. The memories are parametrized blocks that load and store `cMessage` objects and their subclasses. In LDPC decoding modules, the `LDPCMessage` class represents the packets traveling on the NoC. Since the user can decide the width and depth of the memories at initialization time, a single model can be used in very different implementations, and in both turbo and LDPC codes PE. Thus, the `LDPCMessage` and `turboMessage` classes can be modified without requiring a remodeling of memories.

The processing module of the LDPC PE simulates the implementation of the serial pipelined structure similar to the architecture described in Section 3.4. The data needed for the processing of a task are loaded from the data memory: concurrently, the messages computed within the former task are injected in the NoC. The actual computation itself is seen as instantaneous from the simulator point of view, occurring just after the last message has been retrieved from the memory: however, latency parameters are used to model the memory access and the processing hardware. In particular, a set of latencies are used to build the outgoing message's queue, while another is used to simulate the memory access requests.

The turbo codes decoding module is characterized by two operating modes (fw/bw and butterfly), the selection of which is handled at initialization time via parameter. Each of them supposes different hardware resources, resulting in diverse packet injection rates and achievable throughputs [84].

- `fw/bw`: this operating mode models the presence of a single core in each SISO. Each mapped task is divided into two halves, and will take $2 \times W$ cycles to be completed, while messages will be injected in the network only during the last W cycles, one every clock cycle.

- `butterfly`: supposing the presence of two cores within each SISO, this operating mode allows

both halves of each task to be run concurrently. Every one can be completed after W steps: messages begin to be ready for sending after $W/2$ cycles, two every clock cycle. The shortened task duration is a big step towards higher throughput, but the higher packet injection rate can cause more critical traffic patterns in the NoC.

The SISO module makes use of three memory modules. Together with those used by the LDPC PE, a separate intrinsic memory is necessary to store channel intrinsic information, that cannot be overwritten by extrinsic information as in LDPC decoding.

Three additional modules are necessary for both decoding tasks:

- *Initialization module*: at the start of a simulation, this module handles the mapping of the tasks over the topology and the consequent initialization of the interleaving memories. Throughout the decoding, it handles the channel simulation: source bits are created, encoded and sent through a channel according to the selected SNR. Data and intrinsic memories are initialized to the resulting LLRs. This process is repeated for all the SNR points and all the frames considered by a simulation.

- *Global decision module*: this module, at the end of every iteration, gathers the latest information about each bit of the frame from the local PE memories. The resulting received frame is consequently compared with the sent frame, provided by the initialization block: errors are counted and notified to the decoding management module.

- *Decoding management module*: by monitoring the status of the network and of the PEs, it detects the end of an iteration, and warns the global decision block. In case of correct decoding or reached iteration limit, the BER is updated.

The NoC modeled for this case of study is inspired to the ASNoC implemented in Section 3.4 for a flexible turbo/LDPC code decoder. REs are constituted of an $F \times F$ crossbar switch, with F input FIFOs and F output ports: control logic or routing tables implement the routing algorithm, and no additional network interface is present, since packet construction is handled by the PE. The REs modeled for this case of study consider the shortest paths among the nodes, internally storing the information during the initialization phase of the simulation. As with the PE, a set of latencies is passed as parameters by the user to provide correct timing information.

From extensive simulations it has been possible to observe how the on-time delivery of messages is of fundamental importance for the decoding process: to analyze this concept, let us define the Normalized Delivery Time (NDT) as

$$\mathrm{NDT} = \frac{t_d - t_0}{t_u - t_0} \tag{3.17}$$

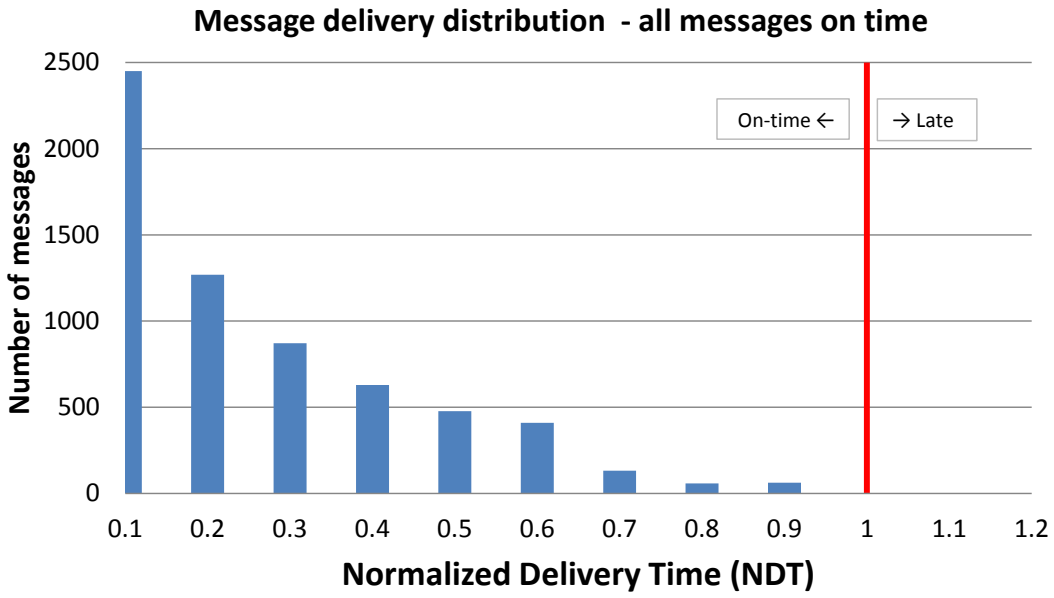

Figure 3.22: Message normalized delivery time distribution - all messages reach the destination on time

where t_0 is the sending time of a message, t_d the arrival time at its destination PE, and t_u the instant when the considered message has to be used at the destination PE. Given a certain decoder architecture, t_u depends on the target throughput while t_d is related to the NoC clock frequency. Messages that are delivered on time, i.e. that reach their destination before t_u, are characterized by NDT$\leq$ 1. Late messages have $t_d > t_u$, leading to NDT$>$ 1. The condition NDT$<$ 1 tends to be very restrictive for codes used in current standards, because of the already mentioned low degree of adjacency in typical inter-PE communication patterns [44]. As a consequence, also a smart mapping of tasks on PEs does not allow for any useful clustering and t_d values have a strongly nonuniform distribution. In order to guarantee NDT$<$ 1 for the totality of messages, either the throughput must be reduced (which means reducing the PEs clock frequency) or the NoC clock frequency should be increased (without altering the PEs clock frequency) [14].

An ideal situation for a decoder is shown in Fig. 3.22, that plots the number of messages with respect to the NDT for a WiMAX code of block size 2304 and rate 1/2, decoded with a 16-PE decoder mapped on a Kautz network: in this decoder, to have NDT$<$ 1 for all messages, the NoC frequency is 420 MHz, while the PEs only run at 280 MHz.

Fig. 3.23 gives the message delivery time distribution for the same code, with the difference that both PE and NoC frequency are 280 MHz. Here, a consistent percentage of messages has NDT$>$ 1. Late messages are extremely disruptive for the performance of the decoder. In case a message has not

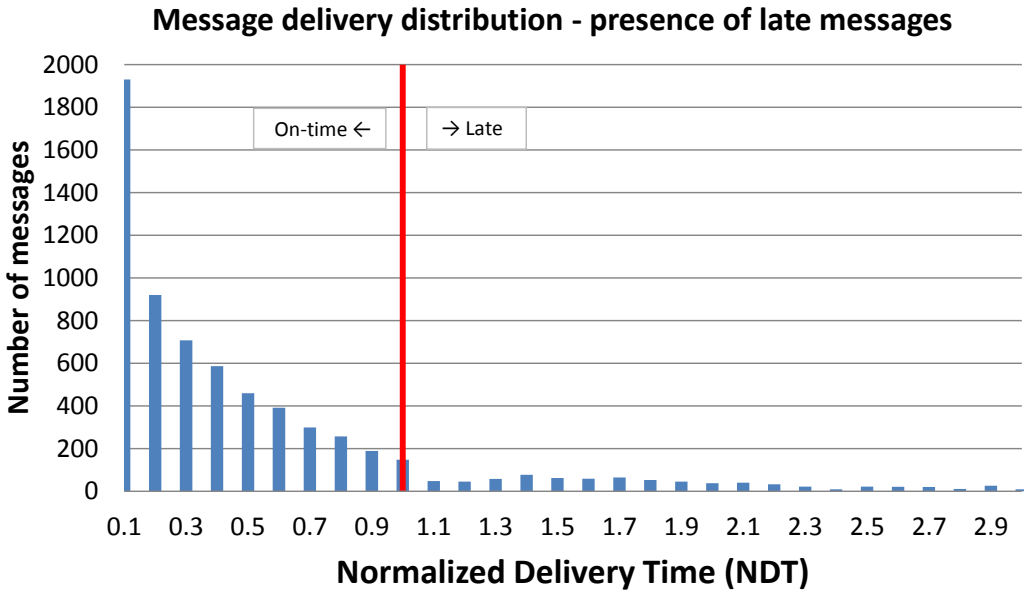

Figure 3.23: Message normalized delivery time distribution - some messages do not reach the destination on time

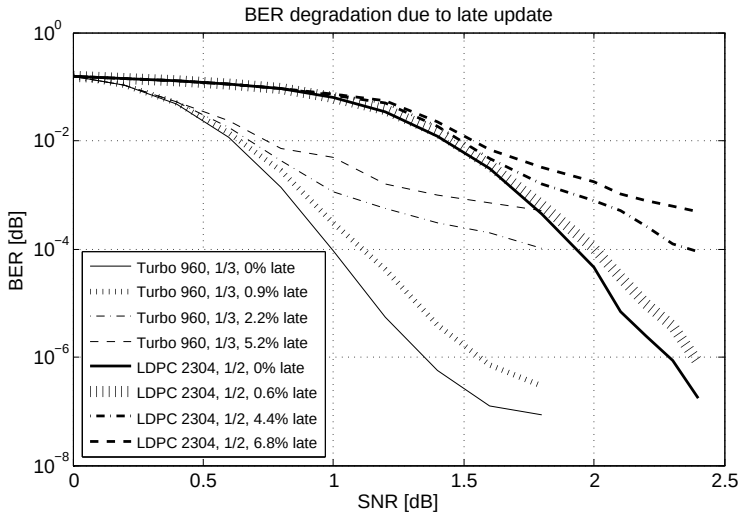

Figure 3.24: BER degradation due to late information update

been delivered at t_u, the destination PE can use the value computed at the previous iteration, but this leads to additional errors. Fig. 3.24 plots the BER curves of WiMAX turbo code of size 960, rate 1/3 and WiMAX LDPC code of size 2304, rate 1/2 under different percentages of late messages for illustration. These are obtained by changing the frequency of the 16-PE Kautz NoC used in Fig. 3.22 and 3.23. Simulations have been performed on an AWGN channel, with Binary Phase Shift keying (BPSK) modulation and fixed point precision (10 bits, 3 of fractional part). It can be seen that very

small amounts of late messages (< 1%) degrade the decoder performance, while larger percentages completely compromise the decoder error correction capabilities. It is clear that late messages can not be simply discarded, however efficient methods to reduce and optimize the traffic are introduced in the Section 3.6.2.

3.6.2 Traffic reduction and optimization

Reducing the number of messages traveling on the NoC is bound to speed-up the delivery of those remaining, with shorter queues and fewer collisions. Two techniques have been devised and tested towards these goals: they are based on the general concept that not all information messages (which are of a probabilistic nature) traveling on the network are essential for the success of the decoding. Two additional methods are instead set towards the optimization of the NoC traffic.

Hard importance

The Hard Importance (HI) method allows to refrain from sending messages that are estimated to be of low impact on the decoding process. In the LDPC decoding case, the messages traveling on the NoC are different updates of $\lambda_k[c]$. The HI checks are performed once per iteration to each of them. Consider the following comparisons:

$$\text{sgn}(\lambda_k^n[c]) = \text{sgn}(\lambda_k^{n-1}[c]) \tag{3.18}$$

$$|\lambda_k^n[c]| \geq |\lambda_k^{n-1}[c]| \tag{3.19}$$

$$|\lambda_k^n[c]| \geq \texttt{Thr}^{HI} \cdot \max(\lambda_k[c]) \tag{3.20}$$

where n expresses the n^{th} iteration and $\max(\lambda_k[c])$ is the maximum possible value of $\lambda_k[c]$ given the number of bits assigned to its representation, while $0 \leq \texttt{Thr}^{HI} \leq 1$ expresses the percentage of $\max(\lambda_k[c])$ involved in the comparison. If all above three conditions are verified, $\lambda_k[c]$ is flagged as `unimportant` and the bit LLR is not updated anymore for the rest of the decoding process. The first two comparisons check the presence of a monotonic divergence from zero in the LLR value, while the third requires a large enough absolute value to be satisfied. Compliance with all three checks confirms that the information is already reliable, and that a change of sign (and consequently a bit flip) is extremely unlikely. Since with layered scheduling a $\lambda^k[c]$ is updated many time within each iteration, a single `unimportant` LLR will result in a traffic reduction of several messages.

For turbo decoding, the choice of stopping or not a message can be made by modifying the Symbol

Reliability Difference (SRD) criterion proposed in [78]. Defining

$$\delta_m^{(i)}(d_k) = L_m^{(i)}(d_k = d_k^1) - L_m^{(i)}(d_k = d_k^2) \tag{3.21}$$

as the difference between the logarithmic extrinsic probabilities of the first and second most probable symbols, the original SRD criterion proposes, for each symbol d_k

$$\phi_{m,m'}^{(i)}(d_k) = |\delta_m^{(i)}(d_k) - \delta_{m'}^{(i)}(d_k)| \tag{3.22}$$

where m' refers to the metrics at the input of the SISO, coming from the previous half-iteration, and m at the output. If condition

$$\phi_{m,m'}^{(i)}(d_k) \leq \texttt{Thr}_{Abs}^{HI} \cdot \max(\phi_{m,m'}^{(i)}(d_k)) \tag{3.23}$$

is satisfied, the message can be stopped. Applying this method as is, however, led to unsettling results. This is due to the fact that it is not taken in account that (3.22) could give very low $\phi_{m,m'}^{(i)}(d_k)$ also if both $\delta_m^{(i)}(d_k)$ and $\delta_{m'}^{(i)}(d_k)$ are very close to 0. In this case $\phi_{m,m'}^{(i)}(d_k)$ expresses just uncertainty about symbols, instead of agreement between SISOs, and the message should not be stopped. For this reason, an additional control has been added for the message to be stopped:

$$|\delta_m^{(i)}(d_k)|, |\delta_{m'}^{(i)}(d_k)| \geq \texttt{Thr}_{Diff}^{HI} \cdot \max(\delta^{(i)}(d_k)) \tag{3.24}$$

where $\texttt{Thr}_{Diff}^{HI}$ assures a degree of reliability on the symbol.

The performance of HI is of large interest, since this method can be applied also to other types of decoders beside NoC-based ones. HI acts on messages by deciding if it is worth updating a value or not, and can be effective also in absence of a NoC. It can consequently be exploited in decoders that rely on shared memory banks: in this case, energy is saved by reducing the number of memory write operations.

Soft importance

The Soft Importance (SI) technique evaluates the state and the evolution of the information exchanged through the NoC, flagging non-essential messages as `expendable`. In case of collisions, i.e. messages that need to be routed through the same output port, the router arbiter will forward a message and discard all the `expendable` messages that were not granted priority. In LDPC decoding, (3.18)-(3.20) are applied in SI with a more relaxed $\texttt{Thr}^{SI}$, while the metrics considered are not $\lambda_k^n[c]$ and $\lambda_k^{n-1}[c]$,

Table 3.13: Percentages of late messages (% late) and stopped or not sent messages (% stop) for no traffic handling (No TH) and combinations of the proposed methods on 16-PE and 32-PE generalized Kautz and 2D-Mesh NoCs, at BER= 10^{-5}

NoC	Code, length, rate	No TH	HI		SI		HI + SI		U	U+ BR	HI+SI+U+BR
		%	%		%		%		%	%	%
		late	late	stop	late	stop	late	stop	late	late	late
16-PE Kautz SP routing	LDPC 2304, 5/6	9.2	4.8	17.5	8.0	12.3	1.6	23.8	8.5	0.8	0.1
	LDPC 576, 5/6	28.8	19.4	16.9	24.3	12.4	15.4	22.3	25.1	11.1	8.2
	LDPC 1440, 1/2	14.5	6.3	19.5	10.9	13.0	5.5	27.0	12.0	2.6	1.1
	LDPC 864, 1/2	22.3	14.2	18.4	19.2	12.6	11.5	26.1	20.7	4.1	3.0
	Turbo 2400, 1/3	3.3	2.0	20.7	2.8	13.9	1.7	22.8	3.0	0.2	0.0
	Turbo 960, 1/3	5.2	2.9	18.3	4.3	12.8	2.6	23.4	4.6	0.3	0.1
	Turbo 6144, 1/3	4.8	3.0	20.1	3.4	12.9	2.1	22.2	3.2	0.2	0.0
32-PE Kautz SP routing	LDPC 2304, 5/6	13.6	7.2	17.4	10.3	14.3	3.7	25.1	9.5	1.1	0.3
	LDPC 576, 5/6	42.6	30.0	16.7	35.3	14.5	25.9	24.3	37.5	18.8	16.8
	LDPC 1440, 1/2	19.9	10.1	19.3	14.8	16.0	8.0	28.3	17.1	6.2	2.9
	LDPC 864, 1/2	31.5	22.2	18.7	27.0	15.8	20.1	27.3	26.2	9.9	8.0
	Turbo 2400, 1/3	6.1	4.2	20.7	5.0	15.5	3.5	23.8	4.8	0.8	0.2
	Turbo 960, 1/3	9.7	5.9	18.3	6.9	15.0	4.4	25.2	7.1	1.2	0.4
	Turbo 6144, 1/3	8.8	6.2	20.0	7.1	14.4	4.6	24.9	6.9	0.8	0.1
16-PE 2D-Mesh X-Y routing	LDPC 2304, 5/6	10.2	5.4	17.3	8.7	12.8	3.3	23.9	8.8	1.4	0.3
	LDPC 576, 5/6	32.6	21.0	17.0	26.0	13.1	18.2	24.4	25.8	12.6	8.9
	LDPC 1440, 1/2	16.0	6.9	19.5	11.5	13.3	6.1	28.1	13.2	2.6	1.1
	LDPC 864, 1/2	24.1	15.5	18.4	20.2	13.1	12.4	26.8	22.1	4.1	3.0
	Turbo 2400, 1/3	3.9	2.3	20.7	3.3	14.4	2.1	23.6	3.4	0.5	0.1
	Turbo 960, 1/3	6.2	3.4	18.3	4.7	13.0	3.2	23.9	5.9	0.9	0.3
	Turbo 6144, 1/3	5.9	3.8	20.1	3.8	13.6	3.0	23.7	3.8	0.4	0.2
32-PE 2D-Mesh X-Y routing	LDPC 2304, 5/6	15.5	9.4	17.4	11.3	14.7	4.2	25.6	9.8	1.1	0.3
	LDPC 576, 5/6	44.8	30.4	16.8	37.7	15.1	27.5	25.3	38.4	18.8	16.8
	LDPC 1440, 1/2	22.3	11.0	19.6	16.0	16.3	9.0	29.1	18.0	6.2	2.9
	LDPC 864, 1/2	33.2	23.3	18.5	27.9	16.0	21.3	27.7	26.9	9.9	8.0
	Turbo 2400, 1/3	7.3	5.3	21.0	5.4	16.1	3.8	25.0	5.0	0.9	0.3
	Turbo 960, 1/3	10.4	7.0	18.4	7.1	15.4	4.7	26.4	7.5	1.5	0.6
	Turbo 6144, 1/3	10.1	6.8	19.9	7.6	15.0	5.0	26.2	7.2	1.0	0.3

but $\lambda_k^{new}[c]$ and $\lambda_k^{old}[c]$. Each PE monitors the evolution of the LLR locally, before and after the processing: if all three comparisons are verified, the message is `expendable`. Since in turbo decoding the HI method already affects each message separately, the SI method can be efficiently implemented with exactly the same mechanism as HI. The `expendable` flag is applied to messages satisfying the modified SRD conditions with more relaxed thresholds $\texttt{Thr}_{Abs}^{SI}$ and $\texttt{Thr}_{Diff}^{SI}$.

Urgency and Buffer reordering

Smarter and more efficient communication on the NoC can be obtained through the identification of urgent and less urgent messages: traffic optimization deals with the late message issue by prioritizing the former against the latter. Priority-based routing is a well-explored path to guarantee QoS: multiple virtual channels are often assigned different priorities to differentiate traffic flows [82,85]. The concept of priority is applied here in an original way, by using a single channel with a reordering buffer.

The Urgency technique (U) implements a priority-based collision management policy. In case

of collision, priority is given to the most urgent message, i.e. the message which is needed by its destination PE sooner. To allow this kind of decision, an urgency field must be added to the message during sending, initialized with an estimate of the number of clock cycles available before the message is needed by another PE. The field must be updated by the routers, taking in account the wait cycles spent in input buffers, and a message is discarded if its urgency reaches zero, avoiding unnecessary switching activity for late messages. With the LDPC case, each PE can perform an estimation based on local knowledge of the instant in which the outgoing message is going to be needed. The precision of the estimate strongly depends on the regularity of the partitioning of the **H** matrix among the PEs. On the contrary, in the turbo case, since the interleaving rule is known to all PEs, the measure can be exact.

In Buffer Reordering (BR) a fast lane can be created by arranging the messages in the input buffers not in arrival order, but according to the urgency field. The most urgent message in a buffer will consequently always be the first one to be pulled out, increasing its chances of arriving on time.

3.6.3 Performance results

The impact of each of the proposed techniques, alone and in combination with one another, has been evaluated with the JANoCS tool described in Section 3.5. Extensive simulations have considered a wide range of codes, NoC topologies, number of PEs, routing algorithms, PE and RE architectures. Table 3.13 lists the percentages of late and stopped messages for a set of LDPC and turbo codes taken from WiMAX [3] and 3GPP-LTE [72] standards, considering a decoder implementing different combinations of the proposed techniques. The codes have been mapped on 16-PE and 32-PE NoCs with generalized Kautz [80] and two-dimensional mesh topologies. Meshes are a common topology for middle-sized NoCs, and the simple X-Y routing algorithm [86] joins good performance with ease of implementation. Kautz networks have been proven effective for turbo and LDPC decoding in [11] and [13] respectively also in presence of REs of degree three: routers implement a shortest path routing algorithm [11]. With all the considered NoCs, each PE produces one $\lambda_k[c]$ message per clock cycle in the LDPC case, and one $\lambda_k[u]$ message in the turbo case. While with single-binary turbo codes $\lambda_k[u]$ consists of a single value, three values are necessary in double-binary turbo codes: the width of the simulated channel is adapted accordingly.

HI and SI show high percentages of stopped messages (up to 29%), with substantial reduction in late messages (down to 1.6%), especially for HI and HI+SI. Results for HI and SI were obtained at the SNR point for which BER= 10^{-5} with 10 maximum iterations for LDPC codes and 8 maximum iterations for turbo codes. Iterations are stopped as soon as the codeword is correct, and the number of

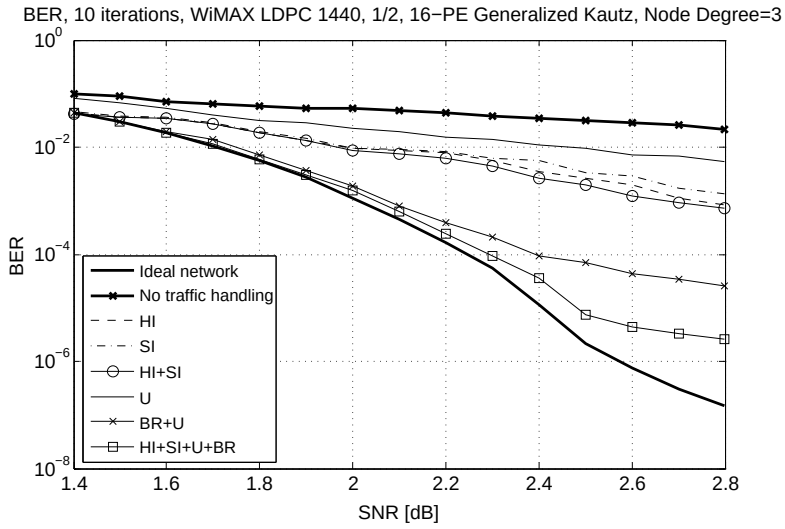

Figure 3.25: Impact of the proposed techniques and their combinations - LDPC codes

stopped messages is averaged over all performed iterations. When this kind of early stopping criterion is not present, HI can work as a valid substitute. In fact, if a codeword is correct and its decoding continues, the magnitude of all LLRs keeps growing and HI effectively prevents all messages from being sent. HI and SI inherently introduce some BER degradation, for which careful threshold calibration is necessary: if set too low, the stopped messages can still carry information about uncertain bits, introducing new errors. A wide range of possible threshold values have been considered and simulated, with the final choice representing a good tradeoff between method effectiveness (i.e. number of stopped messages) and BER degradation. The decoder can incur in additional errors in case a metric update is stopped too early by HI or SI: however, threshold calibration allows for results similar to those shown in Fig. 3.27, where the impact of HI on BER for an LDPC code and a turbo code are presented. The percentages of stopped messages assumed in these simulations are consistent (17% and 18% respectively), but both the LDPC and the turbo code show negligible performance losses. In the plots of Fig. 3.25 the thresholds have been set as $\texttt{Thr}^{HI} = 0.2$ and $\texttt{Thr}^{SI} = 0.1$, while in Fig. 3.26 as $\texttt{Thr}^{HI}_{Abs} = 0.25$, $\texttt{Thr}^{HI}_{Diff} = 0.15$, $\texttt{Thr}^{SI}_{Abs} = 0.15$ and $\texttt{Thr}^{SI}_{Diff} = 0.05$. These thresholds are also used to derive the "Ideal THR" curves in Fig. 3.28 and 3.29, that show how variations in the threshold values affect the BER performance.

The results given by the urgency U method alone are not satisfying: since its effects are mostly appreciated in case of collisions, which can involve also non-critical messages, its effectiveness alone is limited. However, as soon as a message is identified as late it is discarded: this means that the percentage of stopped messages for which U contributes is equal to the percentage of late messages. The U+BR urgency-based buffer reordering, which allows non-urgent messages to be delayed in

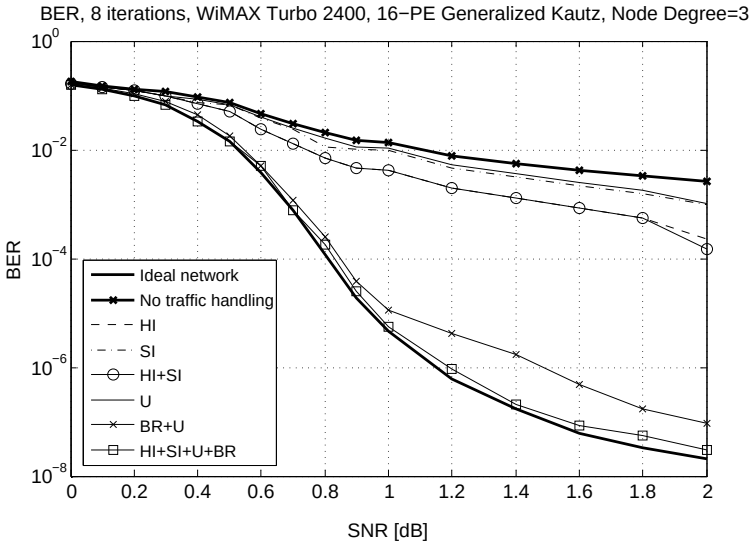

Figure 3.26: Impact of the proposed techniques and their combinations - turbo codes

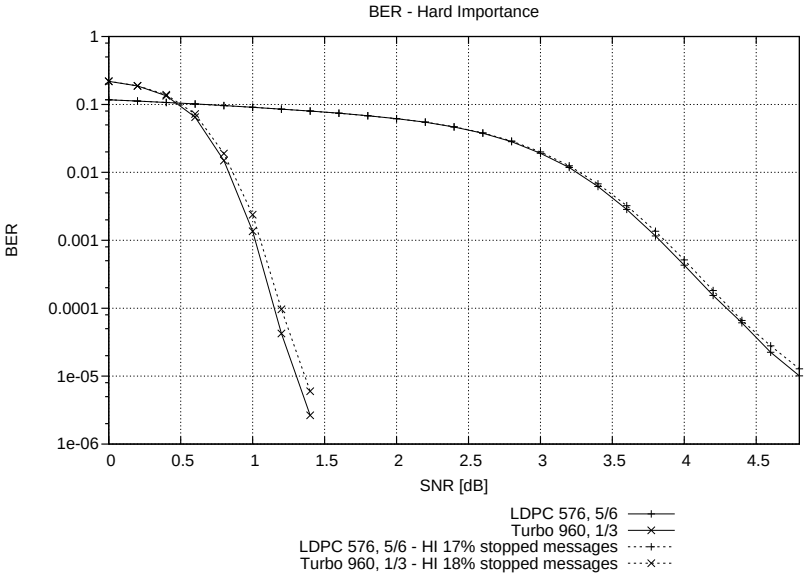

Figure 3.27: Impact of HI on BER

favor of critical ones, drastically reduces the occurrence of late messages (from 11.1 % to 0.2%). Its effectiveness can be improved further by combining the two traffic reduction methods with the traffic optimization ones. The joint application of all four techniques (HI+SI+U+BR) guarantees a late message percentage close to zero in most cases, while substantially reducing their impact in the remaining cases.

From Table 3.13 it is possible to make some important observations. As expected from previous analysis [11, 13], the Kautz topology performs better than 2D-mesh when targeting LDPC and turbo codes. Moreover, turbo codes suffer less from late messages w.r.t. LDPC codes (No TH column), thanks to their less critical communication phase. It can also be noticed how the size and the rate

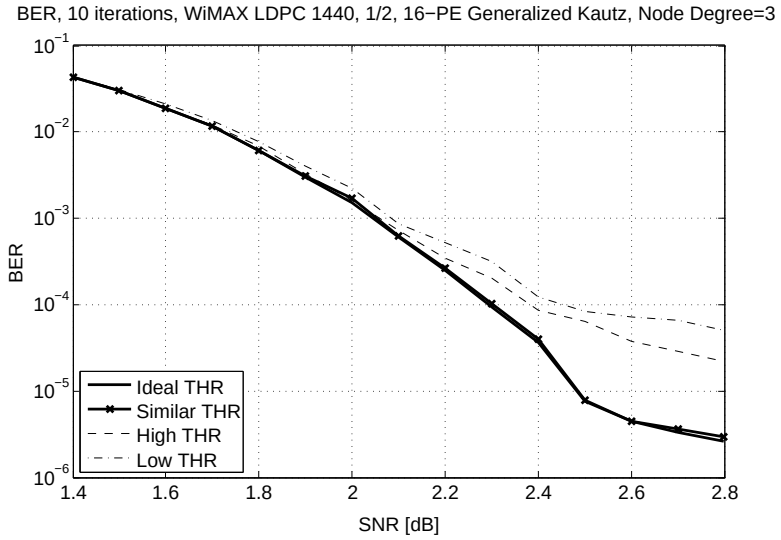

Figure 3.28: Impact of different threshold choices on HI+SI+U+BR - LDPC codes

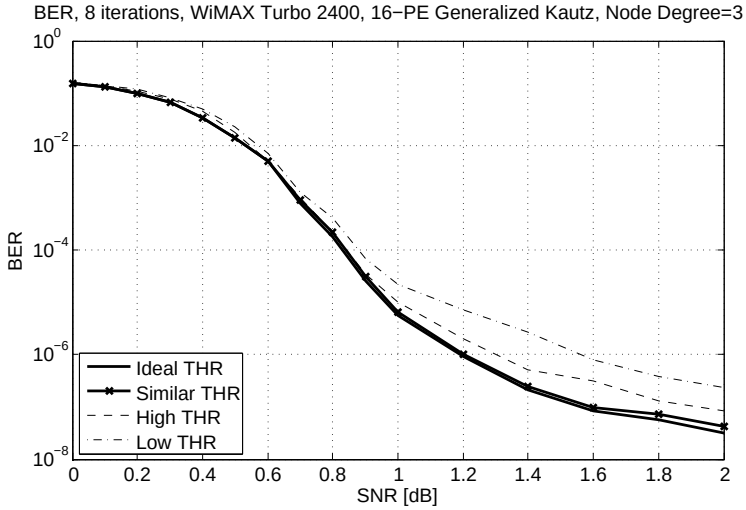

Figure 3.29: Impact of different threshold choices on HI+SI+U+BR - turbo codes

of the code affect the number of late messages, especially when related to the size and topology of the NoC. A small LDPC code has small **H** layers and a small turbo code has short half-iterations: consequently, the available message delivery times are limited. Moreover, small codes mapped on a large NoC suffers from a large number of late arrivals, since the distance between PEs dominates the transmission times. This is the case of the WiMAX LDPC 576, rate 5/6 in Table 3.13: similar effects are encountered with larger codes when mapped on the 32-PE NoCs. On the contrary, queues and collisions are the main sources of delay in case of large codes mapped on small NoCs. These limit cases (e.g. LDPC 576, rate 5/6 in Table 3.13) cannot be completely solved with the implementation of the proposed traffic handling techniques, and need to work in conjunction with alternative techniques:

for example, the code can be mapped on a smaller portion of the NoC, and the unused part of the decoder can be deactivated to save energy.

The percentage of late messages is directly reflected on the decoder performance, as shown in Fig. 3.25 and 3.26. The effects of the different combinations considered in Table 3.13 on the BER of an LDPC and turbo code are plotted, using the same NoC and PE architectural choices. As an example, Fig. 3.25 shows the BER results for a WiMAX LDPC code of rate 1/2 and block size 1440, with 10 maximum iterations; the duo-binary WiMAX turbo code in Fig. 3.26 has an information block size of 2400 symbols and rate 1/3, decoded with 8 iterations. However, the behaviors observed in Fig. 3.25 and 3.26 do not depend on the choice of the codes, but only on the percentage of late messages. In both plots, the "ideal network" curve represents an ideal decoding process, where the interconnection structure does not introduce any latency (lower bound), while the "no traffic handling" curve shows the BER in case of a real decoding process in which no one of the methods is applied (upper bound). It can be noticed how all the performance curves of the proposed solutions span the interval between the upper and lower bound. HI, SI and HI+SI curves do not provide substantial performance improvement, though giving interesting results in terms of traffic reduction, and consequently reducing the switching activity. The U curve is still very close to the "no traffic handling" one, reflecting its limited effectiveness shown in Table 3.13. The U+BR curve shows the performance in presence of the powerful buffer reordering, with a further step towards the reference curve made by the HI+SI+U+BR curve, that combines all four methods. Though the proposed techniques behave coherently in both cases, they yield slightly better results in the turbo case, as expected from Table 3.13.

Fig. 3.28 and 3.29 show the impact of different threshold values on the BER performance of HI+SI+U+BR applied under the same conditions and to the same codes of Fig. 3.25 and 3.26. As mentioned earlier in this section, the "Ideal THR" curves have been obtained by simulating an extensive set of possible threshold values. The final choice has been made by selecting the threshold values that maximize the number of stopped messages without degrading the BER performance, and the obtained percentages of stopped messages are those reported in Table 3.13. The choice of the threshold is not very critical, as shown by curves labeled as "Similar THR" in Fig. 3.28 and 3.29, which have been derived by rising each "Ideal THR" threshold by 10%: it can be seen how the curves are almost superimposed, and similar minor fluctuations are observed in the number of stopped messages as well. Larger threshold variations have much more influence the BER: the thresholds used in the "High THR" curves are obtained by tripling the "Ideal THR" thresholds. Very high threshold values result in a very small percentage of stopped messages, and in the ineffectiveness of both HI and SI. In

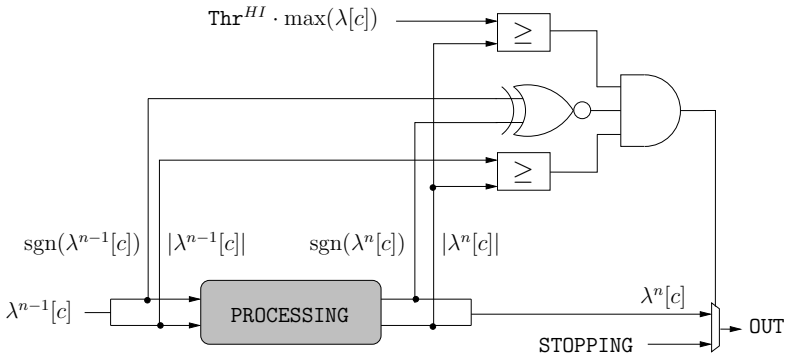

Figure 3.30: HI implementation for LDPC codes - `STOPPING` message

fact, "High THR" BER curves are very similar to the U+BR curves of Fig. 3.25 and 3.26, where HI and SI are not applied. On the contrary, very low thresholds as the ones used in "Low THR" curves (half of "Ideal THR" thresholds) dramatically increase the number of stopped messages. However, a large number of messages carrying useful information are stopped as well, causing consistent BER degradation.

3.6.4 Hardware architecture

The similarity of the calculations involved in HI and SI, and the necessity for controls at each RE for SI, U and BR allow for efficient resource sharing. The multi-mode decoder described in Section 3.4 and [14] has been taken as reference architecture: among the different implementations, the one denoted **A** has been selected and called here $\mathbf{A}_{ref}$. It relies on a 22-PE Generalized Kautz NoC, and complies with WiMAX, HPAV [6] and DVB-RCS [71] standards, with limited support for WiFi [4].

The HI method can be easily implemented in the SISO. At the beginning of each trellis step one or more read operations from the data memory are required, and they provide the data needed to calculate the $\delta_{m'}^{(i)}(d_k)$ member of (3.22). At the end of the trellis steps, also the data needed to calculate $\delta_{m}^{(i)}(d_k)$ are ready, thus making it possible to compute (3.22), (3.23) and (3.24). A memory bit is required for each trellis step to signal if the outgoing messages are `unimportant` and must not be sent. For LDPC codes, since the considered metrics are $\lambda_k^n[c]$ and $\lambda_k^{n-1}[c]$, implementation of HI is less straightforward.The main issue is the fact that the storage of $\lambda_k^n[c]$ is performed by replacing $\lambda_k^{n-1}[c]$. (3.18) to (3.20) can however be executed without any additional memory for the storage of $\lambda_k^{n-1}[c]$ by configuring the data memory as Read-Before-Write. When a message is flagged as `unimportant`, the corresponding memory bit is set, and the `unimportant` flag must be propagated to all other PEs. A dedicated `STOPPING` message is sent in place of the `unimportant` message. Fig. 3.30 shows the circuit responsible for the HI check and eventual creation of the `STOPPING` message:

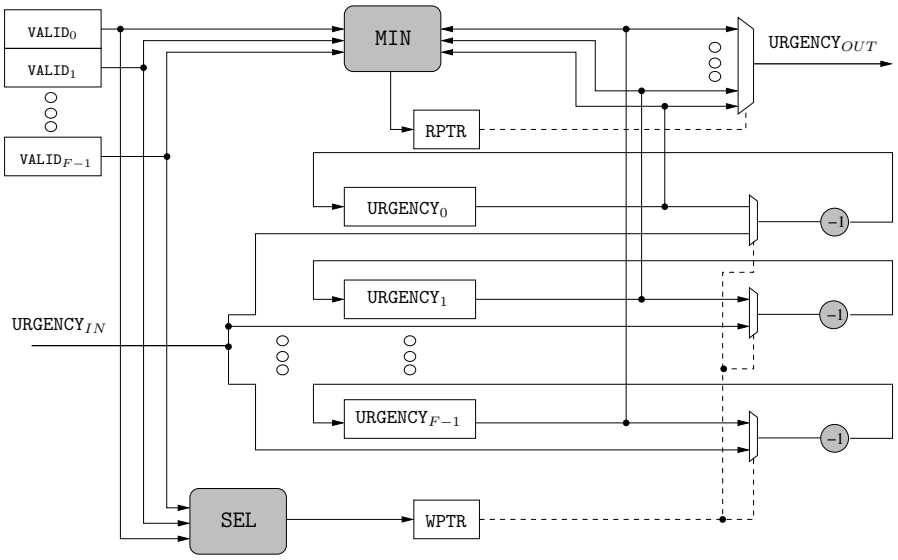

Figure 3.31: Urgency-based buffer reordering

the `PROCESSING` block executes the computations necessary to update each $\lambda_k[c]$. When a `STOPPING` message is received, the `unimportant` memory bit is set in the destination PE, and the `STOPPING` message is propagated. If a PE receives a `STOPPING` message when the `unimportant` memory bit has already been set, the `STOPPING` message is not sent anymore. The `STOPPING` message is mapped to the lowest negative value that can be represented with the allocated number of bits: for example, with 9 bits, the data dynamic range is mapped to the interval (-255, +255), and the value -256 is recognized as a `STOPPING` message.

The HI method can also be applied to save energy by simply reducing the number of memory write operations at the destination PEs. When a given message is received by its destination PE, the writing into the internal memory can be avoided if the message is recognized as unimportant. The implementation of this functionality still exploits the `STOPPING` message, which is used to control the write enable signal of the memory and prevent write operation. However, in this case, the `STOPPING` message must be sent to the memory at every iteration.

The implementation of SI follows that of HI in the turbo case, only requiring two additional comparators for the different thresholds in (3.23) and (3.24). It is instead much simpler than HI for LDPC codes, since both $\lambda_k^{new}[c]$ and $\lambda_k^{old}[c]$ are available during each parity check computation, and there is no need for flag propagation. Together with the simple computational logic in the PEs, a flag bit signaling if a message is `expendable` or not must be added also in all NoC FIFOs and channels, while changes in the write and read pointers (`WPTR` and `RPTR`) of each FIFO are forced by the RE arbiter in case of collisions.

The U method requires, for the initialization of the `URGENCY` field of outgoing messages, the

estimation of the available delivery time. This measure can be obtained, in the turbo case, thanks to the current trellis step together with the globally known interleaving rule. Since each SISO processes a sequential set of trellis steps, the destination memory address is a precise identifier of the time instant a message will be needed. The URGENCY field of each outgoing message can be initialized as

$$U_{turbo} = T_{half} - t_{send} + t_{need} \tag{3.25}$$

where T_{half} is the duration of a half iteration, t_{send} is the time stamp of the sending instant and t_{need} equals to the destination memory address multiplied by the number of cycles needed to complete each trellis step. The destination address is also used in the initialization of U in the LDPC case. By multiplying it by the minimum row degree of $\mathbf{H}$, a lower bound of t_{need} is obtained, thus leading to the following equation:

$$U_{LDPC} = t_{need} - t_{send} \tag{3.26}$$

The urgency field requires additional bits in all the NoC FIFOs and channels, together with the simple initialization logic at each PE. Moreover, each FIFO of length F needs F adders to update U at each clock cycle, while WPTR and RPTR must be updated in case the urgency field reaches zero, and the corresponding message discarded. All the FIFO memory elements must be available for writing at each clock cycle: the FIFO consequently must be implemented with registers, and not with a RAM. Finally, the priority of the RE arbiter is changed from being FIFO-length-based [14] to urgency-based.

The implementation of BR requires all the modifications described for U, plus a novel method for the update of WPTR and RPTR. The RE input buffers in fact lose their FIFO nature, since the input order is not guaranteed to be the output order. Fig. 3.31 shows the simplified structure of the proposed buffer reordering mechanism; white blocks represent registers, while gray blocks indicate additional computation elements. Along with the URGENCY field, each buffer element requires an additional VALID field. Read and write operations on the buffer take in account external signals (PUSH, POP) and the internal state of the buffer (IS_EMPTY, IS_FULL). Every time a write operation is performed, the VALID field of the corresponding buffer element is set, while it is cleared with a read operation. The VALID fields are necessary to keep track of the free and occupied elements, since the irregular input and output orders prevent WPTR and RPTR from being used for this purpose. During write operation, the urgency field from the incoming message $\texttt{URGENCY}_{IN}$ is substituted according to WPTR to one of the stored $\texttt{URGENCY}_X$ during the U update process. Thus, the updated value of $\texttt{URGENCY}_X$ is $\texttt{URGENCY}_{IN} - 1$ instead of $\texttt{URGENCY}_X - 1$. In a concurrent read operation a $\texttt{URGENCY}_X$ value is selected as the buffer output $\texttt{URGENCY}_{OUT}$ according to RPTR. The SEL module chooses the update value of WPTR among

Table 3.14: Effect of traffic handling on area occupation (CMOS 90 nm technology, post-layout results)

	$\mathbf{A}_{ref}$ Area		$\mathbf{A}_{new}$ Area		Overhead
	[mm^2]	%	[mm^2]	%	%
Core Memory	1.46	53%	1.53	49%	4.8%
SISO Logic	0.42	15%	0.45	15%	7.1%
LDPC PE Logic	0.31	11%	0.38	12%	22.6%
NoC	0.56	21%	0.75	24%	33.9%
Total	2.75	100%	3.11	100%	13.1%

Table 3.15: Effect of traffic handling on power consumption (CMOS 90 nm technology, 1.0 V supply)

	$\mathbf{A}_{ref}$		$\mathbf{A}_{new}$		Power Gain
	Pow	f_{clk}	Pow	f_{clk}	
	[mW]	[MHz]	[mW]	[MHz]	%
PEs	68.0	200	70.1	200	+3.1%
NoC	48.6	333	29.1	200	-40.2%
Total	116.6	333-200	99.2	200	-15.0%

the elements with cleared `VALID` field with fixed priority. The `MIN` module, instead, updates `RPTR` as the pointer to the minimum $\texttt{URGENCY}_X$ among the `VALID` ones. The whole operation occurs within a single clock cycle, and correct functionality of the circuit has been tested in 90 nm CMOS technology for up to 10 buffer elements, with 200 MHz as target throughput. The area overhead introduced by the additional operations in the reordering buffer has been evaluated with respect to a typical FIFO buffer. For example, using 90 nm CMOS technology, a reordering buffer with five buffer elements accounts for a little more than 2000 μm^2, against the 850 μm^2 of a regular FIFO, with a $\times$2.35 area increment factor.

3.6.5 Implementation

To help a fair comparison with the state of the art, all proposed optimization techniques have been applied to the $\mathbf{A}_{ref}$ decoder, creating a new implementation $\mathbf{A}_{new}$, targeting 90 nm CMOS technology: starting from VHDL models designed after the exploration performed with JANoCS, synthesis has been carried out with Synopsys Design Compiler, functional simulation and validation with Mentor Graphics ModelSIM, and place and route with CADence SoC Encounter. Table 3.14 dissects the post place and route area occupation of various components of the decoder before and after the implementation of the proposed methods. The small increase in memory occupation is due to the extra memory bit for the `unimportant` flag related to HI, shared between SISOs and LDPC PEs. The simple logic required for both HI and SI in the turbo case results in an additional 7.1% area occupation for SISOs, while the more complex operations involved in the LDPC PE for HI lead to a higher area overhead. The widths of NoC buffers and channels have been increased to accommodate

the `URGENCY` field (five bits), the `VALID` bit of BR and the `expendable` bit of SI. This, together with the additional logic for U and BR, heavily affects the NoC area, with an overhead exceeding 30%. With a total area of 3.11 mm^2, the modified decoder is 13.1% larger than $\mathbf{A}_{ref}$. However, as mentioned in Section 3.6.1, $\mathbf{A}_{ref}$ deals with the late message issue with a NoC clock frequency higher than the PE clock frequency: with the introduction of the traffic reduction and optimization techniques, however, this is not necessary anymore, and the decoder can be clocked with a single frequency. Table 3.15 details the worst case power consumption Pow for $\mathbf{A}_{ref}$ and $\mathbf{A}_{new}$ architectures: in the original decoder the clock frequency f_{clk} is set to 200 MHz for the PEs and to 333 MHz for the NoC. The global power consumption is 116.6 mW, with the NoC accounting for 41.7% of the total. Total power consumption in $\mathbf{A}_{new}$ is reduced of 15% w.r.t. $\mathbf{A}_{ref}$, while the power gain on the NoC alone reaches a very consistent reduction of 40.2%. This result basically derives from the lower clock frequency of the NoC with respect to architecture $\mathbf{A}_{ref}$, but the clock frequency reduction is made possible thanks to the adoption of the described traffic reduction techniques. A final measure of the ratio between costs and advantages in reducing the NoC power consumption can be obtained as

$$(Pow^{NoC}_{\mathbf{A}_{new}} + Pow^{PE}_{\Delta} - Pow^{NoC}_{\mathbf{A}_{ref}})/Pow^{NoC}_{\mathbf{A}_{ref}} = -35.8\% \tag{3.27}$$

where Pow^{PE}_{Δ} is the power consumption increment in PEs due to the contribution of HI, SI and U initialization. The implementation of HI+SI+U+BR, taking in account all the introduced overheads, brings a power reduction on the NoC equal to 35.8% of $Pow^{NoC}_{\mathbf{A}_{ref}}$.

The actual impact of HI on the energy consumption of centralized decoders can also be estimated. In [87] the energy breakdown of a decoder based on memory banks sharing is given. The energy consumption of the γ-memory accounts for 70% of total dynamic energy for WiMAX LDPC code of size 576 and rate 5/6. For this particular code if HI is applied the percentage of stopped messages, and consequently of avoided write operations, is around 17% (Table 3.13). Since write operations contribute for approximately 50% of the energy expenditure, the implementation of HI leads to a 6% reduction in the total decoder energy consumption.

A simple direct way to reduce the traffic on the NoC is to reduce the number of iterations of the channel decoder. Table 3.16 compares the impact of such method with respect to the proposed techniques in terms of energy efficiency and BER degradation. In this table we consider the BER performance and energy consumption of $\mathbf{A}_{red}$, i.e. $\mathbf{A}_{ref}$ with a reduced number of maximum iterations, and compares it with $\mathbf{A}_{new}$ and the original $\mathbf{A}_{ref}$, for two code examples. The number of performed iterations is represented by It_{max}, E_{frame} expresses the energy spent per decoded frame and ΔSNR shows the performance degradation with respect to $\mathbf{A}_{ref}$ when BER=10^{-5}. In $\mathbf{A}_{red}$ the reduced

Table 3.16: Performance and energy consumption comparison (CMOS 90 nm technology, 1.0 V supply)

Code		$\mathbf{A}_{ref}$	$\mathbf{A}_{new}$	$\mathbf{A}_{red}$
LDPC	It_{max}	10	10	9
2304	ΔSNR @ BER=10^{-5} [dB]	0.0	0.02	0.09
5/6	E_{frame} [μJ]	2.03	1.73	1.83
Turbo	It_{max}	8	8	7
6144	ΔSNR @ BER=10^{-5} [dB]	0.00	0.00	0.15
1/3	E_{frame} [μJ]	7.82	6.65	6.84

Table 3.17: LDPC/Turbo architectures comparison: CMOS technology process (Tp), total area occupation (A_{tot}, normalized area occupation for 65nm technology (An_{tot}), clock frequency (f_{clk}), peak power consumption (Pow), energy efficiency (E_{eff}), maximum number of iterations (It_{max}), code length (N) and rate (r), interleaver size (K) and throughput (T), Area efficiency (A_{eff})

Decoder		[1]$\mathbf{A}_{ref}$	[1]$\mathbf{A}_{new}$	[49]	[51]	[1] [45]	[1] [35]
Tp [nm]	LDPC	90	90	65	45	65	-
	CTC					-	130
A_{tot} [mm^2]	LDPC	2.75	3.11	0.62	0.9	2.32	-
	CTC					-	3.57
An_{tot} [mm^2]	LDPC	1.43	1.62	0.62	1.88	2.32	-
	CTC					-	0.89
f_{clk} [MHz]	LDPC	[2]333-200	200	400	150	40	-
	CTC					-	302
Pow [mW]	LDPC	116.6	99.2	76.8	86.1	29.5	-
	CTC					-	788.9
E_{eff} [$\frac{nJ}{bit}$]	LDPC	0.166	0.141	0.032	0.151	0.003	-
	CTC	0.079	0.067	0.826	0.147	-	0.367
It_{max}	LDPC	10	10	10	8	10	-
	CTC	8	8	5	8	-	5.5
N, r	LDPC	2304, 1/2	2304, 1/2	2304, 5/6	N/A	2304, 5/6	-
K	CTC	2400	2400	2400	N/A	-	6144
T [Mb/s]	LDPC	70	70	237.8	71.05	1152	-
	CTC	183	183	37.2	73.46	-	390.6
A_{eff} [$\frac{bits}{mm^2 \cdot kcycles}$]	LDPC	2447	2160	9589	2013	124137	-
	CTC	5119	4519	750	2084	-	7993

[1]post–layout results

[2]f_{clk}^{NoC}

number of iterations (by only one iteration) leads to a proportional reduction of E_{frame}, but non-negligible performance degradation is present. It is especially noticeable in the turbo case, that relies on a smaller It_{max}. The proposed implementation outperforms the iteration reduction in terms of energy savings, while at the same time affecting the BER performance only marginally.

Finally Table 3.17 compares the results of $\mathbf{A}_{new}$ with $\mathbf{A}_{ref}$ and few recent related state-of-the-art LDPC and turbo decoders. The energy efficiency has been introduced to help a fair comparison, and is defined as $E_{eff} = \text{Pow}/(T \cdot \text{It}_{max})$, where T is the achieved throughput and It_{max} is the maximum

allowed number of iterations. This measure expresses the energy spent for each decoded bit. The area efficiency, instead, relates the area occupation normalized to the CMOS 65 nm process (An_{tot}) to T and the clock frequency f_{clk}, and is defined as $\mathrm{A}_{eff} = (T \cdot \mathrm{It}_{max}/f_{clk}) \cdot (1000/\mathrm{An}_{tot})$, where the 1000 multiplication factor changes the unit of measure from $[\frac{bits}{mm^2 \cdot cycles}]$ to $[\frac{bits}{mm^2 \cdot kcycles}]$. The +13.1% in A_{tot} that $\mathbf{A}_{new}$ exhibits w.r.t. $\mathbf{A}_{ref}$ leads to a lower A_{eff}. The effects of the proposed methods, though, can be really appreciated by observing Pow and E_{eff}. The reduction of the NoC clock frequency to 200 MHz overcompensates the increased peak power consumption caused by the additional logic, leading to improved E_{eff} values.

The multi-standard LDPC and turbo presented in [49] has a very small area occupation, with uneven throughput between LDPC and turbo mode. This situation leads to very high A_{eff} and E_{eff} in LDPC mode. $\mathbf{A}_{new}$, instead, is far more efficient in turbo mode, and yields better results also in [49] LDPC mode worst case operating conditions (N=672, r = 1/2, 20 iterations). The flexible LDPC/turbo decoder designed in [51] and $\mathbf{A}_{new}$ achieve comparable throughputs when decoding LDPC codes, with [51] having a larger An_{tot}. This leads to $\mathbf{A}_{new}$ having slightly better A_{eff} and E_{eff}: the gap is much larger in turbo mode.

The high parallelism, single-standard WiMAX LDPC decoder presented in [45] guarantees very high throughput with a 40 MHz frequency, that allows for reduced power consumption (51.6 mW)[0] and great efficiencies. Though [45] has been designed for ultra-low power consumption, and $\mathbf{A}_{new}$ targets multiple code types and standards, the power gap between them is 47.6 mW only. This work fares even better when compared to the dedicated ASIC targeting 3GPP-LTE turbo codes in [35], that yields good A_{eff} and throughput. The provided normalization results allow to estimate a Pow of 278.8 mW with the 90 nm node: power consumption is still higher than $\mathbf{A}_{new}$, with lower efficiency.

[0]Power normalized from 65 nm to 90 nm technology, with a 1.75 normalization factor obtained from [45]

Chapter 4

Early stopping of iterations for LDPC decoding

The energy expenditure of any LDPC decoder is directly linked to the iterative nature of the decoding algorithm. A maximum number of iterations It_{max} is usually set: however, a lower number of iterations is often sufficient to correctly decode a frame, while in other cases additional iterations after It_{max} might be necessary. Early Stopping Criteria (ESCs) aim at identifying such situations where further iterations are not useful, consequently interrupting the decoding and reducing the energy consumption. This chapter proposes an innovative multi-standard early stopping criterion for LDPC decoders: its effectiveness is evaluated against existing techniques both in terms of saved iterations and, after implementation, in terms of actual energy saving.

4.1 Early stopping criteria

Incorporation of ESCs in an iterative LDPC decoder enables energy saving by limiting the average number of decoding iterations [76, 88–91]: the evolution of a certain metric is analyzed over the iterations and a proper stopping rule is set. Quite a large number of ESCs have been described in the open literature: however, in most cases, the real potential of these methods in a multi-standard decoder has not been shown. The performance of ESCs is often provided only in terms of saved number of iterations, for a limited set of codes, and over the AWGN channel.

The hardware implementation of ESCs in a real decoder introduces overheads in terms occupied area and energy dissipated by the additional logic and memory. Therefore, the actual benefit offered by the incorporated ESC needs to be evaluated by taking into account these overheads and accurate post-layout information is required to this purpose. Moreover, ESC performance often depend on code features and channel conditions. Therefore, in the context of multi-standard implementations, it is important to show the obtained performance for a wide set of codes and assuming realistic channel models.

This chapter describes a novel ESC that has been integrated in a complete multi-standard decoder and tested on a wide range of LDPC codes, with both AWGN and Rayleigh fading channel models. First, the Multi-Standard Early Stopping Criterion (MSESC) for LDPC decoding is introduced: it outperforms existing ESCs in terms of number of iterations and energy consumption, and it is adaptive to various LDPC codes and channel conditions, thanks to on-the-fly threshold computation. Then, the MSESC is incorporated into an already available decoder and post-layout accurate evaluations of occupied area and consumed energy are derived. The presented work has been submitted to IET Communications [17].

ESCs allow to evaluate, after each iteration, if it is worth to proceed with the decoding process or not. In case the conditions of the ESC are met, the decoding is stopped, either interrupting the already initiated iteration or not even starting it. The Genie ESC is an unrealizable technique commonly utilized as reference for comparisons: using foreknowledge of the information bits, the decoding is initiated only if the outcome is going to be successful. Practical ESCs can be classified into two subgroups: ESCs identifying *successful decoding* of a frame, and ESCs identifying *impossible decoding*.

An efficient *successful decoding* ESC is the *parity check* method [92]. Since a received vector y is a valid LDPC codeword if and only if $\mathbf{H} \cdot y' = 0$, the syndrome calculation unfailingly identifies correct codewords. The *parity check* is performed after each iteration: if the codeword is correct the decoding is stopped, if not it proceeds until success or until the maximum number of iterations has

been reached.

Much more varied is the pool of *impossible decoding* ESCs. With "impossible" we refer to the decoding processes that are not going to be successful within the maximum number of allowed iterations. In [76] the Convergence of Mean Magnitude (CMM) method is presented. It measures the slope or changing ratio of the a posteriori LLR mean magnitude. In case of *impossible decoding*, this measure converges to and fluctuates around zero. If this situation is detected for long enough, the decoding is interrupted. In [89] the proposed ESC monitors the evolution of the syndrome SYN of the decoder. The decoding is stopped if SYN is larger than a certain threshold for three consecutive iterations, showing no sign of convergence. Two metrics are observed in the ESC described in [90]: the Check Node Mean Magnitude (CNMM) monitors the average value of $Q_{lk}[c]$ messages, while the Check Node Mean Checksum (CNMC) calculates the syndrome substituting $\lambda_k[c]$ with $Q_{lk}[c]$ (for the symbol definition and corresponding equations, please refer to Section 2.2). If CNMM keeps decreasing and CNMC increasing for a set of It_{ESC} consecutive iterations, the codeword is deemed undecodable, and the process is interrupted. The ESC proposed in [91] is tied to the SCMS algorithm, since it keeps track of the number of erased messages (Erased Messages ESC, EMESC). In case of *impossible decoding*, this number does not converge to zero and the decoding process shows a high degree of unreliability.

4.2 Multi-Standard Early Stopping Criterion

The proposed Multi-Standard Early Stopping Criterion (MSESC) relies on the computation of two metrics whose behavior over different iterations can be observed to detect an *impossible decoding*. The two considered metrics are defined as follows:

$$\mathrm{CNMM}^i \equiv \sum_{l=1}^{M} \min_{k \in N(l)} \left| R_{lk}^i \right| \tag{4.1}$$

$$\mathrm{SYN}^i \equiv \sum_{l=1}^{M} \bigoplus_{k \in N(l)} \left(\mathrm{HD}\left(\lambda_k^i[c] \right) \right) \tag{4.2}$$

where i is the iteration number, R_{lk}^i is the value obtained in (2.9) at the current iteration, $\mathrm{HD}(\lambda_k^i[c])$ stands for the Hard Decision on bit k based on the $\lambda_k^i[c]$ metric, and $\oplus$ is the XOR logic operation. The CNMM metric [90] is a measure of the reliability of the codeword as estimated by the parity checks. The CNMM is compared to the previous iteration's value to evaluate its slope. The syndrome computation SYN at i^{th} iteration as expressed in (4.2) is equivalent to (2.11), but highlights the simple operations involved. MSESC exploits CNMM and SYN to detect both *impossible* and *successful*

Algorithm 1 MSESC

```
1: CNT ← 0, Calculate T1, T2, T3
2: Activate IDD (Impossible Decoding Detection)
3: for all iterations 1< i < It_max do
4:     Decode, Calculate SYN^i
5:     if SYN^i = 0 then
6:         Stop Decoding (successful decoding)
7:     else if IDD is active then
8:         Calculate CNMM^i
9:         if i ≥ 2 and (CNMM^i > T2 or SYN^i < T3) then
10:            Deactivate IDD
11:        else if CNMM^i < CNMM^(i−1) and SYN^i > SYN^(i−1) then
12:            CNT ← CNT + 1
13:        else
14:            CNT ← 0
15:        end if
16:        if CNT = It_ESC then
17:            Stop Decoding (impossible decoding)
18:        end if
19:        if i ≥ 0.6 · It_max and SYN^i > T1 then
20:            Stop Decoding (impossible decoding)
21:        end if
22:    end if
23: end for
```

decoding.

The complete MSESC flow within the decoding process is described in Algorithm 1. The Impossible Decoding Detection (IDD) mode is initially activated. At the end of each iteration SYN^i is evaluated. The *parity check* ESC is then run (lines 5 - 6 in Algorithm 1) to see if a *successful decoding* is reached. The second ESC check (lines 8 to 21), aimed at detecting *impossible decoding*, is only run if IDD is active (line 7). The counter CNT is incremented at the end of the current iteration if CNMM is decreased and SYN is increased w.r.t. previous iteration (lines 11 - 15). When CNT reaches threshold It_{ESC}, *impossible decoding* is detected and the decoder is stopped (lines 16 - 17). Moreover, a stopping criterion has been introduced to identify slow convergence behavior, i.e. a codeword that is decodable per se, but not within the currently allowed maximum iterations of the decoder It_{max} (lines 19 - 20).

In case the decoding is going to be successful within It_{max} iterations, it is desirable to deactivate IDD to avoid unnecessary power consumption. For this reason, an adaptive deactivation function has been devised. If after few iterations (for example two) CNMM is larger than a threshold $T2$ or the syndrome SYN is smaller than $T3$, unsuccessful decoding is a very low probability event, and IDD is deactivated for the current codeword, avoiding further computations (lines 9 - 10).

To allow the decoder to dynamically adapt to the used code, on-the-fly computation of $T1$, $T2$

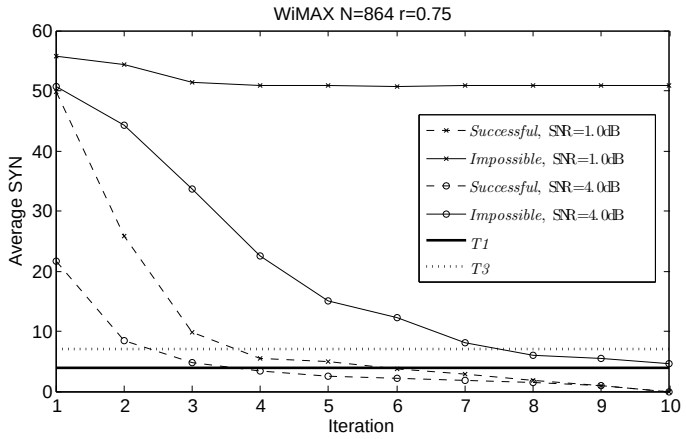

Figure 4.1: Average SYN for WiMAX N=864 r=3/4 in case of *successful* and *impossible decoding*

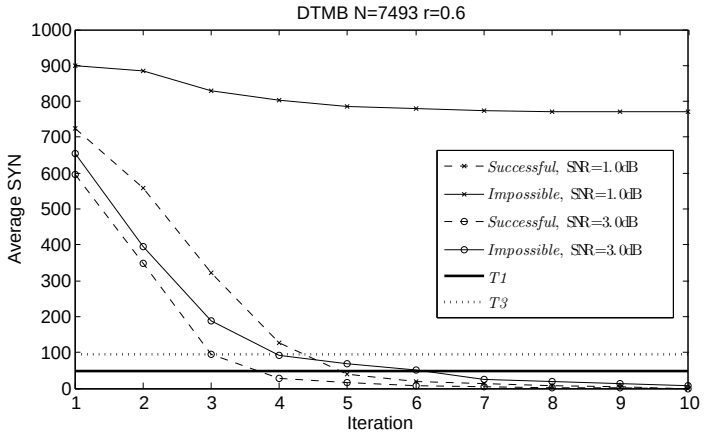

Figure 4.2: Average SYN for DTMB N=7493 r=3/5 in case of *successful* and *impossible decoding*

and $T3$ has been devised:

$$T1 = M \cdot 2^{-6} \tag{4.3}$$

$$T2 = M \cdot 2^{bits_f} \tag{4.4}$$

$$T3 = M \cdot 2^{-5} = T1 \cdot 2 \tag{4.5}$$

where $bits_f$ is the number of bits assigned to the representation of the fractional part of $\lambda_k[c]$.

Threshold $T3$ sets an upper limit for SYN, and expresses a percentage of wrong parity checks that is low enough to be likely corrected in case of *successful decoding*, i.e. 1/32 of M, where M is the number of rows of the LDPC parity check matrix. Fig. 4.1 and 4.2 plot the average SYN for a short WiMAX code and a long DTMB code. With *successful decoding*, SYN is monotonically decreasing and very steep in the first iterations: the number of wrong parity checks quickly descends below $T3$, while it is confined at much higher values in presence of *impossible decoding*. The choice of $T3$ scales well with the frame size N: long codes have high error correction capabilities and are able to sustain large $T3$ values, while short codes are less performing and require consequent $T3$ reduction. The same

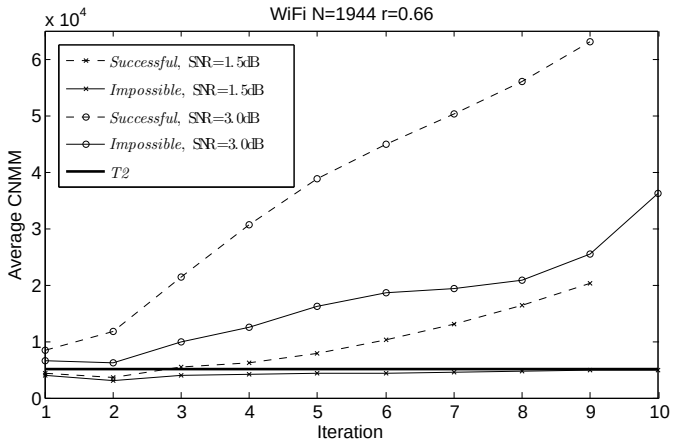

Figure 4.3: Average CNMM for Wifi N=1944 r=2/3 in case of *successful* and *impossible decoding*

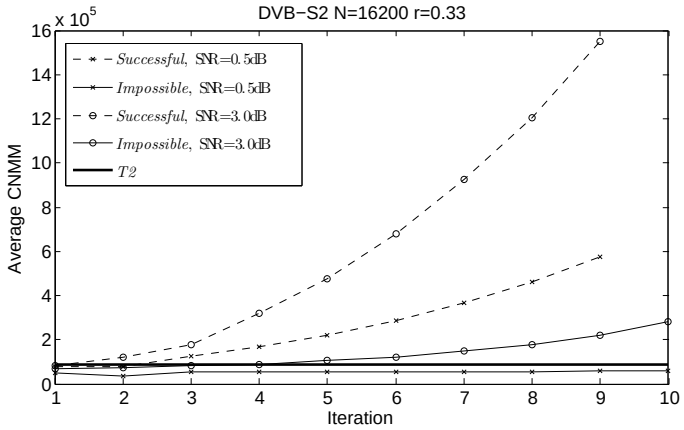

Figure 4.4: Average CNMM for DVB-S2 N=16200 r=1/3 in case of *successful* and *impossible decoding*

concept as $T3$ applies to $T1$, that identifies a percentage of M (1/64) not likely to be corrected within It_{max} due to slow convergence. As shown in Fig. 4.1 and 4.2, at high SNR also *impossible decoding* cases can manage to satisfy the condition in line 9, but this occurrence is taken care by $T1$ (line 19). The trajectories plotted in Fig. 4.1 and 4.2 show how both $T1$ and $T3$ depend on N and M. On the contrary, when min-sum decoding is adopted, the CNMM metric does not depend on the degree of check nodes, since only the minimum of $|R_{lk}^i|$ is considered for every parity check. Finally, $T1$ and $T3$, as well as $T2$, are independent of the considered channel mode, since different models only result in SNR shifts at reception [93].

$T2$ is compared in line 9 of Algorithm 1 to the value of CNMM at each iteration. The $|R_{lk}^i|$ elements that are summed in (4.1) are quantized with $bits_{tot} = bits_{int} + bits_f$: to take in account the initial left-shift of the integer part, in (4.4) also the value of M is left-shifted by the same number of positions $bits_f$. Consequently, for CNMM> $T2$ to be verified, the average value of $\min|R_{lk}^i|$ must be > 1. This is a condition verified early in case of *successful decoding*, while $|R_{lk}^i|$ values in *impossible decoding* tend to float much closer to zero. Fig. 4.3 and 4.4 show the average CNMM for a WiFi and a DVB-S2 code respectively, under different channel conditions. At low SNR, it can be seen how

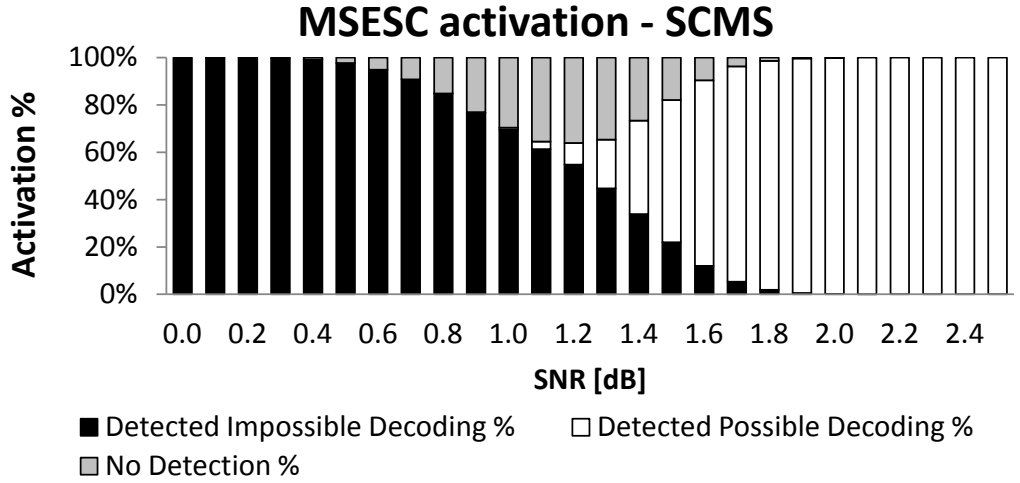

Figure 4.5: Activation percentages of MSESC for WiMAX 2304, 1/2

the average CNMM stays below $T2$ in case of *impossible decoding*, while it quickly rises above it in case of *successful decoding*. At high SNR, where also *impossible decoding* cases have CNMM> $T2$, the decoding is stopped by the condition in line 19.

The curves in Fig. 4.1 to 4.4 have been obtained with $bits_{tot}$ = 10 and $bits_f$ = 3. Typical internal quantization of bit LLRs, that does not cause signicant BER degradation, can be reduced to $bits_{tot}$ = 7 and $bits_f$ = 1, or even to $bits_{tot}$ = 6 and $bits_f$ = 0. The effectiveness of the proposed MSESC has been validated for various quantization parameters. The chosen thresholds maintain their validity with different quantizations: changes in $bits_f$ are taken in account by $T2$, while $bits_{int}$ will only impact the upper bound of the CNMM metric.

4.3 MSESC performance

In Fig. 4.5 statistics are plotted for the activation and deactivation of MSESC with SCMS. Results have been averaged over several millions frames. At each simulated SNR, the black bars indicate the percentage of frames for which an *impossible decoding* has been detected at some point within It_{max}, i.e. an effective early stopping. The white bars, on the contrary, show the percentage of frames for which IDD has been deactivated (line 10 in Algorithm 1). This happens when the algorithm decides that the *impossible decoding* event will not occur for the current frame: in this case, a *possible decoding* is detected and a *successful decoding* will be reached in a later iteration. It is possible to see how, when the SNR increases, the *impossible decoding* detection percentages decrease while the IDD deactivation percentages increase. However, there is a third possible situation, in which IDD is not deactivated but no early stopping is enforced. This case is represented by the gray *no detection* bars,

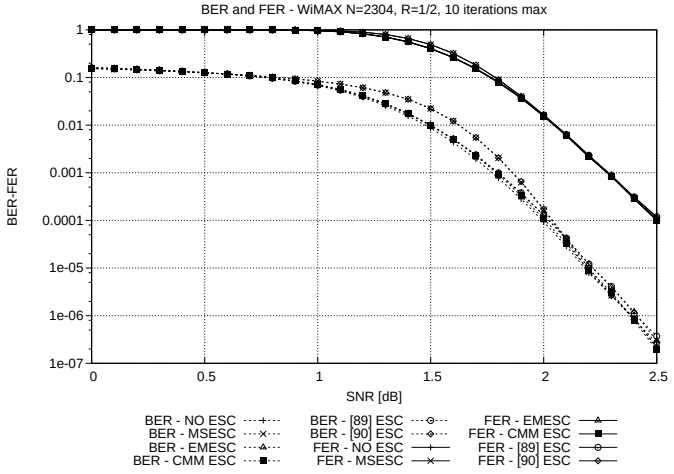

Figure 4.6: BER and FER curves with Early Stopping Criteria

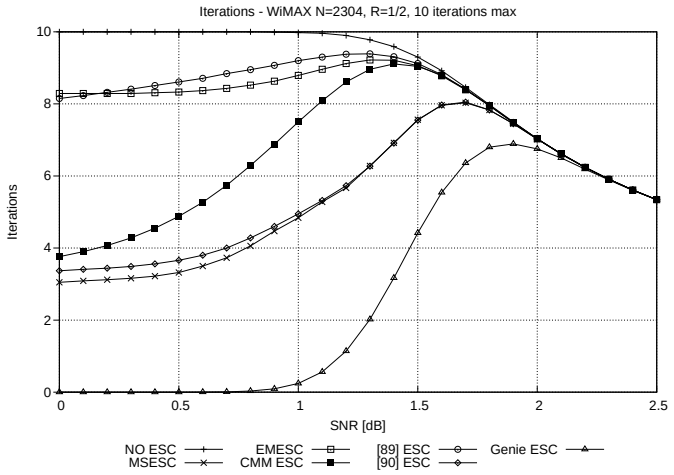

Figure 4.7: Average number of iterations with different ESCs

which only cover a small range of SNRs.

The proposed stopping criterion has been compared to existing ESCs in terms of BER and FER degradation and iteration reduction capabilities. Fig. 4.6 and 4.7 present simulation results for WiMAX code with N = 2304, r = 1/2 on an AWGN channel, with It_{max} = 10. For the threshold $T2$, we considered in these simulations $bits_f$ = 3 as the number of quantization bits to represent the fractional part of the bit LLR $\lambda_k[c]$.

The internal quantization of the decoding algorithm is 10 bits, including $bits_f$ = 3. However, MSESC is affected only by the number of $bits_f$ (taken in account by $T2$) and not by the total number of bits: its effectiveness is consequently guaranteed for all quantizations.

Fig. 4.6 plots the BER (dotted lines) and FER (continuous lines) curves for the *impossible decoding* ESCs analyzed earlier in this chapter: minor performance degradations are present early in the waterfall region.

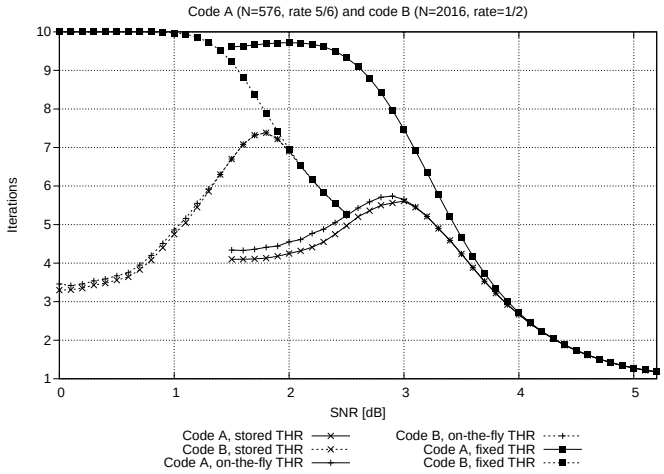

Figure 4.8: Average number of iterations with MSESC and different thresholds

Fig. 4.7 shows the average number of iterations (It_{AVG}) performed by various ESCs: the Genie ESC curve is used as a reference plot. All ESC are supposed to work in conjunction with *parity check* ESC for the sake of a fair comparison. Most of the curves join the "parity check" curve at around SNR=1.6 dB, where the number of detected undecodable frames is close to zero. The [89] ESC and EMESC show similar performance, with two saved iterations at zero SNR, while CMM performs much better at low SNR. A large number of saved iterations can be observed for MSESC and [90] ESC. The dynamic correction in SCMS can in fact result in sudden changes in the CNMM measure, that can occasionally lead to additional errors (Fig. 4.6). However, these rapid metric variations allow for early detection of *impossible decoding*, with consistent saving of iterations. MSESC is the most effective method, with three average iterations performed at low SNR.

In order to illustrate the potential of the on-the-fly threshold computation incorporated in MSESC, in Fig. 4.8 the average iteration number for two codes A (N = 576, r = 5/6) and B (N = 2016, r = 1/2) are evaluated with different $T1$, $T2$ and $T3$ management policies.

- *Stored THR*: in this case, thresholds are individually optimized for each code, via extensive simulations. These thresholds are pre-computed, stored, and read according to the code currently in use, allowing for the best MSESC average iterations results.

- *On-the-fly THR*: in this case, thresholds are computed on-the-fly according to (4.3), (4.4) and (4.5). The differences observed with respect to the stored threshold methods are negligible, and no memory has to be allocated to store the thresholds required for each supported code.

- *Fixed THR*: these curves show the effect on MSESC performance in case thresholds are fixed for all considered codes. In particular, the effects of strongly suboptimal thresholds are reported: to code A are applied thresholds fine-tuned for code B, and vice versa. Since size and rate are

Table 4.1: Effect of MSESC on complexity (A), average power consumption (P) and energy per decoded frame (E_F) (It_{max} = 10, CMOS 90 nm technology, 1.0 V supply, post-layout)

	CODE	SNR	$\mathbf{A}_{\text{REF}}$	$\mathbf{A}_{\text{PC+CMM}}$		$\mathbf{A}_{MSESC}$	
A [mm^2]	-	-	2.40	2.49	+3.8%	2.43	+1.3%
P [mW]	-	-	90.2	95.5	+5.9%	92.9	+3.0%
E_F [μJ]	2304 1/2	0.0 dB	1.49	0.69	-53.7%	0.46	-69.2%
		1.0 dB		0.79	-47.0%	0.75	-50.4%
		2.0 dB		1.10	-26.2%	1.05	-29.5%
	960 2/3	0.0 dB	0.66	0.31	-53.1%	0.20	-69.7%
		1.0 dB		0.35	-47.0%	0.33	-50.0%
		2.0 dB		0.49	-25.8%	0.46	-30.3%
	1944 3/4	0.0 dB	1.32	0.61	-53.8%	0.41	-68.9%
		1.0 dB		0.70	-47.0%	0.66	-50.0%
		2.0 dB		0.97	-26.5%	0.93	-29.6%
	1440 5/6	0.0 dB	0.98	0.46	-53.1%	0.30	-69.4%
		1.0 dB		0.52	-47.0%	0.50	-49.0%
		2.0 dB		0.73	-25.5%	0.69	-29.6%

substantially different for the two codes, the optimal thresholds can vary consistently, causing severe performance degradation.

Similar behavior has been observed for a wide analyzed range of LDPC codes (WiFi, WiMAX, DTMB, CMMB, DVB-S2). The results illustrate how the proposed on-the-fly computation of thresholds combines both high flexibility and high accuracy.

4.4 MSESC Hardware Structure and Implementation

In order to evaluate the benefits of an ESC in terms of energy consumption, it is necessary to integrate it in a real hardware channel decoder and to compare the overheads related to its integration against the benefits related to the reduced number of decoding iterations. To that purpose, we applied the proposed MSESC to a fully characterized multi-standard channel decoder [14]. The implementation referred in [14] as A has been considered after adaptation to support only LDPC codes and to use the SCMS decoding algorithm. This version of [14].**A** has been named $\mathbf{A}_{\text{REF}}$ and has been used as a reference in the following comparisons. It relies on 22 Processing Elements (PEs) connected by means of a Kautz [80] network-on-chip, with routing elements of degree three. The decoder supports all LDPC codes in WiMAX and WiFi standards.

All of the operations required by the proposed MSESC can be implemented with a very low hardware cost. On-the-fly thresholds computation, which is required only when switching from a code to a new one, simply implies shift operations. Therefore, its contribution to power consumption

is negligible. On the contrary, power consumption overhead related to the implementation of parity check ESC is not negligible, although it is rarely reported in existing papers on ESC methods. The great majority of current decoders relies on multiple PEs, and the data need to be gathered from all of them to compute SYN, thus usually implying non negligible overheads. Partially parallel decoders can exploit the idea behind the on-the-fly syndrome calculation proposed in [91]: after the execution of (2.10), the sign bits of the $\lambda_k^{new}[c]$ involved in the current parity check are XORed. These values are concurrently available to the PE without requiring additional memory accesses. During each iteration, the resulting bit in every PE is summed to the bits of the preceding parity checks through an adder modulo $\lceil M/P \rceil$, where P is the number of PEs. At the end of an iteration, each PE holds a partial syndrome value: these are gathered through dedicated connections and summed one at a time to obtain the global syndrome value. Since the number of PEs is typically much smaller than M, such architecture achieves good compromises in terms of complexity and latency. CNMM can be implemented with the same on-the-fly approach by accumulating R_{lk}^{new} of every parity check as soon as they are computed. As with SYN, partial CNMM values are retrieved and summed together.

Table 4.1 reports area, power and energy consumption of different implementations: $\mathbf{A}_{\text{REF}}$ is the reference architecture mentioned above, $\mathbf{A}_{\text{PC+CMM}}$ implements both CMM [76] and *parity check*, $\mathbf{A}_{\text{MSESC}}$ includes the proposed multi-standard ESC. These three architectures have been implemented targeting 90 nm CMOS technology. The simple additional logic required in $\mathbf{A}_{\text{MSESC}}$ results in +1.3% area occupation and +3.0% average power consumption after layout, while the area and power consumption overheads are higher with $\mathbf{A}_{\text{PC+CMM}}$. In fact, in [76] it is described how a single CMM threshold value can be used for all codes, leading to small differences in performance. While this is true for the SNR intervals considered in [76], a single threshold causes large differences in the number of performed iterations at low SNR, as it has been shown in Fig. 4.8. Moreover, BER degradation can be observed at all SNR when very different codes use the same CMM threshold. It is consequently necessary a memory to store a threshold value for every code supported, and in a multi-standard decoder like the one considered the cost of such memory is not negligible. Between WiFi and WiMAX, a total of 131 different LDPC codes are specified: thus, thresholds are calculated off-line and stored in a dedicated memory at least 131 words deep and with width greater or equal to the number of bits assigned to the representation of LLRs. This memory and the more complex CMM computations are the main contributions behind the +3.9% area and +5.9% power consumption in the $\mathbf{A}_{\text{PC+CMM}}$ case. This overhead is avoided in $\mathbf{A}_{\text{MSESC}}$ thanks to on-the-fly threshold computation.

The benefits of ESCs can be appreciated only over the whole decoding process: E_F expresses the

average energy spent in the decoding of a frame for different codes and SNR points. It is defined as:

$$\mathrm{E}_F = \frac{\mathrm{P} \cdot cycles_{it} \cdot It}{f} \tag{4.6}$$

where P is the average power consumption, $cycles_{it}$ the clock cycles needed to complete an iteration, f the clock frequency and It the number of performed iterations. In $\mathbf{A}_{\mathrm{REF}}$ no ESC is present, and $It = It_{max} = 10$ for all the considered codes. Consequently, the consumed energy does not vary with the SNR, but only with the code change. The situation is different in case of $\mathbf{A}_{\mathrm{MSESC}}$, in which the performed iterations follows a pattern close to that of Fig. 4.7. Moreover, thanks to the activation and deactivation function (line 9 of Algorithm 1) the contribution of MSESC to E_F must be weighted according to the percentages displayed in Fig. 4.5. E_F is calculated as

$$\begin{aligned} \mathrm{E}_F = & \frac{(\mathrm{P}_{\mathrm{REF}} + \mathrm{P}_{\mathrm{PC}}) \cdot It \cdot cycles_{it}}{f} \\ & + \frac{cycles_{it} \cdot \mathrm{P}_{\mathrm{IDD}} \cdot (It_D \cdot DP_{\%} + It \cdot (1 - DP_{\%}))}{f} \end{aligned} \tag{4.7}$$

where $\mathrm{P}_{\mathrm{REF}} + \mathrm{P}_{\mathrm{PC}}$ is the power consumption of $\mathbf{A}_{\mathrm{REF}}$ and the *parity check* part of MSESC, $\mathrm{P}_{\mathrm{IDD}}$ is the power consumption due to the IDD part of MSESC, $DP_{\%}$ is the deactivation percentage at the considered SNR point, and It_D the average deactivation iteration. Note that at high SNR $DP_{\%}$ is close to 1, leading to a very small $\mathrm{P}_{\mathrm{IDD}}$. Table 4.1 displays E_F for four codes taken from the supported standards, considering three SNR points for each code. At SNR=0.0 dB MSESC manages to reduce by around 70% the energy consumption with respect to $\mathbf{A}_{\mathrm{REF}}$, while at SNR=1.0 dB the gain is reduced to 50%. At higher SNR the IDD is mostly deactivated, while the *parity check* part of MSESC allows for an average 30% gain. The gains are less significant for $\mathbf{A}_{\mathrm{PC+CMM}}$. At low SNR, CMM implies higher average number of iterations than MSESC, with $\mathbf{A}_{\mathrm{MSESC}}$ saving around 16% more energy than $\mathbf{A}_{\mathrm{PC+CMM}}$, while the increased power consumption due to the threshold memory and the absence of a deactivation function leads to larger E_F (4-5%) also at high SNR.

Another interesting assessment can be done by considering application-specific operational conditions in terms of SNR probability distribution. In fact, real world applications place their SNR working point taking in account large variation margins, in order to have very low probabilities of SNR that prevent the correct functionality. For example, for a typical indoor WiFi connection the industry suggests a minimum of 20 dB SNR to guarantee a very good level of connectivity [94]. Assuming a Rayleigh fading channel model that remains stable for the duration of a frame, it is possible to correlate the corresponding BER curves with those obtained with an AWGN channel [93]. This property can be joined with the instantaneous SNR probability density function (p.d.f.) of a Rayleigh

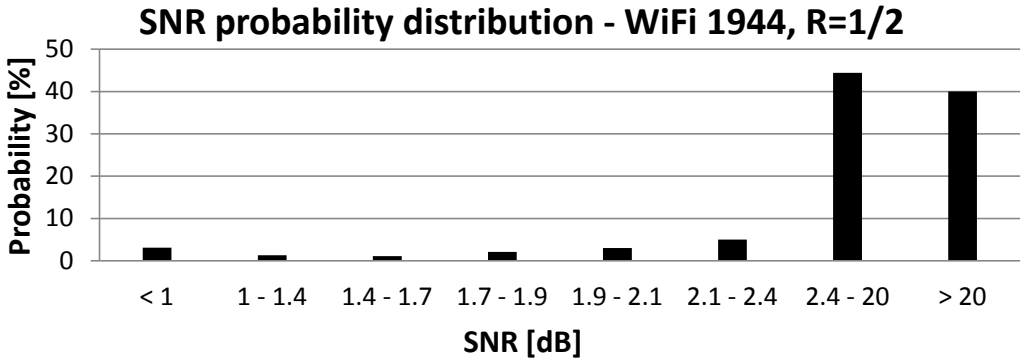

Figure 4.9: SNR probability distribution for WiFi 1944, 1/2

Table 4.2: Implementations comparison on average iterations (It_{AVG}) and energy (E_F) for Wifi N = 1944, r = 1/2 code SNR probability distribution (CMOS 90 nm, 1.0 V, post-layout)

	It_{MAX}	It_{AVG}	E_F [μJ]	$\mathbf{A}_{\text{MSESC}}$ gain
$\mathbf{A}_{\text{REF}}$	10	10	1.285	-77.7%
	20	20	2.571	-87.2%
$\mathbf{A}_{\text{PC}}$	10	2.47	0.321	-10.6%
	20	3.13	0.407	-19.4%
$\mathbf{A}_{\text{MSESC}}$	10	2.20	0.287	-
	20	2.52	0.328	-

channel to map an equivalent SNR p.d.f. over the AWGN channel. For example, using the WiFi code with block N = 1944 and r = 1/2, at 10 dB on a Rayleigh channel the BER is 10^{-5}, and the same level of BER is obtained at 2.2 dB on AWGN. Since on the Rayleigh channel there is a 10% probability of SNR< 10 dB when the average SNR is 20 dB [93], there is an equal probability of SNR< 2.2 dB on the equivalent AWGN. Fig. 4.9 shows the probability distribution of SNR for a WiFi LDPC code [93,94]. It is possible, at this point, to weigh the average number of iterations at each SNR point with the probability for the decoder to work under said conditions, in order to get an It_{AVG} valid for all considered SNR points. Table 4.2 shows the energy consumption and It_{AVG} obtained for the aforementioned WiFi code with It_{max}=10 and It_{max}=20. The implementation labeled as $\mathbf{A}_{\text{PC}}$ was obtained from $\mathbf{A}_{\text{REF}}$ by including the *parity check* ESC. $\mathbf{A}_{\text{MSESC}}$ yields very good E_F figures, with a gain ranging from 77.7% to 87.2% with respect to $\mathbf{A}_{\text{REF}}$, and decreasing the energy consumption of 10.6% and 19.4% compared to $\mathbf{A}_{\text{PC}}$.

Chapter 5

Error resiliency in LDPC decoders

LDPC code decoders heavily rely on memories for their computations: they constitute the majority of the occupied area and account for most of the power consumption. Consequently, the correct behavior of memory elements is of great importance for such an application. Energetic particles hitting memory cells can induce soft errors, i.e. faults in the stored data that are not caused by permanent hardware damage: the probability of their occurrence has known an upward trend within the latest technology nodes. On the other hand, aging and usage can compromise the physical integrity of memories, causing hard errors like stuck-at bits. Given that the purpose of channel decoders is to correct errors, LDPC decoders are intrinsically characterized by a certain degree of resistance to hardware faults. This characteristic, together with the nature of the information typically stored in LDPC decoder memories, allows for ad-hoc error protection techniques. In this chapter, a novel Unequal Error Protection technique is proposed and applied to an LDPC decoding case of study. Its implementation cost is evaluated and compared to existing techniques.

5.1 Unequal Error Protection of memories in LDPC decoders

The characteristic error resilience of LDPC decoders has been exploited in past fault tolerant designs. One of the few works on the topic is [95], where the error resilience of a complete LDPC decoder architecture is analyzed. Different techniques are applied to the decoder modules according to the level of criticalness, and the relative overheads evaluated. Memories obviously result to be some of the most critical modules, and they are protected according to their role in the decoding process. Doubling or tripling of the MSB is employed together with dynamic corrections. A different approach is instead exploited in [96], where statistical error compensation is used to overcome the performance loss brought by voltage overscaling.

The work presented in this section focuses on providing error resilience to LDPC decoder memories in presence of large soft error probabilities or severe physical degradation. An analysis of the impact of errors on the decoding performance is carried out, and the level of criticalness of each bit is determined. Different degrees of protection are employed, leading to the development of an Uneven Error Protection (UEP) methodology. The presented work has originated a collaboration with professor Paolo Montuschi and the targeted publication is IEEE Transactions on Computers.

5.2 Error analysis

Almost all practical implementations of LDPC decoders make use of memories to store LLRs between iterations or, in case of multi-core decoders, to exchange informations between processing elements. The number of memory read and write accesses can be extremely high both considering the total lifetime of the decoder and each decoding process alone. For example, a small WiMAX LDPC code requires around 7000 memory accesses between read and write operations for a single iteration, but this number can surpass the million in case large codes are employed, like in DVB-S2.

The two quantities usually stored in LDPC decoder memories are $\lambda_k[c]$ and R_{lk}, that are read in (2.6) and updated in (2.9) and (2.10). Depending on the involved quantities, errors in read and write operations impact differently on the decoding process. For example, a wrong $Q_{lk}[c]$ (2.6) does not necessarily result in an incorrect R_{lk}^{new}. As (2.12) shows, only the first and second minimum among the $Q_{lk}[c]$ involved in the check node computation are considered for the selection of R_{lk}^{new}: consequently, the error might not be propagated to other $\lambda_k^{new}[c]$. Moreover, since the LDPC decoding process relies on soft information, the performance degradation caused by a flipped bit in $\lambda_k^{old}[c]$ or R_{lk}^{old} depends on the position of the error, as shown in the following analysis. Supposing a quantization on b bits, let us call MSB1 the Most Significant Bit, and MSBb the Least Significant Bit. As a case

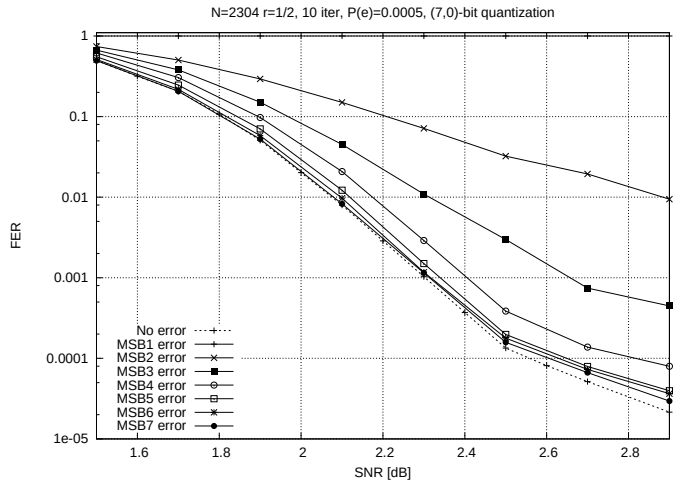

Figure 5.1: FER - errors on different MSBs

of study, both $\lambda_k^{old}[c]$ and R_{lk}^{old} have been quantized with $b = 7$ (7 bits for the integer part and 0 for the fractional part): it is a common choice, since a smaller b will cause non-negligible performance degradation, while in most cases additional bits do not bring sensible improvements.

5.2.1 Variation of MSB

The curves in Fig. 5.1 show the FER for the WiMAX code of size $N = 2304$ and rate $r = 1/2$ on the AWGN channel. The decoding has been performed with the SCMS approximation. The "no error" curve plots the FER under ideal hardware conditions, i.e. without any error in the read and write operations. The other curves have been obtained by allowing errors on a single bit of $\lambda_k^{old}[c]$ and R_{lk}^{old} with a probability equal to P(e)=0.0005. Errors injected on the sign bit (MSB1) are disruptive at every Signal-to-Noise Ratio value (SNR), and cause unrecoverable errors. The criticalness of the sign bit in LDPC decoding is well documented and protection has been employed in the past [95]. The MSB2 and MSB3 error curves show severe degradation with respect to the ideal one. Both MSB2 and MSB3 account for a considerable percentage of the total dynamic, and can cause strong variations in both R_{lk}^{new} and $\lambda_k^{new}[c]$: particularly disruptive occurrences can easily lead to sign bit flips. Less critical is the impact of MSB4 errors, while errors injected in MSB5, MSB6 and MSB7 result in similarly small performance degradations.

5.2.2 Variation of It_{max}

The bit-per-bit error analysis has been extended to different values of It_{max}. The increased correction capability brought by additional iterations can in fact be dampened by the introduction of new errors. Fig. 5.2 plots the FER for three different error bits and three It_{max} values. While the effect of errors

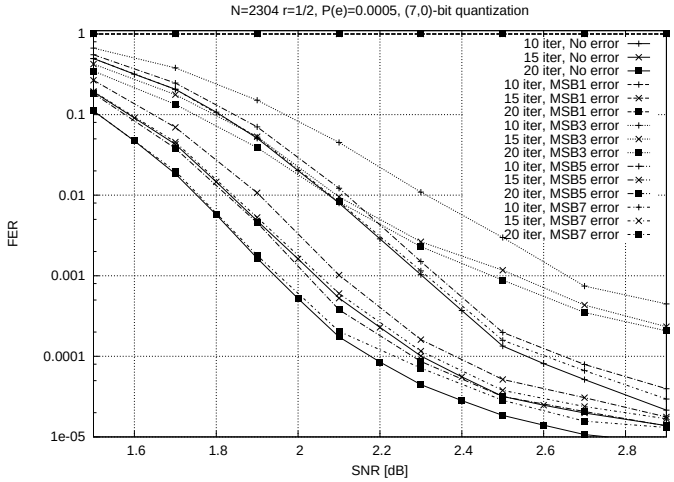

Figure 5.2: FER - errors on different MSBs and variation of It_{max}

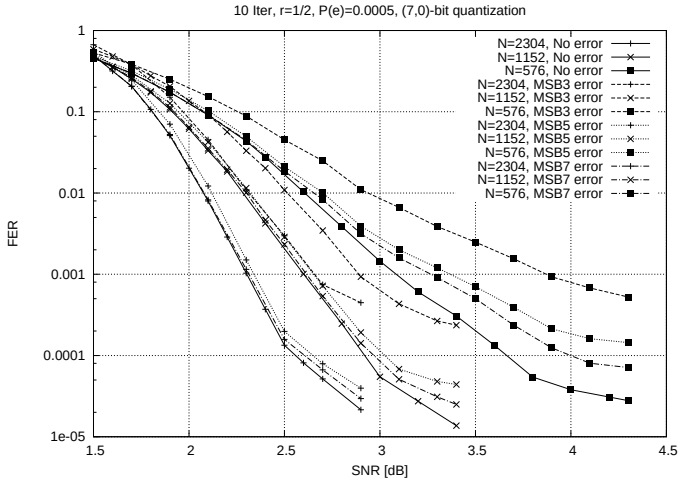

Figure 5.3: FER - errors on different MSBs and variation of code size

is almost the same with It_{max}=10 and It_{max}=15, the degradation caused by erroneous bits is more evident when It_{max}=20: a larger gap can be noticed between the "no error" curve and the MSB7 curve w.r.t. the other two cases, while the MSB5 and MSB3 curves are shifted proportionally.

5.2.3 Variation of code rate r and size N

Code rate and code size play a major role in the criticalness of errors. Fig. 5.3 considers three codes with different sizes and the same rate. The error injection probability has been maintained at P(e)=0.0005 for all the codes: this means that smaller codes will lead to a lower number of wrong bits per frame. However, it can be noticed how smaller codes are more sensitive to faults, regardless of the fewer injected errors. At FER= 10^{-4}, MSB7 errors cause a loss of 0.05 dB to the N=2304 code, while 3.5 dB are lost with N=576.

Fig. 5.4 is complementary to Fig. 5.3, with fixed code size and varying rate. The codes are taken

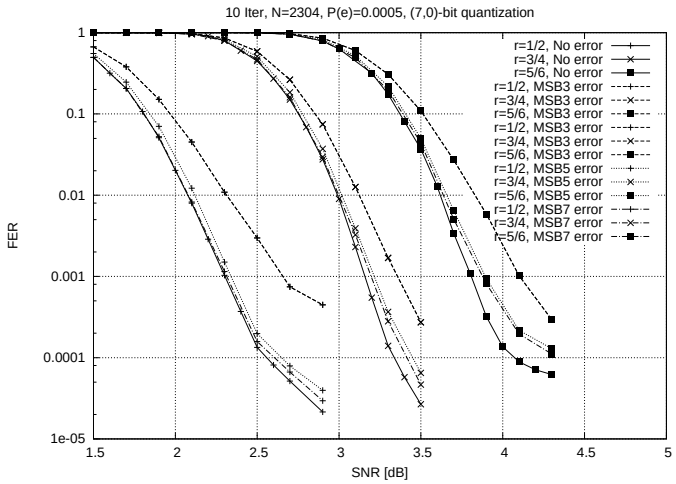

Figure 5.4: FER - errors on different MSBs and variation of code rate

from the WiMAX standard: increasing the code rate, the total number of $\lambda_k^{old}[c]$ and R_{lk}^{old} is kept almost constant, since the rows of the **H** matrix increase their weight. However, the employment of SCMS can often mask errors on R_{lk}^{old}: increments in code rate consequently lead to a lower number of potentially dangerous bits per frame. Rate and error injection variations do not scale proportionally, as it has been already observed with changes in code size: high rate codes are more sensitive to faults. The performance loss caused by MSB7 errors at FER= 10^{-4} is 0.05 dB for code rate 1/2, and 2.5 dB for code rate 5/6.

5.2.4 Variation of quantization and decoding algorithm

Fault tolerance of LDPC decoding has also been analyzed also in terms of quantization. The results meet our expectations, as errors affect the decoding performance in proportion to the weight of the erroneous bit on the overall dynamic range, regardless of the quantization, and FER curves are similar to those in Fig. 5.1. A final experimentation has been carried out by comparing different decoding algorithms. The decoding performance of the SCMS approximation is intrinsically more resistant to hardware errors than more common approximations of the BP algorithm like the NMS approximation [30]. Fig. 5.5 shows the the FER degradation due to P(e)= 0.0005 for a code decoded with both NMS and SCMS, and the inherent resilience of SCMS can be easily noted. This is due to the puncturing of $Q_{lk}^{i}[c]$ in presence of a sign change, that introduces an additional barrier to the propagation of disruptive errors.

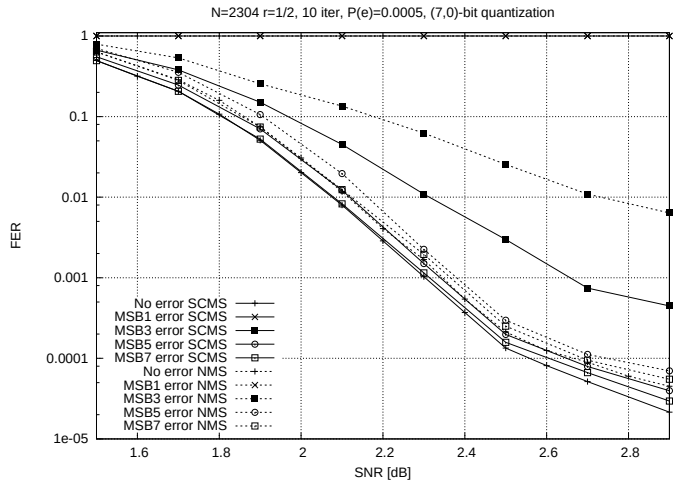

Figure 5.5: FER - errors on different MSBs and variation of decoding algorithm

5.3 Unequal Error Protection

The analysis on the impact of the different errors carried out in Section 5.2 highlighted that not all errors on the bits have the same influence on the FER of LDPC decoders. This is an important result, as it identifies a characteristic of LDPC decoders that can be used to increase the reliability of the decoding process. In particular, based on the study of Section 5.2, we observe that it is possible to apply distinct error protection techniques to each bit or groups of bits depending on their importance and influence on the FER. For this reason we have devised an UEP subdivided into four tiers of possible error protections.

For each Tier we are proposing a corresponding protection choice. Clearly, stronger or weaker protection schemes as well as identification of different "bounds" for each Tier, are possible. However, as we will see in Section 5.4 the proposed choices represent a good solution with a balances tradeoff of performances, costs, computation time & latency, and hardware requirements.

5.3.1 Tier 1 - full recovery

The highest level of protection is applied to bits which reliability is mandatory for a correct decoding i.e. the sign bit and possibly the magnitude MSBs. LDPC decoders base their hard decision on bits on the sign of their respective LLRs. Corrupted sign bits will consequently have catastrophic effects on the decoding, since they may cause an avalanche of undesired bit flipping. The same can happen in case errors occur on bits representing a large part of the total dynamic; sudden increments or decrements in LLRs will be interpreted as changes in the reliability, that will tip the evolution of LLRs towards misleading directions. To provide a high level of reliability and recovery, our choice has been that bits falling within the Tier 1 protection level are tripled during write operations: at load

time, a majority voter selects the most probable output.

5.3.2 Tier 2 - recovery of critical errors

It has been observed through simulations that a very large percentage of cases in which a wrong bit leads to an incorrect $\lambda_k^{new}[c]$ (2.10) within the same iteration falls within a distinct output bit pattern. These abrupt value changes cannot be handled by the inherent error correction of LDPC decoders. It is however possible to observe the occurrence of these patterns in case of errors to be able to determine their impact on the overall decoding process. We have chosen to add a parity bit to the Tier 2 bits, and in case of discrepancy during load operations the pattern recognition system is activated. If the output $\lambda_k^{new}[c]$ matches the critical bit pattern, recovery is possible by observing how $\lambda_k^{new}[c]$ varies if the Tier 2 bits at the input change. The distinctive bit pattern is dependent on the total quantization of the LLRs and on the position of the wrong bit, while the Tier 2 bits must be chosen with care to obtain the maximum effectiveness: thus, every case must be analyzed separately. More details are given in Section 5.4.

Tier 2 is not able to give the same level of protection as Tier 1, but gives a very good percentage of identification and recovery of errors that have been observed to be the main cause for LDPC decoder performance loss.

5.3.3 Tier 3 - error impact limitation

Since the magnitude of an LLR is a measure the information reliability, we have chosen to exploit this concept to the third tier of protection, designed for bits of medium-to-low significance. A parity bit is added to the protected bits during write operations. When the LLR is loaded, parity is recomputed and in case of discrepancy the contribution of all the bits falling within Tier 3 is nulled or reduced. With a 2's complement representation bit puncturing can require bit flipping among Tier 2 and Tier 1 bits as well: to avoid complex operations, depending on Tier 2 and Tier 1 values, a partial puncturing can be employed.

Tier 3 protection does not allow to recover from errors, but reduces their impact by decreasing the LLR magnitude, that in turn induces a conservative behavior in the decoder. This method can not be applied to bits expressing large percentages of the total dynamic, since the LLR magnitude change would be too large and cause errors.

5.3.4 Tier 4 - no protection

As shown in Section 5.2, errors on the least significant bits seldom affect the overall decoding performance. We have chosen to leave a set of low-importance bits unprotected without incurring in degradation.

5.4 A practical case of study

This section presents the performance evaluation of the proposed UEP under the same conditions of the error analysis carried out in Section 5.2. Both R_{lk} and $\lambda_k[c]$ are represented in two's complement and quantized with $b = 7$, all assigned to the representation of the integer part. The SCMS algorithm is considered, for its enhanced error resiliency w.r.t. other min-sum approximations. Figures have have been obtained for a WiMAX $N = 2304$, $r = 1/2$ LDPC code, with a maximum of 10 iterations per frame.

The decoders in [97] and [14] present similarities with many other decoders in the state of the art (serial core, min-sum-based layered decoding, partial parallelism, shared or dedicated memories, high-throughput or flexible design). They are representative examples of the current literature on the subject, and have been considered for the following performance evaluation. The LDPC code decoder architecture described in [97] has been thought for QC-LDPC codes as the ones in WiMAX, and relies on a partial parallelism structure in which as many cores as the **H** matrix layers communicate via shared memory banks. Each core is constituted of a single serial pipelined datapath implementing one of the min-sum modifications, while the overall decoder makes use of the layered scheduling approach. This particular architecture has been implemented in 65 nm CMOS technology, and yields high throughput working at more than 1 GHz frequency. On the contrary, the decoder in [14] targets flexibility: each core makes use of a dedicated memory bank, while being connected to the others through a Network-on-Chip. Implemented in 90 nm CMOS technology, reaches a much lower throughput at 200 MHz, providing support for a wider range of codes.

Errors in memories and logic are mostly identified by the Mean Time Between Failures (MTBF, [s]) or Failure In Time (FIT, number of errors in 10^9 hours). The effect of errors in an LDPC decoder, however, is strongly dependent on the speed of the decoder. For example, a small MTBF will have little effect on the performance of a fast decoder like [97]: the same MTBF could be disastrous for [14], that works at less than 1/5 of the frequency and has a lower degree of parallelism, thus requiring a higher number of clock cycles to complete an iteration. A fair error measure can consequently be the average number of errors encountered by the decoder during each iteration, here defined as Average

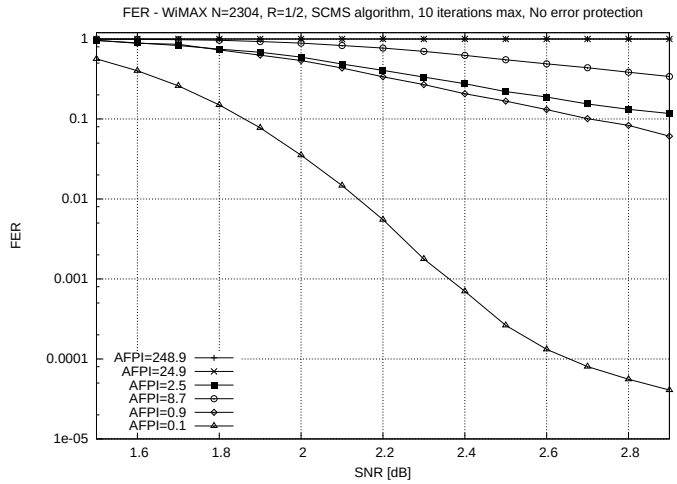

Figure 5.6: FER with different AFPI - No error protection

Failures Per Iteration (AFPI). Their frequencies and memory structure of [97] and [14] have been used to provide three error scenarios each (low, medium and high error probability), resulting in six AFPI values.

Performance

Fig. 5.6 shows the FER of the considered code under the influence of six AFPI values: each bit of the stored R_{lk} and $\lambda_k[c]$ has an equal probability of being flipped in presence of a soft error. While it is understandable how high AFPI irremediably degrade the FER, a lower failure rate should be sustained by a system designed to correct errors: however, it can be seen how also low AFPI values greatly affect the decoder.

Tier 1 protection gives a high degree of reliability, but requires two additional bits for every protected bit. In works like [95], it is shown how the correctness of the sign bit is the minimum requirement to attempt a correct decoder functionality. By applying Tier 1 protection on both R_{lk} and $\lambda_k[c]$ sign bits, the curves shown in Fig. 5.7 have been obtained. A definite improvement can be noticed for all the AFPI values except AFPI=248.9, and no degradation can be observed at all for the AFPI=0.09 case, at the cost of a 28.6% increase in memory bits. It has been observed that errors on R_{lk} are less critical than those on $\lambda_k[c]$ and less prone to propagating, thanks to the local nature of R_{lk} and the masking capabilities of the min-sum algorithm already addressed in Section 5.2. Consequently, Tier 1 on the MSB is sufficient for R_{lk} values.

As mentioned in Section 5.3.2, the pattern recognition and error recovery involved in Tier 2 protection must be evaluated in every situation. In this case study, with $b = 7$ and Tier 1 protecting the MSB, Tier 2 has been chosen to include MSB2 and MSB3. The precision of the identification

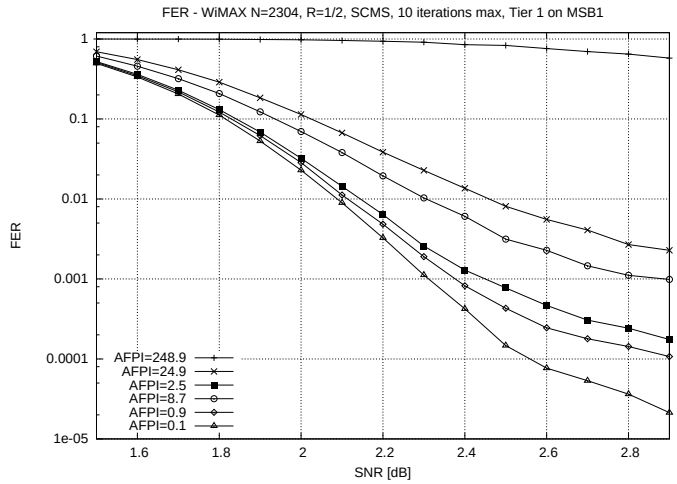

Figure 5.7: FER with different AFPI - Tier 1 on MSB1

of critical errors through output bit patterns is inversely proportional to the number of protected bits: however, a minimum of two bits must be included in Tier 2 to be able to recover from errors. Moreover, limiting Tier 2 to two bits reduces the complexity of the error recovery system, that rises with the number of bits. Let us define $\lambda_k^{new}[c]^C$ the correct result of (2.10), and $\lambda_k^{new}[c]^I$ the incorrect one. $\lambda_k^{new}[c]^C$ is obtained in case all the $\lambda_k[c]$ involved in (2.6) are devoid of errors, while in the computation of $\lambda_k^{new}[c]^I$ there is an error in the MSB2 or MSB3 of one of the $\lambda_k[c]$ in (2.6). From simulation it is revealed that regardless of the channel conditions, the majority of errors on MSB2 or MSB3 that result in uncorrectable codewords are characterized by the following relations:

$$\lambda_k^{new}[c]^C \rightarrow \text{MSB1 = MSB2 = MSB3} \tag{5.1}$$

$$\lambda_k^{new}[c]^I \rightarrow \text{MSB1 = MSB2 = MSB3} \tag{5.2}$$

$$\text{MSB1}(\lambda_k^{new}[c]^C) \neq \text{MSB1}(\lambda_k^{new}[c]^I) \tag{5.3}$$

where k is the same also for the wrong $\lambda_k[c]$ in (2.6). Eq. (5.1)-(5.3) indicate that $\lambda_k^{new}[c]^C$ and $\lambda_k^{new}[c]^I$ have opposite sign and that they have both relatively small magnitude. This pattern is observed either in case a single error is introduced in MSB2 or MSB3 of $\lambda_k[c]$, or in case both MSB2 and MSB3 are wrong: this characteristic is exploited to recover from the error.

1. A parity bit considering MSB2 and MSB3 is added to $\lambda_k[c]$ at storage time.

2. When $\lambda_k[c]$ is loaded from the memory, MSB2-3 are XORed: if the result is different from the parity bit, an error is signaled.

3. In case of error, three different versions of $\lambda_k[c]$ are produced: $\lambda_k[c]_1$ is the one read from the

memory, while in $\lambda_k[c]_2$ and $\lambda_k[c]_3$ the MSB2 and MSB3 are respectively flipped. Two of them are wrong (one has a single wrong bit, the other has two wrong bits) and one is correct, but it is not possible to tell which is which at this stage.

4. Eq. (2.6)-(2.10) are performed with $\lambda_k[c]_1$, $\lambda_k[c]_2$ and $\lambda_k[c]_3$ separately. Three sets of results are produced, containing $\lambda_k^{new}[c]_1$, $\lambda_k^{new}[c]_2$ and $\lambda_k^{new}[c]_3$ respectively, two of which are $\lambda_k^{new}[c]^I$ and one $\lambda_k^{new}[c]^C$.

5. If at least one among $\lambda_k^{new}[c]_1$, $\lambda_k^{new}[c]_2$ and $\lambda_k^{new}[c]_3$ does not follow (5.1) and (5.2) there is a high probability that the error is not critical or that the wrong bit was the parity bit, and the set containing $\lambda_k^{new}[c]_1$ is selected.

6. If $\lambda_k^{new}[c]_1$, $\lambda_k^{new}[c]_2$ and $\lambda_k^{new}[c]_3$ follow (5.1) and (5.2) then two of them ($\lambda_k^{new}[c]^I$) will have the same sign and the other ($\lambda_k^{new}[c]^C$) will have an opposite sign (5.3). The set of results containing the $\lambda_k^{new}[c]$ with the discordant MSB1 is the correct one, and is consequently selected.

The described technique can handle a single Tier 2 error in every parity check computation, that usually involves between five and a few tens of LLRs depending on the LDPC code. While the probability of two Tier 2 errors within the considered LLRs is already very low, it can be reduced even more by protecting memories against burst errors as described in Section 5.4. The curves plotted in Fig. 5.8 have been obtained by joining Tier 1 on MSB1 with Tier 2 on MSB2-3. Error protection is complete for AFPI=(0.1, 0.9, 2.5), and consistent improvements are observed for AFPI=(8.7, 24.9) with respect to Fig. 5.7. However, degradation is still strong for AFPI=248.9. To evaluate the effectiveness of Tier 2 protection against alternative solutions, in Fig. 5.9 Tier 1 is instead applied to both MSB1 and MSB2. It can be seen how Tier 1 on MSB1 and Tier 2 on MSB2-3 gives worse performance than Tier 1 on MSB1-2, while at the same time requiring 42.9% memory increment instead of 57.1%. It is much more effective in presence of severe error conditions, with one order of magnitude smaller FER at SNR=2.9 dB for AFPI=248.9.

To limit the impact of errors on the least significant bits of the representation of $\lambda_k[c]$, the remaining four bits have been divided between Tier 3 and Tier 4: while MSB4-5 are worth considering for error protection and included in Tier 3, MSB6-7 are expendable and left in Tier 4. It does not mean that they do not contribute to the decoding: sporadic error events on these bits, however, do not affect the overall performance. The complete UEP has been employed to obtain the curves in Fig. 5.10. All lower AFPI values are completely taken care of, with the effects of AFPI=248.9 being dramatically attenuated. The memory cost of UEP is the same of Tier 1 on MSB1-2, but shields from errors five bits instead of two: the impact on the decoder architecture and the additional logic

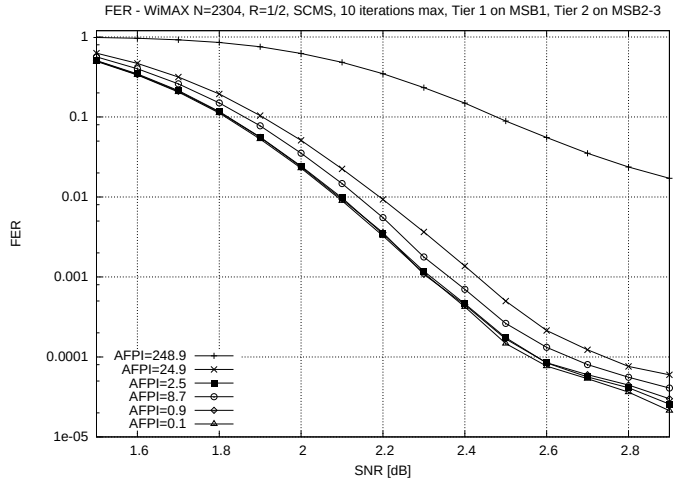

Figure 5.8: FER with different AFPI - Tier 1 on MSB1, Tier 2 on MSB2-3

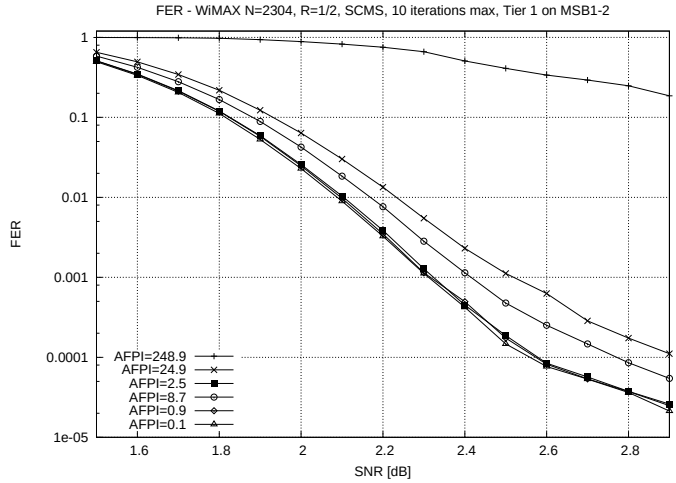

Figure 5.9: FER with different AFPI - Tier 1 on MSB1-2

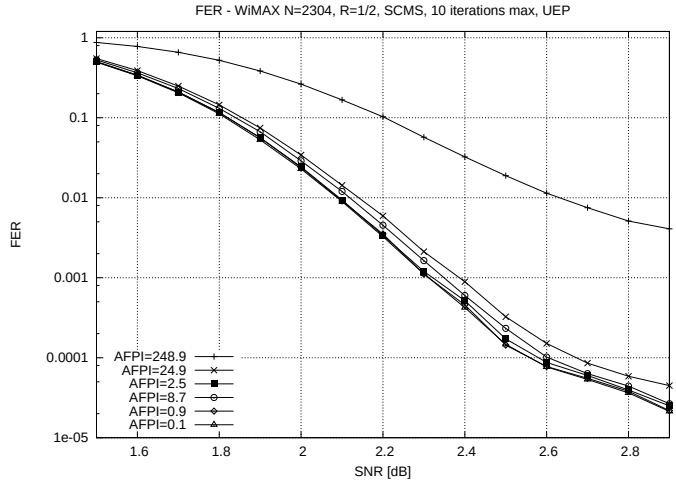

Figure 5.10: FER with different AFPI - Complete UEP

required are discussed in Section 5.5.

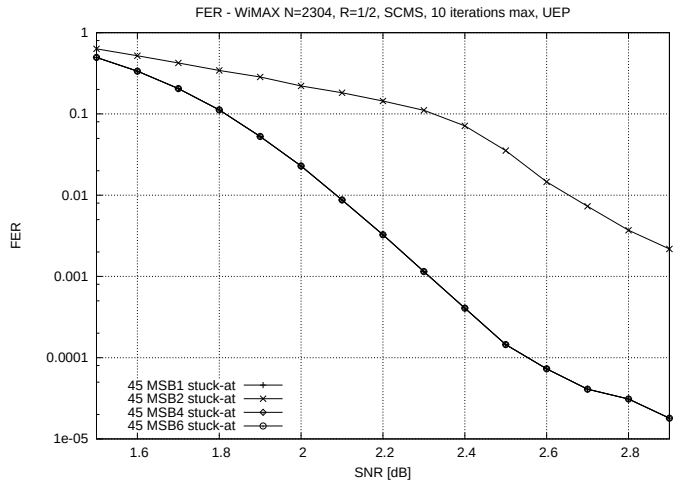

Figure 5.11: UEP performance in presence of stuck-at bits

Table 5.1: UEP - Sustainable single stuck-at bits

Code N, r	MSB Tier 1	MSB2-3 Tier 2	MSB4-5 Tier 3	MSB6-7 Tier 4
2304, 1/2	∞	24	190	256
1152, 1/2	∞	21	175	223
576, 1/2	∞	19	152	206
2304, 5/6	∞	17	138	191
1152 5/6	∞	15	119	152
576, 5/6	∞	12	107	136

Stuck-at bits

The performance of the proposed UEP against stuck-at memory bits is evaluated in Fig. 5.11, where each curve is affected by 45 stuck-at bits affecting different groups of bits. Tier 1 gives total protection from single stuck-at bits, and Tier 3 is sufficient to protect relatively critical MSB4 and MSB5. As with soft errors, MSB6 and MSB7 can be expended without performance degradation. Forty-five stuck-at bits, however, are equivalent to a minimum of AFPI=45, a rate of failures that can not be withstood by Tier 2 when all errors are on MSB2. This is reflected also in the results presented in Table 5.1, where the number of stuck-at bits that can be withstood without performance degradation by different codes on every bit shown. It can be noticed how the number of sustainable stuck-at bits is influenced not only by UEP, but also by the inherent characteristics of the code. Small codes and high-rate codes are more unstable, and prone to stronger degradation. However, small codes have a generally lower probability of incurring in memory errors, due to the lower number of stored LLRs.

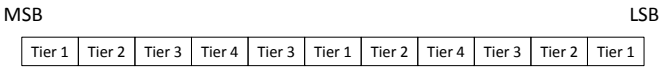

Figure 5.12: LLR rearranged bits for burst error protection

Burst errors

With the level of integration brought by the latest technology nodes, the problem of burst or multi-cell errors has gathered increased interest [98, 99]. All tiers of the proposed UEP are able to detect and possibly recover from single-bit errors, and do not take in account burst errors *per se*. However, it is possible to greatly limit the impact of burst errors by rearranging the order of bits in the stored LLRs: the rearranged order for the case of study is shown in Fig. 5.12. By interleaving the bits belonging to the same tier with those from other tiers, multiple errors are spread over the different protection techniques and can still be handled. Every bit is placed as far as possible from those pertaining to the same tier, with priority being Tier 1 > Tier 2 > Tier 3 > Tier 4. With this order, Tier 1 can sustain up to 9-bit burst errors depending on the involved bits, while it guarantees immunity to burst errors as large as 5-bit. Tier 2 has a maximum resiliency of 8-bit burst errors, and guaranteed immunity to 3-bit bursts, while Tier 3 sustains up to 6-bit bursts and guarantees protection from 2-bit bursts. Bust errors can affect various LLRs concurrently: to avoid multiple wrong LLRs within the same parity check computation (2.6), it is sufficient to store the LDPC codeword $\lambda_k[c]$ in order from the MSB to the LSB. Indeed, the sparse structure of the parity check matrix acts as an interleaver and does not require loading consecutive LLRs.

5.5 Hardware structure, implementation and comparisons

The logic flow of the operations needed by UEP in the conditions portrayed in Section 5.4 is shown in Fig. 5.13. After reading the LLR from the memory, the tripled MSB1 value is obtained through majority voting. The parity of Tier 3 bits is then computed and compared to the parity bit from memory. In case of mismatch, total or partial puncturing is applied to obtain MSB4-5. Three datapaths are necessary to implement the Tier 2 operations: however, they can be used effectively even when no error is detected. The parity comparison is performed on the Tier 2 bits, and if an error occurred, each datapath receives a different version of the LLR ($\lambda_k[c]_1$, $\lambda_k[c]_2$ and $\lambda_k[c]_13$). The outputs are checked according to (5.1)-(5.3) to identify whether the error is critical or not and to select the correct $\lambda_k^{new}[c]$. However, if no error is detected by the parity comparison, all datapaths work with the same set of data. The three outputs can be used to recover from possible errors in the

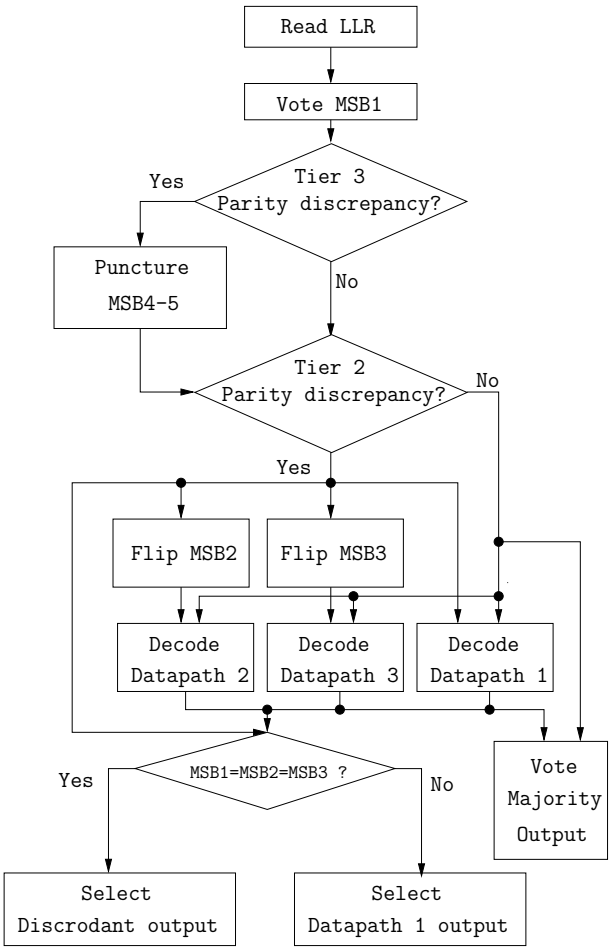

Figure 5.13: UEP logic flow in the considered case of study

datapath logic, and the value of every LLR is decided with a majority voter.

Based on the logic flow of Fig. 5.13, the hardware structure depicted in Fig. 5.14 has been designed. The light gray blocks identify functions pertaining to the different UEP tiers, while an hypothetic LLR is shown at the top of the picture. The three T1 bits enter the `Tier 1` block, that implements a simple majority voting to decide on MSB1. The output of `Tier 1` block is used, along with the rest of the LLR as read from memory, within the `Tier 3` block to decide on what kind of puncturing to apply, if any. The OR operation is applied to bits belonging to the same tier to identify the relative positions of 1s within the LLR. MSB4-5 are either cleared or set, depending on MSB1, the ORed bits, and the parity bit comparison. The dark gray `Datapath` blocks implement the serial datapath depicted in [14]. The computed MSB1 and MSB4-5 are given as inputs to the standard `Datapath` and to the `Tier 2` block, that comprises the two additional `Datapath` blocks. If no discrepancy is detected in the parity comparison among T2 bits, the three datapaths run the same calculations: at the output, majority voters add another layer of reliability to the system against errors within the logic. Two inverters flip MSB2-3 that are fed to the additional datapaths in case

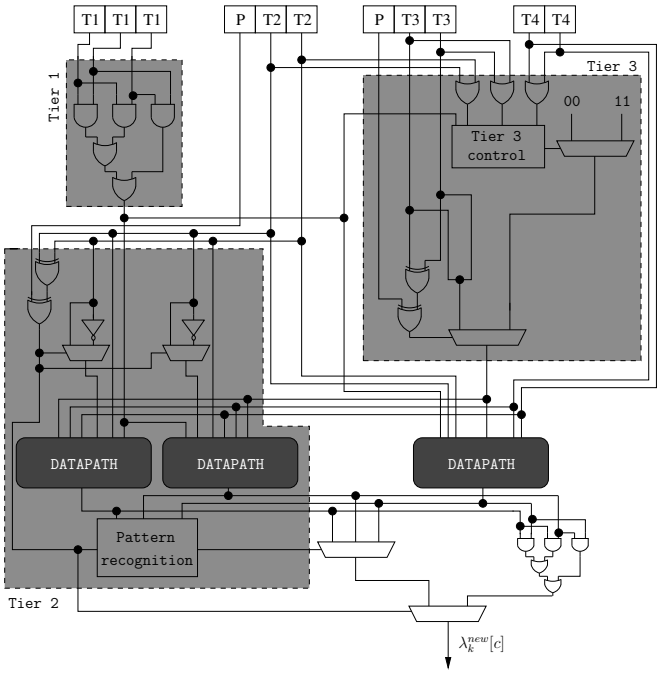

Figure 5.14: UEP hardware structure

the parity comparison presents a mismatch: the output majority voters are cut off, and the pattern recognition system chooses the correct output.

The described architecture for UEP has been implemented in 90 nm CMOS technology with a target frequency of 200 MHz. Table 5.2 reports the area occupation of the different tiers, of a single datapath and of a UEP-protected datapath. It can be seen that while the complexity of Tier 1 and Tier 3 is negligible, Tier 2 introduces a consistent overhead due to the two additional datapaths, that dominate the UEP logic area overhead. The total power and area overhead introduced by UEP over a complete decoder architecture has been reported in the last part of Table 5.2. The considered decoder is the one presented in [14] as **A**, after adaptation to support only LDPC codes and to use the SCMS decoding algorithm, while the quantization of LLRs has been changed to $b = 7$. This version of [14].**A** has been named $\mathbf{A}_{\text{REF}}$ and has been used as a reference in the following comparisons. It relies on 22 Processing Elements (PEs) connected by means of a Kautz [80] network-on-chip, with routing elements of degree three. The decoder supports all LDPC codes in WiMAX and WiFi standards. Between additional logic and extra memory bits, the implementation of UEP requires an additional 36.3% in area occupation, while a 43.9% increment is noticed in power consumption.

The resilient LDPC decoder designed in [95] protects both memories and logic from errors. The decoder works at a frequency of 400 MHz, obtaining a throughput of 333 Mb/s with 30 iterations and the WiMAX code with N=2304 and r=1/2, and implements the λ-min algorithm, that gives better performance than traditional min-sum approximations, since more than two minimums are considered. A dedicated 9-bit RAM is used to store $\lambda_k[c]$ values, and protected with MSB1 tripling

Table 5.2: UEP - area occupation, power consumption (90 nm CMOS, 200 MHz)

	Area	Power
Tier 1	8 μm^2	102.0 nW
Tier 2	13655 μm^2	503.5 μW
Tier 3	102 μm^2	12.7 μW
Datapath	6667 μm^2	278.1 μW
UEP+Datapath	19468 μm^2	659.1 μW
$\mathbf{A}_{REF}$	2.59 mm^2	97.1 mW
$\mathbf{A}_{UEP}$	3.53 mm^2	139.8 mW

(+22% memory increment), while the initial LLRs received from the channel are stored in a 6-bit RAM and protected with MSB1 duplication and puncturing in case of discrepancy (+23% memory increment). The 6-bit R_{lk} values are not protected. The maximum memory error resiliency obtained with this method is MTBF=2 ms, corresponding to AFPI=0.0017. Our solution is more expensive in terms of memory (+57% for 7-bit $\lambda_k[c]$ and +28% for 7-bit R_{lk}), and the tripling of the whole datapath foreseen by Tier 2 adds more complexity than the MSB1 duplication with algorithmic puncturing employed for the datapath in [95]. On the other hand, the proposed UEP targets much more degraded environments, since total error protection is achieved in presence of AFPI four orders of magnitude greater than those in [95]: moreover, this work tackles permanent errors as well, together with burst errors, both neglected in [95].

The two-level UEP devised in [98] splits each codeword in an important partition, where burst errors can be corrected, and a less important partition, where only single errors can be corrected. Supposing to fit our case of study into [98], MSB1-3 are assigned to the important part, while MSB4-7 are left in the other. The number of redundancy bits required by the encoding is linked to the size of the sustained bursts: five bits (a +71% memory increment in our case) are necessary to correct 2-bit bursts, while much larger bursts are considered in our case. The work in [98] can potentially reach performance similar to this work's, but at a much higher complexity cost.

Chapter 6

Channel coding for deep space communications

The world of communications is characterized by a continuous strive for better performance: communication systems are usually pushed towards higher throughput, lower BER and lower power consumption with every generation. A particular application is deep space communications: due to the limited number of complete developments, their evolution in this domain is slower than in other application fields. Moreover, their requirements and constraint can differ substantially from all other communication environments. Transmission between spacecrafts and Earth are supposed to be sporadic events, but the limited amounts of available power and the long distances make failed reception, and consequent retransmission, an unacceptable event. For this reason, deep space missions do not require a high throughput, while at the same time they demand very strict BER and FER performance. This chapter proposes the serial concatenation of turbo and LDPC codes, targeting very high error correction capabilities. The performance of the presented FEC scheme is evaluated against alternative solutions. A unified decoder for the concatenated scheme is subsequently proposed: smart memory and datapath sharing allow for low complexity and low power consumption.

6.1 Concatenation of turbo and LDPC codes

Concatenation between different codes has been frequently considered in order to improve communication performance. "Guaranteed-performance codes" like RS or BCH codes are often used as Outer Codes (OCs) thanks to their measurable error correction capabilities, and joined to Inner Codes (ICs) such as convolutional or LDPC, used in WiMAX and DVB-S2. The same RS+convolutional FEC scheme devised in the CCSDS standard [100] for deep space communications allows these codes to rival with the more powerful LDPC and turbo codes. However, concatenation comes at a usually high implementation cost: decoding support for sometimes very different codes must be provided, increasing area and power consumption. Low-complexity decoders have been designed for many codes [101, 102] but steps have been recently taken towards flexibility, with area efficient multi-code decoders [14, 49].

This section details a novel deep space oriented FEC scheme by serial concatenation of LDPC and turbo codes, which has been published in [18] and has been awarded as the best paper. Performance of the proposed concatenated scheme is compared with the CCSDS standard requirements [100], together with a set of recent works both on deep space communications and on concatenated codes.

6.1.1 Proposed FEC scheme

In the CCSDS recommended standard [103] transmission data rates of up to 2.048 Mb/s are foreseen for the next missions: FEC schemes need to be powerful enough so that retransmission is not necessary, due to the low power available on spacecrafts. The need for effective coding schemes, alongside simple decoding algorithms, makes code concatenation one of the smartest solutions. The powerful Turbo and LDPC codes have been considered for concatenation before [104] but the low level of details provided and the unsatisfying results leave room for further investigation.

The devised FEC scheme is shown in Fig. 6.1: an LDPC code is serially concatenated to a turbo code. The outer encoder encodes the input bits, and the resulting codeword is used as input for the inner encoder. Being responsible of the first, rough decoding, the IC should work well also in presence of a large number of errors. Since turbo codes have better performance than LDPC at low SNR [3], they have been chosen as IC. On the contrary, receiving the updated information from the inner decoder, the OC decoder, that is an LDPC code, can fully exploit its deep waterfall region and low error floor. An interleaver scrambles the output of the OC encoder before the second encoding, while the inverse function is inserted between the decoders. Since, depending on the rate and characteristics of the chosen codes, the block sizes may not be compatible, an optional padding block has been placed after the interleaver: zeros are added to the scrambled codeword to fit the required number of bits. This reduces the coding efficiency but allows for flexible concatenation. The padding bits are removed

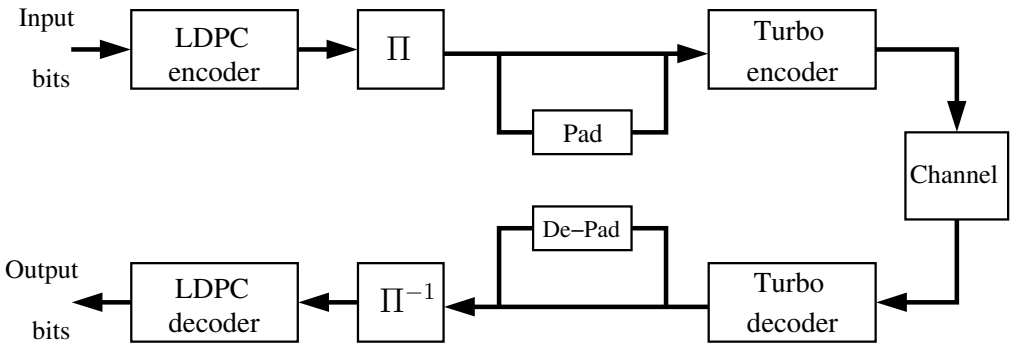

Figure 6.1: Serial concatenation of LDPC and turbo codes FEC scheme

after the IC decoding, before the deinterleaver.

The decoding process starts in the IC decoder, which performs up to It_{max}^{IC} iterations. After It_{max}^{IC} the codeword is stripped of the padding bits, descrambled and passed to the OC decoder. Both turbo and LDPC code decoding algorithms involve soft information: while the IC decoder receives measures of bit error probabilities from the channel estimator, the OC decoder must rely on the metrics updated by the IC decoder. In particular, the LDPC decoder receives as intrinsic information for the initialization of $\lambda_k[c]$ the bit-level extrinsic output of the turbo SISO $\lambda_k[u]$. These metrics are passed through the deinterleaver along with the codeword.

The CCSDS suggests three FEC schemes for space communications in [105]: a RS-convolutional codes concatenated scheme, turbo codes and LDPC codes. Deep space communications requiring very low bit error rates address turbo codes in particular, allowing four code rates ranging from 1/6 to 1/2, and four information block lengths in the range 1784-8920. The WiMAX standard relies on both LDPC and turbo codes. The two code types are mutually exclusive options in the standard: their proven effectiveness and implementation-friendly structure, however, make them ideal candidates for concatenation towards deep-space applications, regardless of the relatively low performance of WiMAX LDPC codes w.r.t. CCSDS LDPC codes. Thanks to the wide variety of available codes, it has been possible to experiment with different code combinations consisting of both WiMAX and CCSDS codes. Though CCSDS SBTCs employ 16 states, it is proven in Section 6.1.2 that also the eight-state DBTCs used in WiMAX guarantee very good results while keeping the decoding complexity low.

6.1.2 Simulations and performance comparisons

To evaluate the effectiveness of the proposed approach, simulations have been run on a proprietary tool. Its deeply customizable structure allows to select codes, channel model, SNR and result reli-

ability level, together with decoding algorithms and related choices (number of iterations, stopping criteria). Moreover, it is possible to tweak a set of implementation-oriented characteristics, like different approximations of the chosen algorithms and number of bits assigned to the representation of the metrics.

In order to comply as much as possible with the requirements of CCSDS, the turbo codes suggested in [105] have been used as ICs in a first batch of simulations, and concatenated with WiMAX LDPC codes. The relatively high rate of the OC results in a concatenated rate that is very close to the CCSDS specifications. Block size compliance is guaranteed by using, if needed, multiple LDPC codewords as a single turbo information block, together with padding bits. The maximum allowed number of iterations has been set to 10 for both IC and OC decoder, and following most of the state of the art on deep space communications, the AWGN channel model has been chosen.

Moving towards a hardware implementation of the proposed FEC scheme, some limitations have been inserted in the simulation environment. To correctly evaluate the impact of soft information quantization on the BER and FER, the dynamic range of all metrics involved in the decoding process has been limited to 10 or 9 bits, with 3 bits of fractional part. For the same reason, both the LDPC and turbo codes have been decoded with the BCJR algorithm [24], thus leading to low decoding complexity.

Early experimentations have shown that the gain that can be obtained with the insertion of a bit interleaver between the two encoders is negligible w.r.t. the additional complexity, and it has not been considered in the plotted curve. This limited effect is mainly due to the sparse structure of the $\mathbf{H}$ matrix, that acts as an interleaver by itself [104].

Fig. 6.2 and 6.3 plot a set of meaningful BER and FER curves respectively. The "+" marker indicates the curves provided by CCSDS in [105] for SBTCs with $r = 1/3$ (continuous) and $1/4$ (dashed). They are obtained with 10 decoder iterations, QPSK modulation and AWGN channel. The ×-marked curves show the performance of these codes when concatenated with a WiMAX $r = 5/6$ LDPC code. It can be seen that both concatenated BER and FER follow very closely the standard's curves: FER results are particularly encouraging, thanks to aggregated error distributions that are addressed later in this section. Moreover, plots in [105] show error floors as early as BER=10^{-6} (block length 8920): the minimum BER simulated with the concatenated scheme is slightly above 10^{-10}, with no signs of error floor.

The continuous, ■-marked curves have been obtained by substituting to the SBTC of CCSDS in the concatenated scheme a WiMAX DBTC of comparable size and same rate. The difference between the two curves is negligible, while the complexity of an 8-state DB turbo decoder is lower than a

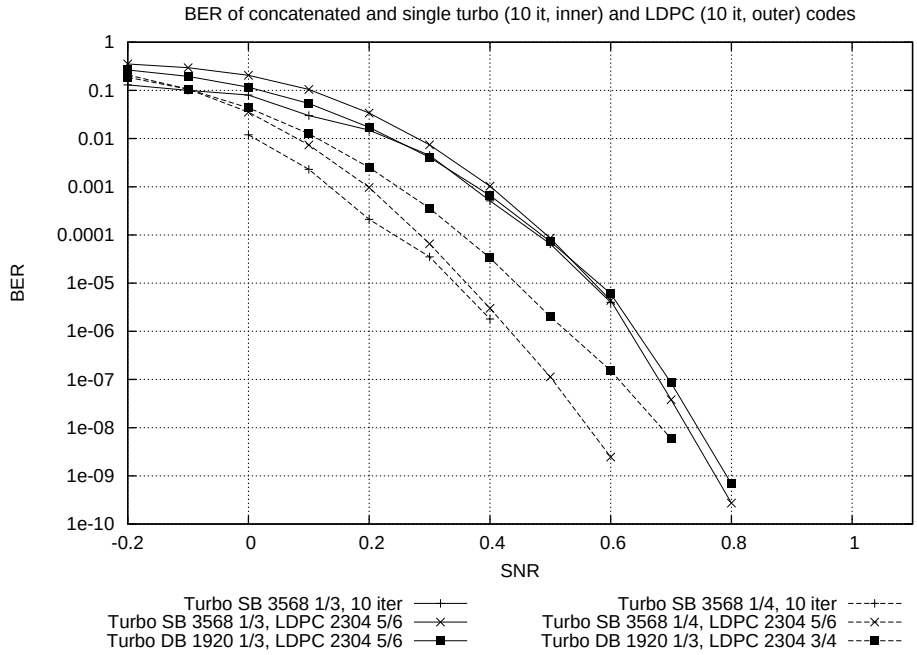

Figure 6.2: Concatenated LDPC and turbo BER, AWGN channel, BPSK modulation

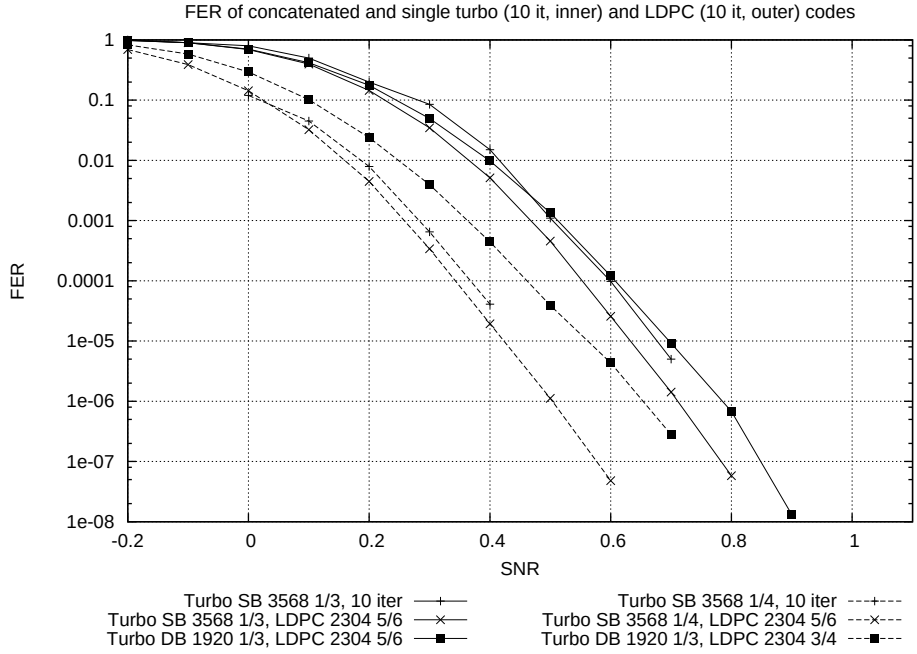

Figure 6.3: Concatenated LDPC and turbo FER, AWGN channel, BPSK modulation

16-state SB one.

To evaluate the influence of IC and OC respective rates on the decoding performance, a second set of simulations has been run: the IC rate has been fixed to 1/3 and by changing the OC rate it has been possible to obtain concatenated rates equal to those of CCSDS turbo codes. In both Fig. 6.2 and 6.3, the dashed, ■-marked curve has a concatenated rate of 1/4, obtained by CTC 1/3 + LDPC 3/4. It can be noticed how the lower rate of the OC fails to deliver the same BER and FER results

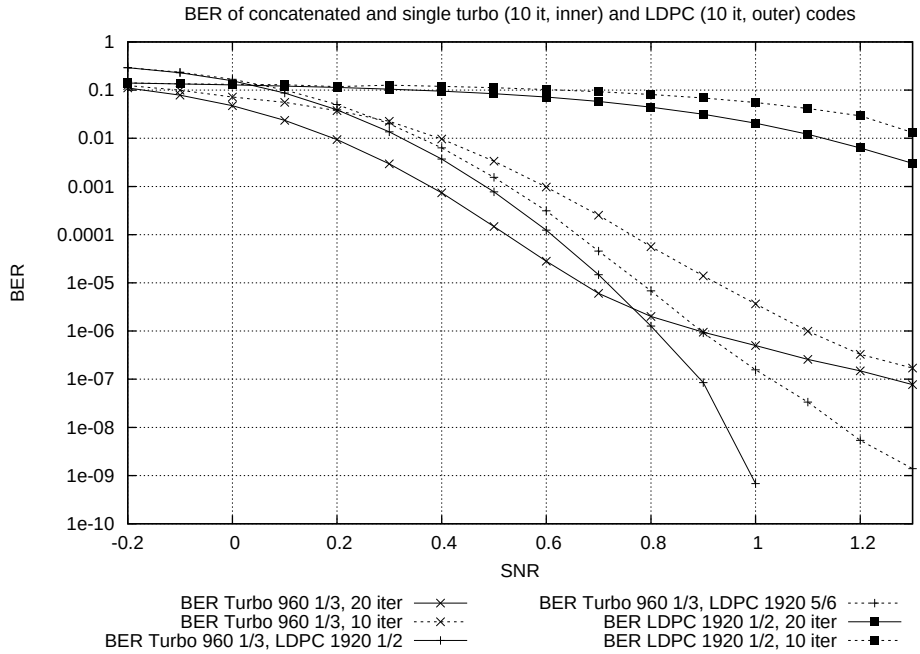

Figure 6.4: LDPC and DB turbo BER, concatenated and single-code, AWGN channel, BPSK modulation

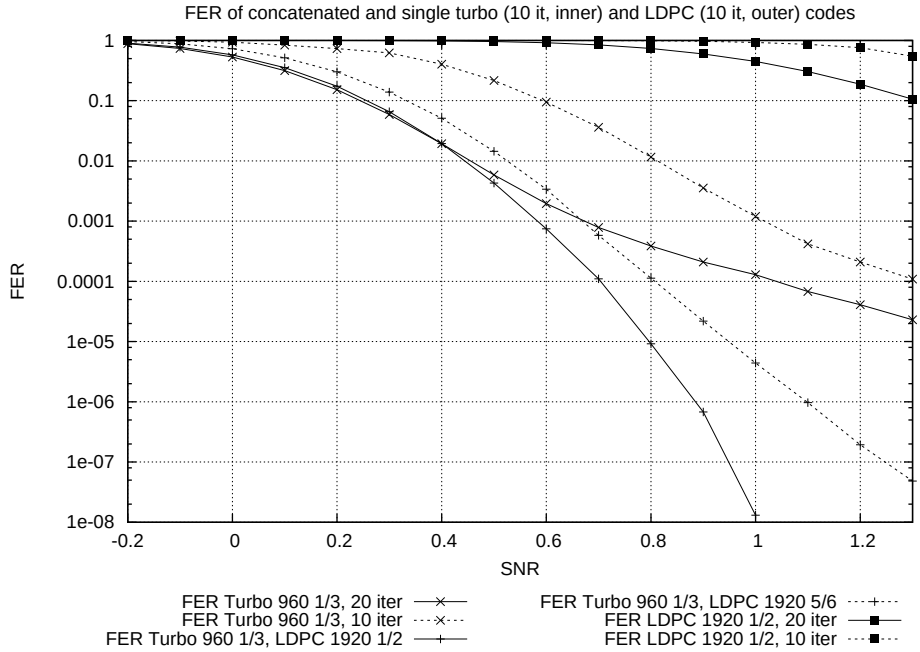

Figure 6.5: LDPC and DB turbo FER, concatenated and single-code, AWGN channel, BPSK modulation

as CTC 1/4 + LDPC 5/6: this behavior, observed also in the CTC 1/6 + LDPC 5/6 against CTC 1/3 + LDPC 1/2 case, reveals how the IC turbo rate is more critical than for the LDPC OC.

Fig. 6.4 plots a set of BER curves to compare the performance of WiMAX concatenated codes against single LDPC and turbo codes. The curves showing the "+" marker refer to the concatenated FEC scheme: both use a WiMAX turbo code of rate 1/3 and 960 two-bit input symbols. The

Table 6.1: Performance comparison among FEC schemes

	A	**B**	[106]	[107]	[108]	[109]	[110]	[104]	[111]
Application	Deep space		3D HDTV	Satellite	Deep space	Deep space	–	Mobile	
Inner	SB CTC		LDPC	NB-LDPC	QC-LDPC	LT	Parallel	SB CTC	
Outer	QC-LDPC		BTC	LT		NBLDPC	RSC/LDPC	LDPC	
r_{in}	1/4	1/3	1/2	2/3	1/3	25/49	1/2	1/3	
r_{out}	5/6	5/6	467/500	9/10		49/50	1/2	7/8	
r	5/24	5/18	467/1000	3/5		1/2	1/3	7/24	
Input_{in}	3568		16 K	1000 sym.	2379	32 K	504	2048	
Input_{out}	1920		15 K	900 sym.		16 K	504	1792	
Inner Alg.	BCJR		BP	FFT-BP	BP	N/A	Log-MAP	BP	
Outer Alg.	BCJR		CHASE	MP			LLR-BP	Log-MAP	
It_{max}^{IC}	10		50	20	15	N/A	8	5	5
It_{max}^{OC}	10		N/A	N/A			10	50	100
Quant.	10-9 bits		Float	Float	Float	Adaptive	Float	Float	
Channel	AWGN		Rayleigh	AWGN	AWGN	AWGN	AWGN	AWGN	
E_b/N_0	0.43	0.65	4.4	1.85	N/A	N/A	2.35	1.75	1.55
FER	$8\cdot10^{-6}$	$8\cdot10^{-6}$	$4\cdot10^{-4}$	N/A			$2\cdot10^{-5}$	N/A	N/A
Δ_{SHN}	1.38	1.43	4.50	1.55			2.90	2.43	2.23
min BER	$2\cdot10^{-9}$	$<10^{-10}$	$2\cdot10^{-7}$	$8\cdot10^{-7}$	$3\cdot10^{-6}$	N/A	$1.2\cdot10^{-7}$	$4\cdot10^{-7}$	$2\cdot10^{-7}$
min FER	$6\cdot10^{-8}$	$1.1\cdot10^{-8}$	$8\cdot10^{-5}$	N/A	N/A	10^{-8}	$6\cdot10^{-6}$	N/A	N/A
E_b/N_0	0.6	0.9	4.6	2.0	0.3	1.05	2.5	1.9	1.6

continuous line has been obtained using a rate 1/2, codeword length 1920 LDPC, while the dashed one with a rate 5/6 LDPC. The higher rate LDPC results in a less steep curve: this degradation can be addressed by rising It_{max}^{IC}, with the two curves superimposed at It_{max}^{IC}=20. These plots are compared to the constituent rate 1/3 CTC and rate 1/2 LDPC with different numbers of allowed iterations. Since the concatenated scheme has at its disposal up to 20 iterations (10 CTC + 10 LDPC), single-code curves are plotted with both 20 and 10 iterations maximum. The concatenated BER shows very good performance at low E_b/N_0, crossing the 10^{-6} threshold at E_b/N_0=0.9 dB when using the rate 5/6 LDPC OC. At higher E_b/N_0, its performances are even more remarkable: a total of $2\cdot10^8$ frames have been simulated for all the shown E_b/N_0 points, for a total of $1.92\cdot10^{11}$ information bits, and no errors were counted for E_b/N_0 higher than 1.0 dB when using the rate 1/2 LDPC code. The curves show no sign of error floor and constant BER decrease. At low E_b/N_0, the 20-iterations single CTC outperforms the concatenated schemes of up to 0.15 dB: the difference is smaller than that reported in [104] with much more favorable conditions, and the crossing point occurs at a higher BER and much lower E_b/N_0.

Fig. 6.5 shows the FER curves for the same parameters as Fig. 6.4: the concatenated FER reaches very low values ($7\cdot10^{-8}$ with r = 5/6 LDPC code). The difference with the 20-iteration single CTC curve at low E_b/N_0 is substantially reduced (0.08 dB maximum) w.r.t. the BER, and the crossing point is moved at lower E_b/N_0. This is due to the fact that very often a failed decoding with the concatenated scheme is due to a high number of wrong bits within the same frame. Consequently, BER and FER scale differently, since errors are clustered together and affect a very low number of frames.

Table 6.1 provides a comparison of the proposed FEC scheme with similar state of the art solutions. Solution **A** refers to CCSDS SB turbo 1/4 + WiMAX LPDC 5/6, while solution **B** to CCSDS SB turbo 1/3 + WiMAX LPDC 5/6, both already shown in Fig. 6.2 and Fig.6.3. To help a fair comparison with coding schemes with different rate, the Δ_{SHN} row identifies the distance of the BER curve from the Shannon limit at BER=10^{-6}: in both cases the distance is less than 1.5 dB. The obtained Δ_{SHN} is similar to that of the AR4JA LDPC codes proposed by CCSDS [105], but has been obtained with a much smaller number of iterations.

In a recent work [106], a FEC scheme for 3D HDTV using an outer block turbo code (BTC) concatenated to an LDPC code is proposed. The scheme is shown to outperform the DVB-T2 standard serial concatenation of BCH and LDPC codes. Contrariwise to CTCs, BTCs are obtained by concatenating various BCH codes, and decoded via CHASE algorithm, whose implementation complexity is estimated comparable to that of BCJR. The BER and FER performance is greatly outperformed by this work's, with approximately 4 dB gain, and a very high Δ_{SHN}. This could partly be due to the higher code rate and to the fading channel model, but [106] sets a very high number of iterations for the inner code, a large block size and floating point precision for the simulations, all factors that contribute to the improvement of results.

Luby Transform (LT) or fountain codes have been used together with non-binary LDPC (NB-LDPC) codes in [107] for satellite communications. The resulting system is very flexible, thanks to LT codes, and the presence of NB-LDPC codes guarantees a high error correction power even at high rates. The decoding complexity, however, suffers from the concurrent FFT-based BP and message passing algorithms, much higher than a single BCJR. The BER shows 1.2 dB loss w.r.t. solution **B**: for the same rate and precision, the two FEC systems should yield comparable results. This is confirmed by the comparable Δ_{SHN} metrics. Both **A** and **B**, however, outperform the LT+binary LDPC codes of [112].

The QC-LDPC construction scheme for deep space communications described in [108] gives very good results without making use of concatenation: although the curves do not show low BER points, a 0.4 dB gain can be observed against the plots with similar rate in Fig. 6.2. These curves have been drawn with floating-point precision and a decoding algorithm devoid of approximations: some degradation of performance is consequently to be expected after the implementation. The complexity of the probability-domain BP algorithm, moreover, is very high, burdening the hypothetical hardware with large area occupation and high power consumption.

The joint source-channel coding scheme described in [109], aimed at deep space image transmission, uses Raptor codes, *i.e.* a concatenation of an LT code with a precode, in this case a very high-rate

NB-LDPC. This powerful coding scheme, also addressed in [107], obtains very good results at high coding rates: FER shows only 0.15 dB loss w.r.t. solution **B**, regardless of the rate difference.

In [110] parallel concatenation of LDPC and RSC codes is explored. The LLR-based BP algorithm implemented for the LDPC part allows good performance also with relatively small block sizes and a few allowed iterations: still, it is outperformed by **A** by 1.70 dB gain at BER=10^{-6} and one dB smaller Δ_{SHN}, with even more difference in the FER curves.

The two closely related works [104] and [111] implement CTCs as inner and LDPC codes as OCs, decoding them with Log-MAP and probability-domain BP algorithms respectively. The work in [111] presents the same system as [104], with the addition of a certain number of global decoder iterations that slightly improve the BER. The plotted curves show a degradation of the concatenated scheme (5+50 iterations) w.r.t. the single turbo code (5 iterations) of up to 0.3 dB that has not been observed in this work (Fig. 6.4). Rates are comparable to that of **B**, and they both use the AWGN channel model, while the block size of **B** is larger. Regardless of the much higher precision and number of iterations allowed in [104], **A** and **B** yield better BER and Δ_{SHN} results, with a gain ranging from 0.90 to 1.10 dB at BER=10^{-6}.

6.2 Unified decoder for concatenated turbo and LDPC codes

To the best of our knowledge no implementation solution for the concatenated scheme described in Section 6.1 has ever been proposed, but decoders for both turbo and LDPC codes are present in the state of the art, mainly targeting wireless communications. Multi-code and multi-standard decoders that make flexibility their primary concern have also been introduced recently [14, 48–52]: they are characterized by different degrees of datapath and memory sharing.

This section proposes a decoder for concatenated turbo and LDPC codes targeting deep space communications, and more in general communications where retransmission can not be afforded, like broadcasting. The usage of the same decoding algorithm for both codes greatly reduces the area overhead of the concatenated scheme decoder with respect to a single LDPC or turbo code decoder. In facts, it allows to exploit a high degree of datapath sharing and obtain very low power consumption and area occupation. The presented work is currently under review by IEEE Transactions on Aerospace and Electronic Systems.

6.2.1 Unified LDPC/turbo decoding architecture

Following the effectiveness of the concatenated FEC scheme presented in the previous section, the decoder architecture for turbo and LDPC codes concatenation shown in Fig. 6.6 has been designed. The gray blocks represent the duplicated datapath described in this section, while the structure of the memory banks, with their alternative usage according to half-iterations, is detailed in Section 6.2.2. The common turbo/LDPC decoding technique depicted in Section 2.2 paves the way for highly shared datapaths, in the wake of works like [52] and [50], as opposed to separate datapath turbo/LDPC decoders like [14, 48, 49]. The proposed decoder relies on an innovative smart memory structure that allows to increase the percentage of module reuse within the datapath and avoid complex interleaving mechanisms between the decoding modes.

Datapath

The structure of the designed LDPC/turbo datapath positions itself in between a completely shared approach and datapath separation. The turbo and LDPC datapaths have great disparities in terms of complexity, with the turbo datapath requiring more resources. As shown later in this section, the LDPC datapath is included within the turbo datapath, while constituting a limited percentage of its overall logic. The concatenated scheme can consequently be decoded at little more than the logic cost of a turbo decoder.

Fig. 6.7 shows a block diagram of the designed datapath. It is characterized by a pipelined

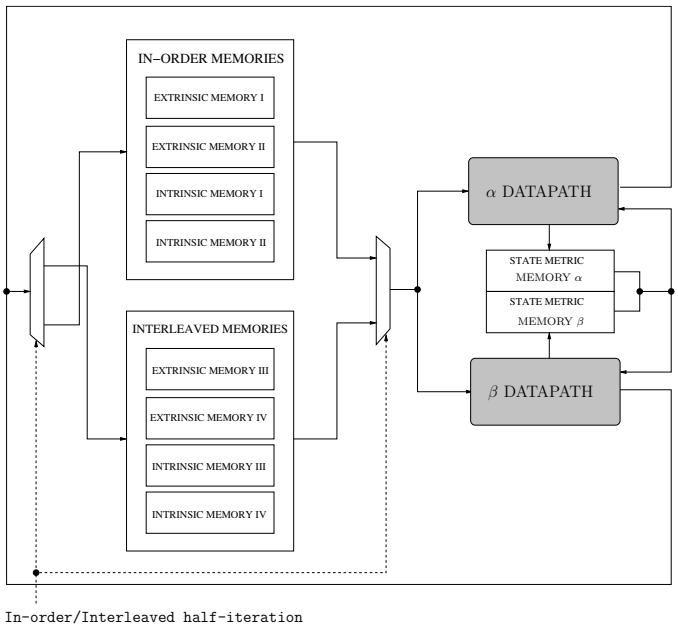

Figure 6.6: Unified LDPC/turbo decoder overall block diagram

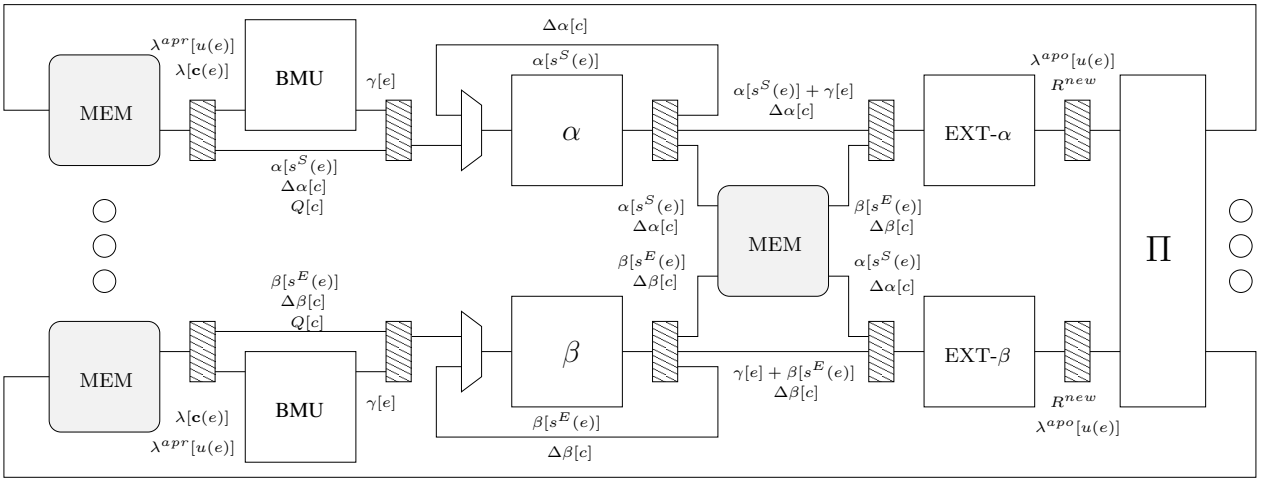

Figure 6.7: Unified LDPC/turbo decoding datapath block diagram

architecture, with registers represented by striped blocks. The turbo decoding process makes use of a *butterfly* structure: the datapath is duplicated in an α and β datapath, respectively entrusted with the concurrent forward and backward scanning of the trellis steps. They implement the modified sliding window technique described in Section 2.2. Each half of the duplicated datapath receives as an input from the memories the $\lambda_k^{apr}[u(e)]$ and $\lambda_k[\mathbf{c}(e)]$ relative to a trellis step: these are used by the Branch Metric Units (BMUs) to perform (2.5) and obtain $\gamma_k[e]$. These are passed to the α and β units, that perform the computations of (2.3) and (2.4) respectively. The structure of the α and β units is similar. Along with the output of BMU, the α unit receives $\alpha_{k-1}[s^S(e)]$ either from the memory (when computing the first trellis step of a window) or from its own outputs (all other trellis

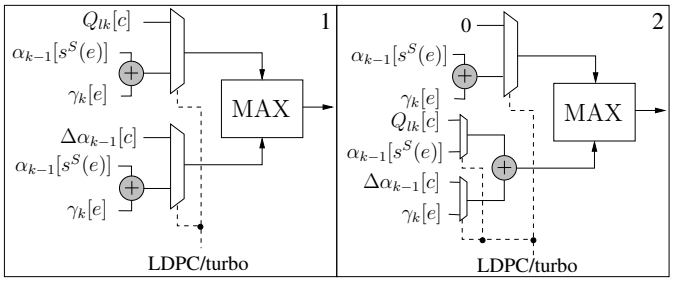

Figure 6.8: Unified LDPC/turbo comparator components, with inputs used in the α unit

steps), as shown by the feedback loop in Fig. 6.7. Together with the updated $\alpha_k[s]$, that are stored in the state metric memory α (Fig. 6.6), the α unit also produces the $\alpha_{k-1}[s^S(e)]+\gamma_k[e]$ partial sums needed by (2.2). These are passed to one of the extrinsic computation units (EXT-α in Fig. 6.7). EXT-α completes (2.2) by taking the $\beta_k[s^E(e)]$ stored in the state metric memory β by the β unit and finally performs (2.1). The same computation is concurrently carried out on another trellis step by EXT-β, to which are given $\alpha_{k-1}[s^S(e)]$ stored by the α unit in the state metric memory α and $\gamma_k[e]+\beta_k[s^E(e)]$ partial sums calculated in the β unit.

The LDPC decoding process makes mostly use of the turbo mode datapath. LDPC codes are characterized by 2-state binary trellises: since the turbo codes considered are either 16-state SBTC or 8-state DBTC, LDPC codes can easily exploit an additional parallelism factor. The BMU is not used and consequently deactivated in LDPC mode, while both α and β units are shared with the turbo mode. In Fig. 6.8, the core components of the unified α unit are depicted. Adders and comparators are shared among the two operating modes. Both structures are equivalent when in turbo mode, while their operations differ in LDPC mode. Architecture '1' implements the $\max(x,y)$ operator of $\Phi(x,y)$ (2.13), while architecture '2' implements $\max(x+y,0)$. The EXT-α and EXT-β units perform (2.14), and rely on the same architectures used for α and β unit (Fig. 6.8). Also in this case they are shared with the turbo datapath.

6.2.2 Memory

As occupied area in both turbo and LDPC decoders is dominated by storage components, efficient memory sharing is very important: for example, a scheme suitable for serial PEs with disjoint turbo and LDPC datapaths has been used in Chapter 3, resulting in large memory saving. However, a different approach is needed with this work. Since the sizing of memories strongly depends on the supported codes, the following analysis is carried out supposing the concatenation of a rate 5/6, N=1920 WiMAX LDPC code with a rate 1/3, K=960 DBTC taken from the same standard, decoded considering a window size w = 80. As already shown in [18] and in Section 6.1, this FEC scheme

guarantees performance comparable to that of more powerful codes. No padding bits are necessary, since the size of the input frame for the DBTC (960 symbols, i.e. 1920 bits) is equal to the size of the LDPC codeword. However, the following discussion on memory requirements stands also in case of padding, as long as the padding bits are added at the end of the LDPC codeword. The memories necessary to support the designed decoder can be observed in Fig. 6.6: two sets of four memory banks serve the in-order and interleaved half-iterations respectively, storing extrinsic and intrinsic information, while two memories are dedicated to the storage of state metrics.

In turbo mode, the duplication of the datapath required by the *butterfly* structure rises the need for concurrent data reading and writing. For the correct computation of a trellis step the following metrics are necessary:

- $\lambda_k^{apr}[u(e)]$ and $\lambda_k[\mathbf{c}(e)]$ for the computation of (2.5). Since WiMAX codes are duo-binary, $\lambda_k^{apr}[u(e)]$ consists of three different metrics, while $\lambda_k[\mathbf{c}(e)]$ of four. However, as explained in [38], symbol-level information in duo-binary codes can be converted to bit-level information and vice versa, with a small performance degradation. This means that the memory requirements for $\lambda_k^{apr}[u(e)]$ can be reduced by approximately 1/3. Due to the *butterfly* structure, eight $\lambda_k[\mathbf{c}(e)]$ metrics and four $\lambda_k^{apr}[u(e)]$ are needed. While $\lambda_k[\mathbf{c}(e)]$ values are received by the decoder at the beginning of a frame and not updated anymore, the $\lambda_k^{apr}[u(e)]$ metric is updated at least once per iteration.

- $\alpha[s^E(e)]$ and $\beta[s^E(e)]$ for (2.2), (2.3) and (2.4). Every trellis step computation requires a number of $\alpha[s^E(e)]$ and $\beta[s^E(e)]$ metrics equal to the number of states of the turbo code: in this case, eight of each. The loading and storing needs for these metrics vary during the decoding process. At the beginning of each trellis window, the $\alpha[s^E(e)]$ and $\beta[s^E(e)]$ values coming from adjacent windows must be read as initialization values. The updated $\alpha[s^E(e)]$ and $\beta[s^E(e)]$ must be stored during the first half of the window, and loaded again in the second half. Finally, the metrics belonging to trellis steps at the edge of a window must be stored for the adjacent windows.

In LDPC mode, for every trellis step computation, a $\lambda_k[c]$ and R_{lk}^{old} pair must be loaded in both parts of the datapath to perform (2.6), along with $\Delta\alpha_k[c]$ and $\Delta\beta_k[c]$ for (2.14), (2.15) and (2.16). Similarly to the turbo case, only the $\Delta\alpha_k[c]$ and $\Delta\beta_k[c]$ metrics at the edge of the trellis need to be stored for further usage, while the $\lambda_k[c]$ and R_{lk}^{new} metrics involved in (2.9) are to be updated once per trellis.

Since the turbo code chosen for this analysis is duo-binary and has an eight-state trellis, its

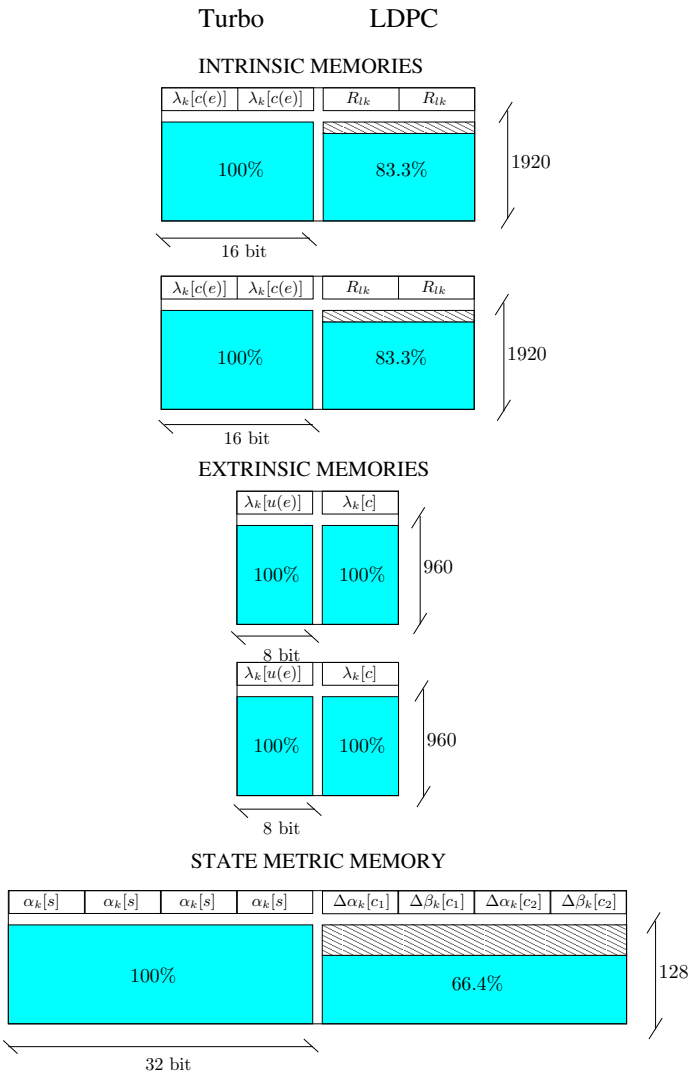

Figure 6.9: Unified turbo/LDPC decoder memory sharing scheme

decoding process needs a much larger number of metrics than the LDPC code, which decoding is similar to that of a single-binary, two-state turbo code. From the LDPC point of view, this translates in an internal level of parallelism in the datapath that is not, however, directly available. In fact, the structure of the **H** matrix and the layered scheduling require the same LLR to be read and updated multiple times during a single LDPC decoding iteration, resulting in complex load and store patterns not found in turbo decoding. Careful planning of the memory structure is consequently necessary to maximize the level of memory sharing and to concurrently allow the LDPC datapath to exploit the internal parallelism. Figure 6.9 shows an in-depth detail of the one the two sets of memory banks depicted in Fig. 6.6. Memories sized to accommodate the considered codes in case a parallelism factor $\times 2$ is to be used in LDPC decoding. They are dual-port, and the usage percentage of each memory is portrayed for both turbo and LDPC codes, along with its depth and width.

In turbo mode, two $\lambda_k[\mathbf{c}(e)]$ metrics are stored at each address in the two 1920×16 bit intrinsic memories: both ports are always kept in read mode, except during initialization. In this way, four

Table 6.2: Memory requirements

	Memory Bits			
	Turbo	LDPC	Total	
Separate memories No smart allocation	176384	210240	386624	100%
Separate memories Smart allocation	161024	138560	299584	77.5%
Shared memories Smart allocation	161024	138560	161792	41.8%

$\lambda_k[\mathbf{c}(e)]$ are concurrently available to the α datapath, and four to the β datapath. These same intrinsic memories are used to store the R_{lk}^{old} values in LDPC mode. At every clock cycle both the datapaths need a R_{lk}^{old} value: during the second half of the trellis, the α datapath will need the values fed to the β datapath during the first half in backward order, and vice versa. This means that by storing at the same memory address R_{lk}^{old} values in symmetrical positions with respect to the trellis half-point (e.g. R_{l1}^{old} with R_{l20}^{old}, R_{l2}^{old} with R_{l19}^{old} etc.) the storage requirements are reduced by 1/2 without decreasing the number of concurrently available metrics. The total number of memory locations required becomes 3200, resulting in a 83.3% usage of each intrinsic memory. Two 960×8 bit extrinsic memories hold the $\lambda_k^{apr}[u(e)]$ values. From the turbo decoding point of view, these two memories could be merged into a single 960×16 bit memory, since $\lambda_k^{apr}[u(e)]$ metrics must be paired to obtain the symbol-level metrics used in the BCJR algorithm. LDPC decoding must hold at least N $\lambda_k[c]$: four of them must be read concurrently. A single 960×16 bit memory would not suffice, since the coupling of values changes with every row of the $\mathbf{H}$ matrix. Both extrinsic memories are used to their full capacity in both decoding modes.

The memories portrayed so far compose the in-order half-iteration memory banks (Fig. 6.6) and need to be kept, during turbo decoding, with both ports in read mode. The in-order half-iteration makes use of two additional extrinsic memories to store the newly computed $\lambda_k^{apo}[u(e)]$, that are used in the interleaved-order half-iteration (Fig. 6.6). Also the $\lambda_k[\mathbf{c}(e)]$ needed by the interleaved half-iteration can be stored in two additional intrinsic memories: while this is not strictly necessary, since they can be obtained by reading in interleaved order those used in the first half-iteration, the extra storage is useful for LDPC decoding. In fact, by having a total of four 1920×16 bit and four 960×8 bit memories, LDPC decoding can exploit a $\times 2$ internal parallelism factor.

Finally, two wider 128×32 state metric memories, as the one shown in Figure 6.9, are used to hold the $\alpha[s^E(e)]$ and $\beta[s^E(e)]$ values for both window initialization and intra-window state metrics in turbo mode. The same memories are used in LDPC mode to store $\Delta\alpha_k[c]$ and $\Delta\beta_k[c]$. Each address holds the values used by each datapath in both levels of parallelism.

Table 6.2 synthesizes the advantages of the devised memory structure. The first row gives the memory bits necessary for the decoder architecture described in Section 6.2.1 to work in both turbo and LDPC mode, while considering separate memories. If smart metric allocation techniques are used (R_{lk}^{old} coupling and reusage, bit-level $\lambda_k^{apo}[u(e)]$), the required bits are reduced of 22.5%. Moreover, by sharing the memories between the two modes, only 41.8% of total bits is necessary, with LDPC mode being completely supported by the memories required by turbo mode.

Interleaving and addressing

Address generation for the described memory structure is in most cases straightforward. In turbo mode, all read operations are sequential, either in forward or backward order, and are consequently handled by simple counters. Write operations to the following half-iteration memories are based on the permutation law associated to the turbo code encoding, and the memory addresses can be obtained via simple operation on the current half-iteration read address. In the considered case study, the interleaving rules are those associated to the WiMAX standard turbo codes: the interleaved addresses are obtained on-the-fly by dedicated logic implementing the WiMAX permutation function. Address generation for the intrinsic memories is sequential in both read and write operations when in LDPC mode, and the counters used in the turbo mode can be reused, but problems arise when dealing with the extrinsic memories. While sequential addressing in intrinsic memories is possible thanks to the local nature of R_{lk}^{old} values, $\lambda_k[c]$ values are read and updated multiple times and in variable order during an iteration. Address generation, however, can still exploit the regular structure of the $\mathbf{H}$ of QC-LDPC codes. By storing in a small memory the shift factors of the constituent $m \times m$ circulant identity matrices, together with the position of the nonzero entry of their first row, read and write addresses can be obtained with modulo-m counters and adders. A single 160×36 bit memory is sufficient to support also the $\times 2$ internal parallelism.

The devised memory structure is particularly advantageous when switching between turbo and LDPC decoding: after the last turbo iteration, the extrinsic memories relative to the in-order half-iteration contain the data needed by the LDPC decoding process in the correct order. The memories relative to the interleaved-order half-iteration in turbo mode, to be used in LDPC mode, will only need to have the read and write addresses pass through the permutation law circuit.

6.2.3 Implementation

The decoder architecture described in Section 6.2.1 has been implemented in 90 nm CMOS technology: synthesis and power estimation have been carried out with Synopsys Design Compiler, while the switch

activity has been analyzed with Mentor Graphics Modelsim.

Several design choices are related to the set of codes that is going to be implemented, in particular the sizing of the memories. The largest codes considered for the implementation are taken from the WiMAX standard: an LDPC code with block size 1920 bits and rate 5/6, and a DBTC with information block size 1920 bits and rate 1/3. Soft metrics have been quantized with nine and eight bits, with two bits of fractional part; the maximum number of iterations has been set as $\mathrm{It}_{OC} = 10$ for LDPC and $\mathrm{It}_{IC} = 6$ for turbo. The CCSDS throughput requirement for spacecraft-to-Earth communication is 2.08 Mb/s, that can be achieved by the proposed architecture at 100 MHz (2.1 Mb/s). With this target frequency, the total area occupation is 1.01 mm^2: thanks to the shared datapath approach, more than 90% of the LDPC datapath is included in the larger turbo datapath, with very few LDPC-exclusive components.

This is also reflected on the power consumption estimate, resulting in 18.4 mW at 100 MHz. Memories occupy 82.6% of the decoder area, and account for 70.6% of the total power consumption. Pipeline stages contribute for 10.1% of the area and 13.3% of power conumption, with the remaining 7.3% area occupation and 16.1% power consumption being taken by processing, addressing and control logic. The implementation results show that this decoder has a smaller area and lower power consumption than most LDPC and turbo decoders [14, 113–115]. Obviously, due to the very reduced throughput target, the obtained throughput-to-area ratio is low. It yields, however, an energy efficiency of 8.76 nJ/bit, outperforming the majority of the state of the art.

Thanks to the low throughput requirements of CCSDS, it has been possible to design a simple decoder structure, and to guarantee very low power consumption. The proposed architecture, however, can sustain higher throughputs: 10.5 Mb/s have been obtained by synthesizing the presented decoder without any modifications targeting a frequency of 500 MHz. The implementation yields an area occupation of 1.06 mm^2 and 111.9 mW power consumption. To achieve even higher throughputs, it is possible to reduce the system critical path by adding a pipeline stage in the EXT-α, EXT-β modules and another in the α, β modules: with these straightforward modifications, up to 14.5 Mb/s can be obtained. Another possible approach can be incrementing the degree of parallelism of the decoder: by subdividing the current memory structure in a number of smaller banks, multiple instances of the datapath can work concurrently, virtually multiplying the achievable throughput.

The state of the art is currently lacking extensive information about decoders aimed at deep space communications, making the comparison between the concatenated FEC scheme implementation and alternative solutions unfeasible. The work in [116] presents an FPGA-based LDPC decoder for space communications: however, the considered near-Earth transmissions involve codes and specifications

very different from deep-space links. Turbo codes are a more mature technology in the deep space field, and various CCSDS-compliant turbo decoders are available on the market [117, 118]. However, very few scientific papers have been published on the subject. The work in [119] discusses the implementation of a CCSDS-compliant turbo decoder, but it is based on multiple off-the-shelf Digital Signal Processors, lacking area occupation and power consumption details. Also evaluating the area, power and energy efficiency gain of the proposed solution with respect to similar architectures is problematic. Shared datapath LDPC and turbo decoders are present in the literature, for which complete implementation results are provided [50, 52]. Their target applications are wireless communication standards like 3GPP-LTE, WiMAX, WiFi and DVB, for which BER and throughput requirements are extremely different from deep-space communications. These decoder designs are often based on high levels of parallelism, favoring speed over performance, especially in video broadcasting. For example, the decoder presented in [52] relies on a completely shared datapath. Since the target throughput ranges between 450 and 600 Mb/s, the internal parallelism of each decoding core can be close to a hundred, while the frequency is set to 500 MHz. Moreover, to give full support to high-throughput communication standards, multiple instances of parallel cores are used. Consequently, while the concept of datapath sharing and turbo/LDPC code decoding behind the presented work and [52] is similar, the difference in throughput requirements results in diverging design choices, that lead to a more than three-fold area occupation and an estimated $\times 20$ factor in power consumption. While it is clear that a fair comparison with the state of the art cannot be performed, it is possible to get a sense of where the proposed decoder stands. The CCSDS-compliant RS decoder [120] and Viterbi decoder [121] yield a total area normalized to 90 nm CMOS technology of 0.63 mm^2. The RS+CC FEC schemes is consequently cheaper to implement than the proposed turbo/LDPC concatenation, but its performance is much worse. An additional evaluation can be made thanks to the resource utilization data given in Lattice Semiconductor FPGA-based CCSDS turbo decoder [117]. Approximately 8000 Look-Up Tables (LUTs) and 4000 flip-flops are necessary for different Lattice devices. This work, implemented on a Xilinx Virtex 6 FPGA, requires 6000 LUTs and 1000 flip-flops, having better performance while at the same time occupying a smaller area than [117].

Chapter 7

Additional contributions and conclusions

7.1 Additional contributions

Additional publications to which the author has contributed are listed below:

- In [20], an extensive review of the state of the art on flexible LDPC decoders is given.
- In [21], an LDPC decoder for euclidean-geometry LDPC codes is designed, with FPGA implementation results being given. The author's contribution lies within the code simulation and decoder architecture design and validation issues.
- In [22], a circuit implementing an optimal routing algorithm for Kautz topology NoCs is presented. The author has carried out the VLSI implementation of the proposed circuit.

7.2 Conclusion and future perspectives

The presented work has been mainly focused on the exploration of alternative solutions for flexible channel decoders. The NoC paradigm has been analyzed and proven effective for both turbo and LDPC code decoding, providing excellent flexible support for multi-standard decoders. Its main drawbacks, i.e. large overheads in area occupation and power consumption, have been tackled by stripping the NoC of general-purpose characteristics and by introducing ad-hoc power reduction techniques. The reconfiguration of NoC based decoders has been studied in depth. Complete decoders have been designed at various steps of the study, resulting in competitive flexible implementations. Additional studies in power reduction and low complexity will reduce the gap between flexible and

dedicated decoders even more. The JANoCS simulator developed within this research is a powerful tool that will ease the design of future MP-SoCs, whatever their purpose might be. In particular, the powerful polar codes have been drawing the interest of the research community in the last few years. Since an important part of future studies will concern smart polar decoder designs, JANoCS will help to significantly speed up the design space exploration.

Regarding the study on early stopping criteria for LDPC decoders, the devised MSESC has been evaluated and compared to the state of the art taking in account both performance and actual energy saving. This is a novel approach unseen before in the literature on the subject, and has confirmed the effectiveness of the proposed criterion. Future developments can be foreseen in the direction of generalization of the on-the-fly threshold computation mechanism to a wider set of codes, and an extension of the implementation-wise energy saving evaluation to other early stopping criteria. While the state of the art on this subject is wide, the lack of an accurate analysis of the actual effectiveness of early stopping criteria prevents the consolidation of these methods within the academical and industrial community.

Also the devised UEP falls within a larger research towards the improvement of LDPC decoders, even using future unreliable fabrication technologies: it has been proven extremely useful under very high error probabilities. The proposed method has shown very good resilience to hard errors as well, and the implementation of UEP has allowed to measure the impact of UEP on a decoder area occupation and power consumption, showing competitive figures. Later research will proceed to extend UEP to different applications other than LDPC decoding: the effectiveness of the proposed solution, while it has been proven in this particular case, needs to be evaluated when applied in different contexts.

Within the research on deep space communications, a new FEC scheme has been designed, together with a low-complexity and low-power decoder. Results are promising in both the scheme's error correction capabilities and the associated decoder complexity. Future research will refine the FEC scheme, while evaluating its usefulness in other communication environments. Broadcasting could particularly benefit from the proposed scheme, since it is impossible to ask for retransmission in case of failed reception.

Bibliography

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

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

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

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

[5] *Multiplexing and Channel Coding, organization =*, Std.

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

[7] A. Morello and V. Mignone, "DVB-S2: The second generation standard for satellite broad-band services," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 210 –227, 2006.

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

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

[10] *TM Synchronization and Channel Coding*, Consulative Committee for Space Data Systems (CCSDS) Std. 131.0-B-2, Aug. 2011.

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

[12] C. Condo and G. Masera, "A flexible NoC-based LDPC code decoder implementation and bandwidth reduction methods," in *Design and Architectures for Signal and Image Processing (DASIP), 2011 Conference on*, nov. 2011, pp. 1 –8.

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

[14] ——, "VLSI implementation of a multi-mode turbo/LDPC decoder architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 6, pp. 1441–1454, 2013.

[15] C. Condo, A. Baghdadi, and G. Masera, "A joint communication and application simulator for NoC-based custom SoCs: LDPC and turbo codes parallel decoding case study," in *Digital System Design (DSD), 2013 Euromicro Conference on*, 2013, pp. 168–174.

[16] ——, "Reducing the dissipated energy in multi-standard turbo and LDPC decoders," *Circuits, Systems & Signal Processing*, under review.

[17] ——, "Energy-efficient multi-standard early stopping criterion for LDPC iterative decoding," *IET Communications*, under review.

[18] C. Condo, "Concatenated turbo/LDPC codes for deep space communications: performance and implementation," in *International Conference on Advances in Satellite and Space Communications (SPACOMM)*, apr. 2013, pp. 1 –6.

[19] C. Condo and G. Masera, "A unified turbo/LDPC code decoder architecture for deep-space communications," *IEEE Transactions on Aerospace and Electronic Systems*, under review.

[20] M. Awais and C. Condo, "Flexible LDPC decoder architectures," *VLSI Design*, vol. 2012, pp. 1–16, 2012.

[21] S. Zaidi, M. Awais, C. Condo, M. Martina, and G. Masera, "FPGA accelerator of quasi cyclic EG-LDPC codes decoder for NAND flash memories," in *Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference on*, 2013, pp. 190–195.

[22] C. Condo, M. Martina, M. Ruo Roch, and G. Masera, "Rediscovering logarithmic diameter topologies for low latency network-on-chip-based applications," in *Proc. of Euromicro International Conference on Parallel, Distributed and network-based Processing*, to appear.

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

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

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

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

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

[28] D. MacKay, *Information theory, inference and learning algorithms.* Cambridge University Press, 2003.

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

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

[31] V. Savin, "Self-corrected Min-Sum decoding of LDPC codes," in *Proc. of IEEE International Symposium on Information Theory*, jul. 2008, pp. 146 –150.

[32] M. Mansour and N. Shanbhag, "Turbo decoder architectures for low-density parity-check codes," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 2, nov. 2002, pp. 1383 – 1388 vol.2.

[33] E. Boutillon, W. Gross, and P. Gulak, "VLSI architectures for the MAP algorithm," *Communications, IEEE Transactions on*, vol. 51, no. 2, pp. 175–185, 2003.

[34] G. Masera, G. Piccinini, M. Roch, and M. Zamboni, "VLSI architectures for turbo codes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 7, no. 3, pp. 369–379, 1999.

[35] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE," *Solid State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 8– 17, Jan. 2011.

[36] Y. Sun and J. R. Cavallaro, "Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder," *Integration, the {VLSI} Journal*, vol. 44, no. 4, pp. 305 – 315, 2011, ¡ce:title¿Hardware Architectures for Algebra, Cryptology and Number Theory¡/ce:title¿.

[37] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *Custom Integrated Circuits Conference, 2009. CICC '09. IEEE*, sept. 2009, pp. 487 –490.

[38] ——, "A 50Mbps double-binary turbo decoder for WiMAX based on bit-level extrinsic information exchange," in *Solid-State Circuits Conference, IEEE Asian*, nov. 2008, pp. 305 –308.

[39] R. Shrestha and R. Paily, "Design and implementation of a high speed MAP decoder architecture for turbo decoding," in *VLSI Design and 2013 12th International Conference on Embedded Systems (VLSID), 2013 26th International Conference on*, 2013, pp. 86–91.

[40] S.-J. Lee, N. Shanbhag, and A. Singer, "Area-efficient high-throughput map decoder architectures," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 8, pp. 921–933, 2005.

[41] H. Arai, N. Miyamoto, K. Kotani, H. Fujisawa, and T. Ito, "A wimax turbo decoder with tailbiting bip architecture," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, 2010, pp. 371–372.

[42] S. M. Karim and I. Chakrabarti, "An improved low-power high-throughput log-map turbo decoder," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 2, pp. 450–457, 2010.

[43] R. Al-Khayat, A. Baghdadi, and M. Jezequel, "Architecture efficiency of application-specific processors: A 170Mbit/s 0.644mm2 multi-standard turbo decoder," in *System on Chip (SoC), 2012 International Symposium on*, 2012, pp. 1–7.

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

[45] Y. Cui, X. Peng, Z. Chen, X. Zhao, Y. Lu, D. Zhou, and S. Goto, "Ultra low power QC-LDPC decoder with high parallelism," in *IEEE International SOC Conference*, sept. 2011, pp. 142 –145.

[46] X. Peng, Z. Chen, X. Zhao, D. Zhou, and S. Goto, "A 115mW 1Gbps QC-LDPC decoder ASIC for WiMAX in 65nm CMOS," in *Solid State Circuits Conference (A-SSCC), 2011 IEEE Asian*, 2011, pp. 317–320.

[47] C.-W. Sham, X. Chen, F. Lau, Y. Zhao, and W. Tam, "A 2.0 Gb/s throughput decoder for QC-LDPC convolutional codes," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 60, no. 7, pp. 1857–1869, 2013.

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

[49] 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.

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

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

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

[53] 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.

[54] The IEEE p802.3an, 10GBASE-T task force. [Online]. Available: www.ieee802.org/3/an

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

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

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

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

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

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

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

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

[63] 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.

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

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

[66] V. Lapotre, P. Murugappa, G. Gogniat, A. Baghdadi, M. Hubner, and J.-P. Diguet, "Stopping-free dynamic configuration of a multi-ASIP turbo decoder," in *Digital System Design (DSD), 2013 Euromicro Conference on*, 2013, pp. 155–162.

[67] P. Murugappa, V. Lapotre, A. Baghdadi, and M. Jezequel, "Rapid design and prototyping of a reconfigurable decoder architecture for QC-LDPC codes," in *Rapid System Prototyping (RSP), 2013 International Symposium on*, 2013, pp. 87–93.

[68] V. Lapotre, P. Murugappa, G. Gogniat, A. Baghdadi, J.-P. Diguet, J.-N. Bazin, and M. Hubner, "A reconfigurable multi-standard ASIP-based turbo decoder for an efficient dynamic reconfig-

uration in a multi-ASIP context," in *VLSI (ISVLSI), 2013 IEEE Computer Society Annual Symposium on*, 2013, pp. 40–45.

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

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

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

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

[73] F. Steenhof, H. Duque, B. Nilsson, K. Goossens, and R. Llopis, "Networks on chips for high-end consumer-electronics TV system architectures," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 2, march 2006, pp. 1 –6.

[74] F. Wang, "Influence of traffic correlation on the performance of network-on-chip designs," *Journal of Circuits, Systems and Computers*, vol. 19, no. 03, pp. 655–669, 2010.

[75] OMNET++ network simulation framework. [Online]. Available: http://www.omnetpp.org/

[76] J. Li, X.-H. You, and J. Li, "Early stopping for LDPC decoding: convergence of mean magnitude (CMM)," *IEEE Communications Letters*, vol. 10, no. 9, pp. 667 –669, sept. 2006.

[77] A. Hunt, S. Crozier, K. Gracie, and P. Guinand, "A completely safe early-stopping criterion for max-log turbo code decoding," in *International Symposium on Turbo Codes and Related Topics*, 2006, pp. 1–6.

[78] O. Muller, A. Baghdadi, and M. Jezequel, "Bandwidth reduction of extrinsic information exchange in turbo decoding," *IET Electronics Letters*, vol. 42, no. 19, pp. 1104–1105, Sep 2006.

[79] W.-H. Hu, J. H. Bahn, and N. Bagherzadeh, "Parallel LDPC decoding on a Network-on-Chip based multiprocessor platform," in *International Symposium on Computer Architecture and High Performance Computing*, oct. 2009, pp. 35 –40.

[80] M. Imase, M.; Itoh, "A design for directed graphs with minimum diameter," *IEEE Transactions on Computers*, vol. C-32, no. 8, pp. 782–784, Aug. 1983.

[81] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic.* Academic Press, 1965.

[82] S.-H. Lo, Y.-C. Lan, H.-H. Yeh, W.-C. Tsai, Y.-H. Hu, and S.-J. Chen, "QoS aware BiNoC architecture, year=2010, pages=1-10, keywords=, issn=1530-2075,," in *IEEE International Symposium on Parallel Distributed Processing.*

[83] A. Banerjee, P. Wolkotte, R. Mullins, S. Moore, and G. J. M. Smit, "An energy and performance exploration of network-on-chip architectures," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 319–329, 2009.

[84] M. Martina and G. Masera, "State metric compression techniques for turbo decoder architectures," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, no. 5, pp. 1119–1128, 2011.

[85] W. Dally, "Virtual-channel flow control," in *International Symposium on Computer Architecture*, 1990, pp. 60–68.

[86] A. V. de Mello, L. C. Ost, F. G. Moraes, and N. L. V. Calazans, "Evaluation of routing algorithms on mesh based NoCs," in *Faculdade de Informatica, Pontifcia Universidade Catolica do Rio Grande do Sul, Tech. Rep*, vol. 40, may 2004.

[87] E. Amador, R. Pacalet, and V. Rezard, "Optimum LDPC decoder: A memory architecture problem," in *ACM/IEEE Design Automation Conference*, 2009, pp. 891–896.

[88] 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 *Proc. of IEEE International Symposium on Circuits and Systems*, jun. 2010, pp. 473 –476.

[89] T. Mohsenin, H. Shirani-Mehr, and B. Baas, "Low power LDPC decoder with efficient stopping scheme for undecodable blocks," in *Proc. of IEEE International Symposium on Circuits and Systems*, May. 2011.

[90] Z. Cai, J. Hao, and U. Sethakaset, "Efficient early stopping method for LDPC decoding based on check-node messages," in *Proc. of Asilomar Conference on Signals, Systems and Computers*, oct. 2008, pp. 466 –469.

[91] E. Amador, R. Knopp, R. Pacalet, and V. Rezard, "High throughput and low power enhancements for LDPC decoders," *International Journal on Advances in Telecommunications*, vol. 4, no. 1, aug. 2011.

[92] F. Kienle and N. Wehn, "Low complexity stopping criterion for LDPC code decoders," in *Proc. of IEEE Vehicular Technology Conference*, vol. 1, 2005, pp. 606–609.

[93] S. R. Saunders and A. A. Zavala, *Antennas and Propagation for wireless communication systems*. Wiley, 2007.

[94] J. Geier. (2005) SNR cutoff recommendations. [Online]. Available: http://www.wifiplanet.com/tutorials/article.php/3468771

[95] M. May, M. Alles, and N. Wehn, "A case study in reliability-aware design: A resilient LDPC code decoder," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 456–461.

[96] E. Kim and N. Shanbhag, "Energy-efficient LDPC decoders based on error-resiliency," in *Signal Processing Systems (SiPS), 2012 IEEE Workshop on*, 2012, pp. 149–154.

[97] K. Zhang, X. Huang, and Z. Wang, "A high-throughput LDPC decoder architecture with rate compatibility," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, no. 4, pp. 839–847, 2011.

[98] K. Namba and E. Fujiwara, "Unequal error protection codes with two-level burst and capabilities," in *Defect and Fault Tolerance in VLSI Systems, 2001. Proceedings. 2001 IEEE International Symposium on*, 2001, pp. 299–307.

[99] C. Argyrides, H.-R. Zarandi, and D. Pradhan, "Multiple upsets tolerance in sram memory," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, 2007, pp. 365–368.

[100] *TM Channel Coding Profiles*, Consulative Committee for Space Data Systems (CCSDS) Std. 131.4-M-1, Jul. 2011.

[101] Z. Wang and Z. Cui, "Low-complexity high-speed decoder design for Quasi-Cyclic LDPC codes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 1, pp. 104 –114, jan. 2007.

[102] A. Ahmed, M. Awais, A. ur Rehman, M. Maurizio, and G. Masera, "A high throughput turbo decoder VLSI architecture for 3GPP LTE standard," in *Multitopic Conference (INMIC), 2011 IEEE 14th International*, dec. 2011, pp. 340 –346.

[103] *Radio Frequency and Modulation Systems Part 1: Earth Stations and Spacecraft*, Consulative Committee for Space Data Systems (CCSDS) Std. 401.0-B-21, Jul. 2011.

[104] S. H. Lee, J. A. Seok, and E. K. Joo, "Serial concatenation of LDPC and turbo code for the next generation mobile communications," in *Wireless and Optical Communications Networks, 2005. WOCN 2005. Second IFIP International Conference on*, march 2005, pp. 425 – 427.

[105] *TM Synchronization and Channel Coding - Summary of Concept and Rationale*, Consulative Committee for Space Data Systems (CCSDS) Std. 130.1-G-2, Nov. 2012.

[106] K. Kwon, H. H. Im, and J. Heo, "An improved FEC system for next generation terrestrial 3D HDTV broadcasting," in *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, jan. 2012, pp. 327 –328.

[107] W. Lei, L. Jing, and W. J. bo, "Design of concatenation of fountain and non-binary LDPC codes for satellite communications," in *Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on*, dec. 2010, pp. 1 –4.

[108] N. Andreadou, F.-N. Pavlidou, S. Papaharalabos, and P. Mathiopoulos, "Quasi-cyclic low-density parity-check (QC-LDPC) codes for deep space and high data rate applications," in *Satellite and Space Communications, 2009. IWSSC 2009. International Workshop on*, sept. 2009, pp. 225 –229.

[109] O. Bursalioglu, G. Caire, and D. Divsalar, "Joint source-channel coding for deep space image transmission using rateless codes," in *Information Theory and Applications Workshop (ITA), 2011*, feb. 2011, pp. 1 –10.

[110] S. Gounai, T. Ohtsuki, and T. Kaneko, "Performance of concatenated code with LDPC code and RSC code," in *Communications, 2006. ICC '06. IEEE International Conference on*, vol. 3, june 2006, pp. 1195 –1199.

[111] S. H. Lee, J. A. Seok, and E. K. Joo, "Serially concatenated LDPC and turbo code with global iteration," in *Systems Engineering, 2005. ICSEng 2005. 18th International Conference on*, aug. 2005, pp. 201 – 204.

[112] W. Yao, L. Chen, H. Li, and H. Xu, "Research on fountain codes in deep space communication," in *Image and Signal Processing, 2008. CISP '08. Congress on*, vol. 2, may 2008, pp. 219 –224.

[113] 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.

[114] M. May, T. Ilnseher, N. Wehn, and W. Raab, "A 150mbit/s 3gpp lte turbo code decoder," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 1420–1425.

[115] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29 mm^2 52 mw multi-mode LDPC decoder design for mobile WiMAX system in 0.13 μm CMOS process," *IEEE Jour. of Solid-State Circuits*, vol. 43, no. 3, pp. 672 –683, 2008.

[116] F. Demangel, N. Fau, N. Drabik, F. Charot, and C. Wolinski, "A generic architecture of CCSDS low density parity check decoder for near-earth applications," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009, pp. 1242–1245.

[117] *Lattice Turbo Decoder*, Lattice Semiconductor Corporation, 2008. [Online]. Available: http://www.latticesemi.com/lit/docs/ip/ipug14.pdf

[118] *TC6000 CCSDS turbo decoder*, Turbo Concept. [Online]. Available: http://www.turboconcept.com/prod_tc6000.php

[119] J. Berner and K. Andrews, "Deep space network turbo decoder implementation," in *Proc. of IEEE Aerospace Conference*, vol. 3, 2001, pp. 3/1149–3/1157 vol.3.

[120] Y.-K. Lu and M.-D. Shieh, "Efficient architecture for Reed-Solomon decoder," in *VLSI Design, Automation, and Test (VLSI-DAT), 2012 International Symposium on*, april 2012, pp. 1 –4.

[121] M. Kawokgy, C. Andre, and T. Salama, "Low-power asynchronous viterbi decoder for wireless applications," in *International Symposium on Low Power Electronics and Design*, 2004, pp. 286–289.