Towards higher speed decoding of convolutional turbocodes

Oscar David Sanchez Gonzalez

N° d'ordre : 2013telb0270

Sous le sceau de l'Université européenne de Bretagne

Télécom Bretagne

En habilitation conjointe avec l'Université de Bretagne-Sud

École Doctorale – SICMA

La montée en débit dans les architectures de turbo-décodage de codes convolutifs

---

Thèse de Doctorat

Mention : « STIC (Science et technologies de l'information et de la communication) »

Présentée par **Oscar David Sánchez González**

Département : Electronique

Laboratoire : Lab – STICC Pôle : CACS

Directeur de thèse : Michel Jézéquel
Co-directeur de thèse : Christophe Jégo

Soutenue le 15 Mars 2013

Jury :

M. Guy Gogniat, Professeur des Universités, Lab-STICC, UBS (Président / Examinateur)
M. Pierre Pénard, Ingénieur, Orange Labs (Examinateur)
M. Bertrand Granado, Professeur des Universités, LIP6, UPMC (Rapporteur)
M. Daniel Chillet, Maître de conférence HDR, CAIRN, ENSSAT (Rapporteur)
M. Michel Jézéquel, Professeur Institut Télécom, Télécom Bretagne (Directeur de thèse)
M. Christophe Jégo, Professeur des Universités, IMS, IPB (Co-directeur de thèse)

DEDICATION

A mis profesores de física y química del colegio. Todo ha sido posible gracias a ellos.

ACKNOWLEDGEMENTS

Je tiens à exprimer ici toute ma reconnaissance aux personnes qui m'ont aidé, encouragé et soutenu tout au long de mon travail de thèse.

Je remercie tout d'abord Guy Gogniat, professeur à L'UBS pour m'avoir fait l'honneur de présider le jury de cette thèse, et Pierre Pénard, ingénieur à Orange labs, pour avoir accepté d'examiner ces travaux de recherche.

J'adresse également mes remerciements à Bertrand Granado, professeur à l'UPMC-LIP6, et à Daniel Chillet, Maître de conférence à l'ENSSAT, qui ont accepté d'être rapporteurs de ces travaux de thèse.

J'ai aussi eu la chance et le plaisir d'avoir Michel Jézéquel, directeur d'études à Telecom Bretagne, comme directeur de thèse : je lui suis reconnaissant pour son accueil au sein du Département Electronique, son écoute et son soutien tout au long de mes travaux de thèse. Les nombreux échanges que nous avons eu tout au long de ce travail m'ont beaucoup apporté et motivé.

Je tiens à remercier également Christophe Jégo, co-directeur de thèse et professeur à L'IMS-IPB, pour son soutien permanent au cours de mes travaux de recherche, et sa disponibilité toujours exceptionnelle malgré une période de travail à distance. Il m'a permis de trouver la bonne direction dans mes activités de recherche. Son implication et son enthousiasme ont été une grande source de motivation.

Je voudrais aussi remercier Yannik Saouter, chercheur associé à Telecom Bretagne, pour les échanges très intéressants que nous avons eu concernant les architectures de décodage de radix élévé.

Je remercie tous les membres du département Electronique pour leur accueil chaleureux et leur convivialité lors de différents moment partagés au bâtiment K2.

Je tiens à saluer les doctorants du département que j'ai eu le plaisir de rencontrer durant cette thèse : Daoud, Roa, Atif, Purush, Salim, Haifa, Ammar, Ronald, Daniel, Quang, Rasheed, Vincent, Ali, Tristan, Meng et Nicolas.

Quiero igualmente agradecer a todos mis amigos por hacer de mi paso por Brest una experiencia agradable y muy divertida: Jorge, Victoria, July Paola, Juan David, Santiago, Juan Pablo, Patricia, Luiz, Soraya, Isabel, Elisa, Roa, Pedro, Yenny, Daniel, Ronald, Javier, Kedar, Vinicius, Omar, Fabian, Manuel. Gracias por todo!

Un agradecimiento muy especial a Hélène por su comprensión, apoyo y ayuda durante estos ya tres años.

Finalmente quiero agradecer toda a mi familia. Gracias a Myriam y Fernando por el buen ejemplo y apoyo. Muchas gracias a Andrés por los consejos y motivación en los momentos más oportunos.

Abstract

La montée en débit dans les architectures de turbo-décodage de codes convolutifs

Oscar David Sánchez González
Department Electronique, Telecom - Bretagne
15 Mars 2013

The turbo codes are a well known channel coding technique widely used because of their outstanding error decoding performance close to the Shannon limit. These codes were proposed using a clever pragmatic approach where a set of concepts that had been previously introduced, together with the iterative processing of data, are successfully combined to obtain close to optimal decoding performance capabilities. However, precisely because this iterative processing, high latency values appear and the achievable decoder throughput is limited.

At the beginning of our research activities, the fastest turbo decoder architecture introduced in the literature achieved a throughput peak value around 700 Mbit/s. There were also several works that proposed architectures capable of achieving throughput values around 100 Mbit/s. Research opportunities were then available in order to establish architectural solutions that enable the decoding at a few Gbit/s, so that the industrial requirements are fulfilled and future high performance digital communication systems can be conceived.

The first part of this work is devoted to the study of the turbo codes at an algorithmic level. Several SISO decoder algorithms are explored, and different parallel turbo decoder techniques are analyzed. The convergence of parallel turbo decoder is specially considered. To this end the EXtrinsic Information Transfer (EXIT) charts are used. Conclusions derived from these kind of diagrams have served to propose a novel SISO decoder schedule to be used in shuffled turbo decoder architectures.

The architectural issues when implementing high parallel turbo decoder are considered in the second part of this thesis. We propose a high throughput low complexity radix-16 SISO decoder. This decoder is intended to break the bottleneck that appears because of the recursive operations in the heart of the turbo decoding algorithm. The design of this architecture was possible thanks to the elimination of parallel paths in a radix-16 trellis diagram transition. The proposed SISO decoder implements a high speed radix-8 Add Compare Select (ACS) unit which exhibits a lower hardware complexity and lower critical path compared with a radix-16 ACS unit. Our radix-16 SISO decoder degrades the turbo decoder error correcting performance. Therefore, we have proposed two techniques so that the architecture can be used in practical applications. Thus, architectural solutions to build high parallel turbo decoder architectures, which integrate our SISO decoder, are presented. Finally, a methodology to efficiently explore the design space of parallel turbo decoder architectures is described. The main objective of this approach is to reduce the time to market constraint by designing turbo decoder architectures for a given throughput.

RÉSUMÉ

La montée en débit dans les architectures de turbo-décodage de codes convolutifs

Oscar David Sánchez González
Département Electronique, Telecom - Bretagne
15 Mars 2013

Les turbocodes sont des codes correcteurs d'erreurs qui présentent des performances remarquables, proches de la limite théorique de Shannon. Ils utilisent un décodage itératif qui permet d'avoir une complexité matérielle limitée. Cependant, à cause de ce traitement itératif le débit de décodage est fortement réduit.

Au début de nos activités de recherche l'architecture la plus rapide de turbo-décodage dans la littérature atteignait un débit autour de 700 Mbit/s. Plusieurs autres travaux proposaient des architectures capables d'atteindre des débits d'environ 100 Mbit/s. Des travaux de recherche devaient donc se faire pour concevoir des architectures qui permettaient le décodage à plusieurs Gbit/s. Ainsi, les besoins de l'industrie peuvent être atteints et des systèmes de communication de haute performance peuvent être conçus dans le futur.

La première partie de cette thèse étudie les turbocodes à partir d'un point de vue algorithmique. Plusieurs algorithmes pour les décodeurs SISO sont explorés, ainsi que les techniques de parallélisme de turbo-décodage. L'analyse de la convergence des turbo-décodeurs parallèles est effectuée, et un nouvel ordonnancement pour turbo-décodeurs shuffled est présenté. Pour ce faire, les diagrammes d'EXIT (EXtrinsic Information Transfer) sont utilisés. Leur emploi nous a permis de concevoir un nouvel ordonnancement pour turbo-décodeurs shuffled.

Dans la seconde partie de la thèse nous considérons les problèmes architecturaux qui apparaissent lors de la mise en oeuvre de turbo-décodeurs. Ainsi, un décodeur SISO radix-16 est conçu pour briser le goulot d'étranglement de l'algorithme de turbo-décodage. C'est un décodeur SISO de faible complexité qui est utilisé comme le principal bloc de calcul d'une architecture turbo-décodage hautement parallèle. Ce décodeur SISO est basé sur l'élimination des chemins parallèles dans le diagramme d'un treillis radix-16. Pour maîtriser la complexité il utilise une unité ACS (Add Compare Select) radix-8, ce qui nous permet aussi de réduire le chemin critique. Deux techniques complémentaires sont introduites afin de surmonter la dégradation qui apparaît lorsque des turbo-décodeurs basés sur le décodeur SISO proposé sont considérées. Nous proposons également des solutions architecturales pour concevoir des turbo-décodeurs hautement parallèles radix-16. Enfin, nous présentons une méthodologie pour explorer efficacement l'espace de conception des différentes architectures de turbo-décodage. Le but principal est de réduire le temps de conception de manière à pouvoir estimer le débit qui peut être attendu dès le début du processus de conception.

Contents

List of Figures

List of Tables

Introduction

Research works carried out through many decades have enable the design of modern digital communication systems that exhibit reliable communication performance and high transmission speed. These works have followed the guidelines originally given by Shannon [1], who established the fundamental principles of the reliable digital transmission of information. Shannon demonstrated that, thanks to an appropriate error correcting code, it is possible to achieve reliable digital transmission of information as long as the information rate of the source is less than the channel capacity. However, this error correcting code was not found by him, attracting in this way the interest of researchers around the world in order to propose efficient error correcting code constructions. Berrou et al., in what is now considered as a remarkable breakthrough in the channel coding theory, have introduced the turbo codes [2]. These codes were proposed using a clever pragmatic approach, where a set of concepts that had been previously introduced, together with the iterative processing of data, are successfully combined in order to approach the theoretical limit established by Shannon. The iterative processing provides good error correction performance with an acceptable computational complexity. However, it induces high latency and limits the achievable decoder throughput. The discovery of the turbo codes has prompted the investigation of other iterative decoding schemes. Thus, the turbo decoding principle has been extended to the product codes [3], and the rediscovery of the Low Density Parity Check (LDPC) codes [4] was carried out [5].

The remarkable performance of the turbo codes in terms of the error correcting capabilities, and their feasibility to be implemented, have led to the adoption of them in several wireless communication standards: Consultative Committee for Space Data Systems (CCSDS) for spatial communications; Universal Mobile Telecommunications System (UMTS), Third Generation Partnership Project 2 (3GPP2), Long Term Evolution (LTE) and LTE-Advanced for mobile phones; Worldwide Interoperability for Microwave Access (WiMAX) for wide area networks and Digital Video Broadcasting - Return Channel via Satellite (DVB-RCS) for digital video broadcasting. The requirements in terms

of the throughput rate have evolved form a few Mbit/s in the first practical applications, to a few hundred Mbit/s in WiMAX and LTE, up to data rates around 1 Gbit/s in four generation cellular communication systems, such as LTE-Advanced. The proposition of efficient architectural solutions to achieve high throughput turbo decoding rates is then a major challenge to accomplish, so that the industrial requirements are fulfilled and future high performance digital communication systems can be conceived. Since mobile devices are of high interest, besides the high throughput requirements, reduction in the system complexity and power consumption is also required. Indeed, in mobile communication systems, the adoption of turbo codes has consistently increased the share of channel decoding in the total receiver energy budget from around 30% to almost 50%. This means that the channel decoder is becoming the main energy bottleneck in the mixed-signal receiver [6].

PROBLEMATIC AND CONTRIBUTIONS

At the beginning of our research activities, the fastest turbo decoder architecture introduced in the literature achieved a throughput peak value around 700 Mbit/s. There were also several works that proposed architectures capable of achieving throughput values around 100 Mbit/s. Research opportunities were then available in order to establish architectural solutions that enable the decoding at a few Gbit/s. These architectural solutions should optimize the required hardware resources, so that the architectures can be exploited in practical applications. In this context, we have made different contributions during our work. These contributions are proposed at the algorithmic and architectural levels.

CONTRIBUTIONS AT THE ALGORITHMIC LEVEL

- A review of different Soft Input Soft Output (SISO) decoder algorithms suitable for turbo decoders implementation has been carried out. We have explored alternative algorithms that can be applied to reduce the hardware complexity or increase the throughput of a SISO decoder architecture. Soft Output Viterbi Algorithm (SOVA) based turbo decoders have been studied. The error correction performance degradation, their main drawback in order to implement practical system, is addressed. Thus, some techniques to reduce the performance degradation are explored.

- An extension of the EXtrinsic Information Transfer (EXIT) charts method in order to take into account the constraints introduced by parallel turbo decoder

implementations has been proposed. The analysis performed with this type of diagrams, associated with Monte-Carlo simulations, gives additional understanding of the convergence process for the design of parallel architectures dedicated to turbo decoding.

- A SISO decoder schedule for the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm has been introduced. This schedule has been proposed based on the convergence analysis carried out with the aid of the EXIT charts. It consists in employing the Butterfly-Replica schedule during the early turbo decoder iterations, and then, use the Butterfly schedule.

- Techniques to reduce the hardware complexity of high radix SISO decoder architectures are proposed. SISO decoders implemented with these techniques exhibit an important error correction performance degradation. Thus, complementary techniques to overcome this problem are also developed.

- A dedicated approach to efficiently explore the design space of parallel turbo decoder architectures has been presented. This approach has been proposed after a collaboration work carried out with the Université de Bretagne-Sud, at Lorient. Using this approach, a tradeoff between the hardware complexity and the throughput can be established in the early stages of the architecture design process. Our approach especially considers memory conflict issues, as well as SISO decoder architectures.

- We have performed a complexity and error correction performance study of receivers that implement full shuffled demapping with turbo decoding. This contribution was the result of a collaboration with a Ph.D student at the electronic engineering department of Telecom Bretagne. A reduction in the complexity in terms of the arithmetic operations performed is achieved when an appropriate SISO decoder schedule is applied.

Contributions at the Architectural Level

- A high speed radix-8 Add Compare Select (ACS) unit architecture has been proposed. It enables to increase the throughput of high radix SISO decoder architectures.

- A high speed low complexity radix-16 Max-Log-MAP SISO decoder architecture has been designed. Based on the elimination of parallel paths in the radix-16 trellis diagram, architectural solutions to reduce the hardware complexity of the different

blocks of the SISO decoder are proposed. In this way the radix-16 SISO decoder is able to use a radix-8 ACS unit. Thus, an increase in the SISO decoder throughput is possible with respect to conventional radix-16 architectures.

- A low complexity extrinsic memory architecture for high radix values has been presented. Taking advantage of the conflict free interleaver properties, a low complexity architecture to generate the memory address is designed. Beside, different banks are arranged in order to provide concurrent access to all the SISO decoders in the system.

- A high parallel turbo decoder architecture targeting throughput values of 1 Gbit/s and beyond is designed. It contains high speed radix-16 SISO decoders. A conflict free memory address architecture is also adopted.

- Field Programmable Gate Array (FPGA) prototyping of high radix turbo decoder architecture was carried out. A first prototype has been done in order to validate our propositions. Currently, a prototype that integrates 32 SISO decoders is under development. Results from this prototype will help to establish the advantages of our architecture with respect to other architectures in the literature.

THESIS BREAKDOWN

This document starts by a study at the algorithmic level of the convolutional turbo decoders. This study is intended to explore alternative decoding algorithms that can be convenient to reduce the decoder hardware complexity, or to increase the decoder throughput. Then, an analysis of the turbo decoder parallelism techniques is performed. The convergence of parallel turbo decoder architectures is also studied. In the last part of this work, architectural solutions to design high throughput turbo decoders architectures are introduced. This document is divided in different chapters organized as follows.

Chapter 1 is devoted to the presentation of the general context of our work. Initially, the basic concepts regarding the error correcting codes are presented. Special attention is given to the convolutional codes, since they are the heart of the turbo codes that we considered. A survey on the SISO decoder algorithms is also presented. For the BCJR algorithm, equivalent and simplified algorithms in the logarithmic domain are derived. The chapter ends with the presentation of the turbo decoding principle.

In Chapter 2, a comprehensive review of the parallel turbo decoding techniques is presented. These techniques are classified according to their granularity level. Afterwards, the turbo decoder transfer characteristics, analyzed by means of the EXIT chart diagrams, are given. Thus, the study of the convergence properties of parallel turbo

decoder architectures is presented. Based on this study, we propose a novel SISO decoder schedule for the Max-Log-MAP algorithm, suitable to increase the convergence of shuffled turbo decoder architectures. The chapter concludes with some results in terms of the error correcting performance for SOVA based turbo decoders.

Our first contribution at the architectural level is presented in Chapter 3. This chapter starts by a presentation of the SISO decoder structure for the Max-Log-MAP algorithm. Then, the architectural issues of high radix SISO decoders are explored. Afterwards, by considering high radix values, we propose the elimination of parallel paths in the trellis diagram in order to overcome the SISO decoder bottleneck. This elimination of parallel paths is further extended to reduce the SISO decoder hardware complexity. Thus, we present a low hardware complexity high speed radix-16 SISO decoder architecture. The use of this SISO decoder in a turbo decoder architecture degrades its error correction performance. Therefore, at the end of the chapter, we introduce two techniques to overcome this problem. In this way, our SISO decoder can be exploited in practical turbo decoding applications.

Chapter 4 concentrates in the issues that prevent a turbo decoder architecture to achieve high throughput values when multiple SISO decoders are implemented. Thus, the main bottleneck is identified as the concurrent access to the extrinsic memory. Consequently, architectural solutions considering the properties of the Almost Regular Permutation (ARP) and Quadratic Polynomial Permutation (QPP) interleavers are proposed. A memory organization to support high radix SISO decoder is explained. Also, a simplified architecture to address the memories is introduced. The memory conflict problems in shuffled architectures are considered as well. We adopt the SISO decoder architecture proposed in Chapter 3 in order to design a high throughput turbo decoder for the LTE standard. This architecture implements the required techniques in order to avoid error correcting performance degradation due to the considered radix-16 SISO decoder. A first FPGA prototype done to validate our ideas is described. The Chapter 4 ends with the presentation of a methodology that have been introduced to explore the turbo decoder design space. Thus, estimations in terms of the achievable throughput and hardware complexity of the final turbo decoder architecture can be established. This methodology is oriented to resolve concurrent access memory conflicts for any turbo decoder parallelism and interleaver.

In the last Chapter, a summary of our contributions is presented. A conclusion and some perspectives for future works are also given.

Chapter 1
Context of Channel Coding

This first chapter starts with an introduction of the main concepts of error correcting codes. Then, the soft decoding is presented. Afterwards, the convolutional codes are introduced by describing their structure and representation diagrams. The decoding of convolutional codes, with a special emphasis on the Recursive Systematic Codes (RSC), is also presented. Different types of convolutional code decoding algorithms are then described. Finally, the turbo coding and the turbo decoding principle are introduced.

1.1 CHANNEL CODING GENERALITIES

1.1.1 INTRODUCTION

Digital communication systems are designed to transmit information over noisy channels in order to provide reliable links between the source and the destination. Due to the noise disturbances, errors may appear during the transmission process, causing the messages produced by the source not to be well interpreted at the destination side. Channel coding techniques are used in order to reduce the probability of transmission errors by adding redundant information to the original message. These coding techniques seek to approach as much as possible the correction capabilities of the communication system to the theoretical limits established by Shannon in his pioneer work [1].

Figure 1.1 depicts the diagram of a classical digital communication system. The source generates a flow of bits $\boldsymbol{d}$ representing a particular message. This message can be related for instance to a video or voice signal, to digital data, or be the samples of a particular analog signal, where an analog to digital converter is needed. At the receiver side, estimates $\hat{\boldsymbol{d}}$ of those bits are provided to the destination. In an ideal scenario, the transmitted and estimated bits are such that $\boldsymbol{d} = \hat{\boldsymbol{d}}$ with *high* probability, *i.e,* a low error rate is achieved. To accomplish this objective, the channel encoder implements a code $\mathcal{C}$. Let $\boldsymbol{d}_k = (d_{k,1}, \ldots, d_{k,m_c})$ be a source message composed of m_c bits at time k. The channel encoder maps each information symbol onto a n_c bit codeword $\boldsymbol{c}_k = (c_{k,1}, \ldots, c_{k,n_c})$, with $n_c \geq m_c$. The ratio $R = m_c/n_c$ denotes the code rate. The coded information is then processed by the digital modulator that transforms digital signals representing codewords, into signals waveforms $\boldsymbol{u}$ to be transmitted through the communication channel. At the receiver side, after the demodulator, noisy symbols $\boldsymbol{r}$ try to be recovered by the channel decoder. Thus, the most probable transmitted symbol $\hat{\boldsymbol{d}}_k$, equivalent to the most probable coded symbol $\hat{\boldsymbol{c}}_k$, is found.

1.1.2 CHANNEL MODEL

For the purpose of our study we have chosen a simple modulation scheme and channel model without compromising the final goal of the work. Therefore, a zero mean memoryless Additive White Gaussian Noise (AWGN) channel is assumed. Furthermore, Binary Phase Shift Keying (BPSK) mapping is used with ideal modulator, demodulator and perfect synchronization. Hence, each codeword bit $c_{k,i} \in \{0, 1\}$ originates a modulated symbol $u_{k,i} \in \{-1, +1\}$, for $i = 1, 2, \cdots n_c$. Let σ_n^2 be the noise variance. Thus, after transmission over the noisy channel the demodulator outputs are $r_{k,i} = u_{k,i} + \eta_{k,i}$ with $\eta_{k,i} \sim \mathcal{N}(0, \sigma_n^2)$. The conditional probability density function (pdf) of the demodulated symbol $\boldsymbol{r}_k = (r_{k,1}, \cdots, r_{k,n_c})$ is then given by:

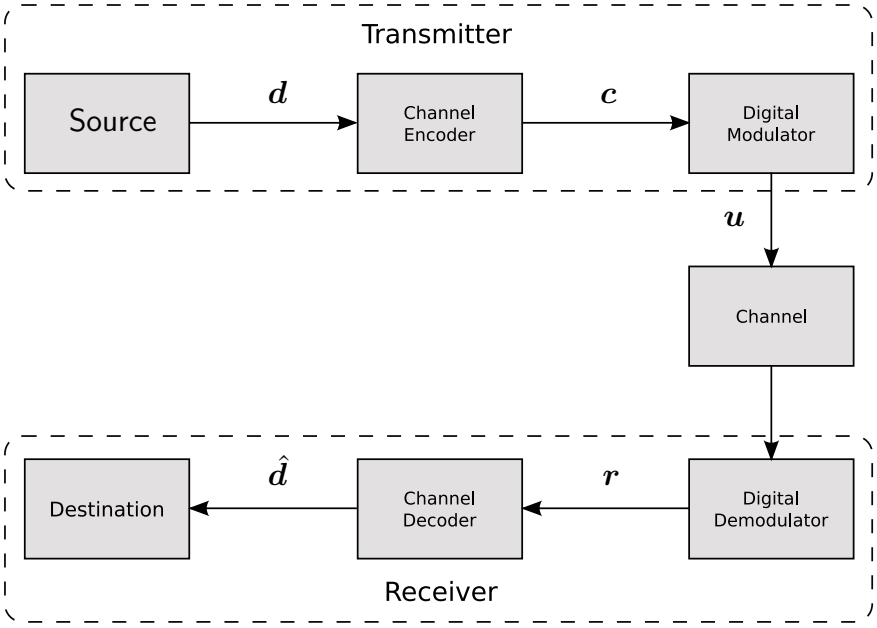

Figure 1.1: Elements of a classical digital communication system.

$$p(\boldsymbol{r}_k|\boldsymbol{c}_k) = p(\boldsymbol{r}_k|\boldsymbol{u}_k) = \prod_{i=1}^{n_c} p(r_{k,i}|u_{k,i}) \tag{1.1}$$

with the conditional probability for each received bit $r_{k,i}$ given by:

$$p(r_{k,i}|u_{k,i}) = \frac{1}{\sigma_n\sqrt{2\pi}} \cdot \exp\left(-\frac{(r_{k,i}-u_{k,i})^2}{2\sigma_n^2}\right) \tag{1.2}$$

For convenience, it can be appropriate to express the logarithm of probabilities rather than the probability itself. Thus, combining (1.1) and (1.2), and taking the natural logarithm of the resulting expression we obtain:

$$\log\left(p(\boldsymbol{r}_k|\boldsymbol{u}_k)\right) = -n_c \log\left(\sigma_n\sqrt{2\pi}\right) - \frac{1}{2\sigma_n^2}\sum_{i=1}^{n_c}(r_{k,i}-u_{k,i})^2 \tag{1.3}$$

The sum in (1.3) is the square of the Euclidean distance E_D:

$$E_D^2(\boldsymbol{r}_k, \boldsymbol{u}_k) = \sum_{i=1}^{n_c}(r_{k,i}-u_{k,i})^2 \tag{1.4}$$

Let E_b/N_0 be the Signal to Noise Ratio (SNR) with E_b the energy per information bit and N_0 the real power spectrum density of the noise. Therefore, the noise variance can be expressed in terms of the SNR and it depends on the code rate:

$$\sigma_n^2 = \frac{1}{2R \cdot E_b/N_0} \tag{1.5}$$

1.1.3 CHANNEL DECODING PROCESS

In order to find the more reliable transmitted bits, two decoding principles can be applied: Maximum A Posteriori (MAP) and Maximum Likelihood (ML) decoding. MAP decoding enables to maximize the probability:

$$P(\boldsymbol{u}_k|\boldsymbol{r}_k) = P(\boldsymbol{c}_k|\boldsymbol{r}_k) \tag{1.6}$$

Therefore, from the received channel output values $\boldsymbol{r}_k$, we try to find the most reliable transmitted signals $\boldsymbol{u}_k$, that corresponds to the codeword $\boldsymbol{c}_k$. Using Bayes' rule we have:

$$P(\boldsymbol{u}_k|\boldsymbol{r}_k) = \frac{p(\boldsymbol{r}_k|\boldsymbol{u}_k)P(\boldsymbol{u}_k)}{p(\boldsymbol{r}_k)} \tag{1.7}$$

with $p(\boldsymbol{r}_k|\boldsymbol{u}_k)$ the conditional pdf of the channel output values given $\boldsymbol{u}_k$, and $P(\boldsymbol{u}_k)$ the probability that the signals $\boldsymbol{u}_k$ were transmitted. $P(\boldsymbol{u}_k)$ is also called the a-priori probability. It gives additional information to the decoding process. Contrary to the MAP decoding principle, the ML decoding principle seeks to maximize $p(\boldsymbol{r}_k|\boldsymbol{u}_k)$ instead of $P(\boldsymbol{u}_k|\boldsymbol{r}_k)$. Both decoding principles lead to a similar decoding performance as long as no a-priori information exists.

From (1.3), note that the ML decoding principle is equivalent to minimize the Euclidean distance in (1.4). Thus, the ML decoding principle enables to converge to the closest transmitted waveforms $\boldsymbol{u}_k$ to $\boldsymbol{r}_k$. MAP decoding is optimum in the sense that it maximizes the symbol by symbol probability, *i.e*, it finds the most probable transmitted bit. On the other hand, ML decoding principle minimizes the errors regarding the sequence of transmitted information as a whole.

1.1.4 CHANNEL CODE PERFORMANCE

The channel code performance indicates the ability of a code to detect and eventually correct possible transmission errors. It depends on the SNR value. It can be improved by a careful selection of the code parameters. The code performance is presented in terms

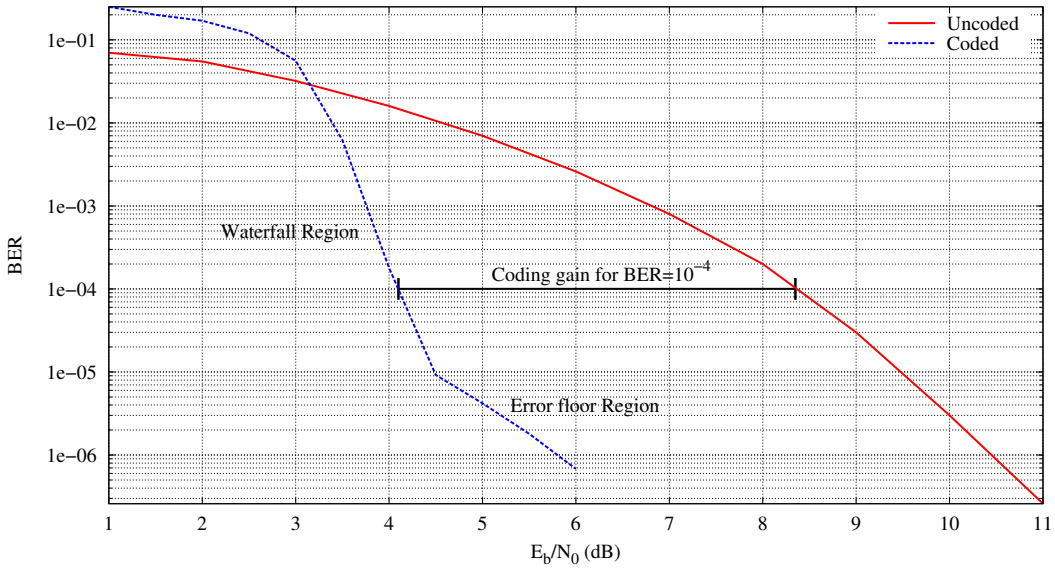

Figure 1.2: BER typical curves for a coded and uncoded system.

of the Frame Error Rate (FER) and Bit Error Rate (BER) values. Unfortunately their analytical expressions are usually very complex. Therefore, Monte-Carlo simulations are frequently used. The analytical expressions are reserved to describe the asymptotical behavior of the code for high E_b/N_0 values.

Figure 1.2 shows a typical BER performance curve. More specifically the BER performance curve of a concatenated code as presented in section 1.4. This curve can be divided into three different regions:

- Low E_b/N_0 region: In this region the code presents bad correction capabilities, having error rates even higher than the uncoded scheme.
- Waterfall region: Range of E_b/N_0 values for which a major reduction in the error rate can be achieved by small increases in the SNR.
- Error floor region: Medium to high E_b/N_0 values. A change in the slope is observed with respect to the waterfall region, reducing the improvement of the error rate as E_b/N_0 grows. For very high E_b/N_0 values, the slope of the coded curve approaches asymptotically the slope of the uncoded curve.

The benefits that a code provides are quantized in terms of the coding gain. It is defined as the SNR difference, usually expressed in decibels (dB), between the coded

and uncoded curves for a given error rate value. For instance, in Figure 1.2 the coding gain for a BER$= 10^{-4}$ is about 4.4 dB.

1.1.5 SOFT DECODING

The information provided by the demodulator to the channel decoder may have different levels. Let us first consider the simplest case were the demodulator is only able to determine if each received waveform corresponds to two possible levels, high or low. In this case the channel decoder receives a sequence of binary (hard) values. From this sequence the source message should be recovered. Such decoding process is called hard-decision decoding. Now, let us increase the demodulator capabilities so that it is able to provide, additionally to the sequence of binary values, reliability (soft) values for each hard value. The soft values represent how confident the demodulator is that each hard value is correctly estimated. The decoder has then more information that, intuitively, allows it to improve the decisions taken all over the decoding process. Even more, the channel decoder can generate reliability values itself. Such a decoder is called a Soft Input Soft Output (SISO) decoder, and the algorithm that it implements a SISO decoding algorithm.

When soft values are considered in the whole communication system, important improvements in the system performance can be observed [7]. It has been shown to be convenient in order to improve the error correction capabilities of the communication system. Specifically, SISO decoding algorithms has been proposed as part of the channel decoder to carry out iterative decoding. The main idea is to replace long and complex codes by simple codes that cooperate between them through an exchange of soft values [2, 8, 9]. The Turbo Codes, presented in section 1.4, are based on this idea.

Log-likelihood ratio and soft channel outputs.

Let V be a random variable in the Galois Field GF(2) with elements $\{+1, -1\}$ and v a realization of V. The probability that the random variable V takes the value $v = \pm 1$ is denoted by $P(v = \pm 1)$. Soft values can be represented by Log Likelihood Ratio (LLR) values, defined by:

$$L(v) = \log\left(\frac{P(v = +1)}{P(v = -1)}\right) \tag{1.8}$$

The sign and the magnitude $|L(v)|$ denote the hard and soft values of the random variable V, respectively. A large $|L(v)|$ value means that there is high certainty that the value estimated for v is correct. LLR values can be defined for conditional variables as well. Let V depends on the random variable Y. Thus, the LLR is:

$$L(v|y) = \log\left(\frac{P(v=+1|y)}{P(v=-1|y)}\right) = \log\left(\frac{P(v=+1,y)}{P(v=-1,y)}\right) \tag{1.9}$$

Using Bayes' rule, equation (1.9) can be written as:

$$\begin{aligned} L(v|y) &= \log\left(\frac{P(v=+1|y)}{P(v=-1|y)}\right) \\ &= \log\left(\frac{P(v=+1)}{P(v=-1)}\right) + \log\left(\frac{p(y|v=+1)}{p(y|v=-1)}\right) \\ &= L(v) + L(y|v) \end{aligned} \tag{1.10}$$

Let us consider once more Figure 1.1. Assuming BPSK mapping, each waveform $u_{k,i}$ transmits one bit. After the transmission over a Gaussian channel, using (1.2), the LLR value of $u_{k,i}$ conditioned on the matched filter output $r_{k,i}$ is then:

$$\begin{aligned} L(u_{k,i}|r_{k,i}) &= L(u_{k,i}) + L(r_{k,i}|u_{k,i}) \\ &= L(u_{k,i}) + \log\left(\frac{\exp\left(-\frac{1}{2\sigma_n^2}(r_{k,i}-1)^2\right)}{\exp\left(-\frac{1}{2\sigma_n^2}(r_{k,i}+1)^2\right)}\right) \\ &= L(u_{k,i}) + L_c \cdot r_{k,i} \end{aligned} \tag{1.11}$$

With $L_c = 2/\sigma_n^2$. L_c is called the reliability value of the channel. The largest the value of L_c is, the less errors will occur.

1.1.6 BLOCK CODES

The block codes are a family of codes without memory dependency, *i.e.*, each codeword depends only on the current source message, and not on previous messages. Thus, a $\mathcal{C}(n_c, m_c)$ block code maps an information block $\boldsymbol{d}_k$ of m_c information symbols into a codeword $\boldsymbol{c}_k$ of n_c coded symbols. $R = m_c/n_c$ is the code rate. The information and coded symbols are defined in GF(2^q). Thus, the code can generate $2^{q \cdot m_c}$ different codewords.

A block code $\mathcal{C}$ is linear if it is a m_c-dimensional subspace of the n_c-dimensional vector space of GF$(2^q)^{n_c}$. Thus, for each information block $\boldsymbol{d}_k$, a codeword $\boldsymbol{c}_k$ is generated following the linear expression $\boldsymbol{c}_k = \boldsymbol{d}_k \cdot \boldsymbol{G}$, with $\boldsymbol{G}$ a $m_c \times n_c$ matrix. This matrix is called the generator matrix of the code. Its rows form a basis of the code.

Let us consider a binary ($q = 1$) linear block code $\mathcal{C}$. The Hamming distance, denoted by $d_H(c^i, c^j)$, between two different codewords $c^i, c^j \in C$ is defined as the number of bits in which these two codewords are different. The minimal code distance is then the minimum Hamming distance between all possible pair of codewords:

$$d_{min} = \min_{c^i, c^j \in C} \left(d_H(c^i, c^j) \right), \ i \neq j \tag{1.12}$$

The Hamming weight $w_H(c^i)$ is defined as the number of nonzero symbols in c^i. Thus, the Hamming distance of two codewords is equal to the Hamming weight of their modulo-2 sum. Since the sum of two codewords is also a codeword for a linear code, we have:

$$d_{min} = \min_{c^i \in C} \left(w_H(c^i) \right), \ c^i \neq 0 \tag{1.13}$$

The minimal distance of a code is a parameter that determines the code correction capabilities. Indeed, the maximum numbers of errors that can be detected by a linear code with minimal distance d_{min} is $(d_{min} - 1)$. Furthermore, a maximum of $\lfloor \frac{d_{min}-1}{2} \rfloor$ errors can be corrected. In the next section, another type of powerful error correcting codes is presented.

1.2 CONVOLUTIONAL CODES

The convolutional codes are a well-known channel coding family proposed for the first time by Elias in [10]. They are widely used in current digital communication systems because of their important error correction capabilities. We present this family of codes in the rest of this section.

1.2.1 DEFINITION

Let m and n be the number of input and output bits of a convolutional code, respectively. Contrary to the block codes, the n bit convolutional encoder output $\boldsymbol{c}_k = (c_{k,1}, \cdots, c_{k,n})$ at time k does not only depend on the m bit information block $\boldsymbol{d}_k = (d_{k,1}, \cdots, d_{k,m})$, but also on the information blocks $\boldsymbol{d}_i$ for earlier times $i < k$. The convolutional encoder receives an infinite sequence of information blocks $\boldsymbol{d} = (\boldsymbol{d}_0, \boldsymbol{d}_1, \boldsymbol{d}_2 \dots)$, and produces an infinite sequence of coded blocks $\boldsymbol{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2 \dots)$. However, these sequences may in practice be finite as presented in section 1.2.5.

The convolutional encoder can be implemented using a set of flip-flops and *XOR* gates. A general representation of this encoder is presented in Figure 1.3, adapted from [11]. p flip-flops $F_1, F_2, \dots F_p$ are arranged as presented in this figure. Each one

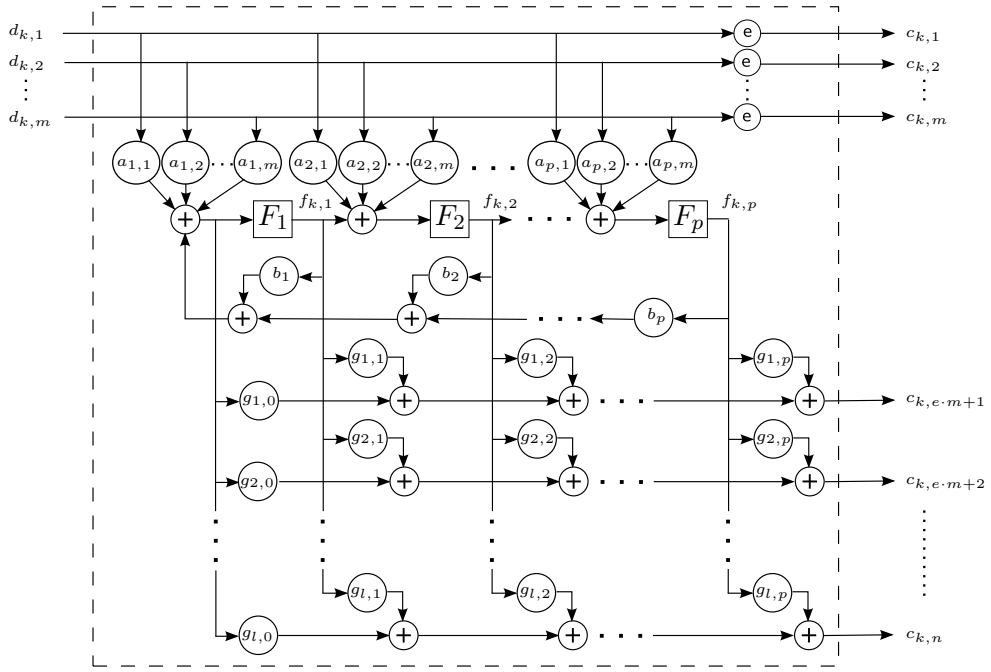

Figure 1.3: General convolutional encoder architecture.

of them stores one bit $f_{k,i} \in \{0,1\}$, that define the encoder state at time k. Let S be the set of possible states of the encoder. Therefore, the state $s_k \in S$, the encoder state at time k, can be expressed as follows:

$$s_k \equiv \sum_{i=1}^{p} f_{k,i} \cdot 2^{p-i} \tag{1.14}$$

The value $p+1$ is called the code constraint length. Since m bits are encoded at each time instant, the code is called a m-binary code. A convolutional code is therefore noted as (p, m, n).

When the encoder is fed with the information block $\boldsymbol{d}_k$, the state transition $s_k \rightarrow s_{k+1}$ occurs. The next state s_{k+1} depends on $\boldsymbol{d}_k$, on the current state s_k and on the code parameters $a_{ij}, b_i \in \{0,1\}$ for $1 \leq i \leq p$ and $1 \leq j \leq m$. Note that there is a feedback loop due to b_i. Indeed, if at least one code parameter $b_i \neq 0$, the code is called recursive. Besides, a convolutional code is systematic if each codeword can be expressed as follows:

$$\boldsymbol{c}_k = (c_{k,1} \dots c_{k,n}) = (d_{k,1}, d_{k,2} \dots d_{k,m}, c_{k,m+1}, c_{k,m+2} \dots c_{k,n}) \tag{1.15}$$

Actually, in a systematic code, the first m bits of the coded block are equal to the

information bits $\boldsymbol{d}_k$. In Figure 1.3, the parameter $e \in \{0, 1\}$ defines if the convolutional code is systematic ($e = 1$) or not. The number of coded bits is then given by $n = e{\cdot}m{+}l$ with l the number of redundant bits generated by the encoder. These l bits depend on the encoder state and the flip-flop F_1 input. The set of coefficients $g_{ij} \in \{0, 1\}$ for $1 \leq i \leq l$ and $0 \leq j \leq p$ establishes the encoder structure to generate the l redundant bits from $c_{k,e\cdot m+1}$ up to $c_{k,n}$.

1.2.2 Polynomial Representation

A convolutional encoder is a discrete Linear Time Invariant (LTI) system. Its outputs are the convolution between the input sequence and the encoder impulse response. Thus, the output $c_{k,i}$ can be expressed as follows:

$$c_{k,i} = d_{k,1} * g_i^{(1)} + d_{k,2} * g_i^{(2)} + \cdots + d_{k,m} * g_i^{(m)} = \sum_{j=1}^{m} d_{k,j} * g_i^{(j)} \tag{1.16}$$

where $*$ is the convolution operation, and $g_i^{(j)}$ is the impulse response of the system considering only the input $d_{k,j}$ and the output $c_{k,i}$, *i.e.*, the output sequence $c_{k,i}$ when $d_{k,j} = (1, 0, 0, \dots)$ and all the other inputs are set to the zero sequence $(0, 0, 0, \dots)$. Applying the $\mathcal{Z}$-transform, $\mathcal{Z}\{x_k\} = X(z) = \sum_{k=0}^{\infty} x_k z^{-k}$, to (1.16):

$$C_i(z) = \sum_{j=1}^{m} D_j(z) G_i^{(j)}(z) \tag{1.17}$$

where $C_i(z)$, $D_j(z)$ and $G_i^{(j)}(z)$ are the $\mathcal{Z}$-transform of $c_{k,i}$, $d_{k,j}$ and $g_i^{(j)}$, respectively. For convenience (1.17) can be arranged in a matrix form:

$$\boldsymbol{C}(z) = \boldsymbol{D}(z)\boldsymbol{G}(z) \tag{1.18}$$

where $\boldsymbol{D}(z) = (D_1(z), D_2(z), \dots, D_m(z))$, $\boldsymbol{C}(z) = (C_1(z), C_2(z), \dots, C_n(z))$ and $\boldsymbol{G}(z)$, called polynomial generator matrix, is the $m \times n$ matrix:

$$\boldsymbol{G}(z) = \begin{bmatrix} G_1^{(1)}(z) & G_2^{(1)}(z) & \dots & G_n^{(1)}(z) \\ G_1^{(2)}(z) & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ G_1^{(m)}(z) & \dots & \dots & G_n^{(m)}(z) \end{bmatrix} \tag{1.19}$$

In the literature, the convolutional codes are ussualy represented using the delay operator D introduced by Forney [12]. In this document, we have used the variable z^{-1} to avoid misunderstanding with the transform of the input $D(z)$. In all cases, both

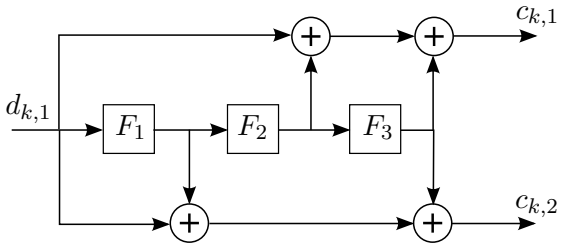

Figure 1.4: Non recursive convolutional encoder (3,1,2) with generator polynomials (13,15).

representations are equivalents and we can pass from one to the other by replacing z^{-1} by D.

1.2.2.1 NON RECURSIVE CONVOLUTIONAL CODES

Figure 1.4 depicts a binary non systematic non recursive (b_i=0) convolutional encoder with constraint length 4 and a code rate $R = 1/2$. This encoder has the parameters $(g_{1,0}, g_{1,1}, g_{1,2}, g_{1,3}) = (1, 0, 1, 1)$ and $(g_{2,0}, g_{2,1}, g_{2,2}, g_{2,3}) = (1, 1, 0, 1)$. Since there is no feedback loop, the impulse response can be is easily expressed. Indeed, it corresponds to:

$$\begin{aligned} g_1^{(1)} &= (g_{1,0}, g_{1,1}, g_{1,2}, g_{1,3}, 0, 0, \dots) = (1, 0, 1, 1, 0, 0, \dots) \\ g_2^{(1)} &= (g_{2,0}, g_{2,1}, g_{2,2}, g_{2,3}, 0, 0, \dots) = (1, 1, 0, 1, 0, 0, \dots) \end{aligned} \tag{1.20}$$

Thus, the polynomial generator matrix is:

$$\begin{aligned} \boldsymbol{G}^{NonRec}(z) &= \left[G_1^{(1),NonRec}(z) \quad G_2^{(1),NonRec}(z) \right] \\ &= \left[g_{1,0} + g_{1,1}z^{-1} + g_{1,2}z^{-2} + g_{1,3}z^{-3} \quad g_{2,0} + g_{2,1}z^{-1} + g_{2,2}z^{-2} + g_{2,3}z^{-3} \right] \\ &= \left[1 + z^{-2} + z^{-3} \quad 1 + z^{-1} + z^{-3} \right] \end{aligned} \tag{1.21}$$

The generator polynomials can be expressed in binary form. For this particular case, 1011_2 and 1101_2 are the binary representation of $G_1^{(1),NonRec}(z)$ and $G_2^{(1),NonRec}(z)$, respectively. An octal representation is more common: $G_1^{(1),NonRec} = 13_8$, $G_2^{(1),NonRec} = 15_8$, or in a more compact form (13,15).

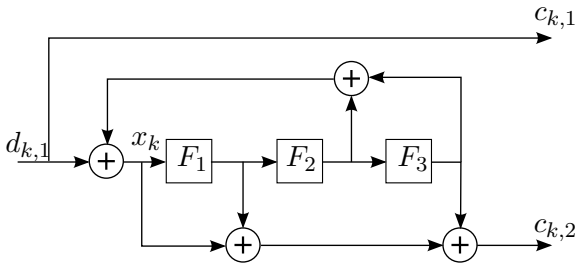

Figure 1.5: Recursive Systematic Convolutional (RSC) encoder (3,1,2) with generator polynomials $[1 \quad 1+z^{-1}+z^{-3}/1+z^{-2}+z^{-3}]$.

1.2.2.2 Recursive Systematic Convolutional Codes

The Recursive Systematic Convolutional (RSC) codes are of special interest due to the properties that the introduction of the feedback loop provides. Figure 1.5 show a binary RSC code built by applying some transformations to the non recursive code in Figure 1.4. Thus, one of the outputs of the original non recursive code is fed back to the flip-flop F_1 input, and a systematic output is added.

A RSC encoder has an infinite impulse response, and thus the definition of the generator polynomials is not straightforward. Since the encoder is systematic, the polynomial related to the systematic output $c_{k,1}$ is trivial $G_1^{(1),Rec}(z) = 1$. In order to deduce the second polynomial let us introduce the variable x (input to the filp-flop F_1). Thus we have:

$$\begin{aligned} d_{k,1} &= x_k + x_{k-2} + x_{k-3} \\ c_{k,2} &= x_k + x_{k-1} + x_{k-3} \end{aligned} \tag{1.22}$$

Accordingly, the next equation holds:

$$d_{k,1} + d_{k-1,1} + d_{k-3,1} = c_{k,2} + c_{k-2,2} + c_{k-3,2} \tag{1.23}$$

Applying the $\mathcal{Z}$-transform to (1.23) the second polynomial can be found:

$$G_2^{(1),Rec}(z) = \frac{C_2(z)}{D_1(z)} = \frac{1+z^{-1}+z^{-3}}{1+z^{-2}+z^{-3}} \tag{1.24}$$

Therefore, for this RSC code, the polynomial generator matrix is:

$$\begin{aligned} \boldsymbol{G}^{Rec}(z) &= \left[G_1^{(1),Rec}(z) \quad G_2^{(1),Rec}(z) \right] \\ &= \left[1 \quad \frac{1+z^{-1}+z^{-3}}{1+z^{-2}+z^{-3}} \right] \end{aligned} \tag{1.25}$$

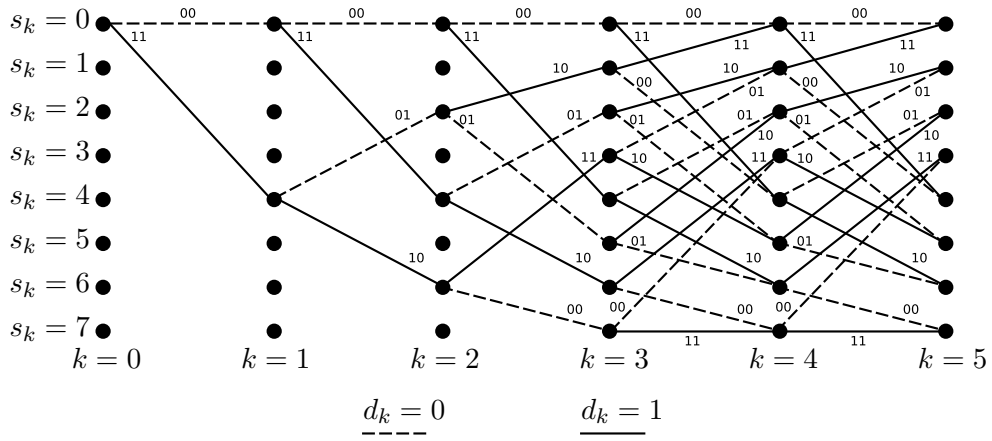

Figure 1.6: Trellis diagram for the RSC code in given in Figure 1.5.

Note that this polynomial generator matrix corresponds to the non recursive generator matrix given in (1.21) divided by $G_1^{(1),NonRec}(z)$.

1.2.3 TRELLIS DIAGRAM REPRESENTATION

Since the behavior of a convolutional encoder depends on its state s_k, given by (1.14), it is important to analyze how the encoder state evolves in function of the time. The polynomial representation, as presented in section 1.2.2, is convenient to describe the structure of an encoder. However, using it, it is difficult to clearly see the encoder state evolution. Some representation diagrams have been proposed to overcome this limitation.

In the literature, three representation diagrams exist: state diagram, tree diagram and trellis diagram. The state and tree diagrams will not be treated in this document. We refer the reader interested in details about them to [11]. The trellis diagram introduced in [13] is the most commonly scheme used to represent convolutional codes. It is especially convenient to illustrate the operation of the decoding algorithms as will be shown in sections 1.3.1 and 1.3.2.

Figure 1.6 shows the trellis diagram of the convolutional encoder detailed in Figure 1.5. The encoder states are represented by black dots • arranged in columns for each time k. Each possible state transition $s_k \rightarrow s_{k+1}$ is represented by an arc connecting two states in two columns at time instant k and $k+1$. Dashed and continuous arcs correspond to transitions due to the input symbol $d_k = 0$ and $d_k = 1$, respectively. Each arc is labeled with the corresponding encoder output. For instance, when the transition $(s_k = 0) \rightarrow (s_{k+1} = 0)$ occurs, the encoder output is $(c_{k,1}, c_{k,2}) = (d_k, c_{k,2}) = (0, 0)$.

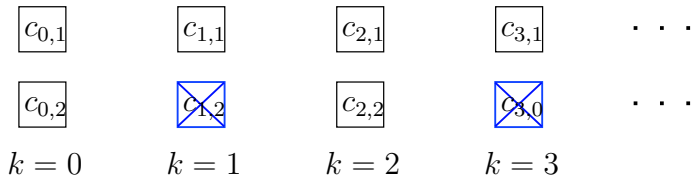

Figure 1.7: Puncturing pattern that enables to build a convolutional code with rate $R_p = 2/3$ from the mother code (3,2,1) in figure 1.5.

A path in the trellis diagram is defined as a sequence of states transitions defined by the structure of the encoder, when a particular sequence of blocks $\boldsymbol{d}$ is coded. Let us assume that the encoder state, at time $k = 0$, is $s_k = 0$. Thus, for time $k < 3$ there are some states that are not permitted. For $k \geq 3$, the encoder can have any state, and therefore any state transition in the trellis diagram may occur. In Figure 1.6, only the possible paths are shown. Note a specific property of this trellis diagram: there are always two arcs arriving to any state, corresponding to the input symbol $d_k = 0$ and $d_k = 1$. This characteristic is due to the recursive structure of the code. Furthermore, a butterfly pattern exist. Thus, each trellis section from time instant k to $k + 1$ consists of four butterfly structures.

1.2.4 PUNCTURING

In order to reduce the number of redundant bits generated by a convolutional encoder, and thus increase the system throughput, it is possible to periodically remove certain bits at the encoder output (punctured code). Naturally, this process has an impact on the error correction capabilities. Puncturing codes are convolutional codes with a code rate R_p that arise from a mother code with code rate $R = 1/n < R_p$.

Let us consider the RSC code in Figure 1.5 with the polynomial generator matrix in (1.25) and a code rate $R = 1/2$. From this mother code, we can derive a punctured code with rate $R_p = 2/3$ by removing the redundant bit $c_{k,2}$ every two input blocks as shown in Figure 1.7. The puncturing pattern is defined by a puncturing table. For this particular case the puncturing table is given in (1.26) where "1" means that the bit is transmitted and "0" that the bit is discarded. Table 1.1 shows different puncturing tables for others code rates. Note that it is also possible to puncture the systematic bit $c_{k,1}$. Thus, the puncturing table is not unique for a given punctured code rate R_p.

$$\boldsymbol{M}_p = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.26}$$

Punctured code rate R_p	$\boldsymbol{M}_p$
3/4	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$
4/5	$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
6/7	$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$

Table 1.1: Punctured table that defines different punctured code rate encoders from the mother code in figure 1.5.

1.2.5 CONVOLUTIONAL CODE TERMINATION

The convolutional codes were originally proposed to code an infinite sequence of information, were future coded blocks depend on the encoder state for previous time instants. On the other hand, the block codes were introduced to be applied over a defined quantity of information bits, generating codewords that do not depend on the previous inputed blocks. Research works were then carried out in order to establish a common framework between both classes of codes.

Let us consider a convolutional code (p, m, n) that receives L m-binary information blocks $\boldsymbol{d} = (\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{L-1})$ generating L n-binary coded blocks $\boldsymbol{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{L-1})$. We can consider this truncated convolutional code to be a block code $\mathcal{C}(n\cdot(L+v), m\cdot L)$, with $v \cdot n$ the number of bits that are generated when the frame is truncated as presented below. In this document the input blocks $\boldsymbol{d}$ are called the information frame. Consequently, the output blocks $\boldsymbol{c}$ are called the coded frame.

The operation of a truncated convolutional encoder can be described as follows. First, the encoder state is initialized to $s_0 = s^{Init}$. Then, the L information blocks of the information frame are encoded. Finally, the termination of the frame is performed in order to take the encoder to a specific and known final state. The initial and final states of the convolutional encoder provide important information to improve the decoder performance. Three alternatives have been proposed in the literature for the termination.

1.2.5.1 DIRECT TRUNCATION

This is the simplest termination method. Once the information frame is encoded, the final encoder state is not modified and thus no additional bits are produced ($v = 0$). Since the final state depends on the information frame, the decoder will not be able to establish it if noise disturbances affect the transmitted information. This lack of

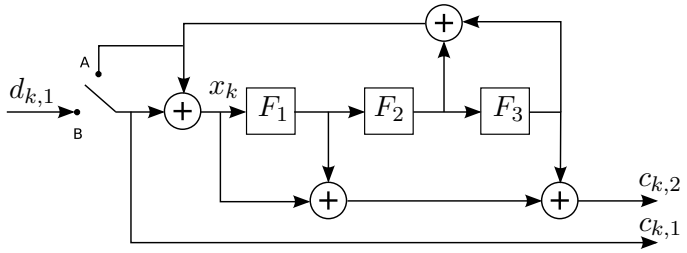

Figure 1.8: Architecture used to force the encoder state to the *all zero* state.

information has an negative impact on the decoder performance, reducing the asymptotic gain of the code.

1.2.5.2 RETURN TO THE *all zero* STATE

In this approach the initial state is set to the *all zero* state $s^{Init} = 0$. Then, at the end of the frame, the encoder is forced to go back to the same initial state by coding v additional information blocks. This approach reduces the code rate since more bits should be sent for the same amount of information bits. The term $v/(v + L)$ is called the rate loss.

For this approach $s_{L+v} = 0$. This can be done for a non recursive convolutional code by inputting $p = v$ zero blocks[1] (tail bits). However, if the code is recursive, the tail bits depend on the state s_L, and thus it is not possible to assure $s_{L+v} = 0$ using the same set of v information block for all possible information frames. In [14] a simple method was proposed to force the encoder state to the *all zero* state, by a small modification to the encoder circuit. This approach is shown in Figure 1.8 for the recursive code considered so far in Figure 1.5. The switch is in position "B" if the information frame is received. Then, it is set to position "A" to put "0" in the flip-flop F_1 input. Therefore, after $v = 3$ clock cycles, the encoder is driven to the state 0.

1.2.5.3 CIRCULAR CODES

This method does not need additional bits for the frame termination ($v = 0$). The encoder is initialized to a state $s_0 = s^c$ such that, after coding the information frame, the encoder retrieves the same state $s_L = s^c$. s^c is called the circular state. It depends on the information frame and also on the encoder polynomials. Since the frame can be seen as a circular frame, all information bits afford the same amount of error protection

[1] Remember that p corresponds to the number of flip-flops used to build the convolutional coder.

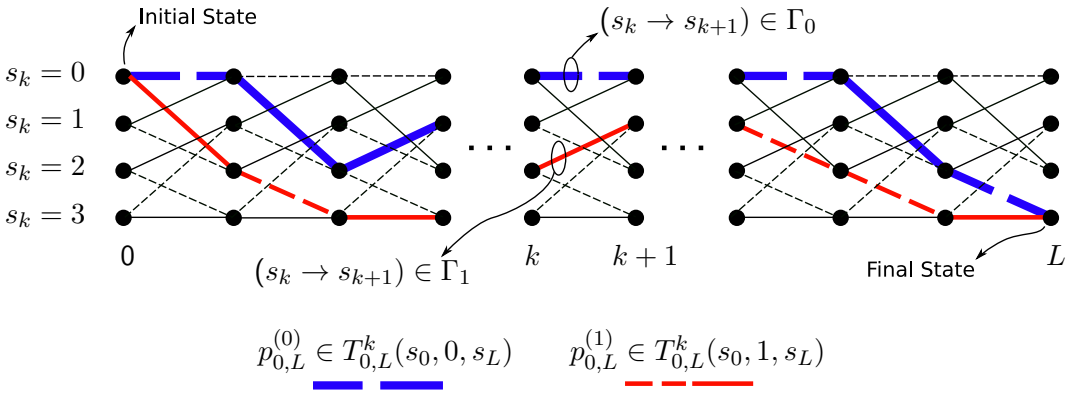

Figure 1.9: Convolutional code trellis diagram.

eliminating undesirable effects in the frame limits. The circular codes were first proposed for non recursive codes in [15], and later extended to recursive codes in [16] in the context of concatenated codes.

1.3 Decoding of Convolutional Codes

In this section we introduce the basic notation that is going to be used to describe the convolutional decoding algorithms as presented in sections 1.3.1 and 1.3.2, where we show the decoding of convolutional codes considering path probabilities in the trellis diagram.

For convenience, each information block $\boldsymbol{d}_k$ is also represented by the integer value $\delta_k \equiv \sum_{i=1}^{m} d_{k,i} \cdot 2^{m-i}$. Thus, $\delta_k \in \{0, 1, \ldots, 2^m - 1\}$. Let Γ be the set of possible state transitions (branches) $s_k \to s_{k+1}$ in the trellis diagram, and $\Gamma_\delta \subset \Gamma$ the set of branches corresponding to the information block $\delta_k = \delta$. A path $p^{(h)}_{i,j}$ in the trellis diagram is a sequence of states $s^{(h)}_i, s^{(h)}_{i+1}, \ldots s^{(h)}_j$ that takes the convolutional encoder for a particular sequence of information blocks $\delta^{(h)}_i, \delta^{(h)}_{i+1}, \ldots \delta^{(h)}_{j-1}$, such that $\left(s^{(h)}_k \to s^{(h)}_{k+1}\right) \in \Gamma_{\delta^{(h)}_k}$. Let $T^k_{i,j}(s', \delta, s)$ be the set of paths $p^{(h)}_{i,j}$ with $s^{(h)}_i = s'$ and $s^{(h)}_j = s$, such that the transition corresponding to the time k is $\left(s^{(h)}_k \to s^{(h)}_{k+1}\right) \in \Gamma_\delta$. For instance, Figure 1.9 presents two paths belonging to $T^k_{0,L}(s', 0, s)$ (thicker line) and $T^k_{0,L}(s', 1, s)$, for a convolutional code $\mathcal{C}(2, 1, n)$. Note that the number of paths in each set $T^k_{0,L}(s', \delta, s)$ grows very fast with L.

Let $\gamma_k(s', s) = p(s_{k+1} = s, r_k | s_k = s')$ be the branch metric for the transition $(s_k = s') \to (s_{k+1} = s)$, *i.e.*, the probability that the transition occurs. If no transition

between s' and s exists in the trellis diagram, $\gamma_k(s', s) = 0$. Using Bayes' relation twice we have:

$$\begin{aligned}\gamma(s', s) &= p(s_{k+1} = s, \boldsymbol{r}_k | s_k = s') \\ &= p(\boldsymbol{r}_k | s_k = s', s_{k+1} = s) p(s_{k+1} | s_k = s')\end{aligned} \tag{1.27}$$

The first term in this equation is the conditional probability of the channel (equation (1.1)). The second term corresponds to the transition $s' \rightarrow s$ given that the encoder is already in the state s'. Therefore, this term is equal to the probability that the symbol $\delta_k = \delta$ was sent, with δ the symbol that produces the transition from s' to s. Thus, (1.27) becomes:

$$\gamma(s', s) = p(\boldsymbol{r}_k | \boldsymbol{u}_k) p(\delta_k = \delta) \tag{1.28}$$

This probability can be conveniently expressed in the logarithm domain:

$$\begin{aligned}M_k^{\gamma}(s', s) &\equiv \sigma_n^2 \log(\gamma_k(s', s)) \\ &= \sigma_n^2 \log(p(\boldsymbol{r}_k | \boldsymbol{u}_k) p(\delta_k = \delta)) \\ &= \sum_{i=1}^{n} r_{ki} \cdot u_{ki} + \sigma_n^2 \log(p(\delta_k = \delta)) + A_k\end{aligned} \tag{1.29}$$

where A_k is a constant that affects all the state transitions in the same manner at time k. Thus, it can be removed from (1.29). Based on the branch metric, the probability of a path in the trellis diagram can be established. Assuming a memoryless channel, the probability of any path in the trellis diagram is:

$$p(p_{i,j}^{(h)}) = \prod_{k=i}^{j-1} \gamma\left(s_k^{(h)}, s_{k+1}^{(h)}\right) \tag{1.30}$$

This probability can also be expressed in the logarithm domain using (1.29):

$$\begin{aligned}\Theta(p_{i,j}^{(h)}) &= \log(p(p_{i,j}^{(h)})) \\ &= \frac{1}{\sigma_n^2} \sum_{k=i}^{j-1} M_k^{\gamma}\left(s_k^{(h)}, s_{k+1}^{(h)}\right)\end{aligned} \tag{1.31}$$

The convolutional decoding algorithms use the path probability to perform the decoding operation. They consider, by different approaches, the different paths that the

encoder can take. Thus, they establish the most reliable information frame $\hat{\boldsymbol{d}}$ according to the ML or MAP decoding principle as discussed in section 1.1.3. We can describe the operations of a decoding algorithm from the *mid-constraint path-partitioning problem* as presented in [17]:

> For all $0 \leq k < L$ and all $0 \leq \delta < 2^m$, the probability of the paths that belong to $T^k_{0,L}(s_0, \delta, s_L), \forall s_0, s_L$ has to be found.

Thus, the probability of each information symbol is $p(\delta_k = \delta) = \sum p(p^{(h)}_{0,L})$, with $p^{(h)}_{0,L} \in T^k_{0,L}(s_0, \delta, s_L), \forall s_0, s_L$. The decoded information symbol is then $\hat{\delta}_k = \delta'$ such that $p(\delta_k = \delta') > p(\delta_k = \delta), \forall \delta \neq \delta'$. Find the paths belonging to $T^k_{0,L}$ and their probabilities is not straightforward. When L is large, there is a huge amount of possible paths to consider, and thus, an exhaustive research is not feasible in practical systems. To overcome this constraint, the decoding algorithms are based on different approaches to reduce the number of paths. The path probabilities in only a section of the trellis diagram are partially computed. In [17] it was shown how different trellis based decoding algorithms can be formulated as path partitioning algorithms, where recursive operations are performed to find the probability of the paths belonging to different sets $T^k_{0,L}$. These recursive operations are performed in the forward (starting at time instant $k = 0$ up to $k = L$) or backward ($k = L$ towards $k = 0$) directions.

The Viterbi Algorithm (VA) [18] is an optimal ML algorithm that can be applied to the decoding of convolutional codes. It performs recursive operations only in the forward direction to compute the path metric (related to the path probability) values[2]. Regarding the MAP decoding principle, we can consider two types of decoding algorithms [13, 19]. Type-I MAP algorithms perform backward and forward recursions while Type-II are forward-only recursion algorithms. Let us consider codes applied over an information frame of L symbols. For Type-I MAP algorithms, in their basic operation without considering any memory optimization technique, the whole sequence should be received before taking any decision about the transmitted symbols. Their memory requirements grow linearly with the frame size. On the other hand, Type-II MAP algorithms take decisions after a delay, and present memory requirements that grow, for the most part of them, exponentially with the decision delay. From a high throughput decoder point of view, forward-only algorithms may be of interest due to their inherent pipelined structure [20]. For the sake of completeness in our study, we have explored the convenience of the VA, Type-I and Type-II MAP decoding algorithms in a high decoding throughput context.

[2]The VA algorithm must also perform an operation in the backward direction: traceback operation. However, it does not consist in a recursive computation of metric values.

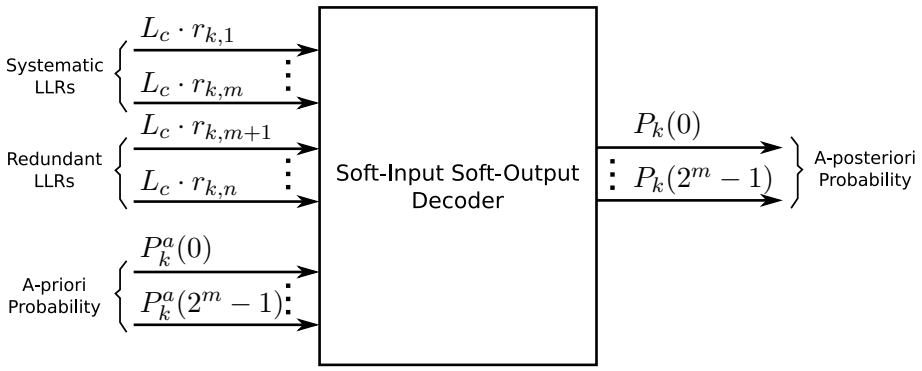

Figure 1.10: RSC SISO decoder black box diagram.

The Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [21] is the most popular Type-I MAP algorithm due to its convenience in turbo decoding (section 1.4). In [22] and [23] Type-II MAP algorithms have been proposed. These algorithms are inspired in the VA but exhibit additional complexity. The Soft-Output Algorithm (OSA) is proposed in [19]. It is another forward-only soft-output MAP algorithm where the memory requirements increase linearly with the decision delay. Unfortunately, works in [19, 22, 23] are only applicable to non recursive codes. Therefore, they are not interested in our context. In [24] and [17] SISO forward-only MAP algorithms suitable to be used in turbo decoders are presented.

We restrict our study to RSC codes. Figure 1.10 shows the black box diagram of the convolutional RSC SISO decoder that we consider. $P_k^a(\delta) = p(\delta_k = \delta)$ is the a-priori probability of the symbol $\delta \in \{0, 1, \ldots, 2^m - 1\}$ at time k. The SISO decoder receives the channel information (systematic and redundant bits) in LLR form (equation (1.11)). After the decoding algorithm, the decoder provides a-posteriori $P_k(\delta)$ values, expressed as probabilities, for the different possible information block values.

1.3.1 VITERBI BASED DECODING

1.3.1.1 OVERVIEW

The VA was proposed as a process to achieve maximum likelihood decoding of convolutional codes [18]. This algorithm finds a solution to estimate the state sequence of a finite state discrete time Markov process observed under the presence of memoryless noise [13]. The VA handles the high number of possible paths in the trellis diagram by keeping only the most probable path for each state at each time k. Let $M_k^{\alpha}(s)$ be the metric of the state s, representing a measure of the probability that the encoder is in

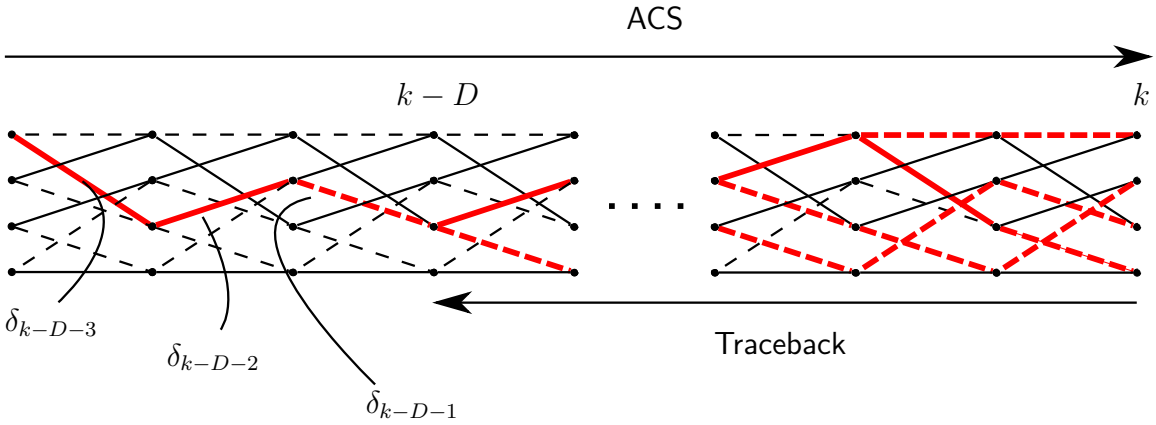

Figure 1.11: The VA operation.

the state s at time k. This metric corresponds to the logarithm of the probability of the most probable path $p_{0,k}^{(h)}$ such that $s_k^{(h)} = s$. It can be computed recursively as follows:

$$M_k^{\alpha}(s) = \max_{(s' \to s) \in \Gamma} \left(M_{k-1}^{\alpha}(s') + M_{k-1}^{\gamma}(s', s) \right) \tag{1.32}$$

Equation (1.32) is a forward recursive operation. It is performed by the so called Add Compare Select (ACS) unit. In the ACS operation, the VA finds the metrics of the paths ending at each state. It stores the largest path metric and discards the others. The path whose metric is stored, for each state, is called survivor path. The other paths are called concurrent paths. Additional to the survivor path metric, the algorithm has to memorize the information symbols for each survivor path at each time. For a sufficiently large number of transitions D in the trellis diagram, it is highly probable that all paths merge when they are rebuilt back [25]. Thus, from time k back to time $k-D$, a traceback operation is carried out by taking the information symbols of the survivors at each state. By this way, a path and the decoded symbol for time $k-D$ is found. Figure 1.11 depicts the VA operation.

The initial state metrics values $M_0^{\alpha}(s)$ depend on the knowledge of the initial convolutional encoder state. If at time $k = 0$ the state of the encoder is $s_0 = s'$, the state metrics are $M_0^{\alpha}(s') = 0$, and $M_0^{\alpha}(s) = -\infty$ for $s \neq s'$. Otherwise, if there is no knowledge about the initial conditions of the encoder, all state metrics are initialized to the same value $M_0^{\alpha}(s) = 0, \forall s$. In this case, after some iterations in the forward recursion process, the algorithm converges to a correct value for the states metrics.

1.3.1.2 THE SOFT OUTPUT VITERBI ALGORITHM

The Soft Output Viterbi Algorithm (SOVA) [26] is an improvement of the VA in order to provide, additional to the estimated information bits, reliability values for each hard decision. Let us define the path metric difference $\Delta_k(s) = \left|\Theta(p_{0,k}^{(h_0)}) - \Theta(p_{0,k}^{(h_1)})\right|$ with $s_k^{(h_0)} = s_k^{(h_1)} = s$, *i.e.*, the difference between the path metrics of the two paths that arrive to the state s at time k. Let $s_k^{(ML)}$ be the state that belong to the ML path at time k, and $p_{0,k}^{(h_1)}$ its concurrent. Therefore, $s_k^{(ML)} = s_k^{(h_1)}$. Let us assume that $\delta_k^{(ML)} = 1$ and $\delta_k^{(h_1)} = 0$. Thus, using (1.31), the probability that the decision of the survivor path is correct at time k, taking into account all the received symbols $\boldsymbol{r}_{i<k}$, is:

$$\begin{aligned} p_1 &= \frac{p(p_{0,k}^{(ML)})}{p(p_{0,k}^{(h_1)}) + p(p_{0,k}^{(ML)})} \\ &= \frac{\exp(\Theta(p_{0,k}^{(ML)}))}{\exp(\Theta(p_{0,k}^{(h_1)})) + \exp(\Theta(p_{0,k}^{(ML)}))} \\ &= \frac{\exp(\Delta_k(s_k^{(ML)}))}{1 + \exp(\Delta_k(s_k^{(ML)}))} \end{aligned} \tag{1.33}$$

Thus, the reliability value expressed as likelihood ratio is:

$$L(\delta_k) = \log\left(\frac{p(\delta_k = 1)}{p(\delta_k = 0)}\right) = \log\left(\frac{p_1}{p_0}\right) = \log\left(\frac{p_1}{1 - p_1}\right) = \Delta_k(s_k^{ML}) \tag{1.34}$$

Therefore, the path metric difference gives the reliability value of the decoded information symbols at time k. However, this reliability value does not take into account the received symbols after time k. The SOVA tries to solve this problem in a rather intuitive way. After the computation of each path difference, once the optimal path is known, two methods are proposed to find soft values: Battail [27] and Hagenauer-Hoeher [26] methods. The first one is more complex compared to the second one. This additional complexity does not provide significative improvements in terms of BER.

Figure 1.12 depicts different paths through the trellis diagram. The ML path, $p_{0,L}^{(ML)}$, merges with the paths $p_{0,L}^{(h_1)}$ and $p_{0,L}^{(h_2)}$ at time $k - j + 1$ and k, respectively. Thus, $\delta_{k-j}^{(ML)} \neq \delta_{k-j}^{(h_1)}$ and $\delta_{k-j}^{(h_2)} \neq \delta_{k-j}^{(h_3)}$. Note that it does not impose any relation between $\delta_k^{(h_2)}$ and $\delta_{k-j}^{(h_3)}$ nor between $\delta_{k-j}^{(ML)}$ and $\delta_{k-j}^{(h_3)}$. Let us suppose that at time $k - j$ the symbol chosen by the ML path and by the path $p_{0,L}^{(h_2)}$ are different. Let us also assume

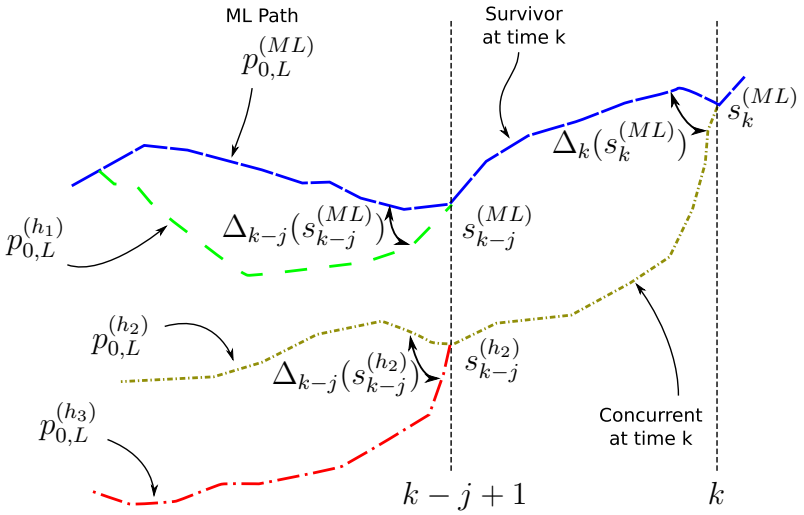

Figure 1.12: SOVA soft values update process.

that, at the same time $k-j$, the soft value of the ML path is much larger that the soft value of the path $p_{0,L}^{(h_2)}$. We have then $\Delta_{k-j}(s_{k-j}^{(ML)}) \gg \Delta_{k-j}(s_{k-j}^{(h_2)})$. If for instance $\Delta_k(s_k^{(ML)}) \to 0$, the probability of choosing the ML path at time k is *low*, and thus, the VA could have chosen the path $p_{0,L}^{(h_2)}$ instead. This change in the decision would produce a completely different decoded bit and reliability value at time $k-j$. This example illustrates the need to appropriately update the soft values in order to avoid reliability overestimations. The update process of soft values is described as follows, where D and U are defined as the traceback and update length, respectively.

Let $p_{0,k}^{(ML_c)}$ be the concurrent path of the ML path at time k. Thus, $s_k^{(ML)} = s_k^{(ML_c)}$ and $\delta_k^{(ML)} \neq \delta_k^{(ML_c)}$. The update process is performed from time $k-D$ to time $k-D-U$. We denote by $L'(\hat{\delta}_k)$ the temporary reliability value for the systematic bit estimation at time k. Initially, $|L'(\hat{\delta}_k)| = \Delta_k(s_k^{(ML)})$ and, when the update process ends, $L(\hat{\delta}_k) = L'(\hat{\delta}_k)$. The algorithm 1 presents the Battail update process. In this algorithm, the reliability value actualization when $\delta_j^{(ML_c)} = \delta_j^{(ML)}$ does not improve significantly the algorithm performance, and thus it can be removed [28]. In this case, the update process corresponds to the Hagenauer-Hoeher approach.

1.3.1.3 SOVA FOR NON-BINARY CONVOLUTIONAL CODES

The concepts introduced so far present the SOVA decoding process for binary codes where each information symbol is equal to one bit. Some works have also been carried

Algorithm 1 - Update process for soft information for the binary SOVA

for each time k **do**
  for all $j \in [k-D-U, k-D]$, **do**
    if $\delta_j^{(ML_c)} \neq \delta_j^{(ML)}$ **then**
      $L'(\hat{\delta}_j) \leftarrow \min\left(L'(\hat{\delta}_j), \Delta_k(s_k^{(ML)})\right)$
    else
      $L'(\hat{\delta}_j) \leftarrow \min\left(L'(\hat{\delta}_j), \Delta_k(s_k^{(ML)}) + \Delta_j(s_j^{(ML_c)}\right)$
    end if
  end for
end for

out in order to extend the SOVA to double binary codes. In [29] an extension of the bidirectional SOVA algorithm [30] is proposed. This algorithm is a MAP like algorithm with forward and backward recursions. The algorithm complexity is claimed to be 1.5 times the VA complexity. In [31] and [32] a transformation of the MAP algorithm is presented with the aid of the path metric values calculated as in the VA. In this case, the algorithm complexity is slightly lower than the MAP algorithm complexity. Thus, no significant advantages exist. The work detailed in [33] presents a windowing SOVA like decoding algorithm where reliability values are defined as the difference of path metric values. For the time k, these paths are not the paths of each state at time k, but those that are connected to the ML state some trellis diagram sections later (at time $k+D$). This algorithm maximizes symbol probability and not the path metric probability. The algorithm complexity is slightly lower than the for simplified MAP like algorithms with similar performance. In [34] a direct extension of the original SOVA in [26] is detailed.

Let us consider a double binary convolutional code. Therefore, there are four arcs in the trellis diagram from time k to time $k+1$ for each state. Consider now the information symbol $\delta_k = \delta$. The path metric difference related to this information symbol is defined as:

$$\Delta_k^{(\delta)}(s) = \max\left(\Theta(p_{0,k}^{(h)})\right) - \Theta(p_{0,k}^{(i)}) \tag{1.35}$$

where the path $p_{0,k}^{(i)}$ is such that $\delta_k^{(i)} = \delta$, *i.e*, this path ends in the state $s_k^{(i)}$ with the information symbol δ, and $s_k^{(h)} = s_k^{(i)} = s$ for all the paths $p_{0,k}^{(h)}$ ending in the state s. Thus, the soft output value expressed as log-likelihood ratio with respect to the symbol $\delta_k = 0$, considering the state $s_k^{(ML)}$ that belongs to the ML path, is:

$$L(\delta_k = \delta) = \log\left(\frac{p(\delta_k = \delta)}{p(\delta_k = 0)}\right) = \Delta_k^{(0)}(s_k^{(ML)}) - \Delta_k^{(\delta)}(s_k^{(ML)}) \tag{1.36}$$

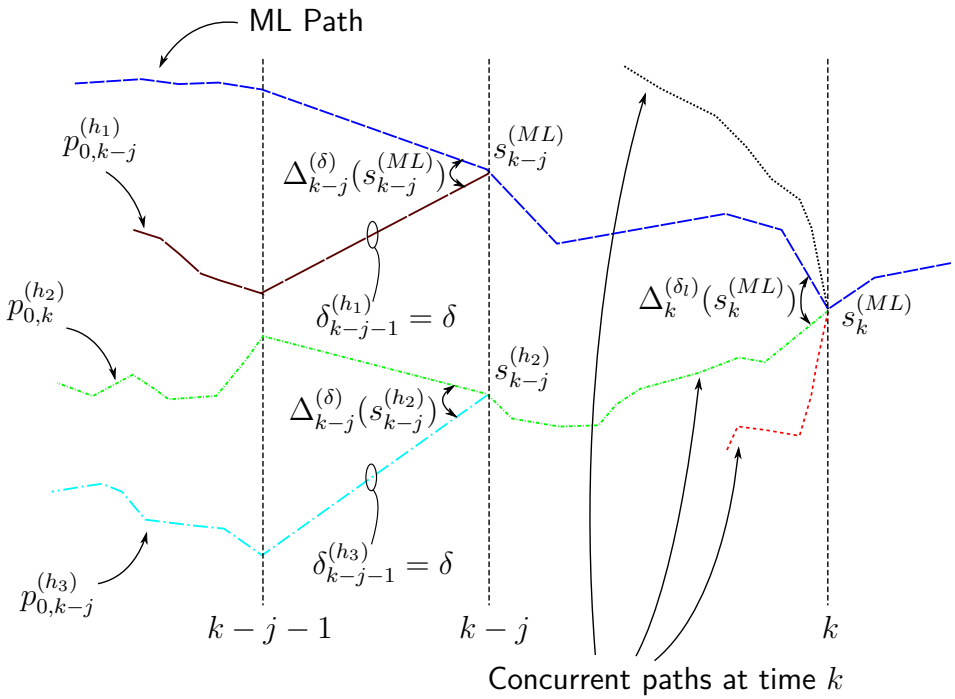

Figure 1.13: SOVA soft values update process for double binary codes.

Figure 1.13 depicts different paths in a trellis diagram for a double binary convolutional code. The ML path at time $k-j$ has the concurrent path $p_{0,k-j}^{(h_1)}$ such that its information symbol at time $k-j-1$ is $\delta_{k-j-1}^{(h_1)} = \delta$. The initial reliability value for the symbol δ at time $k-j-1$ can then be computed as presented in (1.35): $L'(\delta_{k-j-1} = \delta) = \Delta_{k-j}^{(\delta)}(s_{k-j}^{(ML)})$. If the path $p_{0,k}^{(h_2)}$ is concurrent to the ML path at time k, this path has three concurrent paths at time $k-j$. One of them is $p_{0,k-j}^{(h_3)}$ that corresponds to the state $s_{k-j}^{(h_2)}$ when the information symbol is $\delta_{k-j-1}^{(h_3)} = \delta$. Thus, the reliability for the information symbol δ considering the state $s_{k-j}^{(h_2)}$ is $\Delta_{k-j}^{(\delta)}(s_{k-j}^{(h_2)})$. The initial reliability is then updated as follows:

$$L'(\delta_{k-j-1} = \delta) = \min\left(L'(\delta_{k-j-1} = \delta), \Delta_{k-j}^{(\delta)}(s_{k-j}^{(h_2)}) + \Delta_k^{(\delta_l)}(s_k^{(ML)})\right) \tag{1.37}$$

where δ_l is the information symbol of the concurrent path $p_{0,k}^{(h_2)}$ at time k. Equation (1.37) should also be applied to the two other concurrent paths at time k. Since $\Delta_k^{\delta_k^{(ML)}} = 0$, we can write this equation in a general form as follows:

$$L'(\delta_{k-j-1} = \delta) = \min_{\forall \delta_l} \left(\Delta_{k-j}^{(\delta)}(s^{\delta_l}) + \Delta_k^{(\delta_l)}(s_k^{(ML)}) \right) \tag{1.38}$$

where s^{δ_l} is the state of the concurrent path at time $k-j$ that corresponds to the information symbol $\delta_k = \delta_l$. Let us consider a binary code. Without lost of generality the path with all the information symbols equal to 0 is taken to be the ML path. In this case (1.38) can be written as:

$$L'(\delta_{k-j-1} = 1) = \min(\Delta_{k-j}^{(1)}(s_k^{(ML)}), \Delta_k^{(1)}(s_k^{(ML)}) + \Delta_{k-j}^{(1)}(s^1)) \tag{1.39}$$

Thus, if the decision taken by the concurrent path at time $k-j$ is 1 then $\Delta_{k-j}^{(1)}(s^1) = 0$ and therefore (1.39) is reduced to $\min(\Delta_{k-j}^{(1)}(s_k^{(ML)}), \Delta_k^{(1)}(s_k^{(ML)}))$. This is exactly the same algorithm 1 that describes the actualization process for binary codes. Thus, coherent results exist between the update processes for binary and double binary convolutional codes.

1.3.1.4 SOVA Scheduling

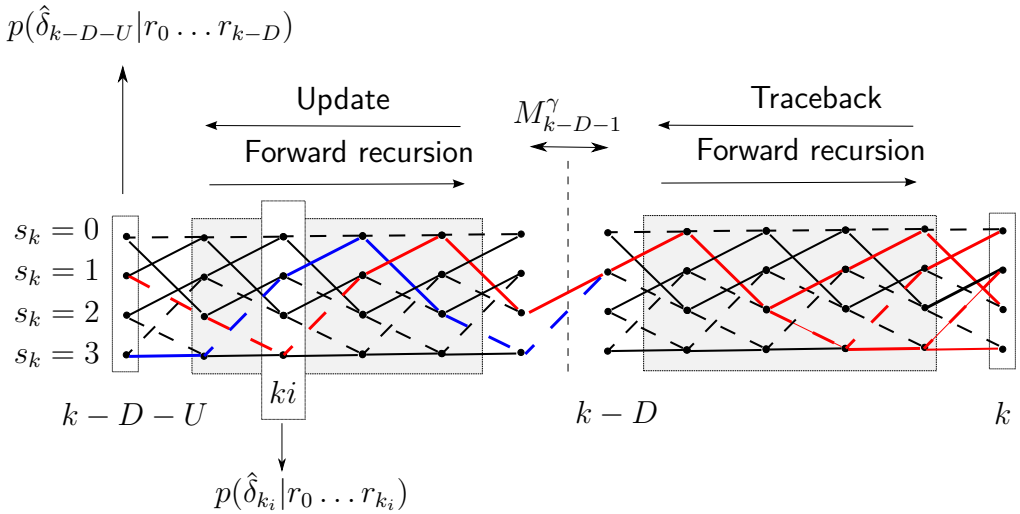

Figure 1.14: SOVA operations.

The diagram in Figure 1.14 depicts the operation of the SOVA on a binary code in a similar way than the diagram in Figure 1.11 for the VA. The forward recursion is performed in order to compute M_k^{α} and the initial reliability values (equations (1.32) and (1.35) respectively). The traceback operation is performed from time k back to time

$k-D$. Then, the survivor and the concurrent paths are found from time $k-D$ back to time $k-D-U$. Afterwards, the update process is executed as described by the algorithm 1. We adopt the graphical representation introduced in [35] in order to illustrate the SOVA operations as presented in Figure 1.15. The horizontal axis represents the time and the vertical axis the symbols in the frame.

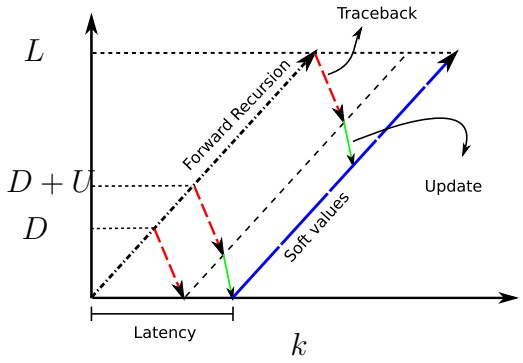

Figure 1.15: Graphical representation of the operations carried out by the SOVA.

1.3.2 MAP Based Decoding

In this section, two types of MAP based decoding algorithms are presented. Section 1.3.2.1 details a well known Type-I MAP algorithm. In section 1.3.2.2, two versions of this algorithm, suited for hardware implementations, are described. Afterwards, a Type-II MAP algorithm is considered in section 1.3.2.3.

1.3.2.1 BCJR Algorithm

The BCJR algorithm [21] is an optimal solution for the decoding of convolutional codes. It provides, besides to the hard decisions, soft values associated to each decoded bit. Thus, no additional manipulation, as done for example in the VA to obtain the SOVA, are required to build a SISO decoder. Moreover, since the BCJR is a MAP algorithm, once the L symbols of one frame are decoded, the path that is reconstructed not necessarily corresponds to a valid path in the trellis diagram.

The BCJR algorithm computes 2^m A-posteriori probability (APP) values corresponding to each possible information symbol. These probabilities are expressed as follows:

$$P_k(\delta_k = \delta|\boldsymbol{r}) = \frac{p(\delta_k = \delta, \boldsymbol{r})}{p(\boldsymbol{r})} = \frac{p(\delta_k = \delta, \boldsymbol{r})}{\sum_{\delta'=0}^{2^m-1} p(\delta_k = \delta', \boldsymbol{r})} \tag{1.40}$$

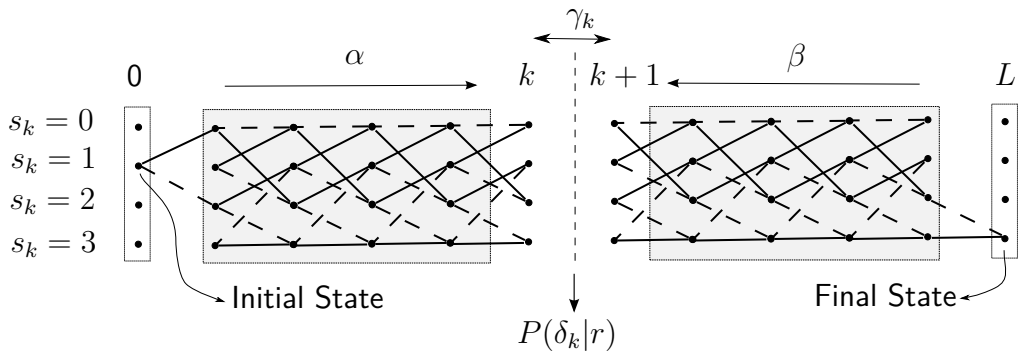

Figure 1.16: BCJR algorithm operations.

where $\delta \in \{0, 1, \dots 2^m - 1\}$ is one of the possible information symbols. The denominator in (1.40) is just a normalization factor, since it equally affects all the possible a-posteriori probabilities. Thus, it can be removed from this equation. Therefore, in practice, only the join probability $p(\delta_k = \delta, \boldsymbol{r})$ is computed. This probability can be calculated by considering all the possible transitions in the trellis diagram from time k to time $k+1$ that are related to the information symbol $\delta_k = \delta$:

$$p(\delta_k = \delta, \boldsymbol{r}) = \sum_{(s \to s') \in \Gamma_\delta} p(s_k = s, s_{k+1} = s', \boldsymbol{r}) \tag{1.41}$$

Let us consider the Figure 1.16. It represents the BCJR algorithm operation over the corresponding convolutional encoder trellis diagram. Since we consider a memoryless channel, the join probability in (1.41) can be expressed from three independent terms:

- Forward recursion $\alpha_k(s) = p(s_k = s, r_{0 \le j < k})$: It corresponds to the probability that the convolutional encoder is in the state s at time k, considering only the channel information from time 0 up to $k-1$.

- Transition probability $\gamma_k(s, s') = p(s_{k+1} = s', r_k | s_k = s)$: This term is the probability that the transition $(s \to s') \in \Gamma_\delta$ occurs at time k.

- Backward recursion $\beta_{k+1}(s') = p(r_{j>k} | s_{k+1} = s')$: It equals the probability that the convolutional encoder is in the state s' at time $k+1$, considering only the channel information received after the time k.

Thus, (1.41) can be expressed as follows:

$$\begin{aligned} p(\delta_k = \delta, \boldsymbol{r}) &= \sum_{(s \to s') \in \Gamma_\delta} p(s_k = s, s_{k+1} = s', \boldsymbol{r}) \\ &= \sum_{(s \to s') \in \Gamma_\delta} \alpha_k(s) \gamma_k(s, s') \beta_{k+1}(s') \\ &= \sum_{(s \to s') \in \Gamma_\delta} p(s_k = s, r_{j<k}) p(s_{k+1} = s', r_k | s_k = s) p(r_{j>k} | s_{k+1} = s') \end{aligned} \tag{1.42}$$

With this equation the APPs in (1.40) can be expressed as follows:

$$P(\delta_k = \delta | r) = \frac{\sum_{(s \to s') \in \Gamma_\delta} \alpha_k(s) \gamma_k(s, s') \beta_{k+1}(s')}{\sum_{(s \to s') \in \Gamma} \alpha_k(s) \gamma_k(s, s') \beta_{k+1}(s')} \tag{1.43}$$

$\alpha_k(s)$ and $\beta_k(s')$ are also referred as state metrics. These metrics are computed by means of the following recursive processes:

$$\alpha_k(s) = \sum_{(s' \to s) \in \Gamma} \alpha_{k-1}(s') \gamma_{k-1}(s', s) \tag{1.44}$$

$$\beta_k(s') = \sum_{(s' \to s) \in \Gamma} \beta_{k+1}(s) \gamma_k(s', s) \tag{1.45}$$

Regarding the transition probability $\gamma_k(s, s')$, note that it corresponds to the branch metric for the transition $(s \to s')$ already defined in (1.27) and (1.28). For convenience, we write an explicit expression for this transition probability by using the conditional probability of the channel in (1.1):

$$\begin{aligned} \gamma_k(s, s') &= p(\boldsymbol{r}_k | \boldsymbol{u}_k) p(\delta_k = \delta) \\ &= p(\delta_k = \delta) \prod_{i=1}^{n} \left(\frac{1}{\sigma_n \sqrt{2\pi}} \exp\left(-\frac{(r_{ki} - u_{ki})^2}{2\sigma_n^2} \right) \right) \end{aligned} \tag{1.46}$$

Developing this expression, and using the channel reliability definition, the next equation holds:

$$\gamma_k(s, s') = B_k \cdot p(\delta_k = \delta) \cdot \exp\left(\frac{1}{2} L_c \sum_{i=1}^{n} r_{ki} \cdot u_{ki} \right) \tag{1.47}$$

where B_k is a constant. The reliability computed by the BCJR algorithm becomes:

$$L(\delta_k = \delta) = \log\left(\frac{p(\delta_k = \delta, \boldsymbol{r})}{p(\delta_k = 0, \boldsymbol{r})} \right) = \log\left(\frac{\sum_{(s \to s') \in \Gamma_\delta} \alpha_k(s) \gamma_k(s, s') \beta_{k+1}(s')}{\sum_{(s \to s') \in \Gamma_0} \alpha_k(s) \gamma_k(s, s') \beta_{k+1}(s')} \right) \tag{1.48}$$

where the symbol 0 is taken as reference. Note that the BCJR algorithm equations require the noise variance. Thus, channel estimation is required, what may be inconvenient in practical communication systems. Indeed, this is a disadvantage for this algorithm that is overcome with suboptimal algorithms as presented in section 1.3.2.2.

MAP algorithm recursion initialization: The initial conditions for the forward and backward recursions, α_0 and β_L, depend on the knowledge of the encoder initial and final states. If for example the encoder state at time $k = 0$ is s_0, then $\alpha_0(s_0) = 1$ and $\alpha_0(s) = 0$ for $s \neq s_0$. If the initial state is unknown, $\alpha_0(s) = 1/2^p, \forall s$, *i.e.*, all the states have the same probability. A similar criterion is applied to the initial values for the backward recursion at time L, $\beta_L(s)$. Note that initial and final state are established by the termination method chosen as presented in section 1.2.5.

1.3.2.2 LOG-MAP AND MAX-LOG-MAP ALGORITHMS

From the implementation point of view, the BCJR algorithm is quite complex due to the large number of multiplications and exponentiations that should be performed. For this reason, a logarithmic version of the algorithm is generally preferred where multiplications become additions and exponentiations are eliminated.

Let us express the forward recursion and backward recursion in the logarithm domain as follows:

$$M_k^{\alpha}(s) \equiv \sigma_n^2 \log(\alpha_k(s)) \tag{1.49}$$

$$M_k^{\beta}(s) \equiv \sigma_n^2 \log(\beta_k(s)) \tag{1.50}$$

Let us also define the soft values at the input and output of the decoder in the logarithm domain. Thus, the a-posteriori log-likelihood and the a-priori log-likelihood are respectively $L_k(\delta) = \frac{\sigma_n^2}{2} \log P(\delta_k = \delta|\boldsymbol{r})$ and $L_k^a(\delta) = \frac{\sigma_n^2}{2} \log p(\delta_k = \delta)$. The transition probability in the logarithm domain was already presented in (1.29). For convenience, we rewrite here this expression using the a-priori log-likelihood definition:

$$M_k^{\gamma}(s, s') = \sum_{i=1}^{n} r_{ki} \cdot u_{ki} + 2L_k^a(\delta) + A_k \tag{1.51}$$

We recall that the constant A_k can be eliminated since it affects all the state transitions in the same manner. Thus, we can expand (1.49) as follows:

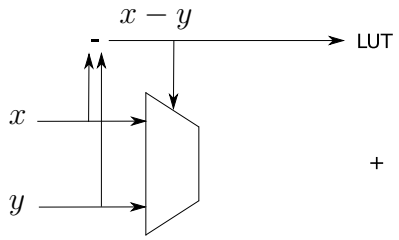

Figure 1.17: Architecture for the implementation of the function $\max^*(x, y)$.

$$\begin{aligned} M_k^{\alpha}(s) &= \sigma_n^2 \log\left(\sum_{(s'\to s)\in\Gamma} \alpha_{k-1}(s')\gamma_{k-1}(s', s)\right) \\ &= \sigma_n^2 \log\left(\sum_{(s'\to s)\in\Gamma} \exp\left(\frac{1}{\sigma_n^2} M_{k-1}^{\alpha}(s')\right) \exp\left(\frac{1}{\sigma_n^2} M_{k-1}^{\gamma}(s', s)\right)\right) \\ &= \sigma_n^2 \log\left(\sum_{(s'\to s)\in\Gamma} \exp\left(\frac{1}{\sigma_n^2}(M_{k-1}^{\alpha}(s') + M_{k-1}^{\gamma}(s', s))\right)\right) \end{aligned} \tag{1.52}$$

The backward recursion $M_k^{\beta}(s)$ can be expressed similarly. Note that the summation in (1.52) is of the form: $\log(\exp(x) + \exp(y))$, and thus it can be computed as follows:

$$\begin{aligned} \log(\exp(x) + \exp(y)) &= \max(x, y) + \log(1 + \exp(-|y - x|)) \\ &= \max^*(x, y) \end{aligned} \tag{1.53}$$

for $x, y \in \mathbb{R}$. The function $\max^*(x, y)$ can be implemented using the architecture presented in Figure 1.17, where a Look Up Table (LUT) is used to compute the function $\log(1 + \exp(-|y - x|))$. Thanks to the associative property of this function, $\max^*(x, y, z) = \max^*(\max^*(x, y), z)$, (1.53) can be easily extended to more that two operands. Thus, by replacing (1.53) in (1.52) we obtain:

$$M_k^{\alpha}(s) = \sigma_n^2 \max^*_{(s'\to s)\in\Gamma}\left(\frac{1}{\sigma_n^2}\left(M_{k-1}^{\alpha}(s') + M_{k-1}^{\gamma}(s', s)\right)\right) \tag{1.54}$$

A similar expression for the backward recursion can also be established:

$$M_k^{\beta}(s') = \sigma_n^2 \max^*_{(s'\to s)\in\Gamma}\left(\frac{1}{\sigma_n^2}\left(M_{k+1}^{\beta}(s) + M_k^{\gamma}(s', s)\right)\right) \tag{1.55}$$

These two last equations represent the recursive operations of the BCJR algorithm in the logarithm domain. They are analogous to (1.44) and (1.45). Now, let us consider the logarithm of the joint probability in (1.42):

$$\begin{aligned}\lambda_k(\delta) &= \sigma_n^2 \log(p(\delta_k = \delta, r)) = \sigma_n^2 \log\left(\sum_{(s\to s')\in\Gamma_\delta} \alpha_k(s)\gamma_k(s,s')\beta_{k+1}(s')\right) \\ &= \sigma_n^2 \max^*_{(s'\to s)\in\Gamma_\delta}\left(\frac{1}{\sigma_n^2}(M_k^\alpha(s) + M_k^\gamma(s,s') + M_{k+1}^\beta(s'))\right) \qquad (1.56)\end{aligned}$$

Finally, the soft output information, expressed in log likelihood value and normalized to the symbol 0 is given by the next equation:

$$L_k(\delta) = \lambda_k(\delta) - \lambda_k(0) \qquad (1.57)$$

The decoded symbol (hard output) is such that it maximizes $\lambda_k(\delta)$ in (1.56):

$$\hat{d}_k = \underset{\delta}{\operatorname{argmax}}\left(\lambda_k(\delta)\right) \qquad (1.58)$$

The algorithm described by the equations (1.51)-(1.58) is the Log-MAP algorithm. If an exact computation of the function $\max^*$ is carried out, *i.e*, if the LUT in Figure 1.17 is implemented without any approximation, the result of the BCJR and the Log-MAP algorithms is the same. However, a high precision for the LUT is usually not required. The works in [36] and [37] propose a two output and four output LUT architecture respectively. It is observed that the degradation introduce by these architectures can be acceptable.

On the other hand, in order to reduce the complexity of convolutional decoders, the LUT can be completely removed. Thus, $\max^*(x, y)$ is replaced by finding the maximum between x and y. The resulting algorithm is called the Max-Log-MAP algorithm. This sub-optimal algorithm is typically used in the most part of practical implementations since its performance is acceptable compared to that of the Log-MAP algorithm. When iterative decoders are considered (see section 1.4), degradation of about 0.5 dB are observed for the Max-Log-MAP algorithm with respect to the BCJR algorithm for binary codes. When double binary convolutional codes are considered, a degradation below 0.1 dB for the Max-Log-MAP algorithm appears [38].

Note that, when $\max^*(x, y)$ is replaced by $\max$ in all the Log-MAP equations, the term σ_n disappears. This is an important advantage of the Max-Log-MAP algorithm. Actually, it does not require to know the noise variance. Therefore, no channel estimators are necessary.

The suboptimal behavior of the Max-Log-MAP algorithm is due to two factors [39]: First of all, the simplification of the $\max^*(x, y)$ function leads to information lost impossible to recover in later stages of the algorithm. Secondly, values in the decoding process are supposed to be probabilities representation. However, in the suboptimal algorithm this probability representation in not longer valid. In the context of iterative decoding systems, the soft output values produced by the Max-Log-MAP algorithm are scaled by coefficients smaller or equal than 1.0, as proposed for instance in [39–41]. In practical systems, the scaling factors are fixed to 0.5, 0.75 or 1.0. Indeed, they can be easily implemented using a binary shift and addition operations.

Log-MAP and Max-Log-MAP recursion initialization: The initial values for the state metrics $M_0^{\alpha}(s)$ and $M_M^{\beta}(s)$ are given by the logarithms of the initial values of the forward and backward recursion of the BCJR algorithm. If there is no knowledge of the initial and final state of the encoder, $M_0^{\alpha}(s) = M_L^{\beta}(s) = 0, \forall s$. On the other hand, if the initial and final conditions are known: $M_0^{\alpha}(s_0) = M_L^{\beta}(s_L) = 0$, $M_0^{\alpha}(s) = -\infty$ for $s \neq s_0$, and $M_0^{\beta}(s) = -\infty$ for $s \neq s_L$, where s_0 and s_L are the initial and finale convolutional encoder states, respectively.

1.3.2.3 FOMAP Algorithms

In [17] the Forward Only MAP (FOMAP) has been proposed. This algorithm can be used as a particular example of other Type-II MAP algorithms. Consider Figure 1.18 which depicts the basic operations of the FOMAP algorithm. This algorithm, similarly to the Viterbi algorithm, stores a set of values for each state at each time, that are updated later in consecutive steps of the algorithm. The FOMAP algorithm is executed by performing three operations: *Extend*, *Update* and *Collect*. The *Extend* operation carries out a forward recursion in order to process the new received symbol. During the second operation, the values inside a window of length D_{FO} are updated. Finally, the *Collect* operation computes soft values and take the hard decision for the symbols at the beginning of a window.

Let $\hat{\alpha}_k(s) = p(s_k = s | r_{0 \leq j < k})$ be the forward recursion state metrics at time k, and let $\hat{\alpha}_{k,i}(s, \delta) = p(s_k = s, \delta_i = \delta | r_{0 \leq j < k})$ be the sum of the probability of the paths $p_{0,k}^{(h)}$ such that $\delta_i^{(h)} = \delta$ and $s_k^{(h)} = s$, *i.e*, the probability that the encoder state is s at time k and that the information symbol is δ at time i. Let us consider the branches in the trellis diagram due to the information symbol $\delta_{k-D_{FO}} = \delta$ (thicker lines in Figure 1.18). For a window size D_{FO} large enough we can apply the following approximation:

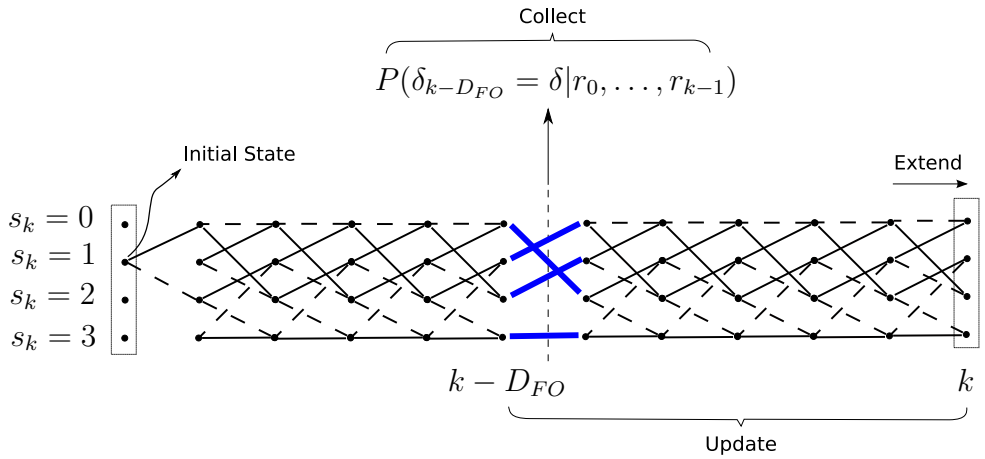

Figure 1.18: FOMAP algorithm operation.

$$\hat{\alpha}_{k,k-D_{FO}}(s,\delta) = p(s_k = s, \delta_{k-D_{FO}} = \delta | \boldsymbol{r}_{0\leq j<k}) \approx p(s_k = s, \delta_{k-D_{FO}} = \delta | \boldsymbol{r}) \quad (1.59)$$

Therefore, considering $\hat{\alpha}_{k,k-D_{FO}}(s,\delta)\ \forall s$, the probability $p(\delta_{k-D_{FO}} = \delta|r)$ can be computed. The three operations that compose the FOMAP algorithm are detailed as follows.

Extend Assume that the state metrics $\hat{\alpha}_{k-1,i}(s,\delta)$ are known for time $i \in [k-D_{FO}, k-2]$. The extend operation finds the probability $\hat{\alpha}_{k-1}(s)$ as the sum of the probabilities of all the paths associated to the state s at time $k-1$:

$$\begin{aligned} \hat{\alpha}_{k-1}(s) &= \sum_{\delta'=0}^{2^m-1} p(s_{k-1} = s, \delta_{k-2} = \delta' | r_{0\leq j<k-1}) \\ &= \sum_{\delta'=0}^{2^m-1} \hat{\alpha}_{k-1,k-2}(s,\delta') \end{aligned} \quad (1.60)$$

Equation (1.60) is now used to find the probability of being in the state s at time k, when $\delta_{k-1} = \delta$. Let the transition $(s' \to s) \in \Gamma_\delta$, we have:

$$\begin{aligned} \hat{\alpha}_{k,k-1}(s,\delta) &= p(s_k = s, \delta_{k-1} = \delta | r_{0\leq j<k}) \\ &= p(s_{k-1} = s' | r_0 \ldots r_{0\leq j<k-1}) p(s_k = s, r_{k-1} | s_{k-1} = s') \\ &= \hat{\alpha}_{k-1}(s') \gamma_{k-1}(s', s) \end{aligned} \quad (1.61)$$

In (1.61) it is assumed that only one path can be associated to the state $s_k = s$ with $\delta_{k-1} = \delta$, as is the case for RSC codes.

Update In this operation, the values $\alpha_{k,i}(s,\delta)$ are updated in the window $k - D_{FO} \leq i \leq k-2$ to include the contribution of the new received symbol r_{k-1}:

$$\begin{aligned}\hat{\alpha}_{k,i}(s,\delta) &= p(s_k = s, d_i = \delta | \boldsymbol{r}_0 \dots \boldsymbol{r}_{k-1}) \\ &= \sum_{s' \to s} p(s_{k-1} = s', d_i = \delta | \boldsymbol{r}_0 \dots \boldsymbol{r}_{k-2}) p(s_k = s, \boldsymbol{r}_{k-1} | s_{k-1} = s') \\ &= \sum_{s' \to s} \hat{\alpha}_{k-1,i}(s',\delta)\gamma_{k-1}(s',s) \end{aligned} \tag{1.62}$$

Collect The *collect* operation enables to compute the soft values and to find the hard decision. By taking the contribution of the probabilities of all the paths connected to all the states at time k, the APP of the information symbol $\delta_{k-D_{FO}}$ is found:

$$p(\delta_{k-D_{FO}} = \delta | r_{0 \leq j < k}) = \sum_{s=0}^{2^p-1} \alpha_{k,k-D_{FO}}(s,\delta) \tag{1.63}$$

The hard decision corresponds to find the symbol δ that maximizes (1.63):

$$\hat{\delta}_k = \underset{\delta}{\operatorname{argmax}} \left(p(\delta_{k-D_{FO}} = \delta | r_{0 \leq j < k}) \right) \tag{1.64}$$

A characteristic of the FOMAP algorithm is its capacity to execute the update operation in parallel for all the values in the window of length D_{FO}, contrary to the SOVA, which carries out a sequential update process thanks to a traceback operation. The updated value, $\hat{\alpha}_{k,i}(s,\delta)$, depends only on the value that has not yet been updated, $\hat{\alpha}_{k-1,i}(s',\delta)$, and on the branch metric $\gamma_{k-1}(s',s)$. No dependence exists between values inside the window, for example between $\hat{\alpha}_{k,i}(\cdot,\cdot)$ and $\hat{\alpha}_{k,j}(\cdot,\cdot)$ for $i \neq j$.

1.3.2.4 MAP Based Algorithm Schedules

Depending on the execution order of the equations (1.44), (1.45) and (1.48), two straightforward schedules for the BCJR algorithm are possible as presented in Figure 1.19. In this Figure, continuous lines symbolize α or β computation, and dashed lines state metrics along with a-posteriori information computation. Since to compute the a-posteriori information, both forward state metric α and backward state metric β are required, a memory is necessary. The dashed area represents the time interval when the

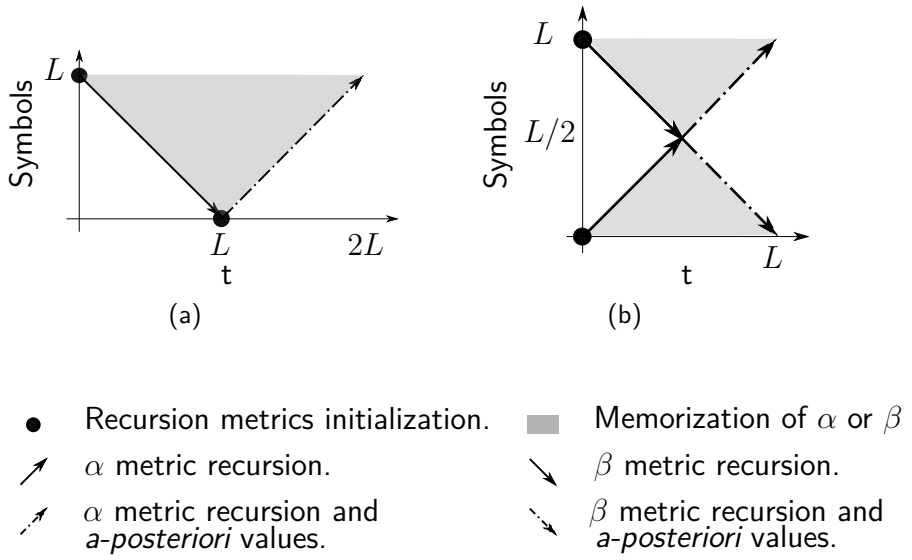

Figure 1.19: Classical BJCR algorithm schedules: (a) Backward-Forward and (b) Butterfly.

memory keeps α or β values. Finally, black dots • represent initial forward or backward state metric values.

The schedule in Figure 1.19(a) is called Backward-Forward. Initially, β state metrics are computed recursively starting from the end until the beginning of the frame. β values are then memorized. When all β values for the frame composed of L information symbols are computed, α state metrics computation can start. In parallel, thanks to the β values previously computed, a-posteriori values are calculated. In the Backward-Forward schedule, the decoding process duration is equal to twice the length L of a frame. This duration is halved by using a Butterfly schedule (Figure 1.19(b)), at the cost of doubling the hardware resources to compute α, β and the a-posteriori information.

Let us consider the schedule for the FOMAP algorithm as presented in Figure 1.20. The *Extend* operation is performed in the forward direction in order to compute $\hat{\alpha}_k$ and $\hat{\alpha}_{k,i}$ as described above. In parallel, the *Update* operation takes place over all the values inside the window of size D_{FO}. Once the *Extend* operation have been done for D_{FO} transitions in the trellis diagram, the a-posteriori values are generated when the *Collect* operation is performed. When *Extend* and *Update* operations are done, at time $k = L$, the calculation of the soft information continues until time $L + D_{FO}$. The delay of the algorithm is equal to the length of the window D_{FO}. Note the similarities between the schedules depicted in Figures 1.15 and 1.20 for the SOVA and FOMAP schedules,

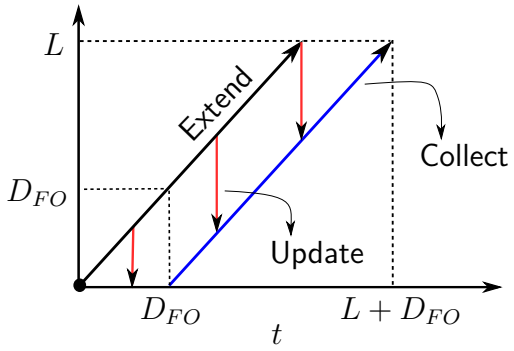

Figure 1.20: FOMAP algorithm schedule.

respectively. However, for the FOMAP algorithm the Update operation is carried out in parallel for all the values of the window, while for the SOVA the Update operations take some time to be performed over the windows of size U since the ML path has to be rebuild back.

1.4 Convolutional Turbo Codes

In this section, we present the convolutional turbo codes basic concepts. The structure of the turbo encoder is introduced. Then, using the SISO decoder algorithms presented section 1.3, the basic ideas behind the turbo decoders behavior are detailed.

1.4.1 Overview

The turbo principle originally proposed for parallel concatenated codes [2] and later extended to serial concatenated codes [42, 43], has demonstrated to be a powerful technique to improve the performance of decoding systems. A turbo encoder is composed of two RSC codes connected through an interleaver. In the decoder side, two convolutional SISO decoders work together by exchanging the so called extrinsic information during an iterative process. Thus, iteration after iteration both convolutional decoders converge to an estimated codeword. In order to do a coherent exchange of information, an interleaved/de-interleaved should be used between the SISO decoders.

1.4.2 Turbo Encoding

Figure 1.21 shows the structure of a turbo encoder. It consists of a parallel concatenation of two identical recursive systematic convolutional encoders RSC1 and RSC2 with code

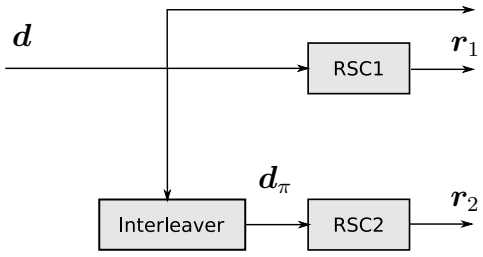

Figure 1.21: Concatenation of two RCS codes using an interleaver.

rate $R = m/n$. An interleaver is used to implement the bijective permutation function (or interleaver function) $i = \pi(j)$ over the information frame $\boldsymbol{d}$ prior to the RSC2 encoding. Thus, at the interleaver output, the interleaved order frame $\boldsymbol{d_\pi}$ is produced. The turbo encoder operation consists then in coding twice the information frame, once in the natural order, carried out by RSC1, and an other time in the interleaved order, done by RSC2. For each information symbol $\boldsymbol{d}_k$, two redundant symbols, $\boldsymbol{r}_{1,k}$ and $\boldsymbol{r}_{2,k}$, are produced. These redundant symbols are sent with the original information symbol (systematic code). Depending on the system requirements, puncturing schemes are also possible. In this case, the redundant symbols are periodically eliminated at the output of the turbo encoder.

The concatenated codes have been adopted in many digital communication standards due to their outstanding error correction capabilities. In satellite communications, such as Digital Video Broadcasting - Return Channel via Satellite (DVB-RCS), Eutelsat (skyplex) and Inmarsat, they have been implemented. Additionally, they are present in mobile communication standards such as Universal Mobile Telecommunications System (UMTS), CDMA2000, Worldwide Interoperability for Microwave Access (WiMAX) (IEEE 802.16), 3GPP-Long Term Evolution (LTE) and LTE-Advanced.

Interleaving

The interleaver plays a key role in the turbo code performance. It is used to scatter in the time the information frame that is going to be coded. By this way, some disorder is introduced in the coding process. The data interleaving helps to mitigate the negative effects of fading channels and burst errors that may occur due to the presence of noise in the channel.

The information frame in the natural order $\boldsymbol{d}= (\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots \boldsymbol{d}_{L-1})$ is inputed to the interleaver which produces the information frame in the interleaved order $\boldsymbol{d_\pi}= (\boldsymbol{d}_{\pi(0)}, \boldsymbol{d}_{\pi(1)}, \ldots \boldsymbol{d}_{\pi(L-1)})$. The interleaver tries to separate as much as possible the information symbols that are mutually affected in the natural order, so that in the interleaved order their mutual effects are significantly reduced. Let us consider an interleaver

function π. For this interleaver, the dispersion factor is defined as the minimal difference between two symbols in the natural and interleaved order:

$$S = \min_{i,j}(|i-j|, |\pi(i) - \pi(j)|) \tag{1.65}$$

Random and pseudo-random interleavers were first proposed in the literature in order to achieve good (high) dispersion factors. In [44] the S-random interleaver has been proposed. This interleaver presents good characteristics in the water-fall region. However, since it does not lead to a large enough Hamming distance, the convergence in the error-floor region is rather poor. In [45], a criterion for the design of pseudo-random interleavers, based on the correlation properties of the extrinsic information, has been introduced. This criterion improves the convergence and the asymptotic gain compared with the S-random interleaver approach.

Unfortunately, from an implementation point of view, random and pseudo-random interleavers involve the memorization of a large amount of data, specially for large frames. Thus, the required memories increase the power consumption and area cost of the turbo encoder/decoder architecture. Structured interleavers have been proposed to overcome this problem. In this type of interleavers, π is easily generated from a small set of parameters. As efficient structural interleavers we can mention the Almost Regular Permutation (ARP) [46], Dithered Relative Prime (DRP) [47] and Quadratic Permutation Polynomial (QPP) [48]. Among them, the ARP and QPP interleavers are of special interest since they have been adopted in some communication systems standards. Additionally, under certain conditions, they are suitable for parallel implementation of turbo decoders as presented in Chapter 3.

1.4.3 TURBO DECODING

The decoding of turbo codes is carried out through an iterative process where two SISO convolutional decoders cooperate by exchanging soft information. Let us consider the basic turbo decoder structure as presented in Figure 1.22. For $i = 1, 2$, the SISOi decoder computes the a-posteriori information $L_{i,k}$ in the form of LLR values, taking as input the systematic LLR values L_k^s, the redundant LLR values $L_k^{r_i}$, and the so called a-priori information $L_{i,k}^a$. For each SISO, a-priori information is provided by the other decoder through an interleaver or de-interleaver. The extrinsic information, $L_k^{e_1} = L_{1,k} - (L_k^s + L_{1,k}^a)$ and $L_k^{e_2} = L_{2,k} - (L_{\pi(k)}^s + L_{2,k}^a)$, is computed from the a-posteriori information by taking out what is already known by the other decoder. These extrinsic values are an estimation of the hard value (the certainty that the hard value is correct), when the channel and a-priori informations are removed. Since the SISO decoders work in different orders, the extrinsic values are additional knowledge of the communication

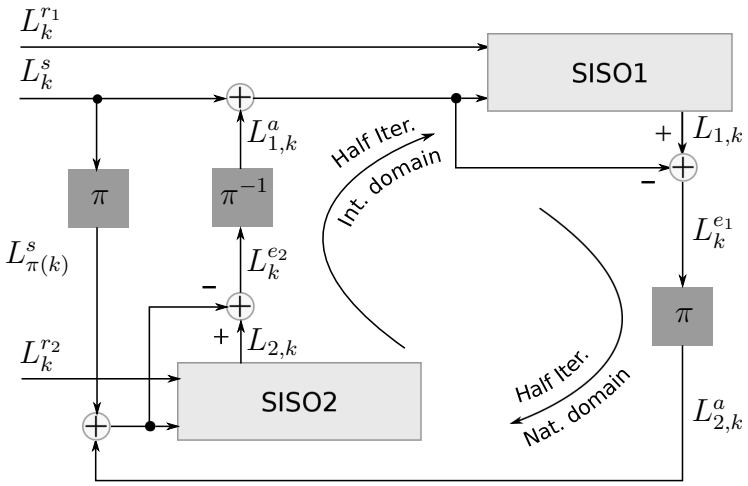

Figure 1.22: Turbo decoder block diagram.

process that can be provided to the SISO decoder in the other order, and thus improve its opportunities to take better decoding decisions. Hence, after interleaving or de-interleaving steps, extrinsic values are used as a-priori information: $L_{1,k}^a = L_{\pi^{-1}(k)}^{e_2}$ and $L_{2,k}^a = L_{\pi(k)}^{e_1}$.

Continuous information exchange is done between the decoders in successively stages referred in this work as half iterations. Each half iteration represents the time that is necessary by any SISO decoder to produce all the extrinsic values in natural or interleaved order. After a certain number of half iterations the iterative process stops and the turbo decoder generates estimates for the systematic symbols. Note that in the half iteration definition only the time is considered. If for example both SISO decoders work in parallel, during a half iteration the extrinsic values for both orders are generated. This definition is adopted since we are mainly concerned with the study of high throughput turbo decoders.

Different SISO convolutional decoder algorithms can be used during the decoding of each constituent convolutional code in the turbo decoder architecture. In section 1.3 we have detailed three algorithms that can be associated to the black box diagram in Figure 1.10, and thus fit perfectly in the turbo decoder architecture. We describe the generation of the extrinsic information by two of the three algorithms (SOVA and BCJR) in the following sub-sections.

1.4.3.1 SOVA SISO Decoder as Turbo Decoder Element

The a-priori information expressed as LLR value is given by the following equation:

$$L_k^a(\delta) = \log\left(\frac{p(\delta_k = \delta)}{p(\delta_k = 0)}\right) \tag{1.66}$$

For the SOVA algorithm, the branch metric was previously defined in (1.29). We can re-write this equation by splitting the terms due to the systematic and redundant bits:

$$\begin{aligned} M_k^{\gamma}(s', s) &= \sum_{i=1}^{m} r_{ki} \cdot u_{ki} + \sum_{i=m+1}^{n} r_{ki} \cdot u_{ki} \\ &+ \sigma_n^2 \log(p(\delta_k = \delta)) + A_k \end{aligned} \tag{1.67}$$

Let us consider the ML path in the trellis diagram up to the time k, $p_{0,k}^{ML}$. Its probability in the logarithm domain is:

$$\begin{aligned} \Theta\left(p_{0,k}^{ML}\right) &= \Theta\left(p_{0,k-1}^{ML}\right) \\ &+ \sum_{i=1}^{m} r_{(k-1)i} \cdot u_{(k-1)i}^{(ML)} + \sum_{i=m+1}^{n} r_{(k-1)i} \cdot u_{(k-1)i}^{(ML)} \\ &+ \sigma_n^2 \log(p(\delta_k = \delta_k^{(ML)})) + A_k \end{aligned} \tag{1.68}$$

where $u_{(k-1)i}^{(ML)}$ and $\delta_k^{(ML)}$ correspond to the information symbol in the ML path. Considering the concurrent path $p_{0,k}^{(h)}$ such that $\delta_k^{(h)} = 0$, the path metric difference at time k, that determines the a-posteriori information, can be expressed as follows:

$$\begin{aligned} L_{k-1}(\delta) &= \Theta\left(p_{0,k}^{(ML)}\right) - \Theta\left(p_{0,k}^{(h)}\right) \\ &= \Theta\left(p_{0,k-1}^{(ML)}\right) - \Theta\left(p_{0,k-1}^{(h)}\right) \\ &+ \sum_{i=m+1}^{n} r_{(k-1)i} \cdot \left(u_{(k-1)i}^{(ML)} - u_{(k-1)i}^{(h)}\right) \\ &+ \sum_{i=1}^{m} r_{(k-1)i} \cdot \left(u_{(k-1)i}^{(ML)} - u_{(k-1)i}^{(h)}\right) \\ &+ L_{k-1}^a(\delta) \end{aligned} \tag{1.69}$$

The last two terms in (1.69) correspond to the intrinsic information, *i.e.*, information that has to be removed from $L_{k-1}(\delta)$ in order to find the extrinsic information that has to be exchanged with the other SISO decoder.

1.4.3.2 BCJR Based SISO Decoder as Turbo Decoder Element

The branch metrics used for the BCJR algorithm in (1.47) can be written as follows (note that the terms due to the systematic and redundant bits are split):

$$\begin{aligned}\gamma_k(s,s') &= B_k \cdot p(\delta_k=\delta) \cdot \exp\left(\frac{1}{2}L_c\sum_{i=1}^{m} r_{ki}\cdot u_{ki}\right)\cdot \exp\left(\frac{1}{2}L_c\sum_{i=m+1}^{n} r_{ki}\cdot u_{ki}\right) \\ &= B_k \cdot p(\delta_k=\delta) \cdot \exp\left(\frac{1}{2}L_c\sum_{i=1}^{m} r_{ki}\cdot u_{ki}\right)\cdot \gamma_k^r(s,s')\end{aligned} \tag{1.70}$$

where $\gamma_k^r(s,s')$ is the contribution of the redundant bits to the branch metric. Now, replacing (1.70) in (1.48), the a-posteriori information becomes:

$$\begin{aligned}L_k(\delta) &= L_c\left(\sum_{i=1}^{m} r_{ki}\cdot u_{ki}\right)+\log\left(\frac{p(\hat{\delta}_k=\delta)}{p(\hat{\delta}_k=0)}\right) \\ &+ \log\left(\frac{\sum_{(s\to s')\in\Gamma_\delta}\alpha_k(s)\gamma_k^r(s,s')\beta_{k+1}(s')}{\sum_{(s\to s')\in\Gamma_0}\alpha_k(s)\gamma_k^r(s,s')\beta_{k+1}(s')}\right)\end{aligned} \tag{1.71}$$

The last term in this equation corresponds to the extrinsic information. $Log\left(\frac{p(\hat{d}_k=\delta)}{p(\hat{d}_k=0)}\right)$ is the a-priori information, expressed as log likelihood, that is inputed to the SISO decoder. Finally, the first term is the information due to the systematic symbols. Since this information is also known by the SISO decoder in the other domain, it is also removed from the a-posteriori inforamation to find the extrinsic information.

1.5 Conclusion

In this first chapter, we have presented the basic definitions of channel coding. Specific considerations were given for the Recursive Systematic Convolutional (RSC) codes. Indeed, this family of codes is the basis of the turbo codes that have been studied in this thesis. A deep study concerning the convolutional codes decoding algorithms, as presented in the literature, has been carried out. The basics of these algorithms were presented in order to establish the type of computations they should perform, and the schedule of these computations. Some comments about their convenience in high throughput decoding were also given. In the last part of the chapter, we have presented the turbo codes as a parallel concatenation of two RSC codes through an interleaver. The decoding algorithms of convolutional codes were then used to describe the decoding

process performed by the turbo decoder. Some well known interleavers were also mentioned. In the next chapters, two of these interleavers (ARP and QPP) are considered in the context of parallel turbo decoders.

Chapter 2

Parallel Processing Exploration for Turbo Decoding

This chapter presents different approaches that have been proposed in the literature in order to design high throughput turbo decoder architectures. The chapter starts by presenting the drawbacks that prevent the achievement of high throughput turbo decoding rates. We define a set of metrics useful to evaluate different architectural solutions. Then, a comprehensive review of the parallel turbo decoding techniques is given. Afterwards, a section of this chapter is devoted to study the convergence of different parallel turbo decode techniques with the aid of EXIT charts diagrams. Finally, we also consider the SOVA based turbo decoders. We describe the structure of a SOVA based SISO decoder, and we give the results obtained in terms of the decoding performance.

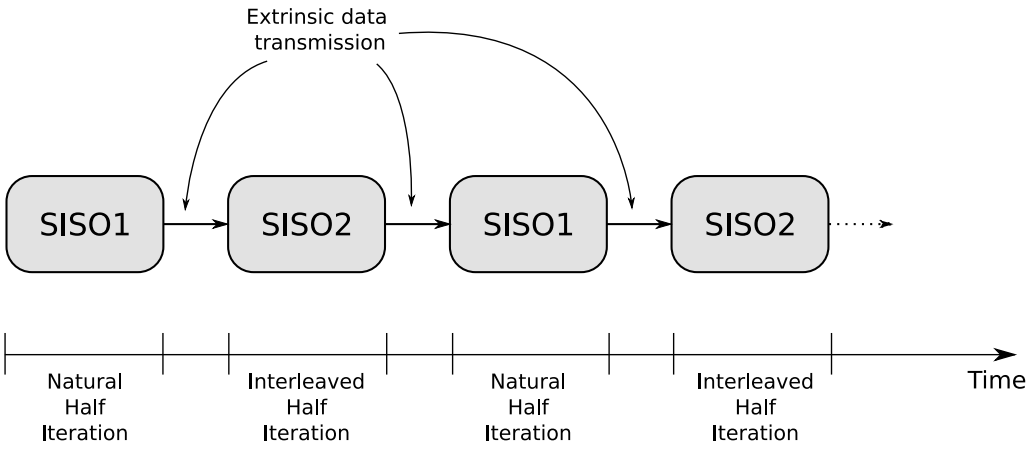

Figure 2.1: Sequential turbo decoding approach.

2.1 CONTEXT

2.1.1 THE INTEREST OF PARALLELISM IN THE TURBO DECODING PROCESS

The decoding of turbo codes is carried out through an iterative process where two SISO decoders, operating in natural and interleaved order, exchange extrinsic information. The turbo decoding approach originally proposed in [2] is sequential: a half iteration in the interleaved (natural) order is performed only when the natural (interleaved) half iteration is completed. This sequential decoding approach is illustrated in Figure 2.1, where SISO1 and SISO2 decoders are assigned to the natural and the interleaved constituent convolutional codes, respectively[1]. In this figure, the iterative process starts by performing a half iteration in the natural order. However, as no preference exists, the interleaved half iteration can be also executed in the first place. For the first half iteration, the SISO1 decoder a-priori information is set to a constant value such as zero. Afterwards, the first natural order half iteration is performed. Then, the interleaved order half iteration starts. During this half iteration the SISO2 decoder exploits the extrinsic information previously produced by the SISO1 decoder to improve its error correcting performance. The decoding process continues in this way for a fixed number of half iterations, or until the conditions established by a stop criterion are met. In Figure 2.1, the time required to transmit the extrinsic information is explicitly considered. This time depends on the turbo decoder architecture. Moreover, the transmission of extrinsic information can start

[1]Note that this sequential decoding approach enables to implement pipelined architectures as proposed in [49]. In this case, multiples frames are decoded concurrently thanks to the pipelined structure.

during the operation of each SISO decoder, *i.e*, it does not necessary starts once each half iteration is completed.

This sequential decoding approach presents two main drawbacks when high throughput is required. First of all, since the half iterations cannot be executed simultaneously, due to the data dependency imposed by the exchange of extrinsic information, the decoding of an information frame is limited by the time required to execute a half iteration. Secondly, the SISO decoder algorithms contain, as presented in section 1.3, recursive operations that prevent a parallel execution of each half iteration. To overcome these constraints, several parallel decoding techniques have been investigated in the literature.

The benefits provided by a parallel turbo decoding technique cannot be only established by considering the gain in terms of throughput. Some others aspects such as the flexibility, the architecture hardware complexity and the power consumption, have also to be taken into account. Consequently, suitable metrics should be defined in order to correctly assess the parallelism techniques.

2.1.2 PARALLEL ARCHITECTURE EVALUATION

The main objective of our work is to design high speed turbo decoders. Hence, the turbo decoder throughput is regarded as the main characteristic to optimize. Additionally, to validate an architectural solution, two other criteria are also considered: error decoding performance and hardware complexity. Flexibility and power consumption are not considered in our work. However, the cost in terms of power consumption is related to the hardware complexity.

2.1.2.1 ALGORITHMIC COMPLEXITY ANALYSIS

With regard to the hardware implementation, there is not an unique measure of the algorithmic complexity. Its definition has to be adapted depending on the application domain where the system designed will be applied. The most common approach to evaluate the algorithmic complexity consists in determining the number of elementary operations performed by the algorithm. In [50], this approach is followed in order to establish the complexity of the Max-Log-MAP algorithm. In this case, additions and comparison-selections correspond to the elementary operations. In [51], in the context of iterative processing, a normalization technique was applied where each addition, subtraction and multiplication is converted into the equivalent number of one-bit full additions. Therefore, the complexity of the receiver is expressed as a multiple of the complexity of a one-bit full adder block. In [52], the algorithmic complexity is determined by the number of *algorithmic* operations that should be performed per time unit.

It is then expressed in Giga Operations Per second (GOPs). In [53], two metrics are proposed to compare the algorithmic complexity of different Forward Error Correction (FEC) schemes: one based on the arithmetic operations and the other on the storage requirements per decoded bit.

2.1.2.2 ARCHITECTURAL COMPLEXITY ANALYSIS

The architectural complexity can be defined as the number of hardware resources, related to a specific target technology, required to implement a given algorithm. For instance, if an Application Specific Integrated Circuit (ASIC) target is considered, the area (in mm^2) is a direct measure of the architecture complexity. This area can also be given in terms of the number of equivalent logic gates of the overall circuit. If a Field Programmable Gate Array (FPGA) target is used, the complexity is therefore measured as the number of functional units in the device that are occupied (RAM blocks, LUTs, Flip-Flops, etc).

It is possible to estimate the hardware complexity from the algorithmic complexity. In this case, the complexity of each elementary operation is first established. Then, the equivalent hardware complexity is computed by multiplying the number of operations by their respective complexities. However, this approach is inaccurate because it assumes that one hardware resource is assigned to each operation. This inaccuracy is specially true for applications dominated by data transfer and with high storage requirements, where the arithmetic computations play a secondary role in the overall complexity. In this case, the hardware complexity should be obtained after logic synthesis.

2.1.2.3 ERROR DECODING PERFORMANCE

As presented in section 1.1.4, the performance of a digital communication system is expressed in terms of its BER and FER for different SNR values. A turbo decoder improves its performance during the iterative process. Classically, if no stop criterion is applied, 6 to 8 iterations are performed by a turbo decoder, providing a good tradeoff between the error correction capabilities and the time required to decode a frame. When parallel turbo decoding process is considered, decoding performance degradation may appear. To limit this degradation, additional iterations may be required, impacting negatively the turbo decoder throughput.

In the context of high throughput architectures, a high parallelism level is necessary. The designer should therefore especially consider the adverse impact on the error decoding performance. In the literature, as presented hereinafter, some techniques are proposed to palliate the performance degradation that may appear, keeping as low as possible the number of additional iterations. However, in some cases, if high throughput

is the most important constraint to achieve, some degradations with respect to a non parallel architecture may be accepted. It depends on the design specifications in accord with the application domain. In our study, we have defined a reference turbo decoder architecture that does not exhibit any parallelism degree. Its decoding performance is established for a fixed number of iterations. Then, the performance of the parallel architectures is compared to that of this reference architecture.

2.1.2.4 METRICS ASSOCIATED TO THE ARCHITECTURE DESIGN

The use of some metrics helps to perform a fair comparison between different architectures so that the most appropriate solution can be selected. The use of algorithmic complexity metrics for this purpose is not sufficient because important implementation issues are not considered.

As stated in [54], the comparison of different decoder architectures is a challenging task. This is specially true when a comparison of published decoder architectures is made, since all the design information is usually not available. We introduce some useful definitions to describe the turbo decoder architectures as follows.

Let I denote the number of iterations executed by a turbo decoder, and t the time necessary to decode a frame. The architecture throughput, defined as the number of decoded bits per time unit, depends on t. It can be expressed as:

$$Throughput = Th = \frac{m \cdot L}{t} \ [Mbit/s] \tag{2.1}$$

where $m \cdot L$ corresponds to the number of bits in a frame[2]. Let us consider a reference turbo decoder architecture with Th_r and t_r the throughput and the frame decoding time, respectively. This reference architecture does not have any parallelism degree. Now, consider a parallel architecture with values Th_p and t_p. For these two architectures, we denote the relative acceleration as:

$$A = \frac{t_r}{t_p} \tag{2.2}$$

Let C_r and C_p be the architectural hardware complexities for the reference and the parallel architecture, respectively. In order to achieve an increase in the turbo decoder throughput ($A > 1$), the parallel architecture presents a higher hardware complexity. We define the relative cost in terms of the hardware complexity between both architectures as follows:

[2]Recall that m and L are the number of bits per information symbol and the number of symbols per information frame, respectively.

$$\rho = \frac{C_p}{C_r} \tag{2.3}$$

In [54], the importance of a good definition of the metrics to evaluate the efficiency of FEC codes has been highlighted. In this work it has been shown how the use of algorithmic complexity metrics, such as GOPs, is inconvenient for this purpose. Thus, two metrics have been defined. One is based on the architecture energy efficiency, and the other one on the hardware complexity (expressed as the overall circuit area in mm^2) efficiency. Since we do not consider the power consumption, in our study we only use the complexity efficiency metric. It is defined as the achievable throughput per area unit:

$$\eta = \frac{Th}{C} \ [Mbit/s/mm^2] \tag{2.4}$$

This metric corresponds to an estimation about how well the hardware resources are utilized by the architecture in order to provide high throughput rates. Since the overall architectural complexity is considered, all the implementation issues such as the memory size and data transfer are entirely considered. Furthermore, the proposed metric is normalized to the number of information bits. Thus, this metric enables to compare competing architectures since the hardware complexity efficiency is independent of the type of operations and the data types used to execute the algorithm [54].

We have also defined the relative efficiency between two architectures by regarding the complexity efficiency in (2.4):

$$E = \frac{\eta_p}{\eta_r} = \frac{Th_p \cdot C_r}{C_p \cdot Th_r} = \frac{t_r \cdot C_r}{t_p \cdot C_p} = \frac{A}{\rho} \tag{2.5}$$

where η_r and η_p are the complexity efficiencies for the reference and the parallel architecture, respectively.

The metrics and definitions introduced above are useful to assess the different turbo decoder parallelism techniques and architectural solutions. We have defined absolute metrics in (2.1) and (2.4) that can be use to compare published decoder architectures. On the other hand, the relative expressions in (2.2), (2.3) and (2.5) help to estimate the impact of the parallelism techniques as presented below.

2.2 PARALLEL PROCESSING IN TURBO DECODING

In [55], a multi-level classification of parallel turbo decoding techniques with three hierarchical levels has been proposed: turbo decoder level parallelism, SISO decoder level parallelism and metric computation level parallelism. These three levels are defined according to the their granularity. Thus, the turbo decoder level parallelism exhibits a

coarse-grained parallelism, while the metric computation parallelism level is fine-grained. The SISO decoder level parallelism corresponds to an intermediate granularity value. Following this classification, a complete review of the parallel turbo decoder techniques is presented in the rest of this section. We also include a technique not considered in [55].

2.2.1 PARALLELISM AT THE TURBO DECODER LEVEL

The parallelism at the turbo decoder level corresponds to a replication of the overall turbo decoder architecture in order to simultaneously decode different frames. In this case, even though a high acceleration is possible, the architectural complexity increase becomes, for the most part of practical applications, unacceptable. This parallelism level provides a linear increase in the throughput with a corresponding linear increase in terms of hardware complexity. Therefore, the relative efficiency is $E = 1$.

Two approaches exist: concurrent frame approach and concurrent iteration approach. In the former approach, multiples dedicated turbo decoders are assigned to decode iteratively multiple information frames. In the latter approach, multiples iterations are executed simultaneously for different information frames. In this case, the decoder consists in a structure composed of $2 \cdot I$ stages. The i^{th} stage executes the i^{th} half iteration of a given frame. When it is completed, it executes the i^{th} half iteration for the next incoming frame. The architecture presented in [49] is based on this approach for a 16-state double binary turbo code.

An intermediate parallelism level between the turbo decoder level and the SISO decoder level (described in section 2.2.2) can also be proposed. In this case, the hardware resources that are available are assigned, during their idle time, to process other frames. In [56], a high throughput turbo decoder architecture exploiting this approach is presented. An appropriate scheduling for the Max-Log-MAP algorithm is proposed to process simultaneously two frames. Even though all the computation resources are not duplicated, the architecture requires a duplication of the whole extrinsic and channel memories, that represent the major hardware cost of the overall architecture.

2.2.2 PARALLELISM AT THE SISO DECODER LEVEL

In the case of parallelism at the SISO decoder level, multiples SISO decoders are assigned to decode a frame. These SISO decoders work simultaneously, and each one is responsible of the decoding operations over a set of transitions in the trellis diagram. There are two approaches at this parallelism level: sub-blocks parallelism [57, 58], and shuffled parallelism [59]. Let P denote the number of SISO decoders that compose the turbo

decoder architecture. We consider a reference turbo decoder architecture with no parallelism at the SISO decoder level. This architecture executes I_r iterations. Therefore, the frame decoding time can be expressed as:

$$t_r \propto L \cdot I_r \tag{2.6}$$

2.2.2.1 SUB-BLOCKS PARALLELISM

Considering the drawbacks that the sequential decoding approach presents, the sub-block technique has been proposed in order to reduce the time required to perform a half iteration. Thus, the frame to decode, composed of L symbols, is divided into Q sub-blocks, each one having $N = L/Q$ symbols. Therefore, the number of SISO decoders should be $P = Q$ to decode simultaneously all the sub-blocks. The sub-blocks are processed in parallel in each order and between both orders sequentially. It means that all the SISO decoders perform a half iteration and, when it is completed, all of them perform a half iteration in the other order. This process is repeated iteratively until all the iterations are executed.

Since the SISO decoder algorithms perform recursive operations along the trellis diagram, the sub-blocks are not independent. Therefore, the sub-block processing imposes a major constraint: the sub-block initialization of state metrics in the limits of each sub-block. A survey of sub-block initialization can be found in [60]. This work is particularly oriented to the BCJR algorithm. However, the techniques can also be applied to the SOVA as explained in [61]. Actually, three initialization methods have been proposed: initialization by acquisition, initialization by message passing and initialization by combining the two previous methods. They will be detailed in the rest of this sub-section.

Initialization by Acquisition This initialization method enables to estimate the state metrics values at the beginning of each sub-block by performing recursive computations in the adjacent blocks. Figure 2.2 depicts this method for the BCJR algorithm. Over a window of length AL (Acquisition Length) in the sub-blocks limits, forward and backward recursive operations are performed. At the end of these recursions, β and α values are estimated for the previous and next block, respectively. In Figure 2.2, the convolutional code is assumed to be circular. Thus, the forward recursion in the last sub-block is used to estimate $\alpha_0(s)$, required to decode the first sub-block. Furthermore, the backward recursion in the first sub-block defines $\beta_L(s)$ used in the last sub-block.

For this initialization, the frame decoding time is:

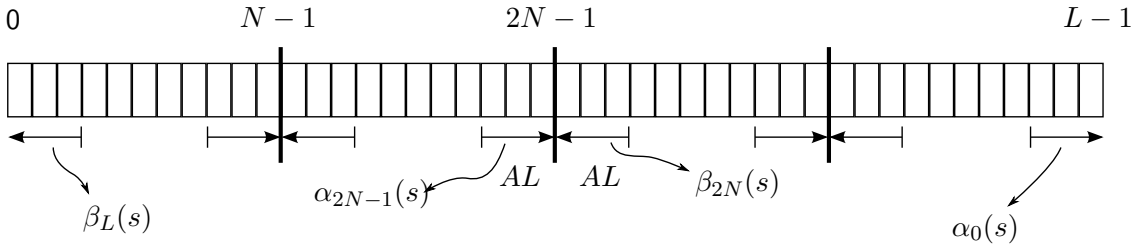

Figure 2.2: Initialization by acquisition.

$$t_d \propto \left(\frac{L}{Q} + AL\right) I_d \tag{2.7}$$

where I_d is the number of iterations. Therefore, the acceleration with respect to the reference architecture with decoding time in (2.6) is:

$$A = \left(\frac{Q}{1 + \frac{Q \cdot AL}{L}}\right)\left(\frac{I_r}{I_d}\right) \tag{2.8}$$

If the acquisition and sub-block length are long enough, the degradation induced by the sub-blocks parallelism can be neglected, and thus no additional iterations are required ($I_r = I_d$). Empirically, the acquisition length to avoid additional iterations should be about 3 to 6 times the convolutional code constraint length [62, 63].

Initialization by Message Passing In this method, also called Next Iteration Initialization (NII), the state metric values in the sub-blocks limits are initialized with the state metrics computed in the adjacent sub-blocks during the previous iteration. For the first half iteration, in the natural and interleaved order, the initial state metrics are set to equiprobable values. In this case, no significative hardware resources or control mechanism are required, contrary to the initialization method by acquisition. We have the frame decoding time:

$$t_d \propto \left(\frac{L}{Q}\right) I_d \tag{2.9}$$

Consequently, the acceleration is:

$$A = \left(\frac{I_r}{I_d}\right) Q \tag{2.10}$$

The acceleration is proportional to the number of sub-blocks in the architecture. However, if no performance degradation with respect to the reference architecture is demanded, additional iterations should be performed ($I_d > I_r$), which limit the achievable acceleration. The message passing method can be complemented by performing, during the first iteration, recursive computations as in the acquisition method [64, 65]. Thus, the number of additional iterations to overcome a decoding performance degradation can be reduced.

In [60, 62] a thoroughly study has been carried out in order to establish the best initialization method by considering their efficiency. We present the main conclusions regarding this point as follows:

- The message passing method is more efficient than the acquisition method. It is especially true when a high parallelism degree (large number of sub-blocks) is considered.

- For a low number of sub-blocks, apply the acquisition method during only the first iteration, and the message passing method in the rest of the iterations, is convenient. It has a positive impact on the system efficiency due to the reduction in the number of required iterations. This positive impact is more important for high code data rates.

- When the parallelism degree increases, by augmenting the number of sub-blocks, the acquisition operation during the first iteration is undesirable. In this case, since the block size is *small* (high parallelism level), the time required to perform the acquisition operation is comparable to the time required to decode each sub-block. This fact is contrary to what is observed for a low parallelism degree, where the acquisition time is, comparatively, negligible. Therefore, the benefits of a reduction in the number of iterations have an important weight.

- In general, for all three initialization methods, the efficiency decreases with the increase of the number of sub-blocks.

From a high throughput turbo decoding point of view, based in these conclusions, the message passing initialization is the most attractive method to manage the state metrics in the sub-blocks limits. Besides, since the acquisition operation during the first iteration is only convenient for a low parallelism degree, its application should be avoided. Thus, we can simplify some architectural constraints such as memory access problems as described in Chapter 3.

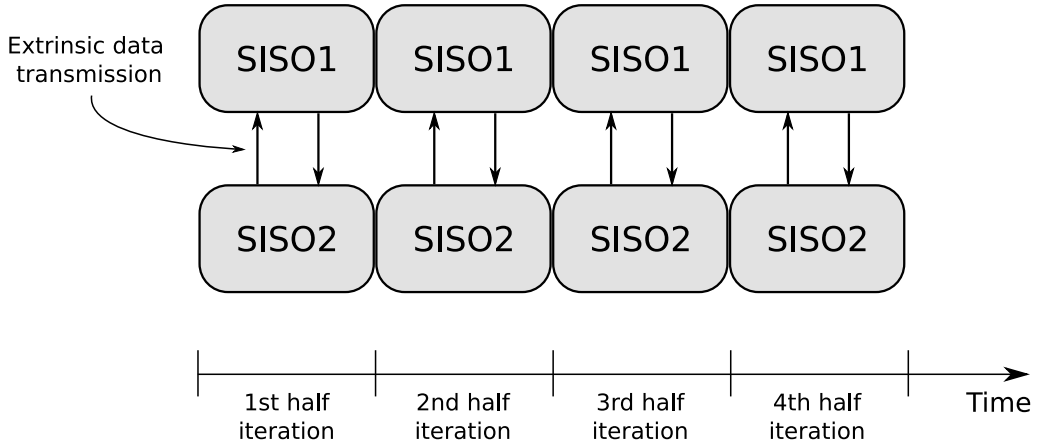

Figure 2.3: Shuffled turbo decoding principle.

2.2.2.2 SHUFFLED PARALLELISM

The basic idea of shuffled decoding is to decode both constituent convolutional codes simultaneously, exchanging extrinsic values as soon as created between both orders. Thus, each SISO component has faster access to update a-priori values, what would enable a faster decoding algorithm convergence. It means that fewer iterations are necessary to achieve the same BER performance with respect to a non-shuffled decoding approach.

Figure 2.3 depicts the shuffled technique principle. SISO1 and SISO2 decoders are assigned to operate in the natural and interleaved order, respectively. Along the decoding of their respective constituent convolutional codes, each decoder sends extrinsic information to the other decoder, and receives from it information used as a-priori information. During each half iteration, the extrinsic values for both orders are generated, contrary to the sequential decoding process. Thus, the duration of a half iteration is the same in the shuffled and sequential approach.

For this parallelism technique the frame decoding time is $t_d \propto L \cdot I_d$. Consequently, the acceleration is just the ratio between the number of iterations:

$$A = \frac{I_r}{I_d} \tag{2.11}$$

Since the convergence is improved, $I_d < I_r$, an increase in the throughput is possible [59,66,67]. Furthermore, sub-block and shuffled parallelism techniques can be combined. Thanks to this combination, several SISO decoders work simultaneously on different sub-blocks in the natural and interleaved domain. This approach is shown in Figure 2.4, were

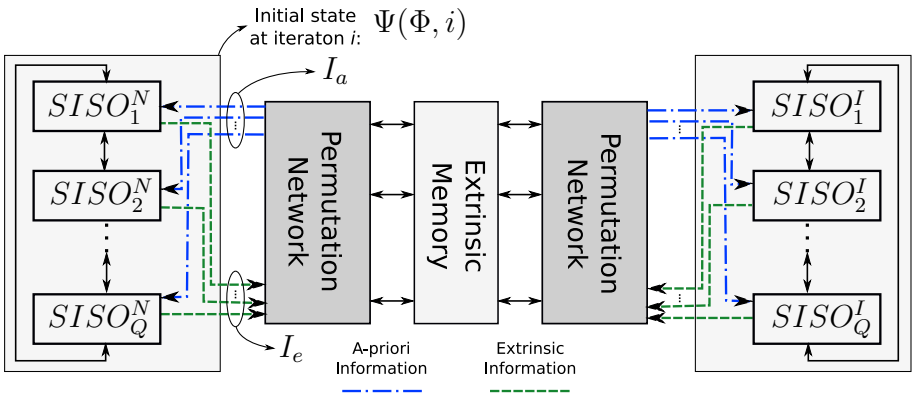

Figure 2.4: Architecture based on shuffled and sub-block parallelism.

$SISO_i^N$ and $SISO_i^I$, for $0 < i \leq Q$, are the decoders in the natural and interleaved order, respectively. An extrinsic memory is necessary to transfer extrinsic values between the decoders in both orders. Two permutation networks are also necessary in order to provide interleaver and de-interleaver functions. In this figure, the message passing method is used to manage the state metric initialization of each sub-block.

The reduction in terms of iterations provided by a shuffled architecture, is possible thanks to the duplication of the number of SISO decoders with respect to a sequential decoder. Thus, the benefits of a shuffled architecture over a sequential one are not easy to spot. Let us detail this point. The number of SISO decoders in a Q sub-block sequential turbo decoder is $P_{sequential} = Q$. In this case, the same set of SISO decoders can be used to decode both orders. With the same hardware cost, without considering any extrinsic memory issue, it is possible to build a shuffled architecture having $Q/2$ sub-blocks. Thus, $P_{shuffled} = P_{sequential} = Q$ SISO decoders are required. In this case, the shuffled architecture would require less iterations than the sequential architecture to achieve the same decoding performance. However, since there are more sub-blocks in the sequential architecture, a sequential half iteration is executed faster than a shuffled half iteration. Therefore, the architecture that provides the best throughput-hardware complexity relation is not straightforward determined.

In [68], sequential and shuffled turbo decoder architectures have been compared by considering their efficiency. In this work, it has been concluded that for a low parallelism degree (low number of SISO decoders) the sequential architectures exhibit a higher efficiency compared with the shuffled architectures. However, when the parallelism grows, for a certain number of sub-blocks, the shuffled architecture becomes more efficient. Thus, if we need a *high* number of SISO decoder to build a hight throughput turbo decoder, we would prefer to distribute them in the natural and interleaved order to

design a shuffled decoder, rather than design a sequential decoder, where all the SISO decoders work in only one order at the same time. This conclusion is reached with an analysis performed at the algorithm level, without regarding important architectural issues that arise when the system is implemented. Indeed, the work in [68] targets a flexible multi-ASIP architecture. Since in our work we consider specific architectures, we have to be careful to establish the convenience of shuffled decoders.

2.2.3 PARALLELISM AT THE METRIC LEVEL

This level of parallelism considers the operations performed inside each SISO decoder. Thus, by exploring the trellis diagram structure and the SISO decoder algorithm properties, it is possible to reduce the sub-block execution time. Let us first consider the parallelism that can be extracted from the trellis diagram structure. The SISO decoder algorithms, during each transition in the trellis diagram, execute a set of identical operations for each convolutional encoder state. Since these operations are independent, they can be executed in parallel. We can then assign 2^p blocks that perform these operations for each state.

Regarding the SISO decoder algorithm, the first high speed SISO decoder architectures were proposed in [69, 70] for nonsystematic codes, and later extended to recursive systematic codes in [71, 72]. In these works, high parallel high pipelined structures that provide real time characteristics are proposed. However, due to their high hardware complexity, we do not consider this solution any further in this document[3]. We describe parallel SISO decoding techniques that increase the throughput rate for an acceptable complexity in the following sections.

2.2.3.1 RADIX-2^{N_T} PARALLELISM TECHNIQUE

All the SISO decoder algorithms are based on recursive computations through the trellis diagram. For the BCJR, Log-MAP and Max-Log-MAP algorithms, these computations are performed in the forward and backward directions, while for the Viterbi algorithm they are only performed in the forward direction. This recursive property is the bottleneck for a high throughput implementation of convolutional decoding algorithms. Thus, it plays a major role to achieve high turbo decoding throughput rates.

The implementation of the recursive operations for the Log-MAP algorithm can be carried out with the architecture presented in Figure 2.5, first shown in Chapter 1,

[3] In chapter 5 we present an idea based in these works that can be explored as a long term perspective of our work

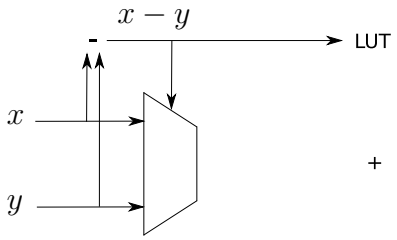

Figure 2.5: Architecture for the implementation of the function $\max^*(x, y)$.

that performs the computation of the $\max^*$ function. On the other hand, the Max-Log-MAP and Viterbi algorithm recursive operations can be implemented with the so called Add Compare Select (ACS) unit as already mentioned in section 1.3.1.1, that can be build from the architecture in Figure 2.5 by eliminating the LUT. Since the ACS unit implements a feedback loop, pipelined architectures do not increase the decoding speed. However, a pipelined ACS unit can be employed in order to reduce the hardware complexity of a decoder implementing the sub-block technique, as proposed for instance in [73, 74] and [50, Ch. 5]. In this case, the pipelined ACS unit can be shared between as much sub-blocks as the number of pipeline stages.

In order to effectively break the bottleneck imposed by the ACS unit, the Radix-2^{N_T} technique can be applied[4] [37, 56, 75–81]. This technique consists in processing during each clock cycle N_T transitions in the trellis diagram. Thus, if the critical path of the resulting Radix-2^{N_T} ACS unit is lower than N_T times the critical path of a non parallel ACS unit (radix-2), a gain for the decoding speed is achieved. Furthermore, since only the state metric values corresponding to time k multiple of N_T are computed, the size of the state metric memories can also be reduced. This memory reduction is achieved with a corresponding increase of the ACS unit complexity.

2.2.3.2 MAP ALGORITHM PARALLELISM: ALGORITHM SCHEDULES

A parallel execution of the BCJR algorithm can be achieved by considering the schedule in which the main computations α, β and extrinsic information are performed. Since the SISO decoding algorithms are intrinsically recursive and data dependent, some constraints are imposed in order to define valid schedules. In section 1.3.2.4 two straightforward schedules for the BCJR algorithm have been introduced: Backward-Forward and Butterfly schedules. In this section, we present two additional schedules that have been proposed in order to improve the convergence of shuffled turbo decoder architectures.

[4] This parallelism technique has not been originally included in the parallelism level classification proposed in [55].

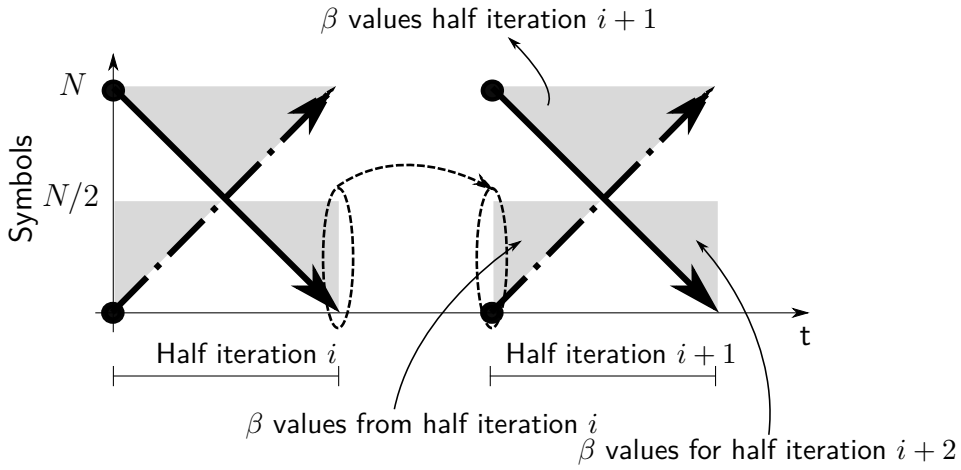

Figure 2.6: Butterfly-Forward schedule.

Let us consider a shuffled turbo decoder with an arbitrary number of sub-blocks. During a half iteration, the shuffled technique seeks to provide to each SISO decoder updated a-priori values that help to improve the efficiency of the decoding algorithm computations. For the information symbol $\boldsymbol{d}_i$, this objective can be achieved if the updated a-priori values are available when:

- The state metrics related to $\boldsymbol{d}_i$ ($\alpha_i(s)$ or $\beta_{i+1}(s)$) are computed.
- The extrinsic value for $\boldsymbol{d}_i$ is calculated.

The Backward-Forward and Butterfly schedules generate extrinsic information only in the second half of the sub-block decoding process. Thus, during the first half the use of the shuffled technique does not provide any advantage. In the second half of the sub-block decoding process, extrinsic values are generated and exchanged between SISO decoders in both domains. However, this exchange may not benefit the most part of the symbols in the sub-block. To solve this problem, two additional schedules have been proposed. These schedules perform an intensive exchange of extrinsic information all over the sub-block decoding process.

Figure 2.6 depicts the Butterfly-Forward schedule [62]. At the beginning of the decoding process, β values for the first half of the sub-block (from symbol 0 to symbol $N/2$), computed during the previous half iteration, are stored in the state metric memory. Then, the α and β recursions are performed simultaneously. In parallel, thanks to the β values available in memory, extrinsic values are generated in the forward direction.

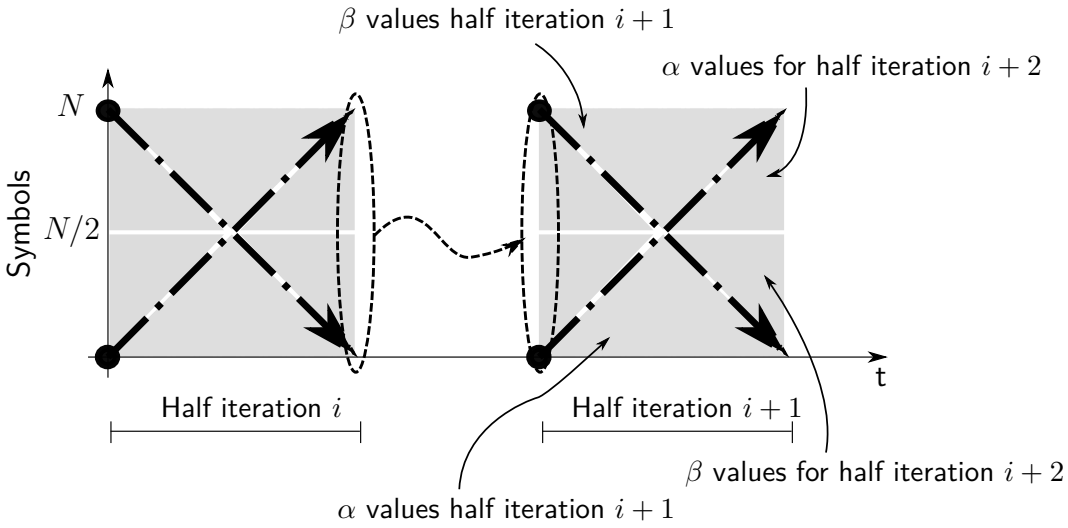

Figure 2.7: Butterfly-Replica schedule.

When the decoding process overpasses the half of the sub-block (symbol $N/2$), extrinsic values are computed with the β values that have been calculated in the backward recursion process during the current half iteration. Note that β values for the first half of the sub-block are stored so that they can be used in the next half iteration. In Figure 2.7, the Butterfly-Replica schedule is shown. In this schedule extrinsic values are continuously generated during all the sub-block decoding process in the forward and backward direction. Thus, for each information symbol, two extrinsic values are generated.

Replica schedules have been introduced in [66]. They consist in producing more than one extrinsic value for each information symbol. The schedule presented in this last work is different of the schedule given in Figure 2.7. In [66], the proposed schedule corresponds to the simultaneous execution of the schedules Backward-Forward (Figure 1.19(a)) and Forward-Backward (α computation is carried out prior to β computations). Thus, the decoding time for this schedule is twice by comparison with the Butterfly-Replica schedule decoding time. Because of this larger decoding time, this schedule is not considered any further in this document.

The different schedules presented so far can be implemented using the three main hardware units:

- Branch Metric Unit (BMU): used to compute the state metric transition $\gamma_k(s, s')$.
- ACS unit.

Schedule	Nb. of BMU	Nb. of ACS	Nb. of SOU	Mem. Size
Backward-Forward	1	1	1	N
Butterfly	2	2	2	N
Butterfly-Forward	2	2	1	$N/2$
Butterfly-Replica	2	2	2	N

Table 2.1: Required hardware units and state metric size for radix-2 to implement the different SISO decoding schedules.

- Soft Output Unit (SOU): in charge of the computation of the extrinsic information and of taking the hard decoding decisions.

These hardware units will be detailed in Chapter 3. Here, we restrict ourselves to indicate each schedule implementation complexity by considering the number of required hardware units, as presented in Table 2.1. In this table, the state metric memory size is also given. In terms of hardware units required, the Backward-Forward schedule is the less complex. This happens since this schedule does not increase the parallelism of the BCJR algorithm schedule. The other three schedules require two BMUs and two ACS units due to the parallel execution of the forward and backward state metric recursions. Note that for the Butterfly and Butterfly-Replica schedules, two SOUs are also required since two extrinsic values are generated simultaneously. Regarding the state metric memory size, the Butterfly-Forward schedule exhibits the lower hardware complexity. This schedule requires a memory block size equal to the half of the sub-block size (maximum shaded area heigh in Figure 2.6) while for the other schedule the state metric memory size should be equal to the sub-block size.

For all the four schedules, the decoding delay and memory requirements grow linearly with the sub-block size. Thus, for a large sub-block size N, the decoding delay, and specially, the memory requirements may be prohibitive in practical decoder implementations. To overcome this problem the sliding window technique has been proposed. Note that it can be applied to the BCJR algorithm [82,83] and to the Viterbi algorithm [84] as well. For the sliding window technique, the sub-block is divided into different *windows*, each one is composed of W symbols. Consecutive windows are decoded sequentially, and thus, the same hardware blocks can be used to decode all the windows. Since the window size is usually much lower than the sub-block size, the state metrics memory can be significantly reduced.

Figure 2.8 depicts the Butterfly schedule when the sliding window technique is applied. Since four sliding windows are used, the required state metric memory size is $W = N/4$. In this figure the black dots • and black squares ■ represent respectively the initial forward and backward state metric values in the window limits. The black dots

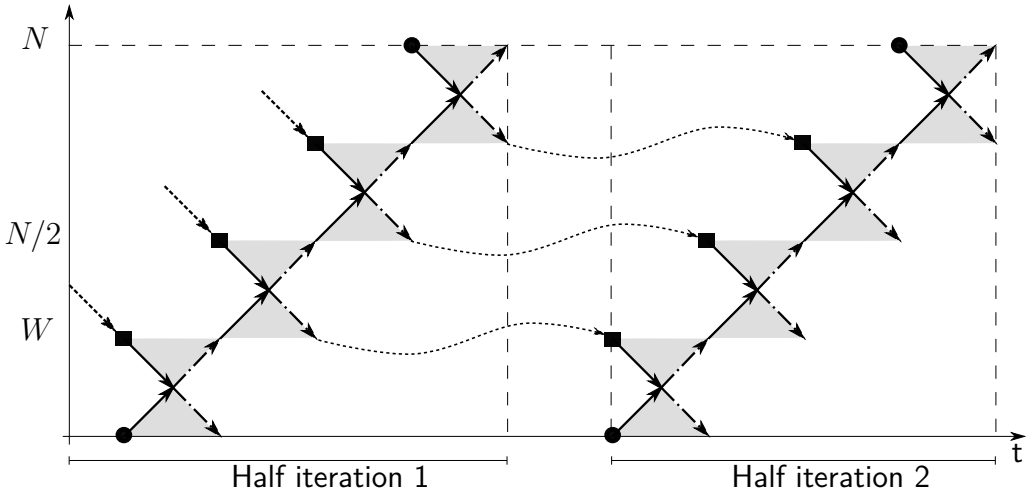

Figure 2.8: Sliding window technique considering the Butterfly scheduled.

also represent the initial state metrics in the sub-block limits, computed with one of the methods as described in section 2.2.2.1. Note that the computation of α is performed continuously through the sub-block length, what is not possible in the β computation. Thus, appropriate initial β values for each window have to be estimated. In figure 2.8 two methods are shown for this purpose. In the first half iteration, an initialization by acquisition is performed, while in the second half iteration a message passing technique is applied.

The window size and the state metric initialization method have an impact on the decoder performance. For a *small* window size, the initialization by acquisition is required during each half iteration in order to avoid an important performance degradation [85, 86]. In this case, the decoding speed is negatively affected. In the context of high throughput turbo decoders, the acquisition operation is avoided, using the message passing initialization instead. In [87], adjacent windows are not processed sequentially, so that the next half iteration in the other domain can be started at the end of the current half iteration, as originally proposed in [88].

2.2.4 GATHERING PARALLEL TURBO DECODING TECHNIQUES

Let the 4-tuple $\Phi_{Q,S_p}^{\xi,N_T} = \{Q, S_p, \xi, N_T\}$ denotes a parallel turbo decoder architecture, with $Q \geq 1$ representing the number of blocks in the natural and interleaved order, S_p the shuffled (*Sh*) or not shuffled (*NoSh*) parallelism, ξ the SISO decoder schedule -Backward-Forward schedule (B-F), Butterfly schedule (B), Butterfly-Forward schedule (B-FW) or Butterfly-Replica schedule (B-R)-, and N_T the number of transitions that

the ACS unit executes during each clock period (radix-2^{N_T}).

We consider the reference turbo decoder architecture $\Phi_r = \Phi^{B,1}_{1,NoSh}$, *i.e.* the turbo decoder is composed of one SISO decoder implementing the Butterfly schedule, and with no parallelism at any other level. After $I(\Phi_r)$ iterations, this turbo decoder provides a decoding performance that will be taken as reference. Thus, in an ideal scenario, a parallel decoder Φ should have at least the same decoding performance after $I(\Phi)$ iterations. For this parallel architecture, let $t(\Phi)$ denote the time required to decode a frame and $A(\Phi) = t(\Phi_r)/t(\Phi)$ its acceleration. Besides, $C(\Phi)$ and $\rho(\Phi) = C(\Phi)/C(\Phi_r)$ are the architectural complexity and relative complexity, respectively. Let $\tau(\Phi)$ be the ACS unit critical path. Therefore, $\tau(\Phi)/N_T$ is the time required to execute one transition in the trellis diagram. Thus, the frame decoding time is $t(\Phi) = \frac{2 \cdot I(\Phi) \cdot \tau(\Phi) \cdot L}{Q \cdot N_T}$. Consequently, from (2.1), the architecture throughput can be expressed as follows:

$$Th(\Phi) = \frac{m \cdot Q \cdot N_T}{2 \cdot I(\Phi) \cdot \tau(\Phi)} = \frac{m \cdot Q \cdot N_T \cdot f_{clk}}{2 \cdot I(\Phi)} \tag{2.12}$$

where f_{clk} is the architecture clock frequency. We assume that all the different turbo decoder blocks are pipelined, so that the ACS unit critical path corresponds to the decoder critical path. The message passing method has been selected when the sub-block parallelism is used. For this reason, the acquisition length AL does not appear in (2.12). The turbo decoder architectural complexity can be expressed as follows [54, 62]:

$$\begin{aligned} C(\Phi) = & \, C_{ctl} \\ & + C_{net}(Q, S, N_T) \\ & + P \cdot C_{SISO}(\xi, N_T, W) \\ & + C_{mem}(Q, N_T, S, L, W) \end{aligned} \tag{2.13}$$

C_{ctl} is the control logic complexity which is usually smaller compared to the other terms. C_{net} is the complexity due to the interconnection network between the SISO decoders and the memory banks. C_{SISO} is the SISO decoder complexity. It is multiplied by the number of SISO decoder that are used in the architecture. Finally, C_{mem} denotes the memory complexity. It encompasses the extrinsic memory complexity C^{ext}_{mem}, the channel information memory complexity $C^{channel}_{mem}$, and internal SISO decoder memory complexity[5] C^{SISO}_{mem}:

[5]The internal SISO memory corresponds to the state metric memory used to store α and β, and to the memories used to temporally store a-priori values, as described in Chapter 3.

$$\begin{aligned} C_{mem}(Q, N_T, S, L) =& C_{mem}^{ext}(Q, N_T, S, L) \\ &+ C_{mem}^{channel}(Q, N_T, S, L) \\ &+ P \cdot C_{mem}^{SISP}(\xi, N_T, W) \end{aligned} \tag{2.14}$$

Note that the channel and extrinsic memories complexity do not only depend on the frame length L, but also on the parallelism at the SISO decoder level and the radix value. Indeed, different memory blocks should be assigned in order to solve memory conflicts. Thus, the complexity of the resulting memory structure is more complex than a single memory that has the same storing capabilities.

In the next section we present a study of the convergence properties of turbo decoders implementing the parallelism techniques described in this section.

2.3 CONVERGENCE OF PARALLEL TURBO DECODERS

The information exchanged during the iterative turbo decoding process is not easy to analyze and to describe. A useful technique to help the designer is the EXtrinsic Information Transfer (EXIT) chart [89]. Unfortunately, this method cannot be directly applied to the decoding convergence analysis if parallel processing has to be exploited for the design of turbo decoders. In this section, an extension of the EXIT chart method is proposed in order to take into account the constraints introduced by parallel turbo decoder architectures. The corresponding analysis associated with Monte-Carlo simulations gives additional understanding of the convergence process for parallel turbo decoder architectures. The results presented in this section have been published in [90].

In section 2.3.1 we present the basic concepts related with the EXIT charts. Afterwards, in section 2.3.2, an extension of the EXIT chart method to analyze parallel turbo decoders is introduced. In section 2.3.3 parallel turbo decoder schemes are analyzed considering their transfer characteristics. Note that a modified schedule for shuffled turbo decoders is also proposed.

2.3.1 THE EXIT CHARTS

The EXIT charts are a useful kind of diagram proposed to analyze the convergence process of iterative decoding systems. It is assumed that both channel observations and extrinsic values can be modeled as conditional Gaussian random variables. Let L^c, L^e and L^a denote respectively the channel information (systematic and redundant bits), the extrinsic information and the a-priori information. Since an AWGN channel is

assumed, we can express the channel information as: $L^c = \frac{p(r|u=+1)}{p(r|u=-1)} = \frac{2}{\sigma_n^2}(u+\eta)$, with $\eta \sim \mathcal{N}(0, \sigma_n^2)$. Moreover, it can be expressed as follows:

$$L^c = \mu_c \cdot u + \eta_c \tag{2.15}$$

with $\mu_c = 2/\sigma_n^2$ and η_c being Gaussian distributed with mean zero and variance $\sigma_c^2 = 2 \cdot \mu_c$. This last equation corresponds to the consistency condition[6] [91]. By a similar analysis the next equation for the a-priori information can be derived:

$$L^a = \mu_a \cdot u + \eta_a \tag{2.16}$$

where $\sigma_a^2 = 2 \cdot \mu_a$. Hence, the consistency condition is also satisfied. Equation (2.16) is based on two observations:

- The a-priori values L^a can be assumed as uncorrelated from the channel observations L^c over many iterations.
- The extrinsic values L^e are Gaussian distributed.

Let I_a denote the mutual information between the information bits $\boldsymbol{d}$ and the a-priori information. Besides, let I_e be the mutual information between $\boldsymbol{d}$ and the extrinsic information L^e. The transfer function that defines the relation between I_a and I_e is denoted by:

$$I_e = Y(I_a, E_b/N_0) \tag{2.17}$$

The EXIT chart consists then in the plotting of I_e vs I_a for the natural and interleaved orders. For a sequential turbo decoder, since both constituent convolutional codes are identical, a transfer characteristics of only one domain can be plotted. Then, the transfer characteristics in the other domain is found by swapping the axes. In order to eliminate possible tail effects due to the code termination, a *long* information frame should be used during the computation of the mutual information.

2.3.2 EXIT CHART DIAGRAM EXTENSION

Figure 2.9 shows a scheme that illustrates the transfer characteristics computation as presented in [67], by Monte-Carlo simulation, considering a no parallel turbo decoder. In this figure, L^c and L^a are given by (2.15) and (2.16) respectively, and they satisfy the consistency condition. The SISO decoder receives L^c and L^a and performs the

[6] A Gaussian density distribution is say to be consistent iff its mean μ and variance σ^2 meet the equation: $\sigma^2 = 2 \cdot \mu$.

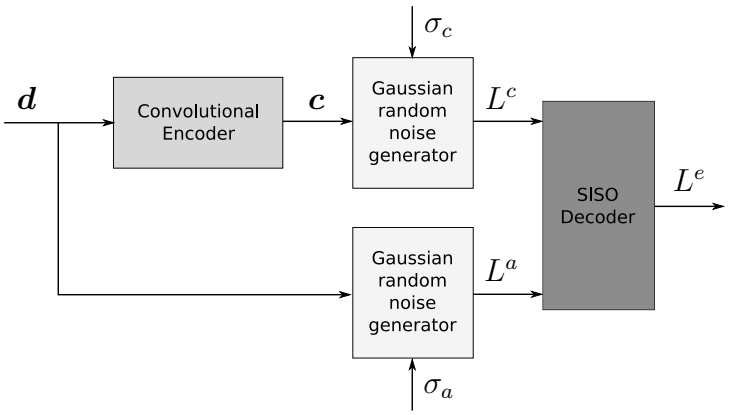

Figure 2.9: Scheme that illustrates the computation of the transfer function by Monte-Carlo simulation of a turbo decoder.

decoding operations in order to generate the extrinsic information L^e. Then, the mutual informations is calculated and the EXIT chart can be plotted.

In the following we consider that $W = N$, *i.e.* the sliding windows technique is not applied. Let us consider the turbo decoder architecture $\Phi_{1,NoSh}^{\xi,N_T}$, *i.e.* a turbo decoder with any SISO decoder schedule ξ and without any parallelism at the SISO decoder level. For this particular case, the mutual information can be computed as shown in Figure 2.9, with the transfer function in (2.17). For a shuffled turbo decoder $\Phi_{1,Sh}^{\xi_m,N_T}$ with a SISO decoder schedule $\xi_m \in \{B\text{-}F, B\}$ the transfer function in (2.17) holds as well. However, the structure for computing the transfer function should be modified. In [67] a method to generate EXIT charts of shuffled turbo decoders has been introduced. Three Gaussian random noise generators are used: one for the channel information and two others to generate the a-priori information. Thus, in Figure 2.9, we can no longer consider only one SISO decoder. We have to replace it by the whole shuffled turbo decoder.

Regarding, the sub-block parallelism in shuffled or not shuffled schemes, (2.17) is not valid anymore. Similarly, this equation cannot be applied to schedules B-FW and B-R with a shuffled parallelism. For these configurations, some values have to be kept during each half iteration in order to execute the next half iteration. It means that the transfer function has to take into account the dependency on additional parameters.

Let $\Psi(\Phi, i)$ denote the state of a turbo decoder Φ at the beginning of the half iteration i. For a sub-block parallelism, $\Psi(\Phi, i)$ represents the initial state metric values α and β in the limits of the sub-blocks (dots • in the schedule diagrams). If schedules B-FW or B-R are considered, $\Psi(\Phi, i)$ also symbolizes α and β values inside the sub-blocks at the beginning of the decoding process of each half iteration (height of the shaded area at the beginning of each half iteration in Figures 2.6 and 2.7). Thus, the

expression of the transfer function becomes:

$$I_e = Y(I_a, E_b/N_0, \Psi(\Phi, i)) \tag{2.18}$$

Figure 2.4 depicts the transfer of information in a parallel turbo decoder architecture. For each SNR value, we obtain a set of transfer curves that depends on the half iteration i. Thus, the transfer function between the input and the output of the SISO decoders in each order depends on the initial turbo decoder state $\Psi(\Phi, i)$ at the beginning of the half iteration i. This initial state corresponds to the final turbo decoder state at the end of the iteration $i-1$. We can then find the mutual information as presented in the algorithm 2. This process is repeated a sufficient number of times. Afterwards, one point in the EXIT chart, corresponding to the mutual input information I'_a, is found by taking the average mutual information at the output for each iteration i. Taking the average of the mutual information is justified. It was also done in previous works [51,92].

Algorithm 2 - Mutual information computation

for a particular mutual information I'_a **do**
  Set the initial conditions $\Psi(\Phi, 0)$ (α and β values are set to equiprobable values)
  for $i = 0$ **to** maximum number of iterations **do**
    Generate a-priori information corresponding to a mutual input information I'_a
    Perform a half turbo decoder iteration
    Compute mutual information I_e
    Set $\Psi(\Phi, i+1)$ to the actual turbo decoder state
  end for
end for

Recall that the EXIT chart method assumes that the extrinsic values produced by the SISO decoders have a Gaussian probability density distribution that satisfies the consistency condition. This Gaussian distribution assumption is valid for large frames. However, for short frames, the Gaussian distribution does not hold. In [93] it was observed that for short frames the extrinsic values have a distribution with two modes. This not Gaussian distribution is more evident for SNR values in the water-fall region and for a high number of iterations. Nevertheless, in [93] it is observed that for each channel realization the extrinsic values have a roughly Gaussian histogram with mean and variance that depend on the seed used to generate the channel noise values. Besides, applying a normalization technique, it is shown how the extrinsic values for a channel realization can be considered a realization sequence of a Gaussian random variable with the consistency property. Therefore, the transfer characteristics have to be defined for each seed that defines the channel realization. Considering these observations, in [94]

the EXIT band charts are proposed. These EXIT charts are composed of a band that contains the transfer characteristics for all the possible channel realizations. The band is depicted by using the mean and the standard deviation of the transfer characteristics.

Thus, when sub-blocks parallelism is considered, with shuffled or non-shuffled parallelism, the Gaussian assumption is not very accurate. However, each sub-block can be considered as a SISO decoder working on a short frame, where the Gaussian assumption holds for each channel realization. Thus, the EXIT charts that we propose, correspond to the mean values of the EXIT band charts proposed in [94].

Figure 2.10(a) shows the extrinsic information transfer characteristics for several iterations of the LTE turbo decoder with parallelism $\Phi^{B,1}_{128,NoSh}$ at $E_b/N_0 = 1dB$, for a frame size $L = 6144$ and a code rate $R = 1/2$. During the first iteration, the mutual information at the output is never equal to one, not even when the mutual information at the input is maximal. Iteration after iteration, the transfer characteristics improve so that the turbo decoder converges. When few iterations are performed, the metrics in the limits of the sub-blocks after each iteration ($\Psi(\Phi^{B,1}_{128,NoSh}, i)$) are not good enough. Hence, even with a perfect knowledge of the information to decode ($I_a = 1$) at the input of the SISO decoders, the decision taken is wrong ($I_e < 1$). As more iterations are executed, better values for $\Psi(\Phi^{B,1}_{128,NoSh}, i)$ are obtained, which enables the turbo decoder to converge.

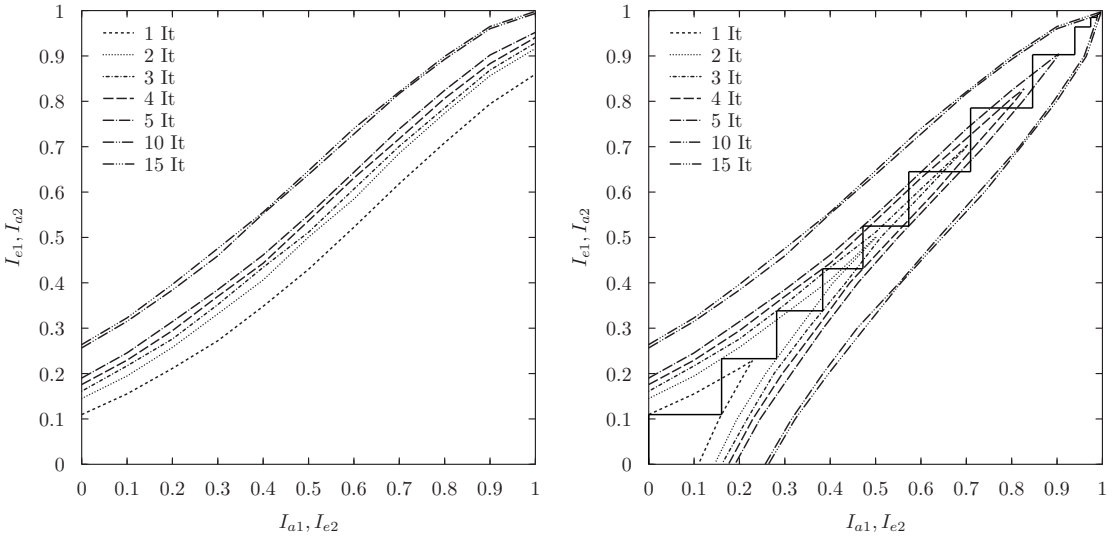

Figure 2.10: (a) EXIT chart for a SISO decoder. (b) Decoding trajectory. (Parallelism $\Phi^{B,1}_{128,NoSh}$, $E_b/N_0 = 1dB$, $L = 6144$ bits).

The decoding trajectory for the turbo decoder $\Phi^{B,1}_{128,NoSh}$ is given in Figure 2.10(b). For convenience the turbo decoder characteristics are only shown for some particular iterations. Each transfer curve is plotted up to its first intersection. The decoding trajectory is thus plotted by "jumping" between the different transfer curves as the iterations are carried out. Thus, the EXIT chart or decoding trajectory of any parallel turbo decoder Φ can be plotted. In this way, convergence of any parallelism scheme can be analyzed.

2.3.3 EXIT CHART BASED ANALYSIS

Let us consider the reference architecture $\Phi_r = \Phi^{B,1}_{1,NoSh}$. The BER performance of this architecture after $I(\Phi_r) = 8$ iterations is taken as reference for an unconstrained turbo decoding process. This performance is within 0.1dB compared to that achieved by an ideal turbo-decoding algorithm compliant with the LTE standard. Thus, any parallel turbo decoder Φ has to achieve similar BER performance after $I(\Phi)$ iterations.

2.3.3.1 SUB-BLOCKS PARALLELISM

The convergences of the turbo decoders Φ_r, $\Phi^{B,1}_{32,NoSh}$, $\Phi^{B,1}_{128,NoSh}$ and $\Phi^{B,1}_{512,NoSh}$, at $SNR = [0.75dB, 1dB]$, are shown in Figure 2.11. For $SNR = 1dB$, the number of iterations required for no BER performance degradation are $I(\Phi^{B,1}_{32,NoSh}) = 9$, $I(\Phi^{B,1}_{128,NoSh}) = 11$, $I(\Phi^{B,1}_{512,NoSh}) = 20$, for each parallel decoder, respectively. Therefore, the sub-blocks technique reduces the decoder convergence, and thus, it is necessarily to execute more iterations to overcome the degradation that is introduced. However, since the duration of an iteration is reduced when more sub-blocks are employed, it is still possible to increase the turbo decoder throughput.

EXIT charts of the different parallel turbo decoders for whom convergence has been shown are presented in Figure 2.12. The EXIT chart at $SNR = 0.75dB$ of the architecture Φ_r is depicted in Figure 2.12(a). For this SNR value, the diagram just starts to open. It corresponds to the beginning of the water-fall region. The decoding trajectory of the turbo decoder $\Phi^{B,1}_{512,NoSh}$ for different iterations is plotted as well. Note that additional iterations are necessary. Figure 2.12(b) shows the decoding trajectories of turbo decoders with sub-block parallelism for $SNR = 1dB$. The decoding trajectory of $\Phi^{B,1}_{32,NoSh}$ is very close to the one that would follow Φ_r. For thirteen iterations, the BER performance of $\Phi^{B,1}_{512,NoSh}$ is worse than that of $\Phi^{B,1}_{32,NoSh}$ with six iterations, which is also worse than that of $\Phi^{B,1}_{128,NoSh}$ with eight iterations. Coherent results between Figure 2.11 and Figure 2.12(b) confirm the correctness of the decoding trajectories that have been plotted as described in section 2.3.2.

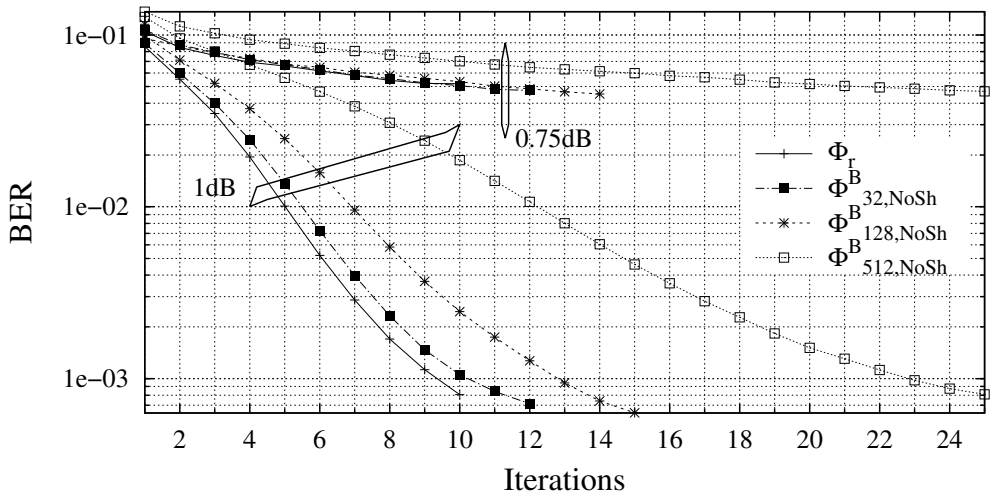

Figure 2.11: Convergence of turbo decoders with sub-block parallelism ($L = 6144$ bits).

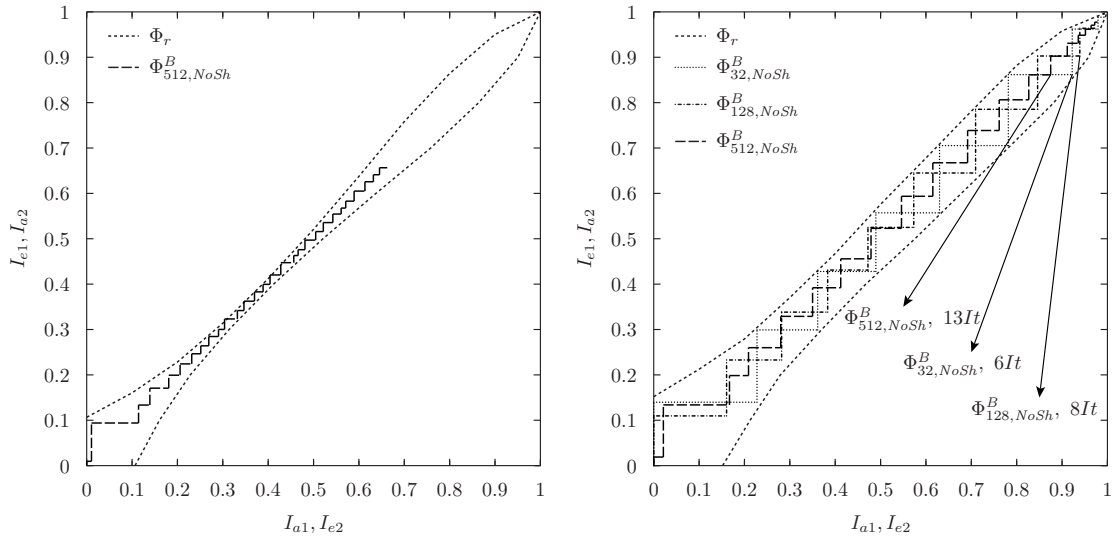

Figure 2.12: a) Decoding trajectory for $\Phi^{B,1}_{512,NoSh}$ at $0.75dB$. b) Decoding trajectories for 32, 128 and 512 sub-blocks no shuffled turbo decoders.

2.3.3.2 SHUFFLED PARALLELISM

Convergence of shuffled turbo decoders $\Phi^{\xi_n,1}_{1,Sh}$ with schedule $\xi_n \in \{B, B-FW, B-R\}$ is shown in Figure 2.13. *B-R* schedule reduces significantly the number of iterations

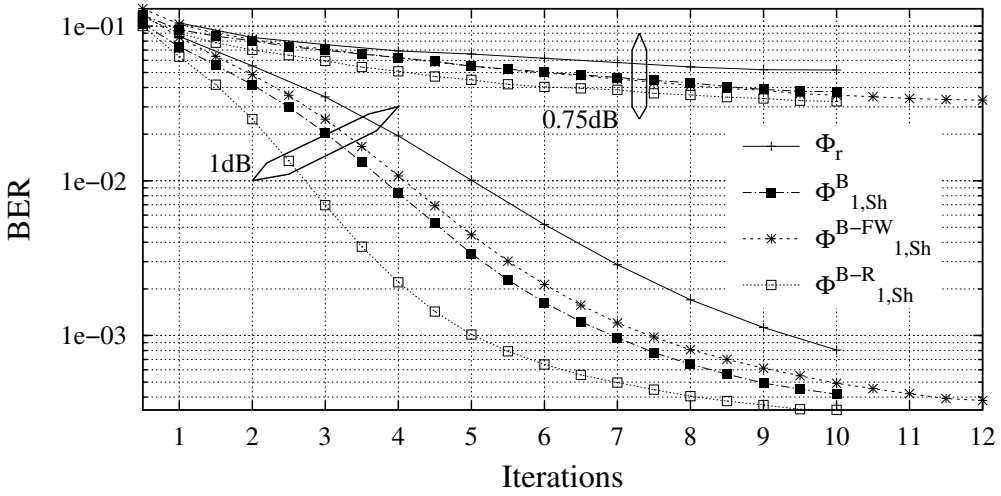

Figure 2.13: Convergence of shuffled turbo decoders ($L = 6144$ bits).

(almost to the half with respect to Φ_r). *B* and *B-FW* schedules have similar behaviors. However, the latter needs about one additional half iteration. For low SNR values both schedules behave almost identically.

Shuffled turbo decoders EXIT charts for $SNR = 1dB$ are presented in Figure 2.14(a). EXIT chart of $\Phi_{1,Sh}^{B,1}$ is wider than the EXIT chart of Φ_r. For the B-R schedule the decoding trajectory exceeds that of the Butterfly schedule in shuffled mode ($\Phi_{1,Sh}^{B,1}$). Even if $\Phi_{1,Sh}^{B,1}$ and $\Phi_{1,Sh}^{\text{B-R}}$ behave similarly after the first half iteration, the decoding trajectory of the *B-R* schedule moves forward faster. $\Phi_{1,Sh}^{\text{B-FW},1}$ presents the worst characteristics to reduce the number of iterations. This occurs mainly due to the poor behavior during the very first half iteration. Since during this half iteration β values of the first half of the sub-block are unknown (Figure 2.6), the a-posteriori values generated during the decoding of this part of the sub-block are not appropriate. After the first half iteration, all values α and β of the sub-block are properly calculated, and the decoding trajectory of the schedule *B-FW* tries to approach that of the EXIT chart of $\Phi_{1,Sh}^{B,1}$. However, the decoding trajectory cannot reach that of the Butterfly schedule and keeps at about one half iteration lower. This behavior does not appear for $\Phi_{1,Sh}^{\text{B-R},1}$. Indeed for the schedule B-R, the a-posteriori values are computed with appropriate α and β values calculated during each half iteration.

Since the EXIT chart (or the corresponding trajectory) for the shuffled architecture is wider than the EXIT chart of the reference architecture, shuffled turbo decoders reach the region of waterfall at lower SNR values with respect to non-shuffled turbo decoders. Figure 2.14(b) illustrates how the decoding trajectory with schedule B-R can

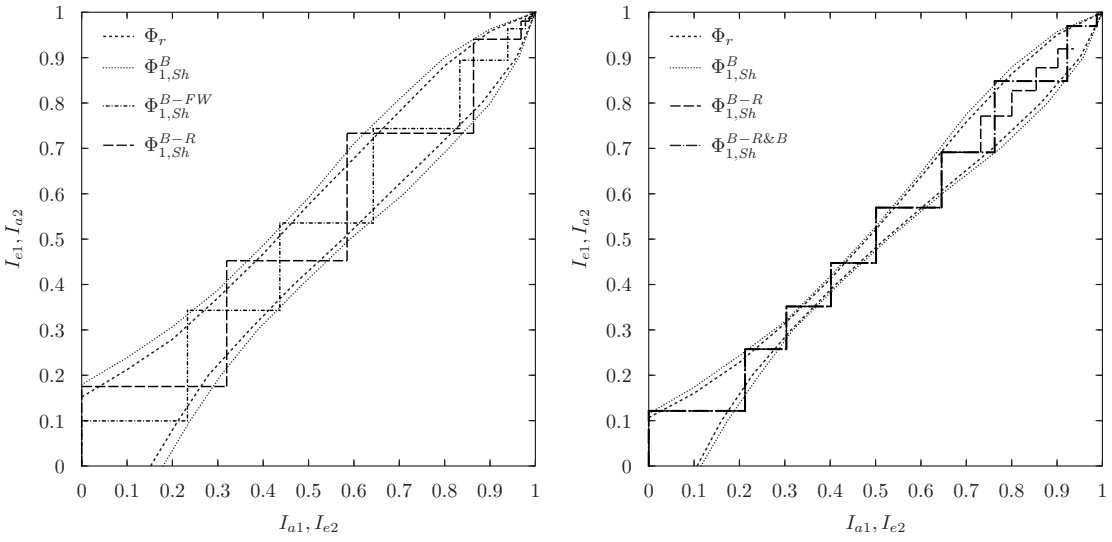

Figure 2.14: a) Decoding trajectories for shuffled turbo decoders at $1dB$. b) Decoding trajectories for schedules *B-R*, *B* and a combination of both at $0.75dB$.

easily converge where the EXIT chart of Φ_r becomes tight. Even more, for $\Phi^{\mathsf{B},1}_{1,Sh}$ the EXIT chart is slightly wider than the EXIT chart of Φ_r. It can be seen however that the decoding trajectory of $\Phi^{\mathsf{B\text{-}R},1}_{1,Sh}$ narrows for high mutual input information.

When the schedule B-R is applied, at the beginning of each half iteration, α and β values inside the sub-block correspond to the initial state of the turbo decoder. If those values are not enough accurate, they might prompt decoding errors even though the mutual information at the input of the SISO decoder is high. This behavior is similar to the one observed in Figure 2.10(a), where not so good α and β values in the sub-block limits lead to errors in the early iterations for high input mutual information. No similar result is observed for $\Phi^{\mathsf{B},1}_{1,Sh}$ since no initial values are kept from iteration to iteration. By taking into account this result, a new schedule can be proposed. It consists in employing B-R schedule during the early iterations, and then switching to Butterfly schedule. The decoding trajectory obtained for this new schedule (B-R&B) is presented in Figure 2.14(b) where after 5 iterations the schedule is switched from B-R to B. Indeed, during the first 5 iterations the decoding trajectories of $\Phi^{B-R,1}_{1,Sh}$ and $\Phi^{B-R\&B,1}_{1,Sh}$ are similar. Thus, fast convergence can be achieved, and since the hardware resources are the same for both schedules, no additional hardware complexity is required.

Butterfly-Replica schedule in shuffled iterative receivers As a collaboration work with Salim Haddad, a Ph.D student at the electronic engineering department of Telecom Bretagne, we have studied the convenience of using the B-R schedule in shuffled iterative receivers. The results of this work have been published in [95]. We consider a full shuffled receiver. It consist in a shuffled sub-block turbo decoder with shuffled iterative demapping. In this case, when the B-R schedule is implemented by the SISO decoders, a reduction in the number of arithmetic operations is observed with respect to the B schedule. Details of this work are no longer discussed in this document. The reader is referred to [95] for additional information.

2.4 EXPLORING SOVA BASED TURBO DECODERS

In this section, we present a general view of the SOVA as SISO decoder algorithm for turbo decoder architectures. The main motivation to consider the SOVA is its lower hardware complexity compared to Max-Log-MAP algorithm implementations. Thus, for a given area budget, a larger number of SOVA based SISO decoders could be implemented compared to the number of Max-Log-MAP based SISO decoders that would fit in the same area. In this way, a higher sub-block parallelism degree would be possible. Considering a binary code ($m = 1$), the Max-Log-MAP algorithm exhibits about twice the complexity of the SOVA [96]. However, for double binary codes ($m = 2$), the SOVA is not attractive from a hardware complexity point of view since its hardware complexity is comparable to the Max-Log-MAP algorithm implementations [97].

Since the introduction of the SOVA, different works have been carried out in order to design VLSI SOVA based SISO decoders [28, 98, 99]. However, the main drawback of these works in practical applications is the poor BER decoding performance. Indeed, a degradation of 0.7 dB or more for the SOVA based turbo decoders compared to MAP based turbo decoders is classically observed.

In this section we present the results obtained in order to improve the decoding performance of SOVA based turbo decoders, so that they can be considered as a valid alternative for practical implementations. We explore binary and double binary turbo decoders. At the beginning of the section we briefly present the SOVA decoder structure. Then, the approach selected to improve the error correction performance is shown.

2.4.1 OVERVIEW

The architecture of a VA decoder is composed of three main blocks as presented in Figure 2.15. The BMU computes the state metric transition $M^{\gamma}(\cdot,\cdot)$ between each pair of states in the trellis diagram. The ACS unit decides for the survivor path at each

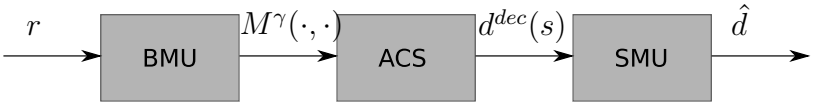

Figure 2.15: VA block diagram.

state s and outputs the corresponding decision bits $d^{dec}(s)$. These decisions are then processed by the Survivor Memory Unit (SMU) which can find the paths associated to each state. Then, it traces back all the paths until they all merge together.

A SOVA decoder can be built with the same structure of a hard decoding VA. However, it is necessary to add to the SMU the update process of soft information as described in section 1.3.1.2. Besides, the ACS unit has to compute the path metric difference $\Delta_k(s)$, that is taken as the initial reliability value. Since this path metric difference is already computed in order to choose the best path associated with each state, no additional hardware resources are required.

2.4.1.1 SMU IMPLEMENTATION

Two algorithms have been proposed for the design of the SMU: Register Exchange Algorithm (REA) and Traceback Algorithm (TBA). The first one enables to increase the throughput, while the second one is adopted for low power designs [61]. The TBA was originally proposed in [100], where an infinite memory is considered. Therefore, some modifications have been proposed for realistic implementations. In [101] several TBA methods are proposed. They execute three types of operations:

- *Traceback Write New Data* (*TWR*): New data from the ACS unit is saved in the traceback memory. A write pointer controls the free memory positions where new decision bits can be stored.
- *Traceback Read* (*TR*): A bit from the memory is read and, together with the present state, a pointer to the previous state is found. No decision bits are generated.
- *Traceback Decode Read* (*TDR*): In a similar way to *TR*, this operation executes a traceback operation but in older data. This operation generates decoded information bits.

In [101–103] four versions of the TBA are proposed. They are based on a set of memory banks organized as a matrix-like structure. In [104] a pipeline TBA architecture based on a REA architecture is introduced.

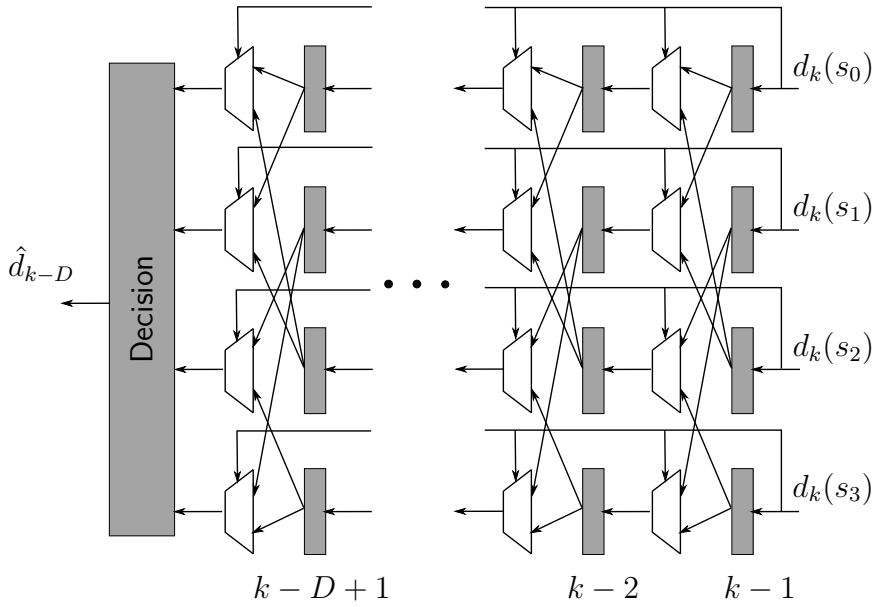

Figure 2.16: REA method circuit.

The REA algorithm can be implemented as a set of registers and multiplexers following the topology of the trellis diagram. A global clock controls all the register. Therefore, the system throughput is determined by the clock frequency. The registers and multiplexers are organized as a matrix-like structure composed of 2^p rows and D columns[7], as presented in Figure 2.16 for a 4-state convolutional code. D is called the survivor depth. Thus, there is a high probability that all the survivors merge, and the symbol of any path that is build back at a depth D can be chosen as the ML path. The memory requirements for the REA algorithm are then $D \cdot 2^p$ bits.

2.4.1.2 SOVA Implementation

Figure 2.17 present a general block diagram to implement the SOVA [28, 98]. The BMU and ACS unit perform the forward recursion (Figure 1.14). Then, a hard decoding SMU execute the Traceback operation to find the state that belongs to the ML path at time $k - D$. Finally, the Update operation of the soft output values takes place. This operation is executed by finding the surviving and concurrent paths considering the state s_{k-D}. When the Update operation is finished, the soft value for the information symbol

[7]Recall that p is the number of flip-flops in the corresponding convolutional encoder, and D the Traceback length in the Viterbi Algorithm.

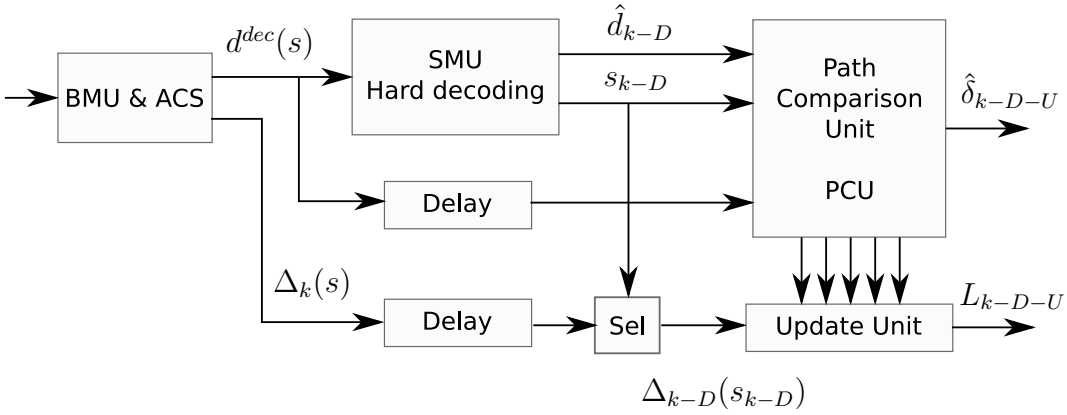

Figure 2.17: SOVA architecture.

at time $k - D - U$ is found. Note that two delay blocks (FIFO) of length D are also necessary to provide the correct values from the ACS unit to the blocks that perform the Update operation.

2.4.2 Improving the SOVA Based Turbo Decoder Performance

SOVA Turbo Decoder Performance for Binary Codes

Two factors are responsible of the low decoding performance of the SOVA [105,106]:

- The SOVA generates too optimistic reliability values.
- There is an inherit correlation between the intrinsic information (SOVA input) and the extrinsic information.

In the literature, several works have addressed these problems. In [106] an attenuator to SOVA output, in order to reduce the distortion produced in the soft values, is applied. In [107], too optimistic extrinsic values are limited by establishing a maximum value. In [61] an attenuator to SOVA output associated with an adaptive upper bound for the metric difference between competing paths is proposed. The attenuator value is a simplification of the method proposed in [106]. In [105] two attenuators are used to reduce the correlation between intrinsic and extrinsic information. Between all these works, those that present the best results are [61] and [105]. Both of them are based on empirical parameters and approximations. Based in these works we have adapted a

technique to improve the error correction performance of SOVA based turbo decoders. It is described as follows.

From (1.69), the SOVA extrinsic information can be computed as:

$$L_k^e(\delta) = L_k(\delta) - L_k^i(\delta) \tag{2.19}$$

where L_k^e and L_k are the extrinsic and a-posteriori information. L_k^i denotes the intrinsic information related to the systematic and a-priori information. For an AWGN channel, L^e and L^i follow approximated Gaussian distributions [26], and are Gaussian correlated [105]. This correlation is claimed to be one of the reasons that explain the poor SOVA behavior. Table 2.2 shows the mean value of the correlation for the Max-Log-MAP and SOVA algorithms obtained after simulating a high enough number of frames. Different iterations are considered.

Iterations	Max-Log-MAP				SOVA			
	SNR (dB)				SNR (dB)			
	0.75	1	1.25	1.5	0.75	1	1.25	1.5
1	0.001	0	0.001	0.003	0.364	0.353	0.287	0.257
3	0.011	0.016	0.018	0.019	0.438	0.337	0.277	0.252
8	0.018	0.028	0.033	0.031	0.440	0.360	0.242	0.253
10	0.019	0.030	0.030	0.027	0.462	0.392	0.324	0.203

Table 2.2: Correlation coefficient between intrinsic and extrinsic information for the Max-Log-MAP and SOVA algorithms.

In order to reduce the correlation coefficient in the SOVA, (2.19) is transformed into (2.20), where $L_k^{e'}$ is the new extrinsic information.

$$L_k^{e'}(\delta) = b_1 \cdot \Big(b_2 \cdot L_k(\delta) - L_k^i(\delta)\Big) \tag{2.20}$$

In this equation b_1 and b_2 are variables used to reduce the correlation between extrinsic and intrinsic information. They depend on each frame to decode. However, we approximate these variables with constants values. From extensive simulations we have found that values around $b_1 = 0.7$ and $b_2 = 0.9$ are optimal for different frame sizes and code rates. Table 2.3 presents the correlation coefficient between $L_k^{e'}$ and L_k^i when the variables b_1 and b_2 are implemented. The LTE turbo code is considered with a frame size of $L = 2048$ bits.

The correlation coefficient decreases specially for a high number of iterations and SNR value. The proposed technique significantly improves the error correction performance of SOVA turbo decoders in the waterfall region. Degradations of about 0.1dB with

Iterations	SNR in dB			
	0.75	1	1.25	1.5
1	0.269	0.246	0.194	0.163
3	0.305	0.254	0.174	0.171
8	0.281	0.153	0.041	0.027
10	0.210	0.086	0.038	0.020

Table 2.3: Correlation Coefficient between intrinsic and extrinsic information for SOVA algorithm. $b_1 = 0.7$ and $b_2 = 0.9$.

respect to Max-Log-MAP algorithm can be achieved for BER of 10^{-5}. However, a bad convergence in the error floor region is observed. For high SNR values (error floor region), extrinsic values grow fast during the iterations, and thus, it is highly probable that the turbo decoder converges to a particular codeword in the very first iterations of the decoding process. If the SOVA algorithm is used, this problem is more critic. Thus, a high error floor may appear. To overcome this problem, we have noticed that an effective solution is to limit the path metric difference between the survivor and the concurrent paths. Furthermore, this limit can be dynamic. Thus, during the first iteration the path metric difference is upper bounded to a given value that is increased as more iterations are performed.

Figure 2.18 shows the BER for the LTE turbo code with a frame size $L = 2048$ bits. There is a degradation of about 0.2dB between SOVA and Max-Log-MAP for a BER$= 10^{-7}$ when 6, 8 and 10 iterations are performed. In the error floor region, the degradation is about 0.25dB.

SOVA Turbo Decoder Performance for Double Binary Codes

In a similar way to the SOVA for binary codes, we have explored the improvement of the error correction performance for double binary codes. The use of the coefficients b_1 and b_2 is extremely important to achieve performance close to those of MAP based algorithms. Figure 2.19 shows the BER performance for the SOVA and Max-Log-MAP algorithms, considering the double binary turbo code with a frame size of 5376 bits. The code rate is set to $R = 1/3$. In this particular case SOVA has a degradation within 0.1dB with respect to the Max-Log-MAP algorithm. The curves are potted in the waterfall region, where the SOVA exhibits and adequate performance. We expect a larger degradation when arriving to the error floor region.

Table 2.4 and 2.5 show the correlation coefficient between intrinsic and extrinsic information for each information symbol, when $b_1 = b_2 = 1$ and $b_1 = 0.7$, $b_2 = 0.9$,

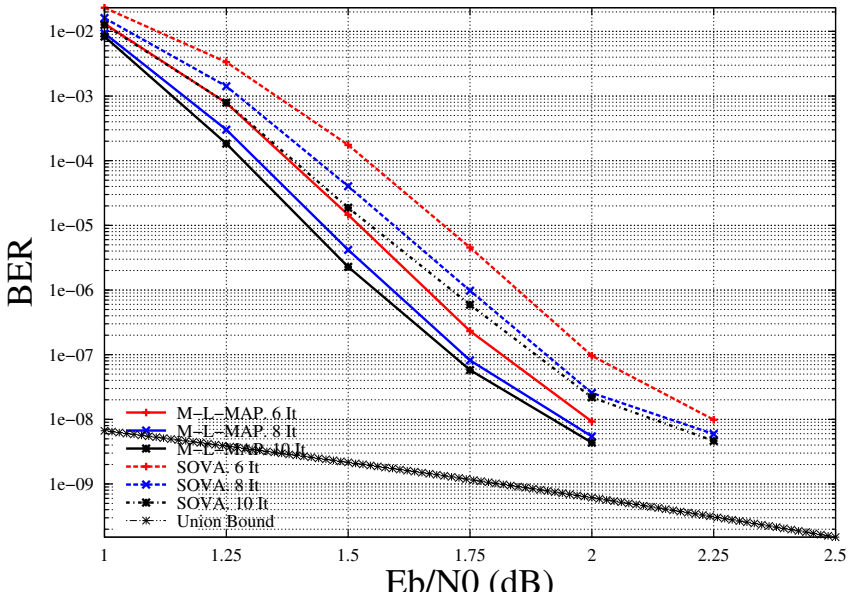

Figure 2.18: BER performance for the LTE turbo code using SOVA and Max-Log-MAP. 2048 bits per frame.

It.	SNR in dB											
	0.2			0.3			0.4			0.5		
	d_k			d_k			d_k			d_k		
	01	10	11	01	10	11	01	10	11	01	10	11
1	0.218	0.228	0.196	0.228	0.249	0.185	0.196	0.242	0.168	0.202	0.235	0.198
3	0.380	0.353	0.200	0.268	0.283	0.273	0.348	0.313	0.201	0.323	0.305	0.240
8	0.315	0.332	0.244	0.386	0.292	0.316	0.319	0.358	0.190	0.299	0.324	0.221
10	0.346	0.349	0.266	0.338	0.273	0.217	0.339	0.331	0.226	0.222	0.297	0.250

Table 2.4: Correlation Coefficient for double binary SOVA between intrinsic and extrinsic information for each possible symbol (d_k) at different SNR. $b_1 = b_2 = 1$.

respectively. In this case, the use of b_1 and b_2 significantly reduces the correlation coefficient. For all the 3 symbols the correlation coefficient is similar for each iteration at different SNR values.

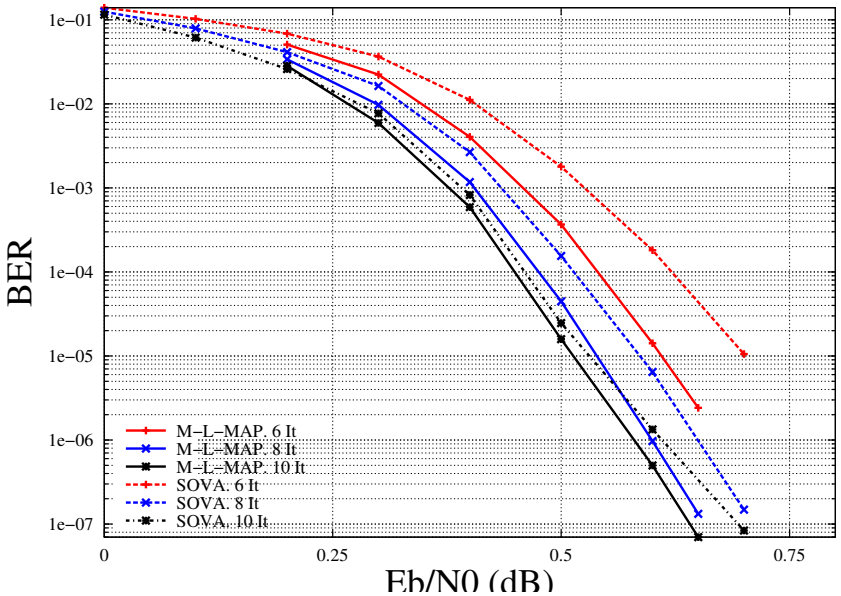

Figure 2.19: BER and FER for a double binary turbo code using SOVA and Max-Log-MAP. $L = 5376$ bits per frame. Code rate $R = 1/3$.

It.	SNR in dB											
	0.2			0.3			0.4			0.5		
	d_k			d_k			d_k			d_k		
	01	10	11	01	10	11	01	10	11	01	10	11
1	0.114	0.100	0.124	0.108	0.090	0.129	0.096	0.075	0.102	0.079	0.085	0.095
3	0.158	0.136	0.050	0.123	0.110	0.022	0.075	0.067	0.023	0.035	0.055	0.029
8	0.023	0.073	0.017	0.081	0.038	0.033	0.072	0.023	0.050	0.080	0.015	0.043
10	0.038	0.016	0.050	0.065	0.025	0.062	0.093	0.027	0.042	0.102	0.023	0.049

Table 2.5: Correlation Coefficient for double binary SOVA between intrinsic and extrinsic information for each possible symbol (d_k) at different SNR. $b_1 = 0.7$, $b_2 = 0.9$.

2.5 CONCLUSION

This chapter presents a complete overview of the parallel turbo decoder techniques considering three hierarchical levels: turbo decoder level parallelism, SISO decoder level parallelism, and metric level parallelism. A set of metrics have been defined in order to evaluate parallel turbo decoder architectures. The EXIT chart diagram have been extended in order to be able to study the parallel turbo decoder architecture convergence.

Thanks to this study, we were able to propose a new SISO decoder schedule for the BCJR algorithm, convenient to increase the convergence of shuffled turbo decoders. Finally, we have presented a brief overview of the SOVA based turbo decoders. We have considered the main problem in SOVA based decoders, *i.e.* the poor decoding performance. A technique to reduce the performance degradations have been tested. Thus, a degradation of about 0.2dB is observed for binary and double binary turbo codes in the water fall region.

Chapter 3

High Throughput SISO Decoder Architectures

As presented in the previous chapters, the SISO decoder is a fundamental part of a turbo decoder architecture. It implements the operations that are the heart of the decoding algorithm, and thus, it largely determines the achievable throughput and overall turbo decoder hardware complexity. In this chapter, the architectures of different constituent SISO decoder blocks are detailed.

At the SISO decoder parallelism level, the sub-blocks technique has shown to be an appropriate alternative to increase the turbo decoder throughput with an additional cost in terms of the hardware complexity. Thus, for a low number of sub-blocks Q, it is possible to achieve an acceleration close to Q with respect to a non parallel architecture. However, as the number of sub-blocks grows, the achievable throughput is limited by the additional number of iterations that should be executed in order to avoid a performance degradation. In the context of high throughput turbo decoders, architectures with a high number of sub-blocks should be considered, where a further increase of Q is not possible. In this case, parallelism techniques at the metric level, that do not affect the turbo decoder convergence, can be used. For this reason, radix-2^{N_T} architectures are explored in this chapter. We demonstrate that the radix technique is useful to increase the SISO decoder throughput by reducing the sub-block decoding time.

This chapter is organized as follows. First, an overview of the general SISO decoder architecture is given. Afterwards, the structure of the different units in a radix-2 SISO decoder is detailed. Then, higher radix architectures are analyzed. Finally, a low complexity radix-16 SISO decoder architecture is proposed. This architecture achieves high throughput values, increasing the radix technique efficiency with respect to conventional high radix decoder implementations.

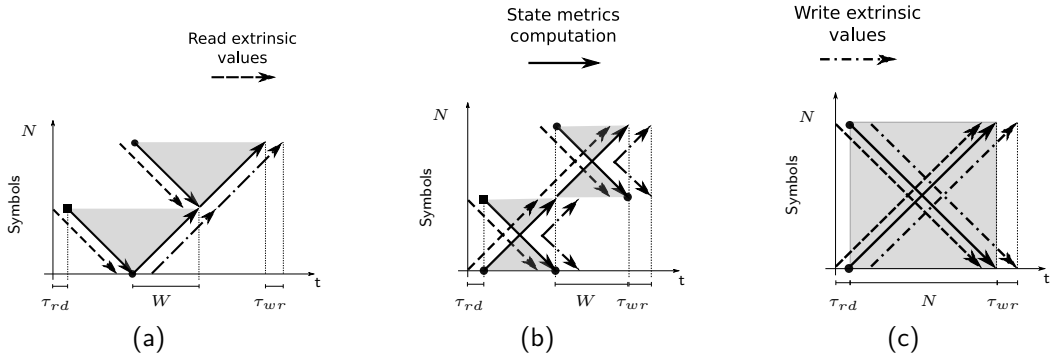

Figure 3.1: SISO decoder architecture. MAP algorithm schedules: (a) Backward-Forward, (b) Butterfly, (c) Butterfly-Replica.

3.1 OVERVIEW

Since hardware implementations are required, a fixed-point representations of the different variables in the SISO decoder algorithm is necessary. This fixed-point representation has been extensively studied in the literature [35, 108–111], where a tradeoff between the decoding performance and the hardware complexity is established. In this document, for the Max-Log-MAP based SISO decoders, w_r, w_{SM} and w_{ext} represent the number of bits required to represent the channel information, the state metrics (M^{α} and M^{β}), and the extrinsic information, respectively. These parameters have an important impact on the hardware complexity due to the cost in terms of computation logic. Besides, they are a major constraint on the size of the overall memory in the turbo decoder.

The implementation of the Max-Log-MAP algorithm implies the execution of three main operations: branch metric, state metric and extrinsic values computation. Thus, a high throughput implementation assigns hardware resources to execute concurrently these operations. Figure 3.1 shows the SISO decoder schedules that we consider. Backward-Forward and Butterfly schedules are presented in Figures 3.1(a) and 3.1(b) respectively. In these cases, the sliding window technique is considered with a window size W. Butterfly-Replica schedule is depicted in Figure 3.1(c). Since Butterfly-Forward schedule has not a good convergence behavior, as explained in section 2.3, it is not considered. These schedule diagrams explicitly consider the time required to provide to the ACS units the values to perform the state metric recursions, and the time spent to compute the extrinsic information once M^{α} and M^{β} are available. Let τ_{rd} denote the number of clock cycles necessary to read the a-priori information and to compute the state metric transitions. Besides, τ_{wr} represents the pipeline latency to compute the extrinsic information and write it to the corresponding memory bank outside the SISO

decoder. τ_{rd} and τ_{wr} model the pipeline stages in the different SISO decoder units and in the network assigned to access the extrinsic information.

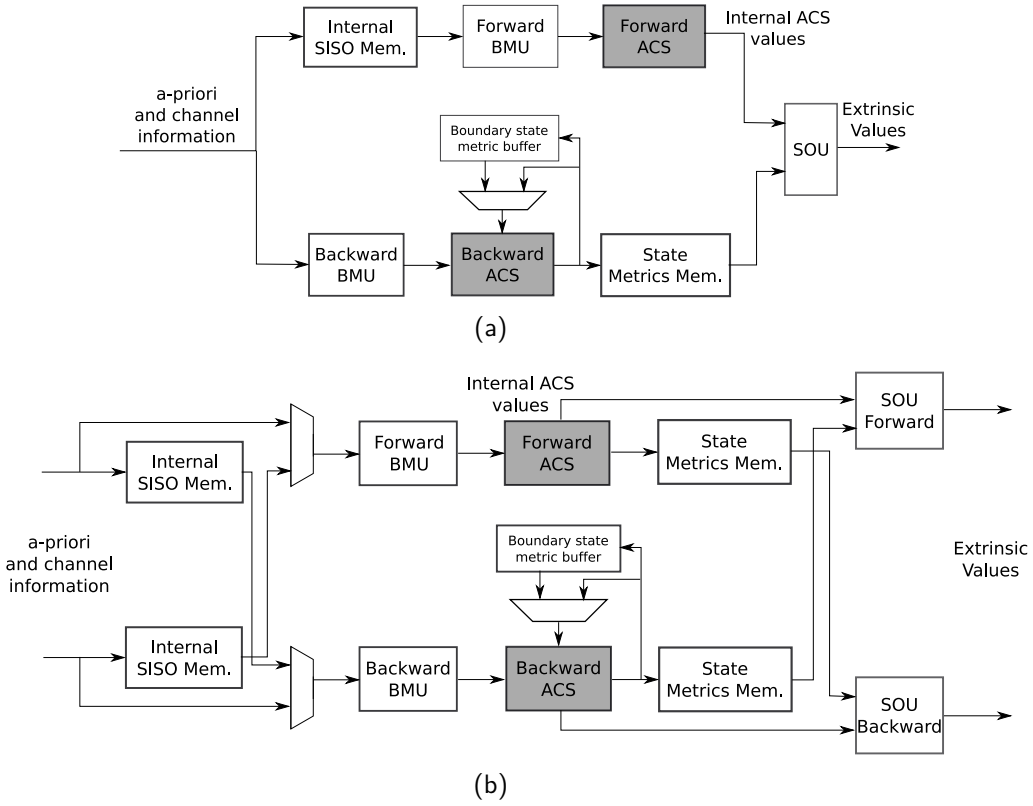

Figure 3.2: SISO architectures: (a) Backward-Forward schedule, (b) Butterfly and Butterfly-Replica (boundary state metric buffer not required) schedules.

Figure 3.2(a) shows the SISO block diagram for the Backward-Forward schedule. Butterfly and Butterfly-Replica schedules can be implemented with the block diagram in Figure 3.2(b). When the sliding window technique is implemented, additional registers should be included in order to store the window boundary state metrics (squares ■ in Figures 3.1(a) and 3.1(b)). Note that since for the Butterfly-Replica schedule the state metrics for all the sub-block should be kept at the end of each half iteration, the sliding window technique cannot be implemented. In this case, the boundary state metric buffer should be removed from Figure 3.2(b). Internal SISO Mem is a buffer that temporally stores extrinsic values. This component enable to avoid a second access to the memory banks for each information symbol. Moreover, this buffer can alleviate collision problems since less memory accesses are necessary. The Branch Metric Unit (BMU) computes

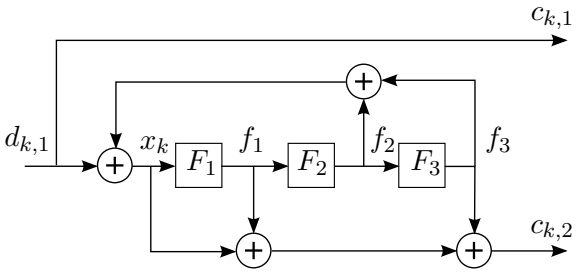

Figure 3.3: Recursive Systematic Convolutional (RSC) encoder (3,1,2) with generator polynomials $\left[1 \quad \frac{1+z^{-1}+z^{-3}}{1+z^{-2}+z^{-3}}\right]$.

metric transition values used by the Add Compare Select (ACS) unit to execute state metrics recursions. State Metric Memories enable to keep M^{α} and M^{β} values (shaded area in Figures 3.1(a), 3.1(b) and 3.1(c)). Finally, the Soft Output Unit (SOU) computes extrinsic values and takes hard decisions. It receives the state metric plus the metric transition values previously computed in the ACS unit. The different SISO decoder architecture blocks are pipelined in order to have the critical path in the ACS unit.

3.2 RADIX-2 SISO DECODER ARCHITECTURE

In this section, the architectures of the different SISO decoder blocks are detailed. To illustrate our propositions, we consider the 8-state binary turbo code compliant with the LTE standard, already considered in Chapter 1. For convenience, the structure of the constituent convolutional encoder is also given in this chapter in Figure 3.3.

Inputs to the SISO decoder are the channel information – systematic L_k^s and redundant L_k^r – and the a-priori information L_k^a. All of them are expressed as LLR values. The SISO decoder computes the extrinsic values L_k^e, and takes hard decisions $\hat{\boldsymbol{d}}_k$.

3.2.1 BRANCH METRIC UNIT

The BMU consists of a set of adders for computing the branch metrics of each possible transition in the trellis diagram representation of the code. For example, in the considered convolutional code, there are four different branches corresponding to the possible values of the coded symbol. Thus, the BMU can be implemented with the architecture proposed in Figure 3.4, where $M_k^{\gamma_{i,j}}$ is the branch metric for the coded symbol $(c_{k,0}, c_{k,1}) = (i, j)$.

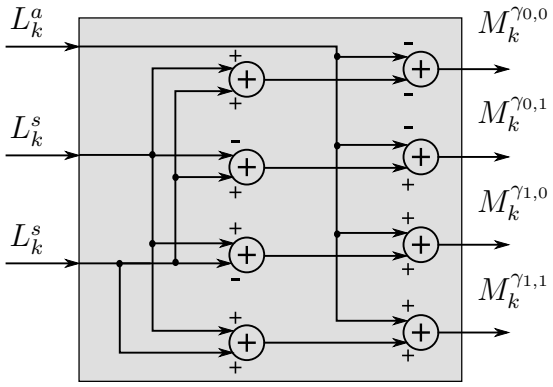

Figure 3.4: BMU radix-2 architecture.

3.2.2 ADD COMPARE SELECT UNIT

Regarding the equations that define the forward and backward state metric recursion, it is clear that the magnitudes of M^{α} and M^{β} increase during the recursive operations along the trellis diagram. Therefore, numeric problems may arise if these values are not bounded so that arithmetic overflows are avoided. M^{α} and M^{β} state metric values should be bounded to be represented with w_{SM} bits. This problem has been widely studied for the Viterbi algorithm [112, 113], where rescaling techniques have been proposed. In this case, when all the state metrics exceed a certain threshold, this threshold is subtracted to all the state metrics. Another rescaling technique consists in the substitution of the minimum state metric, at each time k, to all the state metrics. Note that these rescaling techniques do not affect the decoding performance. However, they are implemented inside the ACS unit, and thus, they have an impact on the ACS unit critical path. Since the ACS unit contains the turbo decoder critical path, the rescaling techniques are inconvenient to achieve high throughput values.

We have decided to avoid rescaling operations by applying the modulo normalization technique. This technique was first proposed for Viterbi decoders [113], and later extended to Log-MAP [114] and Max-Log-MAP [115] SISO decoders. Compared to solutions that employ renormalization techniques, the modulo normalization has proven to be effective in order to reduce the ACS unit critical path [78,115]. Moreover, the number of bits w_{SM} required to represent the state metric scales only logarithmically with the number of encoder states [112]. Thus, the decoder throughput is slightly affected when the number of states increases [116]. Therefore, high throughput decoders with a large number of states can be designed.

Modulo normalization technique is depicted in Figure 3.5. Two's complement representation is used for the state metric values. A state metric value is depicted as a

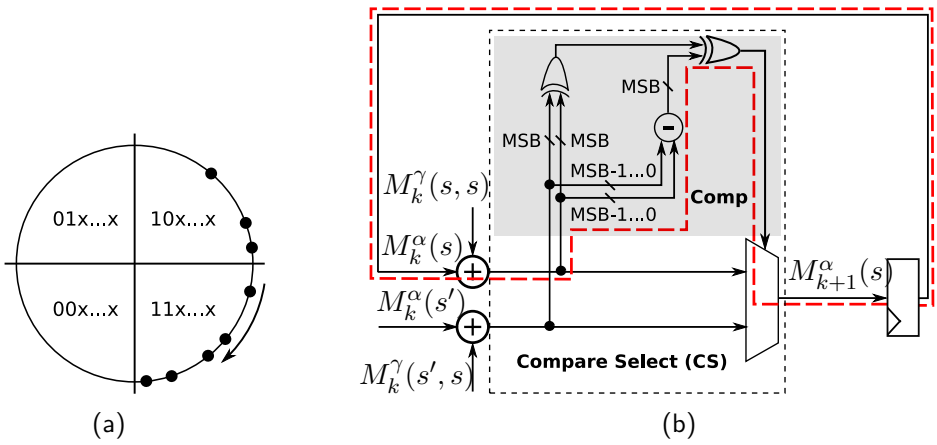

Figure 3.5: Modulo normalization technique for a 8 state binary code a) State metric evolution. b) ACS unit.

point on the circumference in Figure 3.5(a). As the transitions in the trellis diagram are executed, the state metrics move in the clockwise direction over the circumference. Each quadrant corresponds to a range of values determined by the two more significative bits of the state metrics. Since no normalization is performed, the state metric values overflow and pass from the quadrant 01x...x to the quadrant 10x...x. Modulo normalization technique is based in the fact that the maximum difference between all the state metrics at a given time is bounded [113]. Thus, if the number of bits w_{SM} is large enough, all the state metrics at a given time are in adjacent quadrants. In Figure 3.5(b), a radix-2 ACS unit architecture (binary code) implementing the normalization technique is presented. The adequate value is selected at each time, *i.e.* the state metric that has advanced the most over the circumference. In this scheme, the critical path is indicated with a dashed line. Compared with a conventional ACS unit without any normalization nor rescaling technique, the critical path is only increased by a two input *XOR* gate.

In order to reduce the ACS unit critical path, the addition and comparison operations can be performed in parallel. First, by reorganizing the operations in the design, the Compare Select Add (CSA) unit in Figure 3.6(a) can be build. With this architecture no improvements on the critical path is achieved. However, from this design, we can devise the architecture shown in Figure 3.6(b). In this case, the adders were moved before the multiplexer, so that the additions can be executed in parallel with the comparison. Thus, one adder is removed from the critical path. However, the number of adders, registers and multiplexors is increased in the architecture.

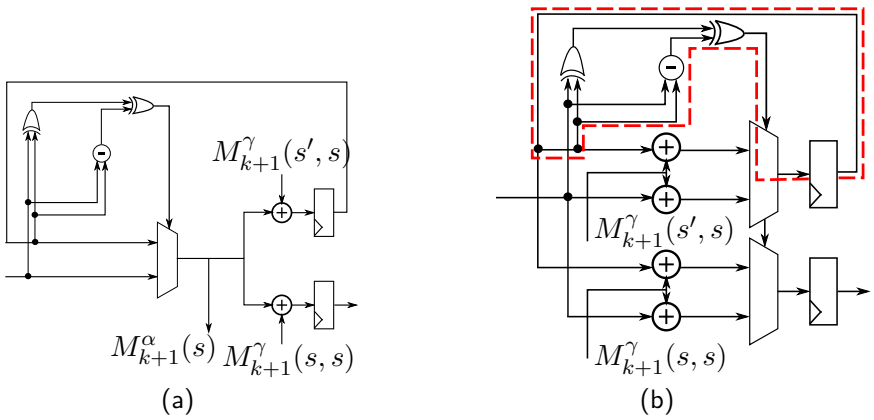

Figure 3.6: (a) radix-2 CSA unit. (b) High speed radix-2 ACS unit.

3.2.3 SOFT OUTPUT UNIT

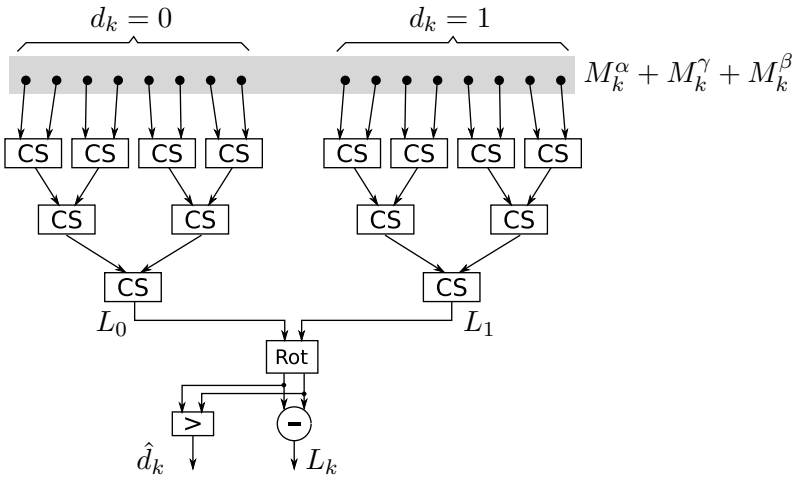

Figure 3.7: SOU radix-2 architecture.

The SOU computes the extrinsic information associated with the systematic bit $c_{k,0}$, that corresponds to the information bit d_k. It consists of a comparison tree as shown in Figure 3.7. The state metrics plus the branch metrics are computed for each possible transition in the trellis diagram. Afterwards, the maximum values corresponding to the information symbols $d_k = 0$ (L_0) and $d_k = 1$ (L_1) are calculated. Note that since the modulo normalization technique has been applied, the maximum values are found by

Decoder Unit	Hardware complexity (gates)
BMU	635
ACS (Area for all the 8 states)	3.4k
SOU	2.2k

Table 3.1: Hardware complexity of the Max-Log-MAP algorithm. Radix-2. RSC code with parameters $p = 3$, $m = 1$, $n = 2$. Quantization $w_r = 6$, $w_{SM} = 10$, $w_{ext} = 9$.

using the Compare-Select (CS) block of the ACS unit (Figure 3.5(b)). Once L_0 and L_1 are found, the a-posteriori information is computed by the subtraction of L_0 from L_1. However, if these two values are in the quadrants 01x...x and 10x...x, the subtraction leads to a wrong result. Therefore, the block Rot (rotation) is necessary to move both values to two other adjacent quadrants. This operation is easily done by adding 01 to the two most significative bits of L_1 and L_0. After the block Rot, the hard decision $\hat{d}_k$ and the a-posteriori L_k information are computed. From L_k, the extrinsic information is computed by the subtraction of the channel and a-priori information.

3.2.4 Implementation Results for the Radix-2 SISO Decoder

Table 3.1 presents the hardware complexity in terms of the equivalent 2-input (NAND) gate count for the different units that compose a radix-2 SISO decoders. The ACS unit in Figure 3.5(b) is considered. These hardware complexity results were obtained using the 90nm CMOS technology from STMicroelectronics. Thus, without considering the sliding window technique (no registers needed to store the window's boundary state metrics), neither the memory cost (Input buffer and state metrics memory), the hardware complexity for a SISO decoder implementing the Backward-Forward schedule is equivalent to 10.3k logic gates[1]. Similarly, when the Butterfly or the Butterfly-Replica schedules are considered, the SISO decoder hardware complexity is equal to 12.5k logic gates. Note that the complexity estimation for the different schedules are obtained by individual logic synthesis of the different units.

3.3 Exploration of High Radix SISO Decoders

In this section, we present an overview of different SISO decoders for high radix architectures. Different ACS units are considered. Their hardware complexity and critical path characteristics are discussed.

[1]This schedule requires 2 BMUs, 2 ACS units and a SOU.

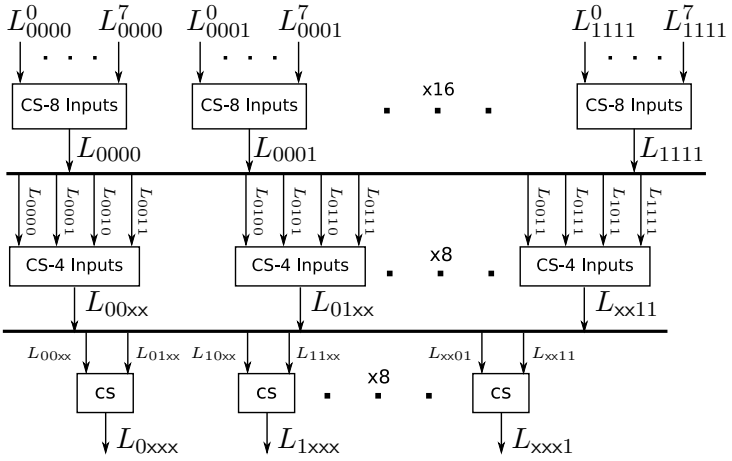

Figure 3.8: Comparison tree for a radix-16 SOU that minimizes the number of Comparison-Selection operation.

3.3.1 HIGH RADIX BMU AND SOU

Considering the hardware complexity of the SISO decoder units in Table 3.1, the BMU represents a rather small fraction of the overall SISO decoder complexity dominated by the SOU and the ACS unit. The SOU accounts for a relative high hardware complexity due to number of CS blocks that are necessary to build the comparison tree, as shown in Figure 3.7. When higher radix values are considered, the percentages of the SISO decoder complexity related to the BMU and SOU become more significative. It is specially true for the SOU, that exhibits a higher complexity than the ACS unit in a radix-16 architecture, if a straightforward implementation is carried out.

For a first estimation of the overhead due to the radix value, we consider the number of $M^{\alpha} + M^{\gamma} + M^{\beta}$ values that has to be processed during a clock cycle for a radix-2^{N_T} trellis transition: $N_b = 2^{p+N_T}$. Thus, $(N_b - 2) \cdot N_T$ 2-inputs Selection-Comparison operations have to be performed by the SOU to generate all the extrinsic values corresponding to the information symbols in a radix-2^{N_T} trellis transition. This is a huge number of comparisons, that involves a high hardware complexity. However, if a careful organization of the comparison operations is done, some intermediate values can be reused. In this way, an important reduction of the SOU hardware complexity can be achieved.

Figure 3.8 shows a radix-16 SOU with minimum hardware complexity. In this figure, $L_{d_k,d_{k+1},d_{k+2},d_{k+3}}$ denotes the maximum $M^{\alpha} + M^{\gamma} + M^{\beta}$ value corresponding to the information symbols $d_k, d_{k+1}, d_{k+2}, d_{k+3}$ in the radix-16 trellis diagram transition. This architecture exhibits 28% of the complexity of a straightforward architecture, and is

14% less complex than the SOU architecture proposed in [81]. Regarding a radix-4 SOU, a similar comparison tree structure leads to a reduction of 46% with respect to a straightforward implementation. Note that the rotation blocks required to deal with the modulo normalization technique, as explained in section 3.2.3, are required at the output of the comparison tree.

Regarding the BMU, the number of path metrics that should be computed is $2^{2 \cdot N_T}$ for $N_T = 1, 2, 3$, and 128 for a radix-16 architecture[2]. Thus, an increase close to an exponential trend is expected for the BMU hardware complexity as N_T grows.

3.3.2 HIGH RADIX ACS UNITS

Different works in the literature have explored the convenience of radix-2^{N_T} SISO decoders for $N_T = 1, 2, 3, 4$ transitions in the trellis diagram. These works are mainly focused on the optimization of the decoding speed. Only a few number of works consider hardware complexity reduction. Regarding the Log-MAP algorithm, radix-4 ACS units have been proposed in [37, 76, 77], a radix-8 ACS unit is introduced in [79], and a low power radix-16 SISO decoder is detailed in [81]. For the Max-Log-MAP algorithm, radix-4 architectures are proposed in [56, 78]. Also, a two-dimensional radix-16 ACS unit has been introduced in [80]. This architecture consists in the concatenation of two radix-4 ACS units that are able to perform in parallel the comparison and addition operations, thanks to a retiming technique. Thus, high throughput rates are achieved with a hardware complexity that is about the half of a conventional radix-16 ACS unit.

Figure 3.9 depicts three different radix-4 ACS unit architectures. A straightforward radix-4 ACS unit (Figure 3.9(a)), consists in a comparison tree composed of three CS blocks. The architecture shown in Figure 3.9(b), proposed in [78], performs in parallel the comparison of all the possible pairs of values $M^{\alpha}+M^{\gamma}$ in the radix-4 trellis transition. Then, a LUT is used to select the final value. For this ACS unit, the critical path is increased by the LUT with respect to a radix-2 ACS unit. Finally, in Figure 3.9(c) a radix-2x2 ACS unit is presented [56]. In this architecture, following a similar approach to the one used in order to design the architecture presented in Figure 3.6(b), the adders are reallocated in order to perform the addition and comparison operations in parallel.

The retiming technique used to design the radix-2 and radix-4 architectures in Figures 3.6(b) and 3.9(c), respectively, can be extended in order to design a radix-16 ACS unit. Thus, two radix-4 ACS units that execute the comparisons and additions in parallel are necessary. We refer to this architecture as a radix-4x4 ACS unit. In this case, there is an additional cost due to number of adders that are required to anticipate the additions.

[2]For $N_T = 4$, the number of branch metrics to compute is 128 instead of 256, since in this case all the values for the coded symbol in the radix-16 trellis transition are not valid.

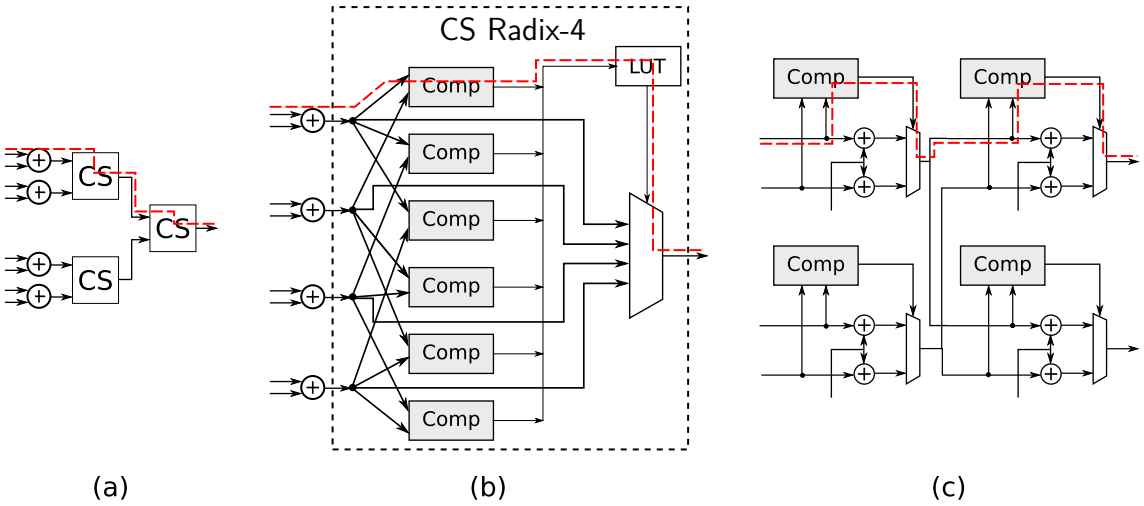

Figure 3.9: Radix-4 ACS unit architectures. (a) Straightforward architecture. (b) Parallel comparison of every pair of $\alpha + \gamma$ values in the radix-4 transition [78]. (c) Radix-2x2 ACS unit [56].

But, the additions are removed from the critical path.

Figure 3.10 shows the Complexity-Timing (CT) characteristics of the radix-2, radix-4 and radix-16 ACS units discussed so far. In this case, the hardware complexity corresponds to eight ACS units (one for each possible convolutional encoder state). Since the state metrics dynamic range is larger if higher radix values are considered, in the radix-4 and radix-16 architectures more bits are required for M^{α} and M^{β}. In this way, correct operation of the modulo normalization technique is ensured. Thus, for radix-2 and radix-4 architectures, $w_{SM} = 10$ and $w_{SM} = 11$, respectively. For the radix-16 architectures, $w_{SM} = 12$. The time required to perform one transition in the trellis diagram (τ/N_T) is shown in the horizontal axis[3]. Thus, we compare the convenience of the architectures to improve the SISO decoder throughput. The complexity (C) is expressed in terms logic gates number. In dashed lines, curves for constant CT product are plotted. Note that the parallel execution of the addition and comparison operations in the radix-2 architecture enables to improve the throughput with an acceptable cost in terms of the hardware complexity. The radix-4 architectures achieve a similar maximum throughput than the fast radix-2 architecture, with the half of the clock frequency. An important increase of the radix-16 (4x4) architecture complexity is observed with respect to the radix-2 architectures. Considering the throughput per area unit, $\eta = Th/C = N_T/(\tau \cdot C) = 1/CT$,

[3]Recall that τ is the ACS unit critical path.

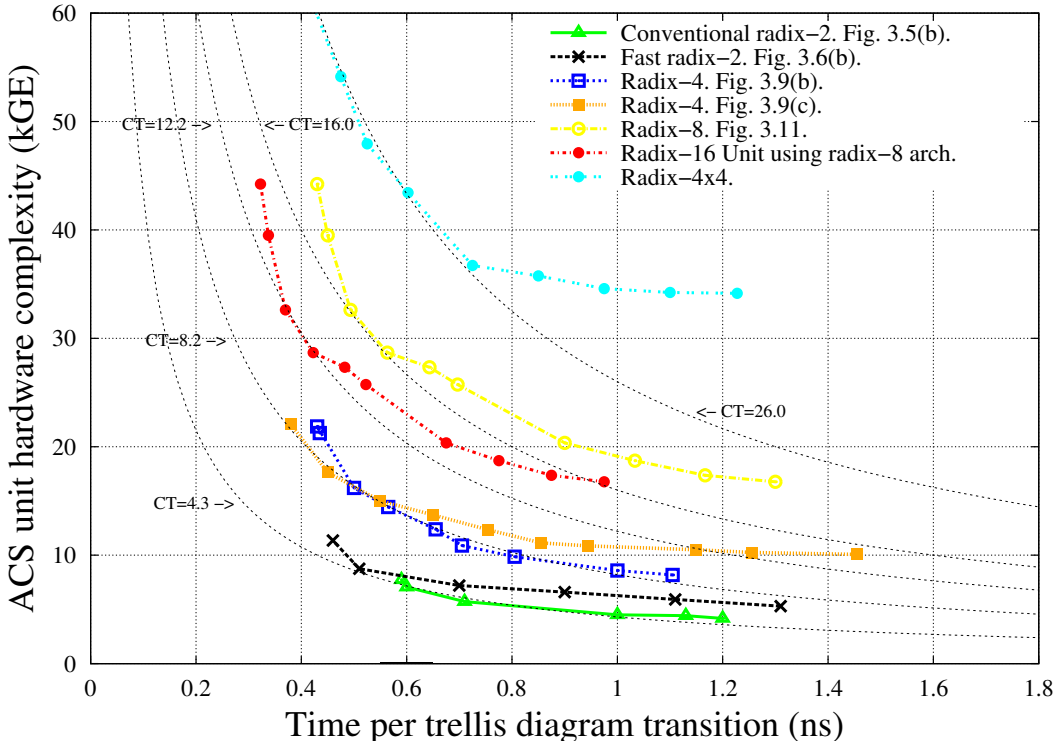

Figure 3.10: Complexity-timing characteristics for radix-2, radix-4, radix-8 and radix-16 ACS units. The architectures are synthesized for 90nm technology. Complexity expressed in terms of the number of logic gates. Horizontal axis: time required to perform a trellis diagram transition τ/N_T.

the radix-4 ($\eta = 0.121$) and radix-16 ($\eta = 0,038$) architectures are, respectively, 48% and 83% less efficient with respect to the conventional radix-2 ACS unit ($\eta = 0.232$).

For low radix values, a high throughput SISO decoder has to be designed with a high clock frequency ACS unit. In this case, since the state metrics generated during the ACS unit operation should be stored in a memory, high speed RAM memories are required. Besides, for high clock frequency values, design challenges in all the other turbo decoder blocks appear, so that the critical path is kept inside the ACS unit. In contrast, high radix values architectures enable to remove the requirements of high clock frequency circuits. Thus, with a relative high critical path, a high radix ACS unit is able to achieves high throughput rates. Moreover, due to the larger ACS unit critical path, a pipeline architecture is possible in order to share the hardware resources between different sub-blocks, as mentioned in section 2.2.3.1. However, the implementation results show

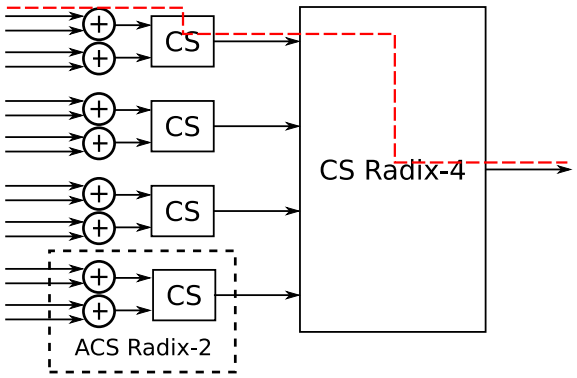

Figure 3.11: Radix-8 ACS unit architecture.

that the radix technique is inefficient in terms of how well the hardware resources are used to improve the decoder throughput. Furthermore, high radix architectures are also energy inefficient, *i.e.* more energy per decoded bit is required when N_T increases [116]. The additional hardware complexity, and the lost in efficiency, are accepted in sake of the achievement of high throughput rates. Nevertheless, the hardware complexity reduction of high radix architectures should be explored in order to increase, if possible, the efficiency.

We propose then, based on the high speed ACS unit given in Figure 3.9(b), the radix-8 ACS unit presented in Figure 3.11. This ACS unit is composed of four radix-2 ACS units and one radix-4 CS unit. First, eight additions are performed (state metrics plus branch metrics). Then, four values are selected. These values are sent to the radix-4 CS to produce the final state metric value. With respect to a conventional radix-2 ACS unit, the critical path is increased by only a CS radix-4 critical path, while performing three transitions in the trellis diagram during each clock cycle. In Figure 3.10, the complexity-timing behavior for this architecture is also presented. It is 73% less efficient with respect to the conventional radix-2 ACS unit. Since for a radix-8 architecture an odd number of transitions are performed during each clock cycle, this radix value is inconvenient to divide the frame to decode in an integer number of transitions. Thus, problems in the limits of the frame may appear. However, we propose to use the presented radix-8 ACS unit to build a high speed low complexity radix-16 SISO decoder. This particular case is also plotted in Figure 3.10. Note that there is an improvement on the maximum achievable throughput with respect to the original radix-8 architecture, since for the same clock frequency and additional trellis diagram transition is performed. The ACS unit efficiency is also improved. In the following section, a low complexity SISO decoder that implements the presented radix-8 ACS unit is detailed.

3.4 HIGH RADIX ARCHITECTURES COMPLEXITY REDUCTION

In this section we present a low complexity radix-16 Max-Log-MAP based SISO decoder intended to increase the efficiency of the radix technique. Moreover, two complementary techniques are proposed in order to overcome BER/FER performance degradation when turbo decoders based on the proposed SISO decoder are considered. The work presented in this section has been published in [117].

3.4.1 LOW COMPLEXITY RADIX-16 SISO DECODER

We recall that the convolutional code given in Figure 3.3 is considered as example. Let $S = \{s^{(0)}, \ldots, s^{(7)}\}$ denote the set of possible encoder states expressed in decimal as $s^{(i)} = \sum_{h=1}^{p} f_h^{(i)} \cdot 2^{p-h}$, with $f_h^{(i)} \in \{0, 1\}$ the value of the flip-flops in the encoder. For the encoder, the next equation holds:

$$D_1(z) = \left(1 + z^{-2} + z^{-3}\right) \cdot X(z) \tag{3.1}$$

where $D_1(z)$ and $X(z)$ are the $\mathcal{Z}$-transform of $d_{k,1}$ and x_k, respectively. Since in the z domain, a multiplication by z^{-l} corresponds to a time shifting, equation (3.2), where $b \in \{0, 1\}$, describes the encoder transition from state $s^{(0)}$ (all zero-state) to the state $s^{(i)}$, then to the state $s^{(j)}$, and finally back to the state $s^{(0)}$. Note that during the state transition $s^{(i)} \rightarrow s^{(j)}$, four information bits are encoded. Therefore, it corresponds to a radix-16 trellis diagram transition.

$$\begin{aligned} D_1(z) = (1 + z^{-2} + z^{-3}) \cdot (f_3^{(i)} + f_2^{(i)} z^{-1} + f_1^{(i)} z^{-2} + b z^{-3} + \\ f_3^{(j)} z^{-4} + f_2^{(j)} z^{-5} + f_1^{(j)} z^{-6}) \end{aligned} \tag{3.2}$$

By considering only the terms related to z^{-3}, z^{-4}, z^{-5}, z^{-6}, which correspond to the radix-16 transition $s^{(i)} \rightarrow s^{(j)}$, we obtain the systematic bit sequence:

$$\begin{aligned} d_{i,j}^{(b)} &= \left(d_k^{(i,j,b)}, d_{k+1}^{(i,j,b)}, d_{k+2}^{(i,j,b)}, d_{k+3}^{(i,j,b)}\right) \\ &= \left(f_3^{(i)} + f_2^{(i)} + b, f_2^{(i)} + f_1^{(i)} + f_3^{(j)}, f_1^{(i)} + f_2^{(j)} + b, f_3^{(j)} + f_1^{(j)} + b\right) \end{aligned} \tag{3.3}$$

This systematic bit sequence produces the following sequence for the redundant bit:

$$\begin{aligned} c_{i,j}^{(b)} &= \left(c_k^{(i,j,b)}, c_{k+1}^{(i,j,b)}, c_{k+2}^{(i,j,b)}, c_{k+3}^{(i,j,b)}\right) \\ &= \left(f_3^{(i)} + f_1^{(i)} + b, f_2^{(i)} + f_3^{(j)} + b, f_1^{(i)} + f_3^{(j)} + f_2^{(j)}, f_2^{(j)} + f_1^{(j)} + b\right) \end{aligned} \tag{3.4}$$

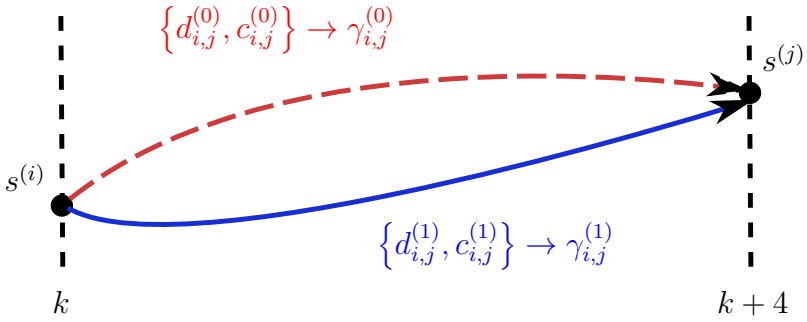

Figure 3.12: Radix-16 ACS unit trellis diagram transition.

$d_{i,j}^{(b)}$ and $c_{i,j}^{(b)}$ are the systematic and redundant bits that are generated during a radix-16 transition, from the state $s^{(i)}$, at time k, to the state $s^{(j)}$ at time $k+4$. Since b has two possible values, there are two paths in this transition, as presented in Figure 3.12. Let $\gamma_{i,j}^{(b)}$ be the branch metric value for the path b. As pointed out in [75], parallel paths should be eliminated prior to the ACS unit. Thus, it is possible to replace a radix-16 ACS unit by a less complex radix-8 ACS unit where each pair of states $(s^{(i)}, s^{(j)})$, at time k and $k+4$ respectively, are connected only by one path. The branch metric value for this path is then:

$$\gamma_{i,j}^{max} = \max\left(\gamma_{i,j}^{(0)}, \gamma_{i,j}^{(1)}\right) \tag{3.5}$$

Note that this simplification does not affect the result of the ACS unit operation. Furthermore, since a radix-8 ACS unit has a lower critical path compared to a radix-16 ACS unit, it enables to increase the clock frequency of the SISO decoder. Note also that although a radix-8 ACS unit is used, the SISO decoder is equivalent to a true radix-16 architecture ($N_T = 4$). We exploit this idea in order to design a novel low complexity high throughput radix-16 SISO decoder as presented in the following sections.

3.4.1.1 Branch Metrics Unit Architecture

From (3.3) and (3.4), for any pair of parallel paths, $d_{k+1}^{(i,j,0)} = d_{k+1}^{(i,j,1)}$ and $c_{k+2}^{(i,j,0)} = c_{k+2}^{(i,j,1)}$ because they are independent of b. Therefore, the channel values of bits d_{k+1} and c_{k+2} do not affect the choice of the maximum branch metric in the radix-16 ACS unit transition $s^{(i)} \rightarrow s^{(j)}$. $\gamma_{i,j}^{max}$ can be then computed as follows. First, a partial term $\xi_{i,j}$, due to the bits that are different in both paths, for $b = 0$, is calculated. If $\xi_{i,j} < 0$ then $\gamma_{i,j}^{(1)}$ is the maximum branch metric and $\epsilon_{i,j} = -\xi_{i,j}$. Otherwise, the maximum branch

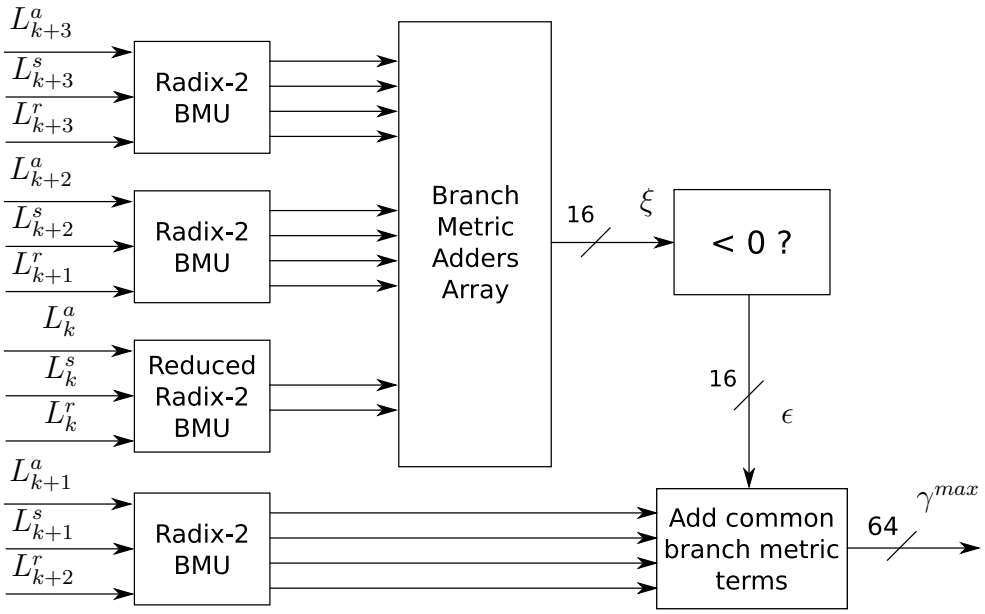

Figure 3.13: Proposed radix-16 BMU architecture.

metric is $\gamma^{(0)}_{i,j}$ and $\epsilon_{i,j} = \xi_{i,j}$. Afterwards, $\gamma^{max}_{i,j}$ is obtained by adding to $\epsilon_{i,j}$ the values of the bits d_{k+1} and c_{k+2}. The proposed BMU architecture is presented in Figure 3.13. Note that there are three radix-2 BMUs (Figure 3.4), and a reduced radix-2 BMU (only two outputs are computed) to reduce the hardware complexity.

The proposed BMU architecture only computes 16 values $\xi_{i,j}$ for all the 64 state transitions $s^{(i)} \rightarrow s^{(j)}$, $i, j = 0, \ldots, 7$. It only costs 53% of the hardware resources of a conventional implementation. A low complexity adder-sharing BMU is also proposed in [81]. It requires a memory to store 128 branch metrics. In our BMU architecture, this memory is reduced to the half (64 $\gamma^{max}_{i,j}$ values). Moreover, the BMU in [81] needs additional registers that increase the latency. Indeed, more execution cycles are required to share the adders. In our architecture, this disadvantage is overcome.

3.4.1.2 ACS Unit Architecture

The proposed SISO decoder implements the radix-8 ACS unit depicted in Figure 3.11. Since four transitions in the trellis diagram are performed during each clock cycle, the efficiency is increased by about 31% with respect to the original radix-8 architecture performing three trellis diagram transitions per clock cycle. Note that the achieve maximum

decoder throughput is higher that in the radix-4 ACS unit architectures.

3.4.1.3 SOFT OUTPUT UNIT ARCHITECTURE

The elimination of parallel paths in the BMU enables to design a faster and less complex radix-8 ACS to be used as part of a radix-16 SISO decoder, without impacting the computations of the forward and backward state metrics. However, since only the half of the branch metrics are produced by the BMU, all the 128 values $M^{\alpha}+M^{\gamma}+M^{\beta}$ cannot be computed by the SOU. Thus, different extrinsic values with respect to a conventional implementation are calculated. We propose a SOU that only uses the 64 branch metrics produced by the low hardware complexity BMU presented in section 3.4.1.1. It reduces the hardware complexity required to compute the extrinsic values and generate the hard decisions.

Since it is not possible to a priori determine the paths to eliminate in each state transition, a static comparison structure such as the one presented in Figure 3.8 cannot be used for the SOU. Note however that we can group the 64 state transitions $s^{(i)} \rightarrow s^{(j)}$ in eight sets Q_l, for $l = 0, 1, \ldots, 7$. Each set is compose of eight elements (pair of states). Thus, the set Q_l is defined as follows:

$$Q_l = \{(s^{(i)}, s^{(j)}) : d_{i,j}^{(0)} = (a_0^{(l)}, a_1^{(l)}, a_2^{(l)}, a_3^{(l)})\} \tag{3.6}$$

where $a_n^{(l)} \in \{0, 1\}$, for $n = 0, 1, 2, 3$. Therefore, the eight branch metric values $\gamma_{i,j}^{max}$ from the state $s^{(i)}$ to the states $s^{(j)}$, such that $(s^{(i)}, s^{(j)}) \in Q_l$, correspond to only two possible systematic bit sequences, according to $d_{i,j}^{(b)}$ in (3.3). For instance, if $(a_0^{(0)}, a_1^{(0)}, a_2^{(0)}, a_3^{(0)}) = (0, 0, 0, 0)$, then the values of $\gamma_{i,j}^{max}$ related to the elements of Q_0 are for the systematic bit sequence $(0, 0, 0, 0)$ or $(1, 0, 1, 1)$. Based on this observation we propose in Figure 3.14 an original SOU architecture that contains comparing-routing elements. The structure of this SOU is explained as follows.

First, the 64 values $L = M_k^{\alpha} + \gamma^{max} + M_{k+4}^{\beta}$, corresponding to the non discarded paths, are computed[4]. These values are then sent to eight *Max-L* blocks. The eight inputs of the l-*th Max-L* block correspond to the values L for the paths defined by the states $(s^{(i)}, s^{(j)}) \in Q_l$. Each *Max-L* block processes its inputs in three stages. In the first stage, four *Switch-I* blocks (Figure 3.14(b)) either find in parallel the maximum values of their inputs or directly send the inputs to the outputs. In the two other stages, *Switch-II* blocks are used (Figure 3.14(c)). These blocks have to find the maximum value or let one of the inputs to pass through them. Thus, at the output of the *Max-L*

[4]Note that $M_k^{\alpha} + \gamma^{max}$ values are already computed inside the ACS unit.

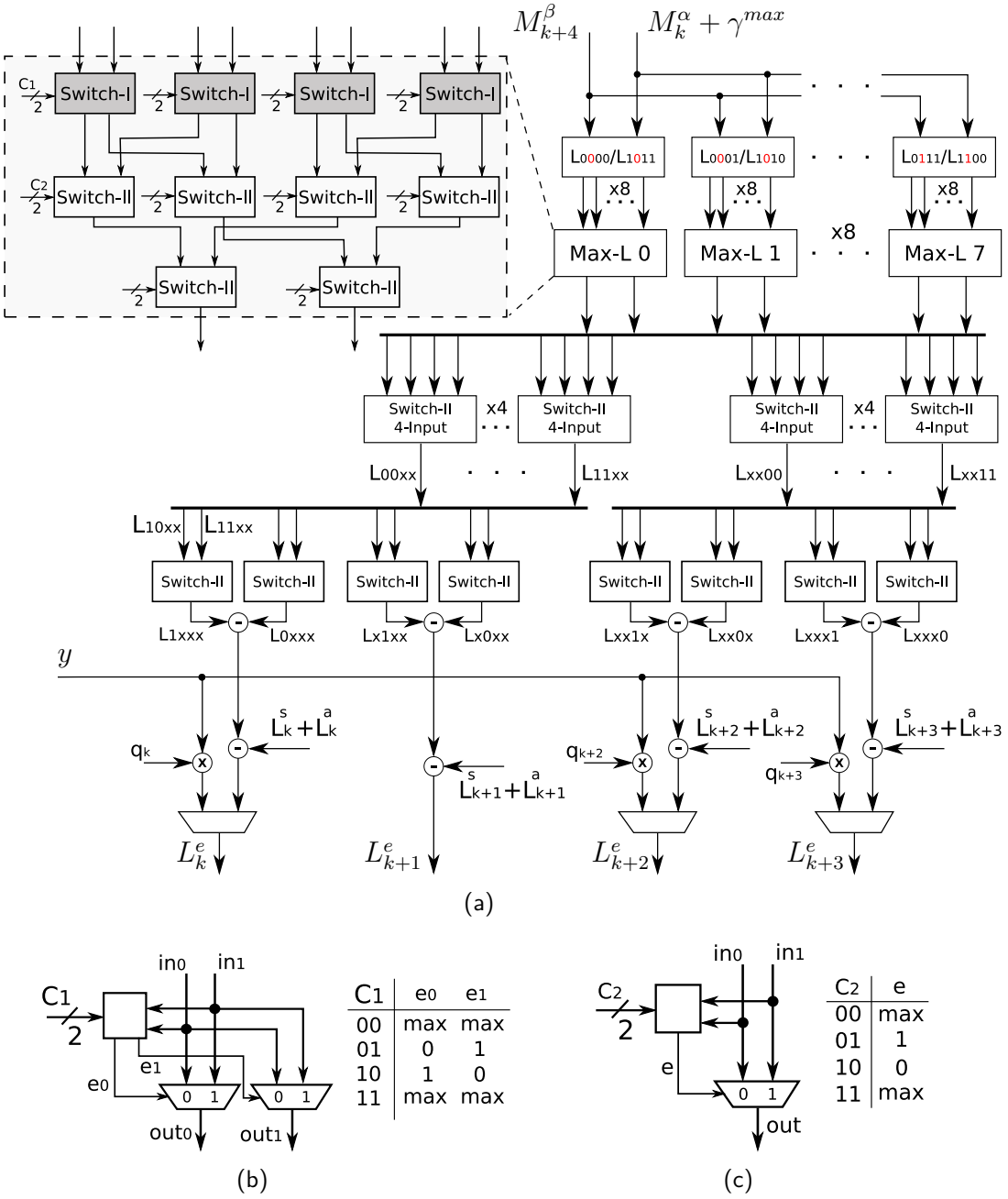

Figure 3.14: (a) Proposed radix-16 SOU. (b) *Switch-I* block. (c) *Switch-II* block. Selection of the maximum value is carried out with a modulo normalization block (Block *Comp* in Figure 3.5(b)).

blocks the maximum value for each sequence of systematic bits is found[5]. These values are then compared thanks to a fixed tree structure composed of *Switch-II* blocks. This comparison tree uses a minimum number of *Switch-II* blocks. Finally, the four extrinsic values are computed.

Due to the elimination of paths, values L^e_{k+h}, for $h = 0, 2, 3$, may not be valid. In this case, a multiplexer enables to replace them by $y \cdot q_{k+h}$, where $q_{k+h} = \pm 1$, $y > 0$, and $\hat{d}_{k+h} = (q_{k+h} + 1)/2$ is the hard decision of the systematic bit d_{k+h}. An expression for y is proposed in Section 3.4.2. Since the systematic bit d_{k+1} does not change in the parallel paths, the value L^e_{k+1} is always valid. Thus, a multiplexor to compute L^e_{k+1} is not necessary.

3.4.2 Low Hardware Complexity Radix-16 SISO Decoder Performance

In section 3.4.1 we have successfully applied the elimination of parallel paths in the radix-16 trellis transition in order to reduce the hardware complexity of all the constituent SISO decoder units. However, when the proposed SISO decoder is used as part of a turbo decoder architecture, a BER/FER performance degradation appears. We have observed a penalty around 0.2 dB for coder rates $R = 1/2$ and $1/3$, at FER of 10^{-6}, with an error floor that remains high compared to the error floor of a radix-2 turbo decoder architecture.

The loss of information due to the elimination of parallel paths is responsible of the performance degradation. Since the SOU cannot consider all the paths in the radix-16 trellis diagram transition, extrinsic values are approximately computed. Thus, during the iterative decoding process, the negative effects of this approximation are amplified, leading to wrong decoding decisions. Let us consider how the extrinsic values are computing in a radix-16 trellis diagram transition.

For high SNR values (error floor region), a-priori values grow fast during the iterative decoding process. Thus, it is more probable to eliminate all the paths for a specific bit value. For instance, if the a-priori value for d_k (L^a_k) is high and positive, it is highly probable that all the paths corresponding to $d_k = 1$ are chosen. It means that there is not enough information to compute L^e_k, since there is no value L corresponding to $d_k = 0$. Furthermore, the selection of the paths that correspond to $d_k = 1$ also affects the computation of the extrinsic values L^e_{k+2} and L^e_{k+3}. Therefore, an undesirable correlation may appear between the hard decisions $\hat{d}_k$, $\hat{d}_{k+2}$ and $\hat{d}_{k+3}$, and between their respective extrinsic values, during the iterative decoding process. To reduce the

[5]Note that it is possible that there is no valid value for a specific systematic bit sequence, since all its related paths could have been eliminated in the BMU unit.

degradation introduced by the proposed SISO decoder architecture, we propose two techniques:

- Select an appropriate value for y in Figure 3.14(a). It is used when there is not enough information to compute a determinate extrinsic value.
- Reduce the mutual interference between symbols in each radix-16 trellis diagram transition.

These techniques are discussed in the two following sections.

3.4.2.1 Expression of y for Unknown Extrinsic Values

Let us consider the four information symbols $d_k, d_{k+1}, d_{k+2}, d_{k+3}$ in a radix-16 trellis transition. Also, let us consider that all the paths corresponding to the information symbol $d_k = 1$ are eliminated. In this case, the turbo decoding process is converging to the hard decision $\hat{d}_k = 0$. Thus, our architecture produces the extrinsic value $L_k^e = -y$. In the iterative decoding process, wrong decision may be taken during the first iterations. Theses decisions may be then corrected during the next iterations. Thus, in order to prevent the turbo decoder to take a possible wrong decoding decision, the y value should not be too high.

We have established an expression for y following a similar approach to the one presented in [118] for the decoding of Block Turbo Codes. In [118], when there is not competing codeword in order to compute the reliability value of a certain bit, the soft output is calculated as the sum of the magnitude of a channel observation set at the input of the decoder. In our case, since there are only four systematic bits for each radix-16 trellis transition, we have modified the summation by a minimum operation. Thus, we avoid too optimistic extrinsic values that are inconvenient during the iterative process. The value of y corresponds to the minimum reliability value at the SISO decoder input (absolute value of the systematic plus the a-priori LLR values) between all the four bits in the radix-16 trellis transition:

$$y = \min_{i=0,1,2,3} \left(\left| L_{k+i}^a + L_{k+i}^s \right| \right) \tag{3.7}$$

Extensive Monte-Carlo simulations have demonstrated the convenience of the expression in (3.7).

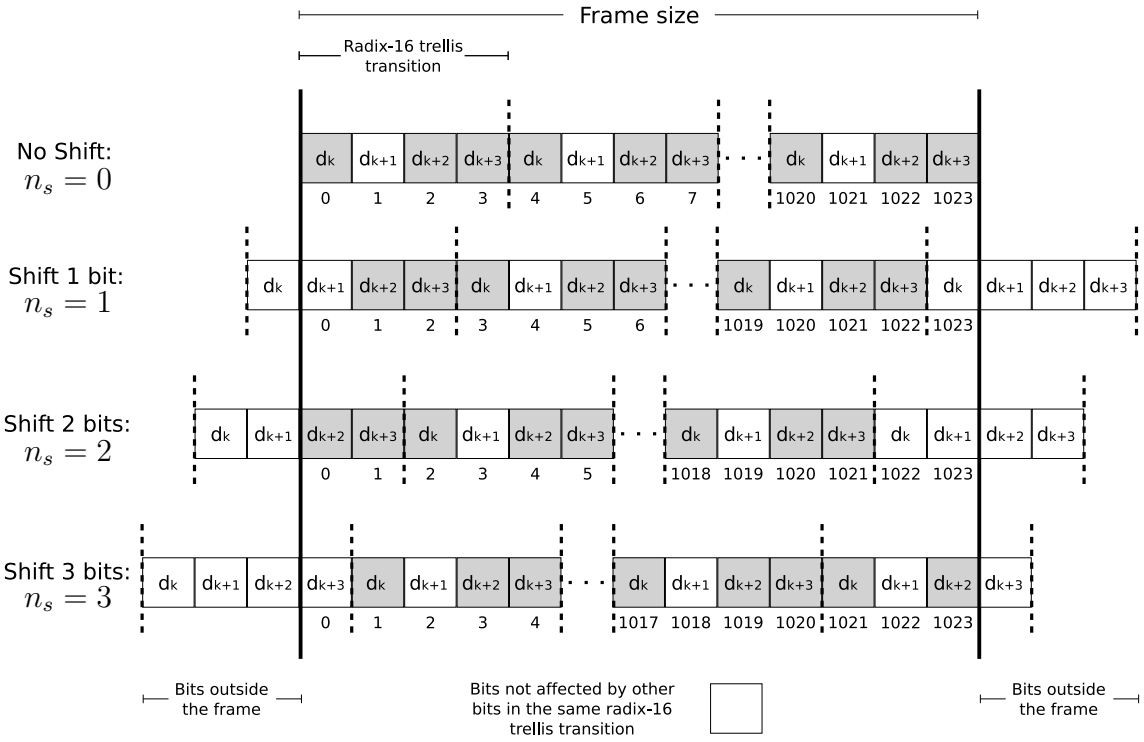

Figure 3.15: Frame shift principle to avoid interference between symbols in a radix-16 trellis transition.

3.4.2.2 REDUCE THE INTERFERENCE OF BITS IN THE SAME TRELLIS TRANSITION

Note that the elimination of parallel paths in the radix-16 trellis diagram transition does not affect the computation of the extrinsic value L^e_{k+1}. Hence, the decoding operations corresponding to the bit d_{k+1} are not affected if the proposed radix-16 SISO decoder architecture is implemented. Thus, we can consider the bit d_{k+1} as having better properties to be correctly decoded, compared to the others bits in the radix-16 trellis transitions. Based in this property, we propose to apply a *shift* operation on the frame processed by the SISO decoder, as presented in Figure 3.15. In this figure, a frame size of 1024 bits is considered. Let n_s denote the number of shifted bits. When no shifting is applied ($n_s = 0$), bits $1, 5, 9, \ldots, 1021$ in the frame are not affected by the simplified architecture. When $n_s = 1$, the first forward radix-16 trellis diagram transition starts one trellis diagram transition before the beginning of the information frame. In this case, bits $0, 4, 8, \ldots, 1020, 1023$ would be better decoded. Similarly, for $n_s = 2$ and $n_s = 3$,

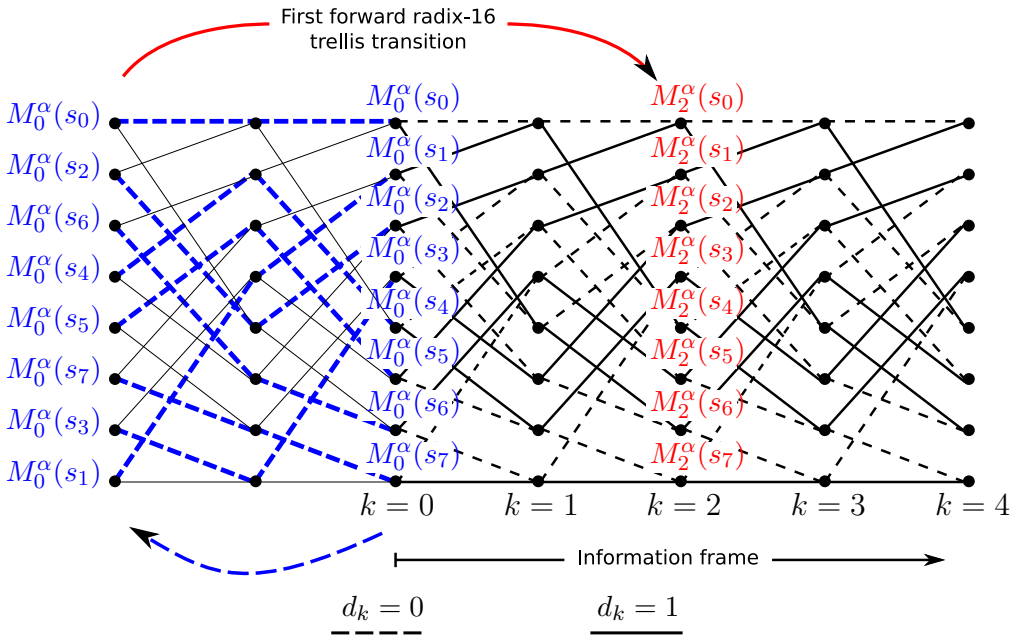

Figure 3.16: Initialization of M^α values for the first radix-16 trellis transition when $n_s = 2$.

other bits in the frame are unaffected by the elimination of parallel paths.

In Figure 3.15, shaded squares denote bits in a given radix-16 trellis transition that mutually interfere when parallel paths are eliminated. The bits that interfere depend on the values of n_s. For instance, when no shifting is applied, bits 4,6 and 7 are mutually affected in the second forward radix-16 trellis transition. When $n_s = 1$, bits 3,5 and 6 affect each other in the same radix-16 trellis transition. Furthermore, when $n_s = 2$ or $n_s = 3$, different bits interfere.

Based on these observations, we propose to perform consecutive turbo decoder iterations with different shift values. Thus, the first, second and third iterations are performed with the information frame shifted by zero, one and two bits, respectively. In the following iterations other shift values are considered. By applying this simple technique, we are able to eliminate the correlation that may appear between bits in the same radix-16 trellis transition. Furthermore, it introduce a diversity since different bits in the frame are protected in function of the iteration.

The number of radix-16 trellis transitions that should be performed when the frame is not shifted, is $L/4$. However, when $n_s > 0$, one additional radix-16 trellis transition should be performed in the forward and backward recursions. In this case, some bits do

not belong to the information frame for the first and last radix-16 trellis transition. For instance, for $n_s = 3$, bits d_k, d_{k+1}, d_{k+2} are outside the information frame for the first forward radix-16 trellis transition. Also, during the last forward radix-16 trellis transition, the bit d_{k+3} is not part of the information frame. If a circular code is considered, bits outside the frame in the first (last) transition correspond to bits inside the frame in the last (first) transition. Thus, during all the radix-16 trellis transitions, a-priori values are easily obtained. However, if the code is not circular, as in the LTE standard, a specific attention should be given to the a-priori values provided to the SISO decoder during the first and last transitions. By this way, M^α and M^β values at the frame limits are not altered. Figure 3.16 depicts the initialization of M^α values if $n_s = 2$. At the beginning of the information frame, the forward recursion metrics (M_0^α) are known for all the states. Since the recursion process starts two trellis transition before, M_0^α values are moved backward two trellis transition following the branches corresponding to the information symbol $d_k = 0$, as shown in the figure. Thus, for the transitions outside the information frame, the a-priori values are negative and their magnitude large enough. In this case, M^α values at the beginning of the information frame are not modified. Note that M^α values at the beginning of the information frame are not re-computed in the first forward radix-16 trellis transition. Actually, the next values computed are M_2^α. A similar initialization process should be applied to M^β values at the end of the information frame if $n_s > 0$.

3.4.2.3 SIMULATIONS FOR LTE TURBO DECODER

Monte-Carlo simulations were performed in order to determinate the performance degradation of turbo decoders using the proposed radix-16 SISO decoder architecture. The parallelism at the SISO decoder level is not applied. Thus, there is $Q = 1$ sub-block, and the decoding is carried out in non-shuffled mode. Besides, the sliding window technique is not considered. Two turbo decoder architectures have been analyzed: one based on a radix-2 SISO decoder, and another one based on the proposed radix-16 SISO decoder. Two different code rates $R = 1/3$ and $1/2$ have been used. The code rate $R = 1/2$ is achieved by puncturing the original code. $w_r = 6$ and $w_{ext} = 9$ bits have been used for the channel and extrinsic values representation, respectively. For the radix-2 SISO decoder, $w_{SM} = 10$. For the radix-16 SISO decoder $w_{SM} = 12$.

Figure 3.17 shows the BER/FER performance curves after six iterations for the LTE turbo decoder, with 1024 bits per frame. The curve with circles corresponds to the performance of our radix-16 SISO decoder using for y the expression in (3.7). In this case, an error floor appears for both code rates. However, for a BER of 10^{-7}, the performance is acceptable by comparison with the radix-2 architecture performance. On

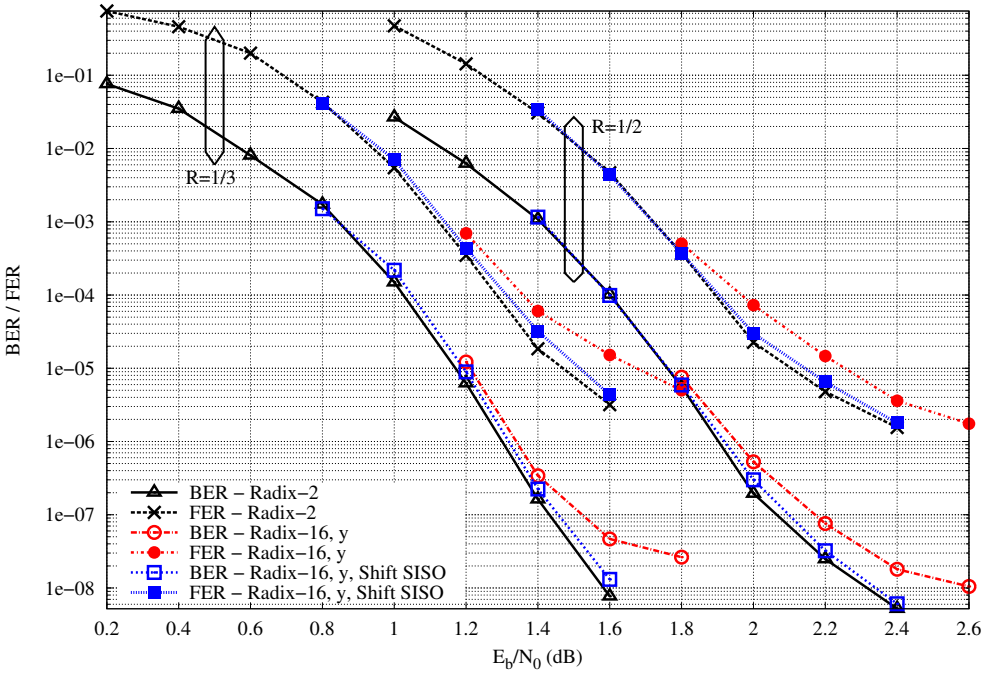

Figure 3.17: Fixed point simulation for the LTE turbo decoder (1024 bits per frame), with a radix-2 and the proposed radix-16 SISO decoders.

the other hand, when the shifting technique is also applied, the architecture based on the proposed radix-16 SISO decoder exhibits a negligible degradation.

3.4.3 Implementation Results

Table 3.2 presents the hardware complexity for the different units of the proposed radix-16 SISO decoder. In order to compare the SISO decoder hardware complexity as function of N_T, the hardware complexity of a radix-4 SISO decoder is also given. The logic synthesis results are obtained for a clock frequency of 200MHz. Even though it is a low clock frequency, the synthesis results give good estimations about the improvements in terms of the complexity provided by our SISO decoder architecture. We also present the hardware complexity results for an alternative radix-16 SISO decoder architecture. For the radix-4 SISO decoder, the ACS unit in Figure 3.9(b) has been used. For the radix-16 SISO decoder, the two-dimensional radix4x4 ACS unit was chosen. Besides, a SOU implementing the static comparison tree that minimizes the number of comparisons (Figure 3.8) has been implemented. Table 3.2 also shows an estimation of the overall

	Radix-4	Radix-16 (radix-16 ACS unit)	Radix-16 (proposed)
BMU	3.2k	28.6k	15.3k
ACS (Area for all the 8 states)	8.2k ([78], 11 bits)	34.1k ([80], 12 bits)	16.6k (12 bits)
SOU	5.3k	23.5k	18.3k
Total estimated SISO area	28.1k	148.9k	82.1k

Table 3.2: Hardware area in terms of equivalent 2-input (NAND) gate count for different SISO decoders (Input and β buffer not included).

SISO decoder complexity when the architecture in Figure 3.2(a) is considered. The input and β buffer complexities are not included in the comparison since their values depend on the window size. However, a radix-2^{N_T} ACS unit enables to reduce the β buffer size by a factor of N_T. Thus, low radix architectures are penalized by the memory complexity.

Thanks to the complexity reduction performed in all the SISO decoder blocks, our radix-16 SISO decoder is about 55% less complex than the considered radix-16 implementation. The proposed ACS unit helps considerably to achieve this result. Furthermore, the proposed BMU and SOU enable to overcome the hardware penalty cost introduced by a radix-16 approach. Compared to radix-2 and radix-4 SISO decoders, the proposed architecture is about 7.8 and 2.9 more complex, respectively. With this additional hardware complexity, the proposed SISO decoder improves by a factor of 4 and 2 the radix-2 and radix-4 SISO decoder throughput, respectively[6].

3.5 CONCLUSION

This chapter details the internal SISO decoder structure. Hence, the architectures of the blocks that compose the decoder are presented. The hardware implementation of high radix architectures was discussed. Thus, the benefits that the radix technique provides in order to improve the SISO decoder throughput are demonstrated. It was observed how this technique is not efficient in terms of throughput per hardware complexity. However, with a high throughput goal in mind, the extra hardware complexity is accepted if throughput improvements are possible.

In the last section of the chapter, a low complexity radix-16 SISO decoder for the Max-Log-MAP algorithm is presented. Besides, two complementary techniques have been proposed in order to limit the BER/FER performance degradation introduced by a turbo decoder architecture based on the proposed radix-16 SISO decoder. We have proposed the elimination of parallel paths in the trellis diagram to be able to implement a radix-8 ACS unit as part of a radix-16 SISO decoder. Even though this idea is not new, we

[6]These throughput improvements factors are achieved if the same clock frequency is considered for all the architectures.

have exploited it in order to reduce the hardware complexity of all the blocks in the SISO decoder. It corresponds to an original contribution, published in ICECS conference [117]. Note that the ideas presented in this chapter can be applied to design higher radix SISO decoders. In this case, the SISO decoder critical path would correspond to a radix-8 ACS unit, but more than four transitions in the trellis diagram would be performed during each clock cycle. In this case, a similar approach as the one presented in this chapter can be used to reduce the hardware complexity of the BMU and SOU. Thus, very significant improvement in throughput rates would be possible with an acceptable efficiency.

Chapter 4

High Throughput Turbo Decoder Architectures

In the previous chapter we have explored the architectural issues that have to be considered for the design of high throughput SISO decoders. We have discussed the convenience of high radix decoders. We have also proposed a low complexity high throughput radix-16 SISO decoder architecture. However, we have not yet detailed the whole turbo decoder architecture, neither the drawbacks that arise in order to achieve high turbo decoding throughput rates. Therefore, this chapter is devoted to the presentation of the architectural solutions that we propose so that multiples SISO decoders, implementing any parallelism at the metric level, can be integrated in a high throughput turbo decoder architecture.

The first part of this chapter describes the extrinsic memory conflicts that are a major bottleneck in the turbo decoder. Then, the different approaches that have been proposed in the literature to overcome this problem are described. Afterwards, the conflict free interleavers are introduced. Thanks to the properties of these interleavers, we propose an optimized extrinsic memory organization. In the second part of the chapter, the design of a high throughput turbo decoder for the LTE standard is detailed. The architectural features are described, and a first prototype to validate our propositions is presented. Finally, at the end of the chapter, a methodology intended to explore the turbo decoder design space is proposed. This methodology allows the designer to reduce the architectural design time. However, high hardware complexity architectures may be designed. Nevertheless, the methodology can be applied as an approach to assess all the parallel turbo decoding techniques presented in Chapter 2.

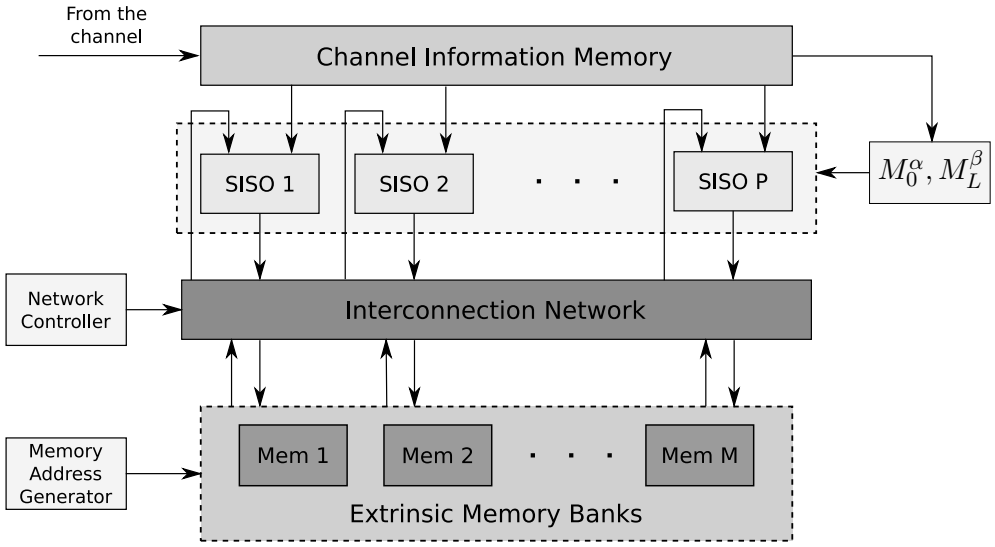

Figure 4.1: Generic parallel turbo decoder architecture block diagram.

4.1 GENERIC PARALLEL TURBO DECODER ARCHITECTURE

Let us consider the turbo decoder architecture in Figure 4.1. With this architecture model, any combination of parallel turbo decoding techniques can be implemented. It is composed of P SISO decoders that implement a given schedule. These decoders can work in shuffled or non-shuffled mode. Each one of them is assigned to decode a sub-block. Through an interconnection network, the SISO decoders have access to M memory banks (single port or dual-port RAM), assigned to store the extrinsic information values produced during each half iteration of the decoding process. The addressing of the memory blocks depends on the interleaver function $\pi(i)$. It also depends on the data required by the SISO decoders during each clock cycle. The network controller produces the required signals in order to correctly exchange the extrinsic values between the SISO decoders and the RAM memory banks. Note that the LLR values received from the channel are store in a memory. In this case, multiples buffers may be required if the frames are received faster than the turbo decoding execution time. The state metrics in the frame limits (M_0^α and M_L^β), whether a circular or non cicular code is being considered, are computed by a different block. Then, they are provided to the SISO decoders at the beginning of the turbo decoding process. For simplicity, in Figure 4.1 the communication links required to implement the message passing sub-block initialization are not shown.

4.2 MEMORY ACCESS CONFLICTS

Since each SISO decoder implements the same decoding schedule, the extrinsic memory banks are acceded simultaneously, for reading or writing, by all the SISO decoders in the architecture. Therefore, due to the interaction of both constituent convolutional codes through interleaver and de-interleaver functions, conflict accesses may appear. For high parallel turbo decoder architectures, these memory conflicts are a major bottleneck.

We recall that $\boldsymbol{d}= (\boldsymbol{d}_0, \boldsymbol{d}_1, \dots \boldsymbol{d}_{L-1})$ and $\boldsymbol{d_\pi}= (\boldsymbol{d}_{\pi(0)}, \boldsymbol{d}_{\pi(1)}, \dots \boldsymbol{d}_{\pi(L-1)})$ denote the information frame in the natural and interleaved order, respectively. For convenience, in this chapter we refer to each extrinsic information value L_k^e, corresponding to the systematic symbol $\boldsymbol{d}_k$, by its index $0 \leq k < L$. Consider the extrinsic value $0 \leq h < N$, with $N = L/Q$ the size of each sub-block, and Q the number of sub-blocks in the natural and interleaved order. Let $\mathcal{V}_h$ and $\mathcal{V}_h^\pi$ be the set of extrinsic values that are acceded simultaneously by the SISO decoders in the natural and interleaved order, respectively. For instance, if we consider that each SISO decoder implements the Backward-Forward schedule with a radix-2 architecture, then $\mathcal{V}_h = \{h, N+h, \cdots, (Q-1)N+h\}$ and $\mathcal{V}_h^\pi = \{\pi(h), \pi(N+h), \cdots, \pi((Q-1)N+h)\}$.

In order to prevent a reduction in the achievable throughput due to the extrinsic memory, the turbo decoder architecture should permit a concurrent access to all the elements of $\mathcal{V}_h$ and $\mathcal{V}_h^\pi$. Three classes of approaches have been proposed in the literature to tackle this problem [119]. The approaches are explained as follows.

Solution during the execution: In this approach, conflict problems are solved either through the addition of extra memory elements and/or complex interconnection networks. Works in [120–124] have adopted this approach with specific structures consisting in buffers, networks and routers.

Solution during the compilation: The objective of this solution is to find a memory mapping that provides conflict free concurrent access. This memory mapping specifies the RAM memory where each extrinsic value can be stored. This kind of approaches where first proposed in [125, 126]. In these works, it was demonstrated that, for any interleaver function $\pi(i)$ and any number of sub-blocks Q, it is possible to find a conflict free memory mapping. Besides, an algorithm to find a memory mapping is proposed. This technique was later extended in [127–129], where other algorithms to find a memory mapping have been proposed.

Solution during the design: In this type of solutions the interleavers are designed under certain constraints, so that the conflict problems can be solved with minimum

amounts of additional hardware resources. Hence, with a trivial memory mapping, the SISO decoders have access to the extrinsic memory without any memory access conflict. The works in [46–48, 130–134] are based in this approach. Thus, very good performance interleavers, in terms of the turbo decode error correction capabilities, have been proposed. In [135] it was demonstrated that only a small fraction of all possible interleavers present conflict-free properties. Thus, a relative low number of parameters should be established to define the interleavers. In this case, an exhaustive search among all the configuration parameters can be done [119]. It enables to find an optimum interleaver configuration for each frame length.

Solutions during the execution or compilation stage have to be applied when the interleaver is already established. Indeed, the designer does not have the possibility to modify its structure in this case. Note that, since the solution during the execution stage requires the use of buffers, the decoder throughput may be reduced. On the other hand, the solutions during the compilation allow high throughput rates. However, since the memory mapping in most of the cases cannot be analytically described, important amount of memories are required to store it. This is specially true for long frames, where the hardware complexity of this approach may be prohibitive.

The solutions during the design are preferred if the interleaver function is not imposed. In this case, the designer is able to use an interleaver that provides a good error correction performance, getting rid of the memory conflict problems for a given turbo decoder parallelism degree. In current digital communication standards, conflict-free interleavers have been adopted since the high throughput is a major constraint. Indeed, future standards have to be proposed following this type of solution.

In the following sections we introduce the conflict-free interleavers, proposed by following a solution at the design stage. The properties of the Quadratic Polynomial Permutation (QPP) and Almost Regular Permutation (ARP) interleavers are specially discussed.

4.2.1 CONFLICT-FREE INTERLEAVERS

An interleaver function applied over a frame of size L is Conflict-Free (CF) for Q sub-blocks ($N = L/Q$ symbols per sub-blocks) if the equation in (4.1) holds, where $g(\cdot)$ is either $\pi(\cdot)$ or $\pi^{-1}(\cdot)$, $0 \leq i < N$ and $0 \leq a_1 < a_2 < Q$ [133]. In this equation, we assume that a radix-2 architecture is used.

$$\left\lfloor \frac{g(i + a_1 N)}{N} \right\rfloor \neq \left\lfloor \frac{g(i + a_2 N)}{N} \right\rfloor \tag{4.1}$$

This CF interleaver definition guaranties that there is no conflicts when extrinsic

values are acceded through both interleaver and de-interleaver functions. Thus, interleavers that meet this condition are well suited for turbo decoder architectures where the exchange of extrinsic information is not carried out through a bank of memories, but through a Network-on-Chip (NoC) [62]. However, for the architecture that we consider in Figure 4.1, if a trivial memory mapping is applied[1], the memory conflicts in the natural order are resolved. Thus, we only need that (4.1) is satisfied for $g(\cdot) = \pi(\cdot)$.

An interleaver is said to be Maximum Conflict-Free (MCF) [134] if it is CF for each sub-block length N which is a factor of the information frame length L. Interleavers that fall into this category are very interesting in order to design high throughput turbo decoders. Indeed, the sub-block technique can be used at a high parallelism degree[2].

Let us assume that the SISO decoders have the Internal SISO Mem given in Figure 3.2. Let us also consider a non-shuffled architecture. Hence, to avoid memory conflict problems, $M = Q$ ($M = 2Q$) memories are required when the Backward-Forward (Butterfly) schedule is implemented, if a CF schedule is applied. For an ASIC technology, if these memories are physically separated, an important complexity cost is added due to the logic implemented inside each memory. Besides, the Memory Address Generator in Figure 4.1 has to produce, for each memory, a different address for each memory access. As pointed out in [136], it is possible to implement a single memory that stores, in each memory position, Q extrinsic values if the following equation is satisfied:

$$\pi(a_1 N + i)(\mathrm{mod} N) = \pi(i)(\mathrm{mod} N), \text{ for } 0 \leq a_1 < Q,\ 0 \leq i < N. \tag{4.2}$$

An interleaver that fulfills (4.2) is refered to as vectorizable. It is important to mention than an interleaver is CF if and only if it is vectorizable [136]. In the rest of this section, we present the QPP and ARP interleavers, and their relevant properties to design high throughput turbo decoders.

4.2.1.1 QPP INTERLEAVER

The QPP interleaver is defined by a quadratic polynomial function:

$$\pi(i) = f_1 i + f_2 i^2 (\mathrm{mod} L) \tag{4.3}$$

where f_1 and f_2 are nonnegative integers, selected from a restricted set defined by L. The de-interleaver $\pi(i)^{-1}$ is also a polynomial function, but not necessarily quadratic

[1] In this trivial mapping, the extrinsic value $0 \leq k < L$ is assigned to the memory bank $\lfloor k/N \rfloor$ at the memory location $k - \lfloor k/N \rfloor N$.

[2] It is important to keep in mind that a very high sub-blocks parallelism implies an important number of additional iterations. Thus, the decoding performance is the major constraint for very high sub-block parallelism architectures.

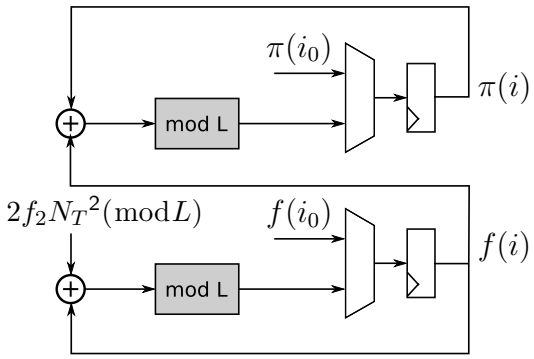

Figure 4.2: Architecture to generate the QPP address in the forward direction.

[137]. In [134], it was demonstrated that any valid interleaver defined by (4.3) is MCF, and thus, also vectorizable. The QPP interleaver function can be computed recursively [136, 138]. Let us consider the interleaver value for $i + N_T$, expressed in terms of the interleaver value for i:

$$\begin{aligned}\pi(i+N_T) &= f_1(i+N_T) + f_2(i+N_T)^2(\mathrm{mod}L) \\ &= f_1 i + f_2 i^2 + f_1 N_T + 2f_2 i N_T + f_2 N_T^2(\mathrm{mod}L) \\ &= (\pi(i) + f(i))(\mathrm{mod}L)\end{aligned} \tag{4.4}$$

where $f(i) = f_1 N_T + 2f_2 i N_T + f_2 N_T^2(\mathrm{mod}L)$. Note that $f(i)$ can also be expressed recursively:

$$f(i+N_T) = f(i) + 2f_2 N_T^2(\mathrm{mod}L) \tag{4.5}$$

With (4.4) and (4.5), and inspired by the architecture proposed in [138], we designed the architecture detailed in Figure 4.2. We propose this architecture in order to generate the interleaver address in the forward direction. Thus, starting at the value i_0, this architecture enables to generate, during each clock cycle, the interlever address every N_T trellis diagram transitions. The modulo blocks are easily designed with an adder and a multiplexor. This architecture should be replicated two and four times if radix-4 and radix-16 architectures are considered, respectively. Note that a similar architecture can be applied in order to generate the addresses in the backward direction.

4.2.1.2 ARP INTERLEAVER

The expression in (4.6) defines an ARP interleaver, where P_0 is an integer prime with L, and $\omega(i)$ is a periodical function with period T. T is called the disorder cycle. It has

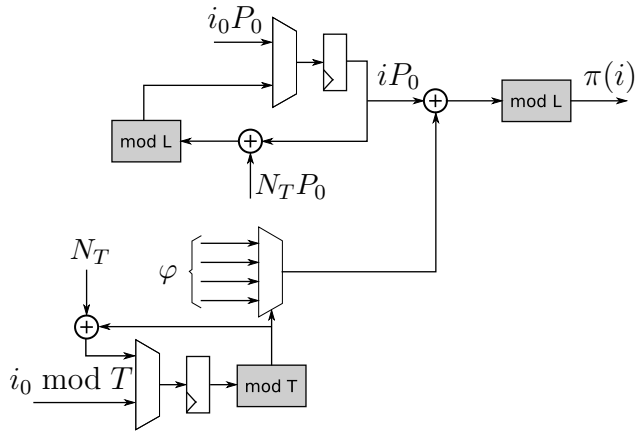

Figure 4.3: Architecture to generate the ARP addresses in the forward direction. $T = 4$.

to be a divider of L.

$$\pi(i) = iP_0 + \omega(i)(\mathrm{mod}\, L) \tag{4.6}$$

The periodical function can be expressed as follows.

$$\omega(i) = E[i \bmod T] + P_0 F[i \bmod T] \tag{4.7}$$

where E and F are vectors of length T. It is assumed that the elements of E and F are multiples of T. Indeed, it is a sufficient, but not a necessary condition, to ensure that the interleaver function is valid [46]. An ARP interleaver is CF and vectorizable, if the number of symbols in each sub-block, $N = L/Q$, is a multiple of T. Thus, this interleaver is less parallelizable than the QPP interleaver, which is MCF.

Figure 4.3 shows the architecture used to generate the interleaver values every N_T trellis diagram transitions, starting at i_0, in the forward direction. A period $T = 4$ is considered. A similar architecture can be applied in order to generate the interleaver addresses in the backward direction.

4.2.2 MEMORY ORGANIZATION FOR CONFLICT-FREE INTERLEAVERS

The definition of CF interleavers in (4.1) guarantees that no conflict problems appear when the sub-block technique is used, as long as a radix-2 architecture is considered. However, for higher radix values, the conflict free property is not necessarily true. For instance, in general the QPP and ARP interleavers are no CF if radix-4 or radix-16 architectures are implemented [124]. Since the results of Chapter 3 have shown the

convenience of high radix values architectures to achieve high throughput rates, we have to propose solutions for radix-4, and specially radix-16 architectures. Moreover, an appropriate architecture for addressing all the memory blocks should be designed, trying to reduce as much as possible its hardware complexity. Let us first tackle the addressing scheme of all the memory banks.

4.2.2.1 MEMORY ARCHITECTURE FOR SUB-BLOCKS TECHNIQUE

In this section, we consider a non-shuffled turbo decoder, where each SISO decoder implements a radix-2 Backward-Forward schedule. We recall that the QPP and ARP interleavers are vectorizable. Therefore, these interleaver functions can be expressed as [135]:

$$\pi(i) = \varphi(i \bmod N) + N \cdot \phi_{\lfloor i/N \rfloor}(i \bmod N) \tag{4.8}$$

$\varphi(i \bmod N) = \pi(i)(\bmod N)$ is a fixed function that defines the position inside each sub-block. $\phi_{\lfloor i/N \rfloor}$ is a function which depends on the the sub-block $\lfloor i/N \rfloor$. This last function determines in which sub-block the extrinsic value i is mapped. Therefore, the set of functions:

$$\phi(j) = \{\phi_0(j), \phi_1(j), \cdots, \phi_{Q-1}(j)\} \tag{4.9}$$

for $0 \leq j < N$ defines the j-th size-Q permutation that determines the sub-block where the j-th data inside each sub-block is mapped. Thus, the j-th data of the sub-block $0 \leq h < Q$ is mapped to the sub-block $0 \leq \phi_h(j) < Q$.

Let $\pi(i)$ be an interleaver function that is CF for Q sub-blocks. In this document, we refer to this interleaver as rotatable with degree n_p, if n_p is the minimum number of permutations $\phi^{(l)} = \left\{\phi_0^{(l)}, \phi_1^{(l)}, \cdots, \phi_{Q-1}^{(l)}\right\}$, for $l = 1, 2, \cdots, n_p$, such that for any j the permutation $\phi(j)$ in (4.9) can be obtained with a barrel shift operation over $\phi^{(l)}$ for a particular value of l. We refer to the set of permutations $\Lambda = \{\phi^{(0)}, \phi^{(1)}, \cdots, \phi^{(n_p)}\}$ as the kernel. Let us explain this definition with an example. Suppose that $\pi(i)$ is a rotatable CF interleaver for $Q = 4$. Let the two permutations in the kernel be:

$$\begin{aligned} \phi^{(1)} &= \{1, 2, 0, 3\} \\ \phi^{(2)} &= \{0, 2, 1, 3\} \end{aligned} \tag{4.10}$$

Therefore, for any $0 \leq j < N$, $\phi(j)$ is equal to either $\{1, 2, 0, 3\}$, $\{3, 1, 2, 0\}$, $\{0, 3, 1, 2\}$, $\{2, 0, 3, 1\}$, $\{0, 2, 1, 3\}$, $\{3, 0, 2, 1\}$, $\{1, 3, 0, 2\}$, or $\{2, 1, 3, 0\}$. $n_p = 2$ is the minimum number of permutations that can be defined as kernel.

For the size-Q permutation ϕ we define the distance between the sub-blocks 0 and 1 as:

$$\Omega(\phi) = \max(\phi_1 - \phi_0, Q + \phi_1 - \phi_0) \tag{4.11}$$

We do not provide a mathematical proof to establish the conditions under which an interleaver is rotatable. However, following a pragmatic approach, we have tested this property for all the frame sizes in the LTE and DVB-RCS standards. Indeed, for the DVB-RCS standard, the ARP interleaver is always rotatable with degree $n_p = 1$. Regarding the LTE standard, for $Q \leq 64$, the ARP interleaver is rotatable with degree $n_p \leq 16$. It is worth to mention that for the LTE interleaver, if $n_p > 1$, for any two different permutations $\phi^{(l_1)}$,$\phi^{(l_2)} \in \Lambda$, the following is valid:

$$\Omega\left(\phi^{(l_1)}\right) \neq \Omega\left(\phi^{(l_2)}\right) \tag{4.12}$$

Therefore, the permutation $\phi(j)$ for any j can be established by only computing $\phi_0(j)$ and $\phi_1(j)$. With these two values, the permutation in the kernel $\phi^{(l)}$ that, after a barrel shifter operation generates $\phi(j)$, can be unambiguously determined. Furthermore, if $n_p = 1$, as in the DVB-RCS or in some cases for LTE, only the computation of $\phi_0(j)$ is required.

Based on these observations, we propose the architecture in Figure 4.4 to generate the extrinsic memory address, and to control the interconnection network. This scheme considers a rotatable interleaver with degree $2 \leq n_p$. It is assumed that the condition in (4.12) is fulfilled. Thus, the Memory Address Generator receives the interleaver addresses for the two first sub-blocks. It is composed of two blocks named Get Interleaver Functions, that compute $\varphi(\cdot)$ and $\phi(\cdot)$ according to (4.8). Thanks to the vectorizable property, $\varphi(\cdot)$ corresponds to the extrinsic memory address. This memory stores in each memory location Q extrinsic values. Note that the output of the priority encoders are sent to the Network Controller. This block generates the Interconnection Network control signals in three steps. First, the permutation $\phi^{(l)}$ in the kernel, such that $\Omega(\phi^{(l)}) = \Omega(\phi(i \bmod N))$, is found[3]. Afterwards, the number of positions that $\phi^{(l)}$ should be shifted in the barrel shifter operation in order to get $\phi(i)$, is determined. Finally, the control signals to perform the barrel shifter operation are generated.

Regarding the interconnection network, we have adopted the architecture proposed in [139], shown in Figure 4.5 for $Q = 8$. In this case, seven control signals $c_0, c_1, \cdots, c_6$ are required. It has been demonstrated that this network is able to manage conflict free memory access in QPP interleavers [139]. Moreover, note that this network is able to

[3]Note that $\Omega(\phi(i \bmod N))$ can be easily computed with $\phi_0(i \bmod N)$ and $\phi_1(i \bmod N)$ according to (4.11).

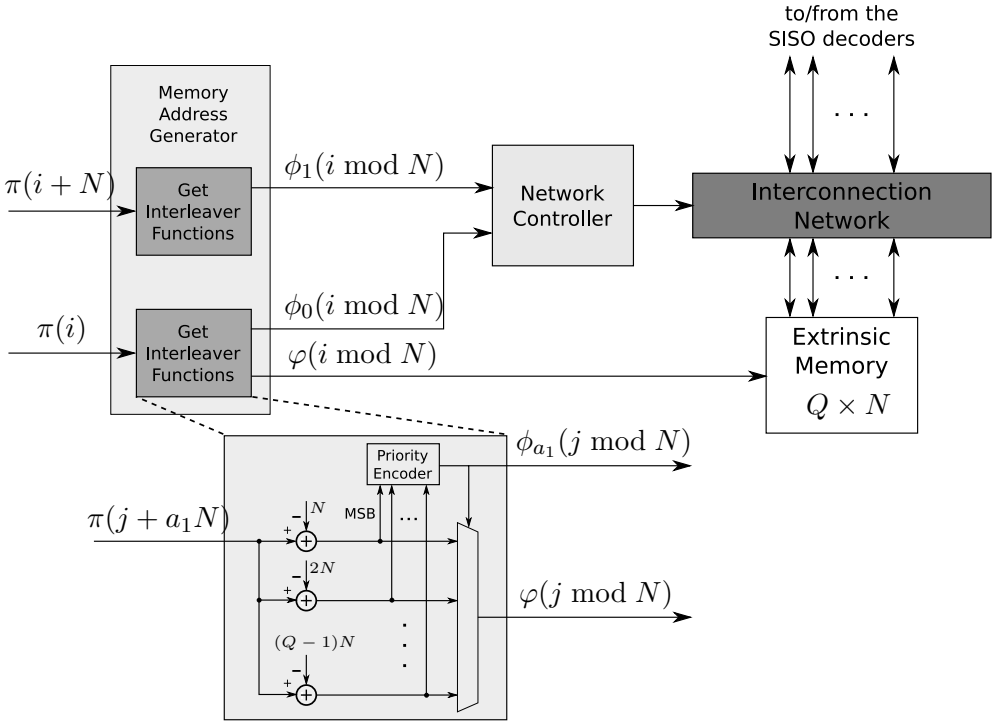

Figure 4.4: Extrinsic memory structure for a rotatable interleaver with degree $2 \leq n_p$, assuming that the condition in (4.12) is fulfilled. A radix-2 architecture is considered with Q sub-blocks. If the degree is $n_p = 1$, only one block *Get Inteleaver Functions* is required.

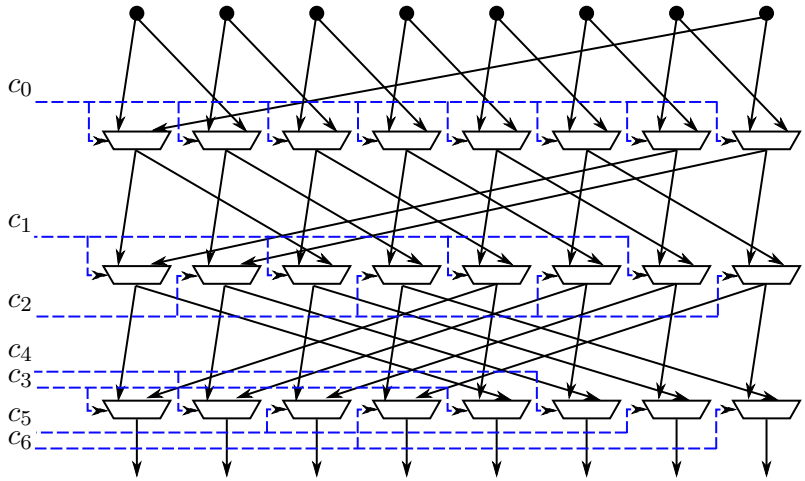

Figure 4.5: Three-stage network proposed in [139] for QPP interleavers.

perform a barrel shift operation for any number of positions up to $Q-1$. Thus, It can be used for any CF rotatable interleaver with degree 1. Note that in [78] a master-slave network has been proposed. It can be used for any CF interleaver. However, more stages are required, at least for $Q=8$, with a higher hardware complexity. Similarly, in [140], a butterfly network is proposed. In this case, the function $\phi(\cdot)$ in (4.8) should be defined under certain restrictions.

With the architecture in Figure 4.4, we have eliminated the necessity of computing the interleaver function for all the sub-blocks, contrary to the architectures proposed in [78, 138]. For high number of sub-blocks, an important reduction in terms of the hardware complexity to generate the extrinsic memory address is achieved[4].

4.2.2.2 MEMORY ACCESS WITH RADIX TECHNIQUE

In order to obtain conflict free memory access in the architectures that contain radix-2^{N_T} schemes using the sub-blocks technique, the condition in (4.1) has to be fulfilled with the following restriction:

$$\left\lfloor \frac{g(i+j)}{N} \right\rfloor \neq \left\lfloor \frac{g(i+j)}{N} \right\rfloor \tag{4.13}$$

with $i(\mathrm{mod}\ N_T)=0$ and $j=1,2,\cdots,N_T-1$. Regarding the ARP interleaver in the DVB-RCS standard, the odd/even rule is satisfied [141]. Thus, if i is even, then $\pi(i)$ is odd and vice-versa. Therefore, radix-4 architectures conflict free can be achieved if the memory is divided to store even and odd values in different memories. Considering the QPP interleaver in the LTE standard, the parameter f_1 and f_2 are always odd and even, respectively. Thus, $\pi(2i)(\mathrm{mod}2)=0$ and $\pi(2i+1)(\mathrm{mod}2)=1$. This property enables to implement conflict free radix-4 turbo decoder architectures. Furthermore, the following equations are valid [138]:

$$\begin{aligned}
\pi(4i)(\mathrm{mod}4) &= 0 \\
\pi(4i+1)(\mathrm{mod}4) &= \begin{cases} 1 & \text{if } (f_1+f_2)(\mathrm{mod}4)=1 \\ 3 & \text{if } (f_1+f_2)(\mathrm{mod}4)=3 \end{cases} \\
\pi(4i+2)(\mathrm{mod}4) &= 2 \\
\pi(4i+3)(\mathrm{mod}4) &= \begin{cases} 3 & \text{if } (f_1+f_2)(\mathrm{mod}4)=1 \\ 1 & \text{if } (f_1+f_2)(\mathrm{mod}4)=3 \end{cases}
\end{aligned} \tag{4.14}$$

[4] However, we have to keep in mind that the address generator generally represent a small percentage of the overall turbo decoder complexity.

Considering a radix-16 architecture, memory conflict problems can be avoided by implementing four memory banks. The size of each memory is $N/4$. In each position, Q extrinsic values are stored. Each memory is assigned to store the values determined by the conditions in (4.14). In this case, we can implement the simplified addressing architecture proposed in section 4.2.2.1.

When iterleavers that are not well suited to avoid memory conflicts in high radix architectures, a solutions during the execution or the compilation has to be applied. Since the latter one is more appropriated for high throughput decoding, this solution is adopted in this study. Thus, in section 4.4 we introduce a methodology applied during the compilation, to design high throughput turbo decoders for any interleaver function.

4.2.3 SHUFFLED TURBO DECODERS MEMORY CONFLICT PROBLEMS

Shuffled turbo decoder implementations impose additional memory constraints. A careful analysis of the extrinsic information values exchanged between SISO decoders in the natural and interleaved order has to be carried out in order to avoid a complete duplication of the extrinsic information memory.

The memory access sequence performed by a shuffled turbo decoder with one sub-block $Q = 1$ $(P = 2)$ is presented in Figure 4.6. Both SISO decoders implement the Butterfly-Replica schedule. In the left side of the figure, the SISO decoder schedules are depicted $-$ natural order SISO decoder in the top and interleaved order SISO decoder in the bottom $-$. In the right side of the figure, the memory accesses performed by both SISO decoders are represented. Indeed, in the figure two possible cases are depicted. First, let us consider the information symbol j in the natural order, that is mapped by the interleaved to $\pi^{-1}(j)$. For this information symbol, the extrinsic value is read by the SISO decoder in the natural order, followed by a writing operation done by the same SISO decoder. After that, the SISO decoder in the interleaved order performs twice a reading operation followed by a writing operation. Afterwards, the natural SISO decoder reads and writes the extrinsic value. In this case, the first data written by the SISO decoder in the interleaved order is not consumed by the SISO decoder in the natural order. Thus, it can be removed without affecting the turbo decoder performance. If the SISO decoder contains an internal memory (Figure 3.2(b)), the second reading operation executed by the interleaved SISO decoder can also be removed. In this case, the extrinsic memory architecture is simplified thanks to an additional SISO decoder internal memory.

Information symbol i in Figure 4.6 presents what is called a consistency problem. During the decoding process, a reading operation done by a SISO decoder in one domain is followed by a reading operation performed by the SISO decoder in the other domain. This occurs for a writing operation as well. Thus, if only one memory position is used for

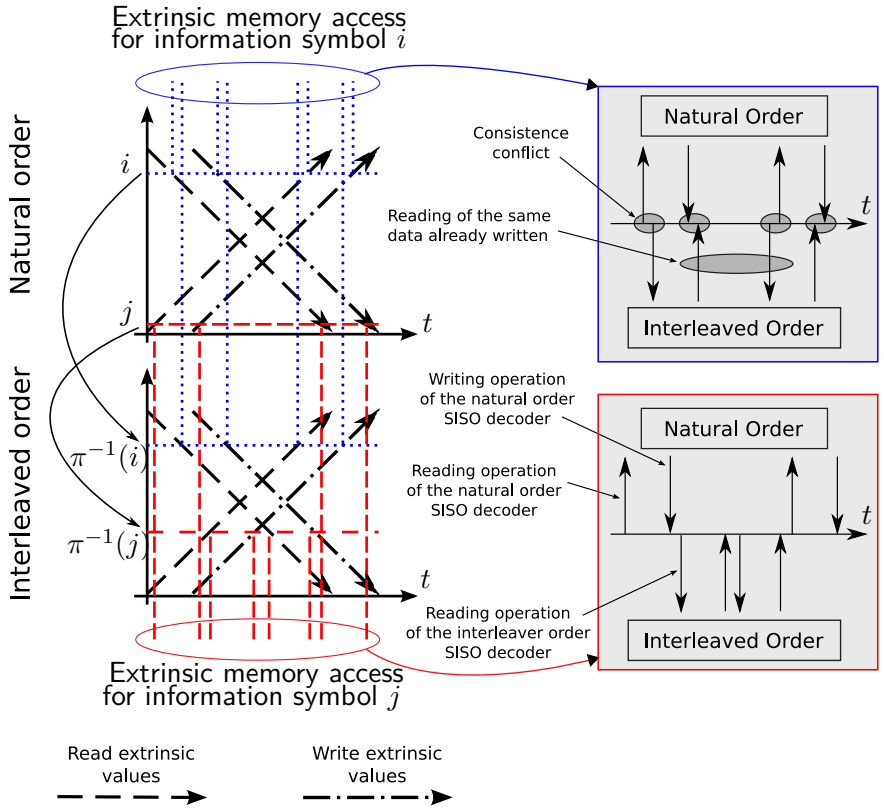

Figure 4.6: Shuffled turbo decoder memory issues.

the extrinsic information of symbol i, the same extrinsic value is read in both orders, and the extrinsic value that has to be produced by one SISO decoder is missed. In this case, a sever performance degradation in terms of the correction capabilities occurs. Note also that the interleaved order SISO reads always the same data that it has written before.

Concurrently access to the same extrinsic value is also possible in a shuffled turbo decoder. Indeed, the extrinsic information for the same symbol can be computed in the natural and interleaved orders during the same clock cycle. However, this kind of memory conflict does not appear in non-shuffled architectures, since any information symbol does not belong to more than one sub-block.

In order to solve consistency conflicts and concurrent accesses to the same memory location, additional extrinsic memory locations are required. Note that by avoiding unnecessary reading and writing operations, as long as it does not affect the turbo decoder performance, the whole extrinsic memory is not duplicated.

4.3 LTE HIGH THROUGHPUT TURBO DECODER ARCHITECTURE

In the previous section we have described the memory conflict problems in parallel turbo decoder architectures. Conflict free interleavers were specially considered, and a novel architecture to address the extrinsic memory was introduced. In this section, we present some architectural solutions to design a high throughput turbo decoder for the LTE standard. They take advantage of the QPP interleaver properties. The SISO decoder presented in the previous chapter is used.

4.3.1 SISO DECODER ARCHITECTURE

We adopt the radix-16 SISO decoder architecture presented in Chapter 3, section 3.4. This architecture enables to achieve high throughput rates. However, since the shifting technique described in section 3.4.2.2 has to be implemented, some architectural constraints appear in the turbo decoder architecture design. Let us first consider the most appropriate SISO decoder schedule for the implementation.

We implement a non-shuffled turbo decoder. The justification for this choice is presented latter on, from the results obtained in section 4.4. Thus, we have to decide for the best schedule between Butterfly and Backward-Forward. It is also necessarily to choose if the sliding window technique is applied, and the sub-blocks and windows initialization method.

Considering the sliding windows technique, the Butterfly schedule is less efficient than the Backward-Forward schedule when medium to large sub-blocks are used [50]. However, if the sub-block size decreases, the Butterfly schedule becomes more efficient. It is due to the larger sub-block decoding time required by the Backward-Forward schedule, by comparison with the Butterfly schedule. This decoding time becomes more significant for small sub-blocks. For high throughput turbo decoders, the sub-block technique with a large number of SISO decoders is mandatory. Hence, the sub-blocks have a medium to small size. In this case, the Butterfly schedule becomes more attractive.

The sliding window technique is necessary to reduce the hardware complexity overhead due to the implementation of large state metric memories. This is specially true when a low number of sub-blocks is used. However, as the number of SISO decoders increases, and thus, the sub-block size decreases, the benefits of the sliding window technique are less important. In [138], the Butterfly and Backward-Forward schedules, without and with sliding windows, respectively, have been compared. It has been reported that for a large sub-block size, the hardware complexity of the non-sliding window Butterfly schedule is much larger compared to hardware complexity of the sliding window

Backward-Forward schedule. In this case, the state metric memory accounts for a very high hardware complexity if the sliding windows is not used. However, as the number of SISO decoders increases significantly, the difference in complexity between both schedules decreases rapidly. Thus, for a frame size $L = 6144$ and $Q = 64$ sub-blocks, the non sliding window Butterfly schedule is more efficient than the sliding window Backward-Forward schedule (window size $W = 64$), in terms of the Complexity-Timing (CT) product. Note that in [138] it was also reported that the use of a radix-4 architecture improves the CT product of the Butterfly schedule, but not necessarily that of the Backward-Forward schedule. Thus, for a low number of sub-blocks, the CT product of both schedules becomes closer as the radix values increases.

We propose then to use the Butterfly schedule. The sliding window technique is avoided as long as a high number of sub-blocks are implemented (small sub-block size), and a high radix value is used. Regarding the sub-block initialization technique, from the conclusions presented in section 2.2.2.1, the message passing initialization is required. It should be avoided any acquisition operation, since it penalizes the throughput for a high number of SISO decoders [62, 142].

4.3.2 Extrinsic Information Memory Access

Since we are using a radix-16 SISO decoder implementing the Butterfly schedule, eight extrinsic values are produced or consumed during each clock cycle (four in the top and four in the bottom of the Butterfly). Thus, the extrinsic memory architecture has to be designed with conflict free access to all the $8P$ extrinsic values required by the SISO decoders in the turbo decoder architecture. Also, note that since we are using the proposed low complexity radix-16 SISO decoder, the shift technique has to be implemented in order to avoid decoding performance degradation. Figure 4.7 depicts the decoding operations over a frame, when a shift of $n_s = 2$ bits is applied. Four SISO decoders are considered. In each sub-block, four radix-16 trellis diagram transitions should be performed. Due to the shifting operation, the SISO decoder 4 performs five radix-16 trellis diagram transitions, while the other SISO decoders perform only four. In this case, memory conflict problems appear if all the SISO decoders start to work at the same time. In order to overcome this problem, the SISO decoder 4 has to perform the first backward radix-16 trellis diagram transition prior to any other SISO decoder in the architecture. Then, all the SISO decoders operate simultaneously. Thus, the conflict free properties of the interleaver can be exploited. Note that this solution works well for $n_s = 1, 2, 3$.

Figure 4.8 presents the turbo decoder architecture that we propose. For simplicity, only the communication links from the extrinsic memories to the SISO decoders are

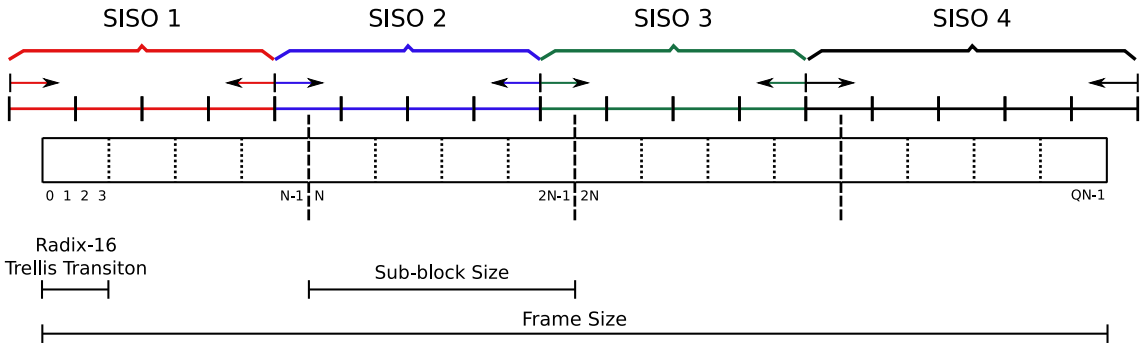

Figure 4.7: Frame shift principle when the sub-block technique is used. Radix-16 SISO decoder architecture. $n_s = 2$, $Q = P = 4$. Four radix-16 trellis diagram transition per sub-block.

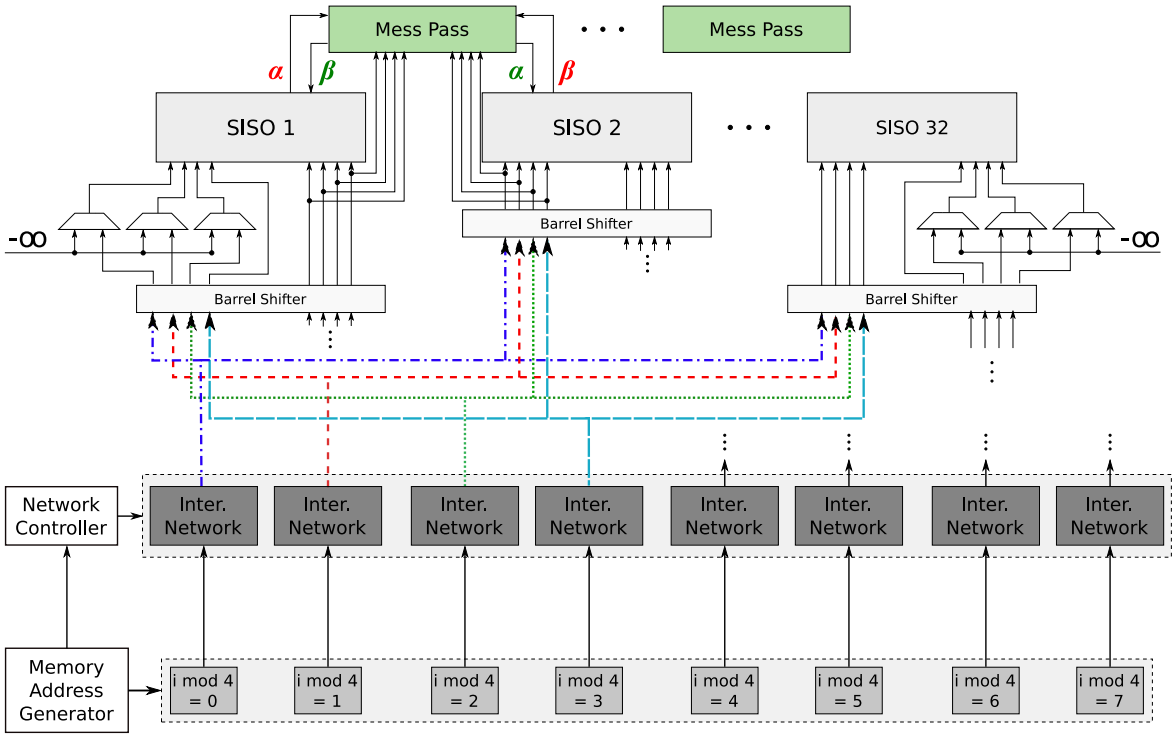

Figure 4.8: Conflict free access for a turbo decoder that is composed of radix-16 SISO decoders and a QPP interleaver.

plotted. Note that some logic has to be added in order to perform the message passing between adjacent SISOs. Indeed, due to the shift technique, the limits of the sub-blocks are different in consecutive iterations with different n_s values. In this case radix-2 BMUs and ACS units are used to compute α and β in the limits of the sub-blocks for the next iteration.

Eight memories are required to avoid memory conflicts. Each one store the extrinsic values for a given value $i \mod 4$. In each memory location Q extrinsic values are stored. Therefore, $W_{RAM} = N/8$ different values can be addressed in each memory.

The Memory Address Generator is based on the architecture proposed in section 4.2.2.1. Eight interconnection networks, implemented with the architecture in Figure 4.5, are used. Note that the SISO decoders receive the extrinsic values through a Barrel Shifter. It is required due to the shift technique. Therefore, when $n_s > 0$, the barrel shifters have to rotate the extrinsic values n_s positions. By this way, we are able to simplify the interconnection network architecture, since no additional constraints between any pair of extrinsic values i and j have to be taken into account if $(i \mod 4) \neq (j \mod 4)$. Note that three multiplexors are necessary at the input of the first and $P\text{-}th$ SISO decoder. They are used to select either the values that come form the extrinsic memories, or a negative value with a large magnitude. Thus, the radix-16 trellis transitions in the frame limits are correctly managed, as previously explained in section 3.4.2.2.

4.3.3 FPGA PROTOTYPING OF THE TURBO DECODER

In order to validate the architectural solutions proposed in this section, an FPGA prototype has been carried out. The board *DN9000K10PCI* [5], available at the electronics department of Telecom Bretagne, has been used. This board is composed of 6 Xilinx Virtex 5 devices with appropriate communication channels. Also, an USB interface is available to communicate with a host computer.

Two different FPGAs in the board were assigned to implement the transmitter, the channel and the receiver, as shown in Figure 4.9. In the transmitter side, a pseudo-random bit sequence is generated by a Linear Feedback Shift Register (LFSR) in order to emulate the information frame produced by the source. This sequence of bits is then received by the turbo encoder. Two buffers are implemented in order to store the information frame in the natural and interleaved order before the convolutional encoders operation. In order to emulate the communication channel, a Gaussian noise generator, based in the Wallace method [143], was implemented as presented in [144]. The host computer sets the noise variance in function of the SNR value. In the FPGA dedicated

[5] http://www.dinigroup.com/new/DN9000k10PCI.php

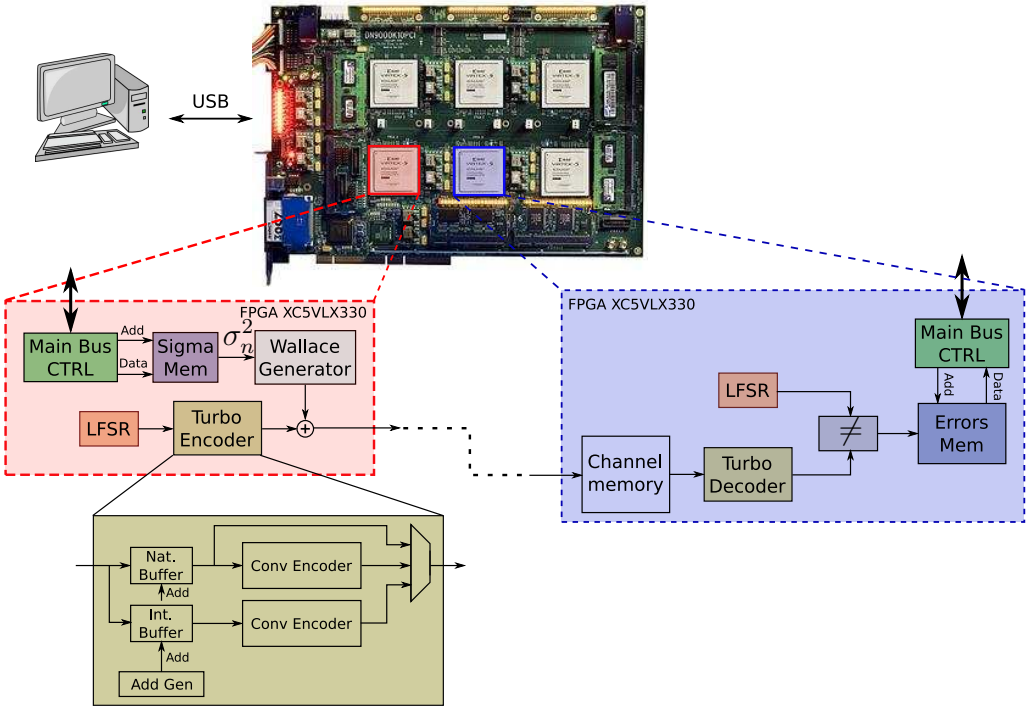

Figure 4.9: Turbo Encoder/Decoder on-board prototype.

to the implementation of the receiver, a memory stores the LLR values that correspond to the channel information. After the turbo decoding, the number of bits incorrectly decoded is computed. An LFSR, identical to the LFSR in the transmitter, is implemented to enable to do the comparison. The host computer has access to the number of erroneous bits and computes the BER and FER values. It contains a graphical user interface in order to setup various parameters of the board, download the corresponding configuration bitstreams, and display the BER/FER decoding performance.

Table 4.1 presents the synthesis results of a first FPGA implementation when $Q = 1$ sub-block is considered. The frame size is set to $L = 1024$ bits. The required hardware resources for both FPGAs are given. The architecture can work at a frequency of 100 MHz. The measured BER and FER performance are shown in Figure 4.10 after 6 turbo decoder iterations.

To assess our ideas in a very high parallel context, the turbo decoder complexity when $Q = 32$ sub-blocks are used was estimated using 90nm ASIC technology from STMicroelectronics. A frame size of $L = 1024$ bits was considered. Thus, for a clock frequency of 200MHz a throughput of 1,13Gbit/s is expected. Taking advantage of the pipeline stages of the SISO decoders, it is possible to decode two frames simultaneously

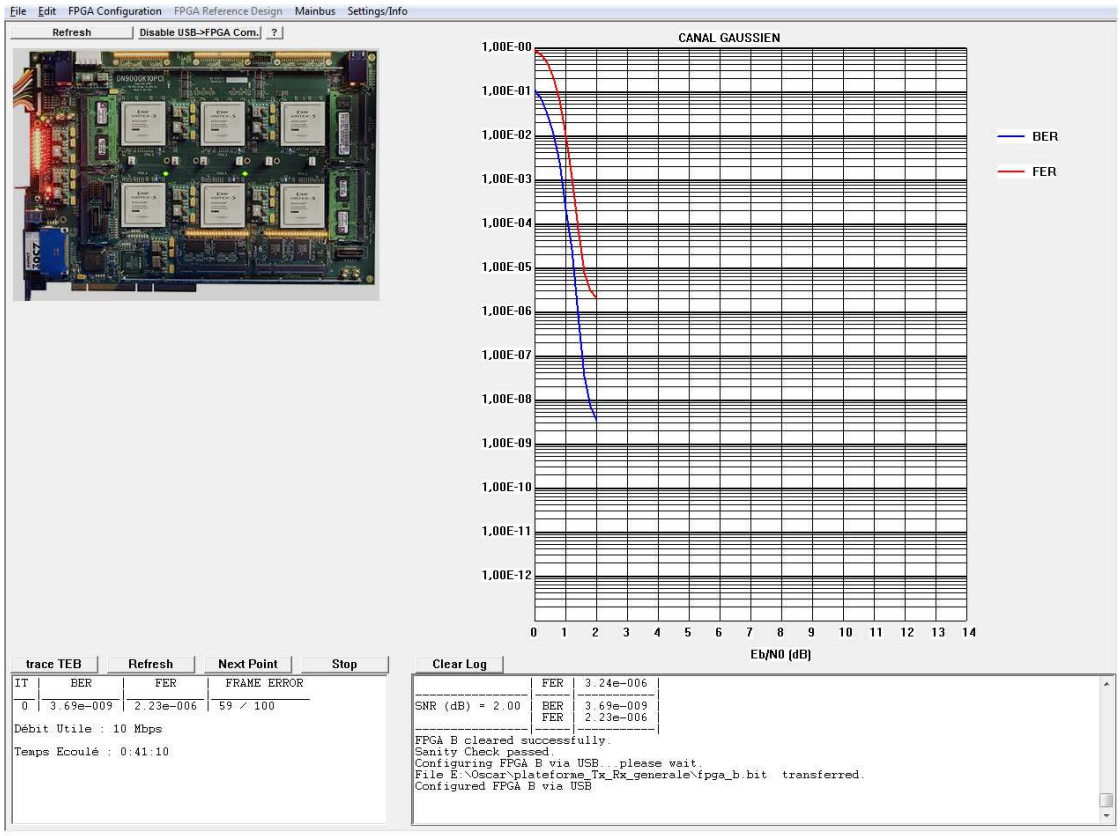

Figure 4.10: BER and FER of the SISO decoder measured on on-board prototype with 6 Iterations.

by duplicating the extrinsic and channel memory resources. In this case the expected throughput is about 2,27Gbit/s with an estimated complexity of 5,13M logic gates. The system complexity is distributed as presented in figure 4.11.

4.4 TURBO DECODER DESIGN SPACE EXPLORATION

The impact on the hardware complexity and throughput, that results when a set of parallel turbo decoding techniques are combined, is usually determined at the end of the design process, after the turbo decoder architecture has been synthesized. Thus, the time to market is penalized and the probability of designing a suboptimal system increases. In this section we address this problem by introducing a dedicated approach to efficiently explore the design space of parallel turbo decoder architectures. Thanks

	Transmitter & Channel	Receiver
Slice Registers (out of 207,360)	746 (<1%)	5,948 (<3%)
Slice LUTs (out of 207,360)	1,419 (<1%)	19,382 (9%)
DSP48Es (out of 192)	7 (3%)	0

Table 4.1: Logic synthesis results of the FPGA Xilinx Virtex 5 xc5vlx330 device. The achievable frequency is 100 MHz.

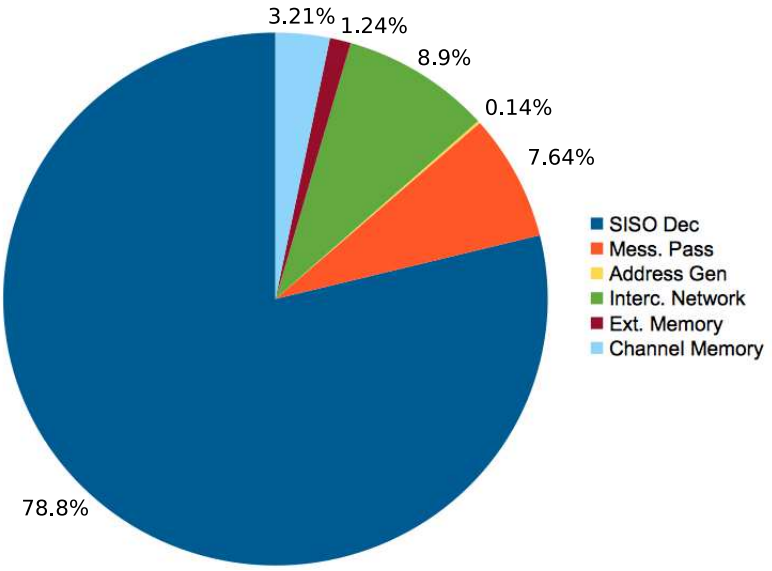

Figure 4.11: Resources for a parallel turbo decoder with $Q = 32$ sub-blocks and the proposed radix-16 SISO decoder.

to this approach, a tradeoff between the hardware complexity and the throughput can be established in the early stages of the architecture design process. The approach, contrary to the architecture presented in the previous section, implements a solution during the compilation to solve the memory access conflicts. In this way, we are able to design a high throughput architecture for any parallelism and interleaver. However, a penality in terms of the hardware complexity is expected. Note that the most part of the contents of this section have been published in [145]. This work has been carried out in collaboration with S. ur Rehman, A. Sani, C. Chavet and P. Coussy, who are with the Université de Bretagne-Sud, at Lorient.

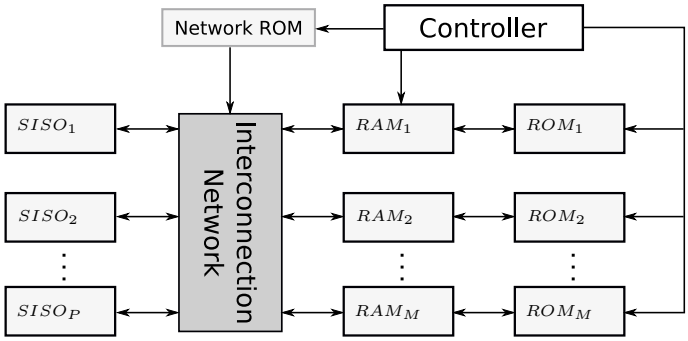

Figure 4.12: Turbo decoder architecture model.

4.4.1 TURBO DECODER ARCHITECTURE MODEL

Our approach considers P SISO decoders, that can implement any parallelism at the metric level, and that can work in either shuffled or non-shuffled mode. The turbo decoder architecture is given in Figure 4.12. In order to reduce the memory complexity, single port RAM are allocated to store the extrinsic information values. Read only memories are used to address each memory block and also to control the signals of the interconnection network. The addressing of the memory blocks is based on the interleaved function and on the data required by the SISO decoders during each clock cycle. Thus, a controller has to be designed to address the ROM memories and to generate the control signals of the memory blocks. Note that channel information memories are omitted in Figure 4.12.

We recall that each sub-block is formed by $N = L/Q$ symbols with $Q = P$ for non-shuffled turbo decoders, and $Q = P/2$ for shuffled turbo decoders. Let T_c denotes the number of clock periods during which each SISO decoder performs writing or reading memory access in order to execute one iteration (for a non-shuffled turbo decoders), or a half iteration (for a shuffled turbo decoders). Let W_{RAM} represents the size of each RAM memory. Then, the size of each addressing ROM is: $T_c * \lceil \log_2(W_{RAM}) \rceil$. Note that an iteration in a shuffled and non-shuffled turbo decoders has the same duration if both turbo decoders have the same sub-block size N.

We consider Butterfly and Backward-Forward schedules for non-shuffled architecture, and Butterfly-Replica schedule for shuffled architectures. The SISO decoder architectures that are detailed in Chapter 3, section 3.1, are used. Radix-2^{N_T} ACS architectures are considered. Let σ be the maximum number of reading or writing extrinsic values accesses performed by each SISO decoder during each clock cycle. If the Internal SISO Mem is used in the Backward-Forward schedule, then a maximum of $\sigma = N_T$ extrinsic values should be accessed simultaneously from the memory banks. However, if this memory

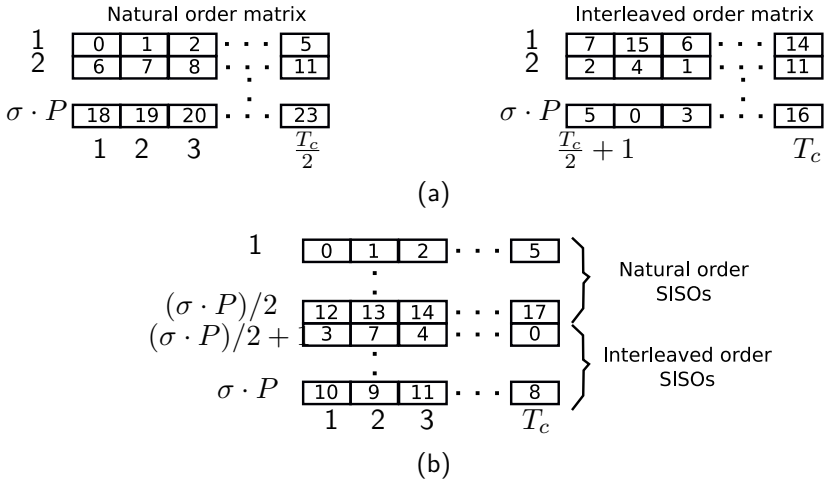

Figure 4.13: Data access matrices. (a) Non-shuffled turbo decoders. (b) Shuffled turbo decoders.

type is not used, or if the other two schedules are considered, then $\sigma = 2N_T$. Thus, the number of memories that should be assigned to facilitate a contention free memory mapping is given in (4.15). Radix values for $N_T = 1, 2, 4$ are considered.

$$M = \sigma \cdot P \tag{4.15}$$

4.4.2 A Dedicated Approach to Explore the Design Space

In the first part of this section the adopted solution to solve memory conflict problems is presented. The second part is devoted to present the approach that we propose.

4.4.2.1 Solving Memory Conflicts

Shuffled and non-shuffled turbo decoders have two different types of memory conflict problems. To tackle them, two approaches are proposed in this section: one for non-shuffled decoding and another one for shuffled decoding in turbo decoders with any radix value and any schedule. Both approaches can be applied to find conflict free memory mappings.

The algorithm proposed in [129], based on the transportation problem, is executed to solve memory conflicts for non-shuffled turbo decoders. Data access order can be illustrated through data access matrices as shown in Figure 4.13(a). In this Figure, one matrix is related to the natural order access and another one is related to the

interleaved order access. Each matrix has $\sigma \cdot P$ rows for the extrinsic values accessed by the SISO decoders, and $T_c/2$ columns for the time instances. Data elements in each row are processed by the same SISO (each SISO accesses σ data elements, σ rows in the matrix). Similarly, the $\sigma \cdot P$ data elements in each column have to be accessed in parallel by P SISO decoders.

The algorithm presented in [129] finds memory mapping in two steps. In a first step, a bipartite graph $= (T_c \cup F, E)$ is built in which vertex set T_c represents all the time instances and vertex set F represents all the data elements. An edge (t_c, f) is incident to the data element vertex f and to the time instance vertex t_c, if f has to be processed at t_c. During a second step, transportation problem is used to partition this bipartite graph into different semi 2-factors and to allocate data in different memory banks.

For shuffled turbo decoders, both natural and interleaved orders are processed concurrently as shown through data access matrix in Figure 4.13(b). The approach proposed in [128], based on edge coloring of tripartite graph, is used to solve the memory mapping problem. In this approach, data can be read from one memory bank and then it can be written in a different bank, in order to allocate data in minimum number of memory banks. Initially, data access matrix is modeled as a tripartite graph $G = (T_R \cup T_W \cup F, E)$ in which vertex sets T_R and T_W represent all the time instances during which data are read and written, respectively. In this graph model, vertex set F represents all the data used in the computation. After the construction of the tripartite graph G, subgraph partitioning technique is applied to divide this graph into different subgraphs with specific characteristics to solve mapping problem. Finally, edges of each subgraph are colored separately to allocate data in different memory banks.

4.4.2.2 Proposed Design Flow

The proposed design flow is detailed in Figure 4.14. Inputs are the interleaver function $\pi(i)$, the parallelism type of the turbo decoder architecture (P, shuffled/non-shuffled), the selected SISO architecture (schedule, window size W if any, internal SISO Mem, number of pipeline stages and N_T that defines the radix used) and the parameters of the interconnection network (number of pipeline stages that defines τ_{rd}). Note that the number of pipeline stages in the SOU and in the interconnection network define the value of the parameter τ_{wr}.

The first step in the design flow is the generation of the memory access description files. These files contain the sequence of extrinsic information values that are read or written by each SISO decoder during each clock cycle. They are generated in three steps: extrinsic values that are acceded concurrently by all the SISO decoders (depending on P, N_T and $\pi(i)$) are listed for each clock cycle, eliminating unnecessary reading and writing

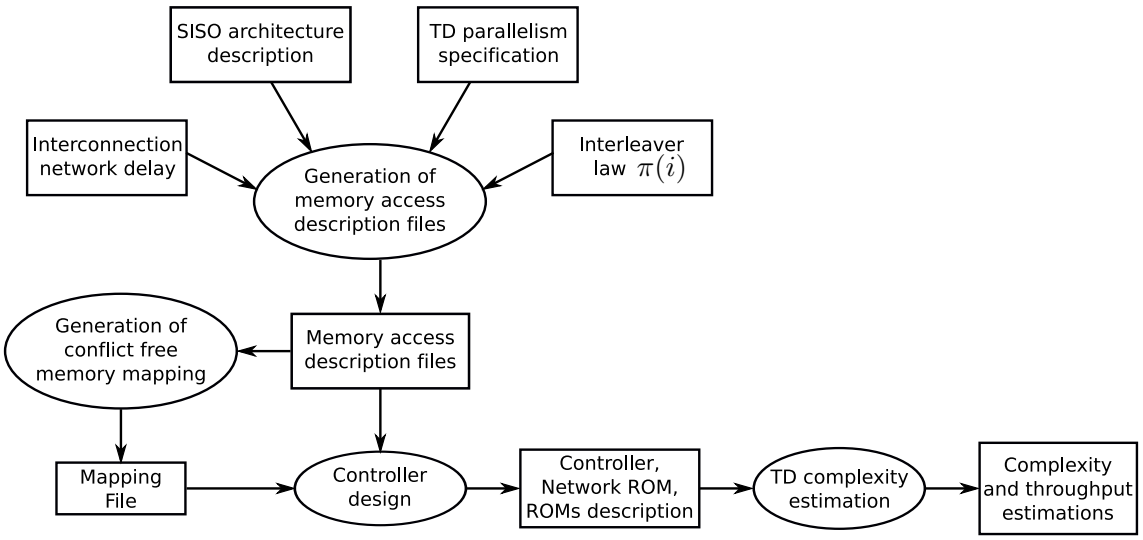

Figure 4.14: Design Flow for architectural space exploration.

accesses (section 4.2.3). Additional extrinsic memory locations are used if concurrent access to the same extrinsic value occur ensuring that a correct exchange between natural and interleaved order is achieved. Finally, consistency problems are resolved by adding extrinsic memory locations as well. Let l denotes the number of additional memory positions used to solve concurrent access to the same extrinsic value and consistency problems. Note that the memory access description files are generated automatically by the executions of some scripts.

Memory access description files enable to carry out conflict free memory mapping as presented in section 4.4.2.1. Thus, $L + l$ extrinsic values are assigned to M memory banks. From this memory mapping, the controller can be designed and the content of ROM memories (Figure 4.12) can be established. Finally, estimations of the turbo decoder throughput and hardware complexity are done.

4.4.3 CASE STUDY: TURBO DECODER FOR LTE STANDARD

To illustrate our approach, we have applied it to a turbo decoder for the LTE standard. A frame size $L = 1024$ and a code rate $R = 1/2$ are considered. Parallelism is explored for $P = 16$ and 32 SISO decoders that can implement radix-2, radix-4 or radix-16 ACS unit architectures. Shuffled and non-shuffled architectures are considered with schedule Butterfly-Replica and Butterfly, respectively. In non-shuffled case, there are $Q = 16$

and 32 sub-blocks. In shuffled case, since the processors are distributed in both orders, $Q = 8$ and 16. Thus, we consider sub-blocks sizes $N = 32, 64$ and 128. Sliding window technique is used in non-shuffled decoders with a window size of 32. We have chosen $w_r = 5$ bits to quantify the channel information, and $w_{ext} = 6$ bits to represent the extrinsic information. $w_{SM} = 9$ bits are fixed for M^{α} and M^{β} state metrics values in radix-2 architectures. These quantization values where established through Monte Carlo simulations. Here we have accepted a degradation lower than 0.1 dB with respect to a floating point model in order to reduce the hardware complexity.

The interconnection network is implemented by a $M \times M$ beneš network [146], with M given by (4.15). To prevent the critical path of the turbo decoder to be in the interconnection network, pipelining stages have to be introduced. From logic synthesis results we have established that one pipeline stage is enough for $M = 32$ and 64, while two stages are necessary for $M = 128$ and 256. These pipeline stages define the value of τ_{rd}.

For the design space exploration, we have fixed the clock frequency to 500MHz, where an ASIC target (90nm) is considered. All the blocks in the SISO decoder, other than the ACS unit, are pipelined to respect the clock frequency constraint. Thus, the critical path is in the ACS unit. 6 pipeline stages are necessary for radix-2 and radix-4 SISO decoders. Since the radix-16 version is more complex in terms of logic gates, 8 pipeline stages are required.

Turbo decoders with different parallelism levels are compared as long as there is no performance degradation. To this end, a non-shuffled turbo decoder with $Q = 1$ sub-block with 6 decoding iterations is taken as reference. All the parallel turbo decoders should execute a certain number of iterations such than their BER performance is not worst by comparison with the reference. Through extensive Monte-Carlo simulations, the number of iterations for each turbo decoder architecture was determined. Shuffled decoders should execute 3.5 and 4 iterations when 16 and 32 SISO processors are chosen respectively, if the SISO decoder internal memory is not used. Furthermore, 2 additional iterations are required when internal memories are added. Non-shuffled turbo decoders should execute 7 and 9 iterations for 16 and 32 SISO decoders, respectively. Note that, contrary to shuffled architectures, the use of internal memory in the non-shuffled configurations does not affect the turbo decoder performance.

Nine turbo decoder configurations are selected. They have been studied for 16 and 32 SISO decoders. The approach presented in section 4.4.2.2 has been applied to the nine configurations. The hardware cost of the different components of the resultant architecture was estimated using 90nm ASIC technology from STMicroelectronics. The corresponding area is expressed in terms of logic gates. Figure 4.15 shows the equivalent number of logic gates for all the configurations with respect to the number of clock

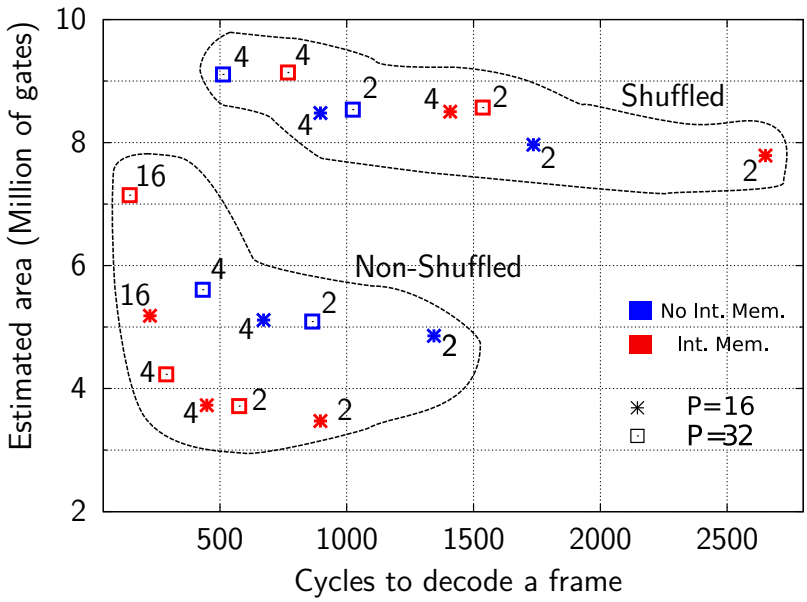

Figure 4.15: Area estimations of the considered turbo decoder architectures in function of the number of clock cycle to decode a frame. The numbers by each point indicate the radix value.

cycles necessary to decode a frame (directly related to the turbo decoder throughput), for equivalent BER performance. Numbers by each point denotes the radix value. In blue and red are indicated the configurations that implement or not an internal memory, respectively.

Note that non-shuffled configuration with internal memory are Pareto-optimal architectures for 2, 4 and 16 radix values. In non-shuffled turbo decoder configurations, the use of internal SISO decoder memories is convenient since it has a positive impact on the turbo decoder throughput. Indeed, it reduces the number of reading accesses. It also helps to reduce the hardware complexity of the whole architecture. However, for the shuffled configurations, the turbo decoder hardware complexity is slightly reduced. But the throughput is significantly affected due to the 2 additional iterations that should be executed in order to avoid BER performance degradation. Since less reading accesses are performed when the internal SISO memory is used, the sizes of the ROM memories in Figure 4.12 can be decreased. Consequently, the whole turbo decoder hardware complexity is reduced.

For radix-2 and radix-4 configurations, a significant decoding time enhancement is obtained when switching from 16 towards 32 SISO decoders with minimal area overhead. It leads to the conclusion that the relative area of the SISO decoders is not the dominant

term in the whole turbo decoder complexity. However, for the radix-16 architecture this area overhead becomes more important, since the SISO decoders exhibit a higher complexity.

From the considered configurations, no advantage was observed for the shuffled turbo decoder architectures. They correspond to more complex architectures, that do not present significative throughput improvements. This is due to the additional extrinsic memory positions used to solve consistency problems and concurrent access to the same memory position. Additionally, since single port RAM memories have been considered, reading and writing accesses can not be performed simultaneously. This penalizes the number of cycles per iteration in the Butterfly-Replica schedule, that performs intensive reading and writing operations. ROM memories in the designed architecture are the most costly element. Since they are used to address the RAM memories, future works have to be oriented toward the proposition of alternative addressing schemes. Conclusion from this investigation could restore the interest in shuffled turbo decoders.

4.5 CONCLUSION

In this chapter an effort is made in order to overcome the architectural problems that arise when multiples SISO decoders are assigned in a parallel turbo decoder. Regarding the extrinsic memory conflicts, we have proposed two approaches to build high throughput turbo decoders. In the first one, taking advantage of the conflict free interleavers properties, we propose a low complexity memory addressing architecture, a memory organization scheme, and an interconnection network architecture in order to use radix-16 SISO decoders. Besides, we have demonstrated that the low complexity radix-16 SISO decoder proposed in Chapter 3, can be integrated in high parallel turbo decoder architectures. In the second approach, a solution during the compilation is adopted. In this case, a dedicated approach is proposed to explore the design space, and propose architectures that support any parallel turbo decoding technique, regardless the properties of the interleaver function.

The main objective of the dedicated approach introduced in section 4.4.2.2 is to reduce the time to market constraint by designing turbo decoder architecture for a given throughput. To validate our approach, different configurations based on shuffled and non-shuffled schemes have been investigated. A novel architecture to decode turbo codes using shuffled scheduling is also proposed. Memory conflict problem, concurrent memory access problem and consistency conflict problem are considered to guarantee the BER performance and also to reduce the hardware architecture cost. In future works, an optimization in the addressing and control logic has to be proposed. Note that the shuffled architectures have not shown significative improvements with respect

to non-shuffled ones.

This chapter has also presented the results of a first prototype designed to validate our propositions. This prototype has demonstrated the feasibility of using the radix-16 SISO decoder architecture presented in the previous chapter. Currently, a prototype that integrates 32 SISO decoders is under development. Results from this prototype would help to establish the advantages of our architecture as presented in section 4.3. A throughput of 1Gbit/s is expected for 6 turbo decoder iterations. Besides, since the SISO decoder complexity has been reduced, an efficient architecture in terms of the achieved throughput per hardware complexity is anticipated with respect to other architectures proposed in the literature.

Chapter 5
Conclusion and Perspectives

The work accomplished during this Ph.D is motivated by the increasing demand of high throughput decoders that are required in current and future digital communication systems. We concentrate our efforts in the design of high speed convolutional turbo decoder architectures. For this purpose, a complete exploration of the design space, from the algorithmic level to the architecture details, has been performed.

The turbo codes are a well known channel coding technique widely used because of their outstanding error decoding performance close to the Shannon limit. Their discovery occurred around twenty years ago. At present, their understanding has reached a mature level thanks to the huge number of research works that have been carried out. The turbo codes consist in a concatenation of several constituent codes related by interleaver functions. Their decoding is performed throughout an iterative process where different Soft Input Soft Output (SISO) decoders exchange the so-called extrinsic information. This iterative decoding process enables to achieve excellent error decoding performance. However, it prevents the achievement of high throughput values. While turbo decoder architectures that provide low (a few Mbit/s) and medium (around 100 Mbit/s) throughput rates have been already proposed, there is a lack of architectural solutions that support high (1 Gbit/s and higher) throughput rates. Therefore, design challenges appear in order to cope with the high speed requirements of current digital communication systems.

We have started by an introduction of the basic concepts concerning the convolutional codes and the turbo decoding principle. Then, a review of different convolutional decoding algorithms was done. The main motivation of this review was to establish the convenience of other decoding algorithms, different to the de-facto Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm, in order to reduce the hardware complexity or increase the decoding speed. Thus, algorithms based on the Maximum Likelihood (ML) and Maximum A Posteriori (MAP) principles have been considered. Regarding the ML principle, the Soft Output Viterbi Algorithm (SOVA) has been analyzed. This algorithm is spe-

cially interesting for binary codes, where a reduction in the hardware complexity can be achieved. Since the main problem of the SOVA is its poor error correcting performance, techniques to reduce the algorithm degradation have been explored. Thus, degradations around 0.2dB are observed for SOVA based binary and double binary turbo decodes in the water fall region, when AWGN channels are considered. However, a bad convergence in the error-floor region exists.

MAP based algorithms are classified depending on the kind of recursive operations that they perform.Type-I MAP algorithms, such as the BCJR algorithm, implement recursive operation in the forward and backward direction. On the other hand, Type-II algorithms execute recursive operations in only one direction. The Forward Only MAP (FOMAP), which implements a forward-only recursion, falls in this last category. This algorithm is suitable for a high pipelined structure. However, since its memory requirements grow exponentially with the decision delay, a high hardware complexity is required for its implementation. Moreover, its forward-only recursion characteristic does not provide significative advantages over the BCJR algorithm, which can execute the backward and forward recursions in parallel. We have then kept the BCJR algorithm as the SISO decoding algorithm. Consequently, the Max-Log-MAP algorithm, suitable to be implemented, is applied.

The turbo decoder parallelism have been explored following a multi-level classification. Different parallel turbo decoding techniques have been studied. The convergence of a turbo decoder was identified as a major characteristic to improve the throughput. Therefore, a study of the convergence of parallel turbo decoder architectures was performed with the aid of the EXtrinsic Information Transfer (EXIT) chart diagrams. From this study, a novel SISO decoder schedule for the BCJR algorithm was proposed. It enables an increase of the convergence of shuffled turbo decoders.

We have demonstrated that the radix technique provides an effective way to overcome the bottleneck in the SISO decoder architecture. This bottleneck is due to the feedback loop required to implement the forward and backward recursive operations. The implementation of the radix technique enables to achieve high throughputs with a relative low clock frequency. It also helps to decrease the number of pipeline stages in the different turbo decoder blocks. Besides, the required state metric memory can be significantly reduced: to a half and to a quarter for radix-4 and radix-16 architectures, respectively. However, radix architectures have shown to be inefficient in terms of the energy per decoded bit and achieved throughput per area unit. In order to overcome this problem, we have proposed a low complexity radix-16 SISO decoder architecture. The design of this architecture was possible thanks to the elimination of parallel paths in a radix-16 trellis diagram transition. The proposed SISO decoder implements a high speed radix-8 Add Compare Select (ACS) unit which exhibits a lower hardware complexity

compared with a radix-16 ACS unit architecture. Also, improvements in the critical path are possible. The elimination of parallel paths has been further extended to reduce the hardware complexity of the Branch Metric Unit (BMU) and Soft Output Unit (SOU). It corresponds to an original contribution of our work. Since the proposed radix-16 SISO decoder degrades the turbo decoder error correcting performance, we have proposed two techniques so that the architecture can be used in practical applications: 1) an heuristic parameter to compute the extrinsic information for a given bit when there is not enough information due to the elimination of paths, 2) a shift technique to reduce the correlation between extrinsic values computed during the same radix-16 trellis diagram transition.

When considering a whole turbo decoder architecture, the main issue that prevents the achievements of high throughput rates corresponds to the memory conflicts. They appear when multiple SISO decoders have simultaneous access to the extrinsic memory in the system. To overcome this problem, solutions during the execution, the compilation and the design have been proposed in the literature. Our contributions correspond to solutions during the compilation and the design. In the former kind of solutions, we have proposed a dedicated approach to explore the turbo decoder design space. This approach enables conflict free memory access for any turbo decoder parallelism and interleaver function. The main objective of the dedicated approach is to reduce the time to market constraint by designing turbo decoder architectures for a given throughput. Regarding the solution at the design, we have proposed an extrinsic memory architecture for conflict free interleavers. To this end, we have introduced the definition of a rotatable interleaver. The properties of these interleavers have been exploited in order to simplify the extrinsic memory addressing logic.

Shuffled turbo decoding parallelism has been studied as well. This parallelism technique has been proposed in order to increase the turbo decoder convergence. Thus, an increase of the throughput is possible for the same Bit Error Rate (BER)/Frame Error Rate (FER) performance. Prior to this thesis, the benefits of shuffled turbo decoders were observed as long as a high number of sub-blocks was considered. This statement is based on an analysis performed at the algorithm level, leaving out important turbo decoder implementation issues. In a shuffled turbo decoder architecture, additional extrinsic memory problems appear: concurrent access to the same memory location and consistency problems. Besides, to ensure conflict free memory access, more complex extrinsic memory structures are required. We have analyzed the exchange of extrinsic information in a shuffled turbo decoder architecture. Thus, we are able to avoid a complete duplication of the extrinsic memory and reduce the number of writing and reading access. However, shuffled turbo decoder architectures are in general more complex, without demonstrating a significant advantage in terms of a higher throughput with respect to non-shuffled ones.

In a high throughput turbo decoding context, the use of the sub-block technique with a high parallelism degree is required. In this case, the best schedule to be implemented by the SISO decoders is the Butterfly schedule without sliding windows. The use of the radix technique is then required in order to improve the (Complexity-Timing) (CT) product of the architecture. Using this schedule, a high parallel turbo decoder architecture for the Long Term Evolution (LTE) standard has been designed. The proposed low complexity high speed radix-16 SISO decoder architecture has been adopted to perform the Max-Log-MAP algorithm computations. The QPP interleaver properties have been exploited in order to avoid memory conflicts. We have taken advantages of the vectorizable and rotatable properties of the interleaver in order to reduce the memory addressing architecture complexity. A set of barrel shifters are also used in order to implement the shift technique with minimum hardware complexity overhead.

A first FPGA prototype using the development board *DN9000K10PCI*, which integrate Xilinx Virtex 5 devices, has been carried out. This prototype demonstrates the suitability of the proposed radix-16 SISO decoder as part of a turbo decoder architecture. Besides, a high turbo decoder architecture, which integrates 32 SISO decoders, has been designed and synthesized to demonstrate the validity of our ideas.

PERSPECTIVES

SHORT TERM PERSPECTIVES

The main bottlenecks in a parallel turbo decoder architecture are the memory access conflicts and the recursive computations performed by the ACS units inside each SISO decoder. To tackle the ACS bottleneck, we have successfully used the elimination of parallel paths in a radix-16 trellis diagram transition to increase the SISO decoder throughput. It is also possible to further extend this idea to radix-2^{N_T} SISO decoder architectures, for $N_T > 4$. Thus, a fast radix-8 ACS unit is used, while performing N_T trellis diagram transitions per clock cycle. For high N_T values, very high throughput improvements would be possible. Since an important increase in the hardware complexity is expected, the elimination of parallel paths should be also exploited in order to reduce the complexity of the BMU and SOU, as it was done for the proposed radix-16 SISO decoder. The techniques introduced to avoid error correcting performance degradation should be study in this high radix context as well. We believe that this approximation would be very useful in order to design even higher throughput turbo decoder architectures for future digital communication systems.

Future turbo decoder architectures should implement CF interleavers. They provide excellent error correction performance, relieving the memory access bottleneck. Since

CF properties are usually ensured for low radix values, extrinsic memory structure and interconnection networks should be proposed for $N_T > 4$.

Long Term Perspectives

In [72], complementing the ideas in [71], highly pipelined SISO decoder architectures are considered. In this case, a feedback loop in the ACS unit is not implemented. Instead, a set of ACS units serially connected through pipeline registers are designed. At the beginning of our research activities, we have considered this architectural solution because of the high throughput values that it provides. However, due to the excessive hardware complexity overhead, we did not continue exploring this ideas any further.

In a very recent work [147], a deeply pipelined digital-serial Low Density Parity Check (LDPC) decoder is proposed. Very high throughput values are reported. The digit-online arithmetic [148] is exploited to reduce the hardware complexity of the computations units, and also to enable the use of a high number of pipeline stages.

We propose then, as a long term perspective, to explore the convenience of digital-serial highly pipelined SISO decoder architectures, that use the digital-online arithmetic in order to keep the hardware complexity in an acceptable value. Thus, a high number of pipeline stages can be applied, what may enable to increase the throughput, for example, by processing several information frames simultaneously.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, vol. 27, pp. 623–656, 1948.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in *IEEE ICC '93, Geneva*, pp. 1064 – 1070, 1993.

[3] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of product codes," in *Global Telecommunications Conference, 1994. GLOBECOM '94. Communications: The Global Bridge., IEEE*, pp. 339 –343 vol.1, nov- 2 dec 1994.

[4] D. MacKay, "Low Density Parity Check Codes," *Transactions of the IRE Professional Group on Information Theory*, vol. IT-8, pp. 21–28, jan 1962.

[5] D. MacKay, "Good error-correcting codes based on very sparse matrices," *Information Theory, IEEE Transactions on*, vol. 45, pp. 399 –431, mar 1999.

[6] B. Bougard, *Cross-layer energy management in broadband wireless transceivers*. PhD thesis, Katholieke Universiteit Leuven, 2006.

[7] J. Hagenauer, "Soft-in soft-out the benefits of using soft values in all stages of digital receivers," in *Proceedings of the 3rd Int Workshop on Digital Signal Process. Techniques Applied to Space Comm., ESTEC, Noordwijk*, Institute for Communications Technology. German Aerospace Reseach Establishment, Sept 1992.

[8] G. Battail, "Building long codes by combination of simple ones, thanks to weighted-output decoding," in *Proc. URSI ISSSE (Erlangen, Germany)*, pp. 634–637, Sept 1989.

[9] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on information theory*, vol. 42, no. 2, 1996.

[10] P. Elias, "Coding for noisy channels," *IRE Convention Record*, vol. 4, pp. 37–47, 1955.

[11] C. Berrou, K. A. Cavalec, M. Arzel, A. G. I. Amat, M. Jezequel, C. Langlais, R. L. Bidan, Y. Saouter, E. Maury, G. Battail, E. Boutillon, A. Glavieux, Y. O. C. Mouhamedou, S. Saoudi, C. Laot, S. Kerouedan, F. Guilloud, and C. Douillard, *Codes and Turbo Codes*. IRIS international series, Paris: Springer-Verlag, 2010. Ouvrage collectif sous la direction de Claude Berrou.

[12] J. Forney, G., "Convolutional codes I: Algebraic structure," *Information Theory, IEEE Transactions on*, vol. 16, pp. 720 – 738, nov 1970.

[13] J. Forney, G.D., "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, pp. 268 – 278, march 1973.

[14] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Communications, 1995. ICC '95 Seattle, 'Gateway to Globalization', 1995 IEEE International Conference on*, vol. 1, pp. 54 –59 vol.1, jun 1995.

[15] H. Ma and J. Wolf, "On tail biting convolutional codes," *Communications, IEEE Transactions on*, vol. 34, pp. 104 – 111, feb 1986.

[16] C. Berrou, C. Douillard, and M. Jézéquel, "Multiple parallel concatenation of circular recursive convolutional (CRSC) code," *Ann. Telecomm*, pp. 166–172, March-April 1999.

[17] X. Ma and A. Kavcic, "Path partitions and forward-only trellis algorithms," *Information Theory, IEEE Transactions on*, vol. 49, pp. 38 – 52, jan 2003.

[18] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260 – 269, April 1967.

[19] Y. Li, B. Vucetic, and Y. Sato, "Optimum soft-output detection for channels with intersymbol interference," *IEEE Transactions on Information Theory*, vol. 41, pp. 704–713, 1995.

[20] R. Ratnayake, A. Kavcic, and G.-Y. Wei, "A high-throughput maximum a posteriori probability detector," *IEEE Journal of solid-state circuits*, vol. 43, pp. 1846–1858, 2008.

[21] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, 1974.

[22] L. Lee, "Real-time minimal-bit-error probability decoding of convolutional codes," *Communications, IEEE Transactions on*, vol. 22, pp. 146 – 151, feb 1974.

[23] J. Hayes, T. Cover, and J. Riera, "Optimal sequence detection and optimal symbol-by-symbol detection: Similar algorithms," *Communications, IEEE Transactions on*, vol. 30, pp. 152 – 157, jan 1982.

[24] B. Bai, X. Ma, and X. Wang, "Novel algorithm for continuous decoding of turbo codes," *Communications, IEE Proceedings-*, vol. 146, pp. 271 –274, oct 1999.

[25] I. Onyszchuk, "Truncation length for Viterbi decoding," *Communications, IEEE Transactions on*, vol. 39, pp. 1023 –1026, jul 1991.

[26] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Global Telecommunications Conference, 1989, and Exhibition. Communications Technology for the 1990s and Beyond. GLOBECOM '89., IEEE*, pp. 1680 –1686 vol.3, nov 1989.

[27] G. Battail, "Pondération des symboles décodés par l'algorithme de viterbi," *Ann. Télécommun.*, pp. 31–38, Jan 1987.

[28] C. Berrou, P. Adde, E. Angui, and S. Faudeil, "A low complexity soft-output Viterbi decoder architecture," in *Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, vol. 2, pp. 737 –740 vol.2, may 1993.

[29] D. Bera and J. Sen, "SOVA based decoding of double-binary turbo convolutional code," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, pp. 757 –761, may 2009.

[30] V. Branka and Y. Jinhong, *Turbo codes : principles and applications / Branka Vucetic, Jinhong Yuan*. Kluwer Academic, Boston ; London :, 2000.

[31] J. Tan and G. Stuber, "Soft output Viterbi algorithm (SOVA) for non-binary turbo codes," in *Information Theory, 2000. Proceedings. IEEE International Symposium on*, p. 483, 2000.

[32] J. Tan and G. Stuber, "A MAP equivalent SOVA for non-binary turbo codes," in *Communications, 2000. ICC 2000. 2000 IEEE International Conference on*, vol. 2, pp. 602 –606 vol.2, 2000.

[33] J. Liu and G. Tu, "Iterative decoding of non-binary turbo codes using symbol based SOVA algorithm," in *Communications, Circuits and Systems Proceedings, 2006 International Conference on*, vol. 2, pp. 689 –693, june 2006.

[34] L. Gong, W. Xiaofu, and Y. Xiaoxin, "On SOVA for nonbinary codes," *Communications Letters, IEEE*, vol. 3, pp. 335 –337, dec. 1999.

[35] E. Boutillon, W. Gross, and P. Gulak, "VLSI architectures for the map algorithm," *Communications, IEEE Transactions on*, vol. 51, pp. 175 – 185, feb 2003.

[36] W. Gross and P. Gulak, "Simplified MAP algorithm suitable for implementation of turbo decoders," *Electronics Letters*, vol. 34, pp. 1577 –1578, aug 1998.

[37] Z. Wang, "High-speed recursion architectures for MAP-based turbo decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, pp. 470 –474, april 2007.

[38] C. Douillard and C. Berrou, "Turbo codes with rate-m/(m+1) constituent convolutional codes," *Communications, IEEE Transactions on*, vol. 53, pp. 1630 – 1638, oct. 2005.

[39] A. Alvarado, V. Nunez, L. Szczecinski, and E. Agrell, "Correcting suboptimal metrics in iterative decoders," in *Communications, 2009. ICC '09. IEEE International Conference on*, pp. 1 –6, june 2009.

[40] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *Electronics Letters*, vol. 36, pp. 1937 –1939, nov 2000.

[41] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of ldpc codes and extension to turbo decoding," in *Information Theory, 2001. Proceedings. 2001 IEEE International Symposium on*, p. 189, 2001.

[42] S. Benedetto and G. Montorsi, "Serial concatenation of block and convolutional codes," *Electronics Letters*, vol. 32, pp. 887 –888, may 1996.

[43] S. Benedetto and G. Montorsi, "Iterative decoding of serially concatenated convolutional codes," *Electronics Letters*, vol. 32, pp. 1186 –1188, jun 1996.

[44] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," pp. 56–65, 1995.

[45] J. Hokfelt, O. Edfors, and T. Maseng, "A turbo code interleaver design criterion based on the performance of iterative decoding," *Communications Letters, IEEE*, vol. 5, pp. 52 –54, feb 2001.

[46] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel, "Designing good permutations for turbo codes: towards a single model," in *Communications, 2004 IEEE International Conference on*, vol. 1, pp. 341 – 345, june 2004.

[47] S. Crozier and P. Guinand, "Distance upper bounds and true minimum distance results for Turbo-Codes designed with DRP interleavers," *Annales des Télécommunications*, vol. 60, no. 1-2, pp. 10–28, 2005.

[48] J. Sun and O. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *Information Theory, IEEE Transactions on*, vol. 51, pp. 101 –119, jan. 2005.

[49] M. Jezequel, C. Berrou, C. Douillard, and P. Penard, "Characteristics of a sixteen-state turbo-encoder/decoder (turbo4)," in *International Symposium on Turbo Codes & Related Topics, 3-5 September 1997 - Brest, France*, pp. 280 – 283, 1997.

[50] D. Gnaedig, *High Speed decoding of convolutional turbo Codes*. PhD thesis, L'université de Bretagne du Sud, 2005.

[51] S. Haddad, A. Baghdadi, and M. Jezequel, "On the convergence speed of turbo demodulation with turbo decoding," *Signal Processing, IEEE Transactions on*, vol. 60, pp. 4452 –4458, aug. 2012.

[52] C. van Berkel, "Multi-core for mobile phones," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pp. 1260 –1265, april 2009.

[53] S. Galli, "On the fair comparison of FEC schemes," in *Communications (ICC), 2010 IEEE International Conference on*, pp. 1 –6, may 2010.

[54] F. Kienle, N. Wehn, and H. Meyr, "On complexity, energy- and implementation-efficiency of channel decoders," *Communications, IEEE Transactions on*, vol. 59, pp. 3301 –3310, december 2011.

[55] O. Muller, A. Baghdadi, and M. Jezequel, "On the parallelism of convolutional turbo decoding and interleaving interference," in *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pp. 1 –5, 27 2006-dec. 1 2006.

[56] C.-C. Wong, M.-W. Lai, C.-C. Lin, H.-C. Chang, and C.-Y. Lee, "Turbo decoder using contention-free interleaver and parallel architecture," *Solid-State Circuits, IEEE Journal of*, vol. 45, pp. 422 –432, feb. 2010.

[57] J.-M. Hsu and C.-L. Wang, "A parallel decoding scheme for turbo codes," in *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, vol. 4, pp. 445 –448 vol.4, may-3 jun 1998.

[58] Z. Wang, Z. Chi, and K. Parhi, "Area-efficient high speed decoding schemes for turbo/MAP decoders," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, vol. 4, pp. 2633 –2636 vol.4, 2001.

[59] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *Communications, IEEE Transactions on*, vol. 53, pp. 209 – 213, feb. 2005.

[60] O. Muller, A. Baghdadi, and M. Jezequel, "Exploring parallel processing levels for convolutional turbo decoding," in *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, vol. 2, pp. 2353 –2358, 0-0 2006.

[61] Z. Wang and K. Parhi, "High performance, high throughput turbo/SOVA decoder design," *Communications, IEEE Transactions on*, vol. 51, pp. 570 – 579, april 2003.

[62] O. Muller, *Architectures multiprocesseurs monopuces génériques pour turbo-communications haut-débit*. PhD thesis, Université de Bretagne-Sud, 2007.

[63] T. Wolf, "Initialization of sliding windows in turbo decoders," in *International Symposium on Turbo Codes & Related Topics*, pp. 219–222, sep 2003.

[64] A. Dingninou, F. Raouafi, and C. Berrou, "Organisation de la mémoire dans un turbo decodeur utilisant l'algorithme SUB-MAP," in *Dix-septième colloque GRETSI*, pp. 71–74, sep 1999.

[65] J. Dielissen and J. Huisken, "State vector reduction for initialization of sliding windows MAP," in *International Symposium on Turbo Codes & Related Topics*, pp. 387–390, sep 2000.

[66] J. Zhang, Y. Wang, M. Fossorier, and J. Yedidia, "Replica shuffled iterative decoding," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, pp. 454 –458, sept. 2005.

[67] J. Zhang, Y. Wang, M. Fossorier, and J. Yedidia, "Iterative decoding with replicas," *Information Theory, IEEE Transactions on*, vol. 53, pp. 1644 –1663, may 2007.

[68] O. Muller, A. Baghdadi, and M. Jezequel, "Parallelism efficiency in convolutional turbo decoding," *EURASIP journal on advances in signal processing*, november 2010.

[69] H. Dawid, G. Gehnen, and H. Meyr, "Map channel decoding: Algorithm and vlsi architecture," in *VLSI Signal Processing, VI, 1993., [Workshop on]*, pp. 141 –149, oct 1993.

[70] H. Dawid and H. Meyr, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding," in *Personal, Indoor and Mobile Radio Communications, 1995. PIMRC'95. 'Wireless: Merging onto the Information Superhighway'., Sixth IEEE International Symposium on*, vol. 1, pp. 193 –197 vol.1, sep 1995.

[71] A. Worm, H. Lamm, and N. Wehn, "A high-speed MAP architecture with optimized memory size and power consumption," in *Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on*, pp. 265 –274, 2000.

[72] M. Mansour and N. Shanbhag, "VLSI architectures for SISO-APP decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, pp. 627 –650, aug. 2003.

[73] S.-J. Lee, N. Shanbhag, and A. Singer, "A 285-MHz pipelined MAP decoder in 0.18-μm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 40, pp. 1718 – 1725, aug. 2005.

[74] 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, pp. 921 –933, aug. 2005.

[75] G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: breaking the ACS-bottleneck," *Communications, IEEE Transactions on*, vol. 37, pp. 785 –790, aug 1989.

[76] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC. 2003 IEEE International*, pp. 150 – 484 vol.1, 2003.

[77] Y. Zhang and K. Parhi, "High-throughput radix-4 logMAP turbo decoder architecture," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, pp. 1711 –1715, 29 2006-nov. 1 2006.

[78] 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, pp. 8 –17, jan. 2011.

[79] F. Jin, J. Tang, Z. F. Wang, and L. Guo, "A radix-8 Log-MAP recursion VLSI architecture," in *Communication Technology, 2008. ICCT 2008. 11th IEEE International Conference on*, pp. 347 –350, nov. 2008.

[80] C.-H. Tang, C.-C. Wong, C.-L. Chen, C.-C. Lin, and H.-C. Chang, "A 952MS/s Max-Log MAP decoder chip using radix-4 x 4 ACS architecture," in *Solid-State Circuits Conference, 2006. ASSCC 2006. IEEE Asian*, pp. 79 –82, nov. 2006.

[81] K.-T. Shr, Y.-C. Chang, C.-Y. Lin, and Y.-H. Huang, "A 6.6pj/bit/iter radix-16 modified log-MAP decoder using two-stage ACS architecture," in *Solid State Circuits Conference (A-SSCC), 2011 IEEE Asian*, pp. 313 –316, nov. 2011.

[82] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "A Soft-Input Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes," *Telecommunications and Data Acquisition Progress Report*, vol. 127, pp. 1–20, July 1996.

[83] C. Schurgers, F. Catthoor, and M. Engels, "Memory optimization of MAP turbo decoder algorithms," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, pp. 305 –312, april 2001.

[84] A. Viterbi, "An intuitive justification and a simplified implementation of the map decoder for convolutional codes," *Selected Areas in Communications, IEEE Journal on*, vol. 16, pp. 260 –264, feb 1998.

[85] R. Dobkin, M. Peleg, and R. Ginosar, "Parallel interleaver design and VLSI architecture for low-latency MAP turbo decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, pp. 427 –438, april 2005.

[86] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, "Architectural strategies for low-power VLSI turbo decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 10, pp. 279 –285, june 2002.

[87] C.-C. Wong and H.-C. Chang, "High-efficiency processing schedule for parallel turbo decoders using QPP interleaver," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, pp. 1412 –1420, june 2011.

[88] D. Gnaedig, E. Boutillon, J. Tousch, and M. Jezequel, "Towards an optimal parallel decoding of turbo codes," in *4th International Symposium on turbo codes and related topics, April 3-7, Munich, Germany*, 2006.

[89] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *Communications, IEEE Transactions on*, vol. 49, pp. 1727 –1737, oct 2001.

[90] O. Sánchez, C. Jégo, and M. Jézéquel, "Analysis of the convergence process by EXIT charts for parallel implementations of turbo decoders," *Accepted for publication. To appear on IEEE Communications Letters*.

[91] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of provably good low-density parity check codes," in *Information Theory, 2000. Proceedings. IEEE International Symposium on*, p. 199, 2000.

[92] C. Nour and C. Douillard, "Cth11-4: On lowering the error floor of high order turbo bicm schemes over fading channels," in *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pp. 1 –5, 27 2006-dec. 1 2006.

[93] J. W. Lee and R. E. Blahut, "Convergence analysis and ber performance of finite-length turbo codes," *Communications, IEEE Transactions on*, vol. 55, pp. 1033 –1043, may 2007.

[94] J. Lee and R. Blahut, "Lower bound on BER of finite-length turbo codes based on EXIT characteristics," *Communications Letters, IEEE*, vol. 8, pp. 238 – 240, april 2004.

[95] S. Haddad, O. Sánchez, A. Baghdadi, and M. Jezequel, "Complexity reduction of shuffled parallel iterative demodulation with turbo decoding," in *Telecommunications (ICT), 2012 19th International Conference on*, pp. 1 –6, april 2012.

[96] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Communications, 1995. ICC '95 Seattle, 'Gateway to Globalization', 1995 IEEE International Conference on*, vol. 2, pp. 1009 –1013 vol.2, jun 1995.

[97] Y. Saouter and C. Berrou, "Fast soft-output Viterbi decoding for duo-binary turbo codes," in *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, vol. 1, pp. I–885 – I–888 vol.1, 2002.

[98] O. Joeressen, M. Vaupel, and H. Meyr, "High-speed VLSI architectures for soft-output viterbi decoding," in *Application Specific Array Processors, 1992. Proceedings of the International Conference on*, pp. 373 –384, aug 1992.

[99] D. Garrett and M. Stan, "Low power architecture of the soft-output Viterbi algorithm," in *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*, pp. 262 –267, aug. 1998.

[100] C. M. Rader, "Memory management in a viterbi decoder," *IEEE Transactions on communications*, vol. COM-29, pp. 1399–1401, September 1981.

[101] O. Collins and F. Pollara, "Memory management in traceback Viterbi decoders," *TDA Progress Report 42-99*, July-September 1989.

[102] G. Feygin and P. Gulak, "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," *Communications, IEEE Transactions on*, vol. 41, pp. 425 –429, mar 1993.

[103] R. Cypher and C. Shung, "Generalized trace back techniques for survivor memory management in the Viterbi algorithm," in *Global Telecommunications Conference, 1990, and Exhibition. 'Communications: Connecting the Future', GLOBECOM '90., IEEE*, pp. 1318 –1322 vol.2, dec 1990.

[104] E. Angui, *Conception d'un circuit intégré VLSI turbo-décodeur*. PhD thesis, L'Université de Bretagne Occidentale, 1994.

[105] C. X. Huang and A. Ghrayeb, "An improved SOVA algorithm for turbo codes over AWGN and fading channel," in *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*, vol. 2, pp. 1121 – 1125 Vol.2, sept. 2004.

[106] L. Papke, P. Robertson, and E. Villebrun, "Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme," in *Communications, 1996. ICC 96,*

Conference Record, Converging Technologies for Tomorrow's Applications. 1996 IEEE International Conference on, vol. 1, pp. 102 –106 vol.1, jun 1996.

[107] L. Lin and R. Cheng, "Improvements in SOVA-based decoding for turbo codes," in *Communications, 1997. ICC 97 Montreal, 'Towards the Knowledge Millennium'. 1997 IEEE International Conference on*, vol. 3, pp. 1473 –1478 vol.3, jun 1997.

[108] G. Montorsi and S. Benedetto, "Design of fixed-point iterative decoders for concatenated codes with interleavers," *Selected Areas in Communications, IEEE Journal on*, vol. 19, pp. 871 –882, may 2001.

[109] Y. Wu and B. Woerner, "The influence of quantization and fixed point arithmetic upon the BER performance of turbo codes," in *Vehicular Technology Conference, 1999 IEEE 49th*, vol. 2, pp. 1683 –1687 vol.2, jul 1999.

[110] 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, pp. 369 –379, sept. 1999.

[111] J.-M. Hsu and C.-L. Wang, "On finite-precision implementation of a decoder for turbo codes," in *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*, vol. 4, pp. 423 –426 vol.4, jul 1999.

[112] C. Shung, P. Siegel, G. Ungerboeck, and H. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *Communications, 1990. ICC '90, Including Supercomm Technical Sessions. SUPERCOMM/ICC '90. Conference Record., IEEE International Conference on*, pp. 1723 –1728 vol.4, apr 1990.

[113] A. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *Communications, IEEE Transactions on*, vol. 37, pp. 1220 –1222, nov 1989.

[114] Y. Wu, B. Woerner, and T. Blankenship, "Data width requirements in SISO decoding with module normalization," *Communications, IEEE Transactions on*, vol. 49, pp. 1861 –1868, nov 2001.

[115] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, "Design and optimization of an HSDPA turbo decoder ASIC," *Solid-State Circuits, IEEE Journal of*, vol. 44, pp. 98 –106, jan. 2009.

[116] C. Studer, S. Fateh, C. Benkeser, and Q. Huang, "Implementation Trade-Offs of Soft-Input Soft-Output MAP Decoders for Convolutional Codes," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, pp. 2774 –2783, nov. 2012.

[117] O. Sánchez, C. Jégo, M. Jézéquel, and Y. Saouter, "High Speed Low Complexity Radix-16 Max-Log-MAP SISO Decoder," in *Electronics, Circuits, and Systems, 2012. ICECS 2012. 16th IEEE International Conference on*, dec. 2012.

[118] P. Adde and R. Pyndiah, "Recent simplifications and improvements in Block Turbo Codes," in *2nd International Symposium on Turbo Codes & Related Topics, September 4-7, Brest, France*, pp. 133 – 136, 2000.

[119] E. Boutillon, C. Douillard, and G. Montorsi, "Iterative decoding of concatenated convolutional codes: Implementation issues," *Proceedings of the IEEE*, vol. 95, pp. 1201 –1227, june 2007.

[120] M. Thul, F. Gilbert, and N. Wehn, "Optimized concurrent interleaving architecture for high-throughput turbo-decoding," in *Electronics, Circuits and Systems, 2002. 9th International Conference on*, vol. 3, pp. 1099 – 1102 vol.3, 2002.

[121] M. Thul, N. Wehn, and L. Rao, "Enabling high-speed turbo-decoding through concurrent interleaving," in *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, vol. 1, pp. I–897 – I–900 vol.1, 2002.

[122] M. Thul, F. Gilbert, and N. Wehn, "Concurrent interleaving architectures for high-throughput channel coding," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, vol. 2, pp. II – 613–16 vol.2, april 2003.

[123] C. Neeb, M. Thul, and N. Wehn, "Network-on-chip-centric approach to interleaving in high throughput channel decoders," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 1766 – 1769 Vol. 2, may 2005.

[124] G. Wang, Y. Sun, J. Cavallaro, and Y. Guo, "High-throughput Contention-Free concurrent interleaver architecture for multi-standard turbo decoder," in *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*, pp. 113 –121, sept. 2011.

[125] A. Tarable and S. Benedetto, "Mapping interleaving laws to parallel turbo decoder architectures," *Communications Letters, IEEE*, vol. 8, pp. 162 – 164, march 2004.

[126] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures," *Information Theory, IEEE Transactions on*, vol. 50, pp. 2002 – 2009, sept. 2004.

[127] C. Chavet, P. Coussy, P. Urard, and E. Martin, "Static Address Generation Easing: a design methodology for parallel interleaver architectures," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 1594 –1597, march 2010.

[128] A. Sani, P. Coussy, C. Chavet, and E. Martin, "An approach based on edge coloring of tripartite graph for designing parallel LDPC interleaver architecture," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pp. 1720 –1723, may 2011.

[129] A. Sani, P. Coussy, C. Chavet, and E. Martin, "A methodology based on transportation problem modeling for designing parallel interleaver architectures," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 1613 –1616, may 2011.

[130] A. Giulietti, L. van der Perre, and A. Strum, "Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements," *Electronics Letters*, vol. 38, pp. 232 –234, feb 2002.

[131] D. Gnadieg, E. Boutillon, V. Gaudet, P. G. Gulak, and M. Jezequel, "On multiple slice turbo codes," *3rd International Symposium On Turbo Codes and Related Topics*, pp. 343 – 346, sept 2003.

[132] Y.-X. Zheng and Y. Su, "A new interleaver design and its application to turbo codes," in *Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th*, vol. 3, pp. 1437 – 1441 vol.3, 2002.

[133] A. Nimbalker, T. Fuja, J. Costello, D.J., T. Blankenship, and B. Classon, "Contention-free interleavers," in *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, p. 54, june-2 july 2004.

[134] O. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *Information Theory, IEEE Transactions on*, vol. 52, pp. 1249 –1253, march 2006.

[135] A. Nimbalker, T. Blankenship, B. Classon, T. Fuja, and D. Costello, "Contention-Free Interleavers for High-Throughput Turbo Decoding," *Communications, IEEE Transactions on*, vol. 56, pp. 1258 –1267, august 2008.

[136] A. Nimbalker, Y. Blankenship, B. Classon, and T. Blankenship, "ARP and QPP Interleavers for LTE Turbo Coding," in *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pp. 1032 –1037, 31 2008-april 3 2008.

[137] J. Ryu and O. Takeshita, "On quadratic inverses for quadratic permutation polynomials over integer rings," *Information Theory, IEEE Transactions on*, vol. 52, pp. 1254 –1260, march 2006.

[138] Y. Sun and J. R. Cavallaro, "Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder," *Integration*, vol. 44, no. 4, pp. 305–315, 2011.

[139] C.-C. Wong, Y.-Y. Lee, and H.-C. Chang, "A 188-size 2.1mm^2 reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system," in *VLSI Circuits, 2009 Symposium on*, pp. 288 –289, june 2009.

[140] C.-C. Wong, C.-H. Tang, M.-W. Lai, Y.-X. Zheng, C.-C. Lin, H.-C. Chang, C.-Y. Lee, and Y.-T. Su, "A 0.22 nj/b/iter 0.13 μm turbo decoder chip using inter-block permutation interleaver," in *Custom Integrated Circuits Conference, 2007. CICC '07. IEEE*, pp. 273 –276, sept. 2007.

[141] ETSI, "Digital video broadcasting (DVB): interaction channel for satellite distribution systems." Standard, EN 301 790 (V1.3.1), 2003.

[142] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15GBit/s turbo code decoder for LTE advanced base station applications," in *Turbo Codes and Iterative Information Processing (ISTC), 2012 7th International Symposium on*, pp. 21 –25, aug. 2012.

[143] C. S. Wallace, "Fast pseudorandom generators for normal and exponential variates.," *ACM Trans. Math. Softw.*, vol. 22, no. 1, pp. 119–127, 1996.

[144] O. Sánchez, M. Arzel, C. Jégo, A. García, and M. Guerrero, "Design and implementation of a MIMO channel emulator onto FPGA device," in *XV Iberchip Workshop, IWS'09, Argentina*, March. 2009.

[145] O. Sánchez, S. ur Rehman, A. Sani, C. Chavet, P. Coussy, C. Jego, and M. Jezequel, "A dedicated approach to explore design space for hardware architecture of turbo decoders," in *Signal Processing Systems, 2012. SiPS 2012. IEEE Workshop on*, oct. 2012.

[146] V. E. Benes, *Mathematical Theory of connecting network and telephone traffic*. Academic Press, 1965.

[147] P. A. Marshall, V. C. Gaudet, and D. G. Elliott, "Deeply pipelined digit-serial LDPC decoding," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, pp. 2934 –2944, dec. 2012.

[148] M. Ercegovac, "Online arithmetic: An overview," *Proc. SPIE V.495: Real Time Signal Processing VII*, pp. 86–93, 1984.

List of Publications

JOURNALS

- O. Sánchez, C. Jégo, and M. Jézéquel, "Analysis of the convergence process by EXIT charts for parallel implementations of turbo decoders," *Accepted for publication. To appear on IEEE Communications Letters*.

CONFERENCES

- O. Sánchez, M. Arzel, C. Jégo, A. García, and M. Guerrero, "Design and implementation of a MIMO channel emulator onto FPGA device," in *XV Iberchip Workshop, IWS'09, Argentina*, March. 2009.

- S. Haddad, O. Sánchez, A. Baghdadi, and M. Jézéquel, "Complexity reduction of shuffled parallel iterative demodulation with turbo decoding," in *Telecommunications (ICT), 2012 19th International Conference on*, pp. 1 –6, april 2012.

- O. Sánchez, S. ur Rehman, A. Sani, C. Chavet, P. Coussy, C. Jégo, and M. Jézéquel, "A dedicated approach to explore design space for hardware architecture of turbo decoders," in *Signal Processing Systems, 2012. SiPS 2012. IEEE Workshop on*, oct. 2012.

- O. Sánchez, C. Jégo, M. Jézéquel, and Y. Saouter, "High Speed Low Complexity Radix-16 Max-Log-MAP SISO Decoder," in *Electronics, Circuits, and Systems, 2012. ICECS 2012. 16th IEEE International Conference on*, dec. 2012.

- O. Sánchez, C. Jégo, and M. Jézéquel, "Décodeur radix-16 à entrées et sorties pondérées pour un turbo-décodage à haut débit," *XXIVème Colloque Gretsi, Brest*, sep 2013.