DISSERTATION

In Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
from TELECOM ParisTech

Specialization: Communication and Electronics

Erick Amador

Aspects of Energy Efficient LDPC Decoders

Defended on the 31st of March 2011 before the committee composed by:

Reviewers	Prof. Emmanuel Boutillon, Université de Bretagne-Sud
	Prof. Michel Jezequel, Télécom Bretagne
Examiners	Prof. Cécile Belleudy, Université de Nice-Sophia Antipolis
	Dr. Bertram Gunzelmann, Intel Mobile Communications
	Vincent Rezard, Intel Mobile Communications
Thesis supervisors	Prof. Raymond Knopp, EURECOM
	Prof. Renaud Pacalet, Télécom ParisTech

THESE

présentée pour obtenir le grade de

Docteur de TELECOM ParisTech

Spécialité: Communication et Electronique

Erick Amador

Aspects des Décodeurs LDPC Optimisés pour la Basse Consommation

Soutenue le 31 Mars 2011 devant le jury composé de :

Rapporteurs	Prof. Emmanuel Boutillon, Université de Bretagne-Sud
	Prof. Michel Jezequel, Télécom Bretagne
Examinateurs	Prof. Cécile Belleudy, Université de Nice-Sophia Antipolis
	Dr. Bertram Gunzelmann, Intel Mobile Communications
	Vincent Rezard, Intel Mobile Communications
Directeurs de thèse	Prof. Raymond Knopp, EURECOM
	Prof. Renaud Pacalet, Télécom ParisTech

Abstract

Iterative decoding techniques for modern capacity-approaching codes are currently dominating the choices for forward error correction (FEC) in a plethora of applications. Turbo codes, proposed in 1993 [1], triggered the breakthrough in channel coding techniques as these codes approach the Shannon capacity limit. This was followed by the rediscovery of low-density parity-check (LDPC) codes in the 1990s, originally proposed by Gallager [2] in 1963. These codes are presently ubiquitous in the context of mobile wireless communications among other application domains.

In this dissertation, we focus on the aspects and challenges for conceiving energy efficient VLSI decoders aimed at mobile wireless applications. These nomadic devices are typically battery-operated and demand high energy efficiency along with high throughput performance on the smallest possible footprint. Moreover, these iterative decoders are typically one of the most power intensive components in the baseband processing chain of a wireless receiver.

We address the aspects for designing energy efficient LDPC decoders. At the algorithmic level we investigate the tradeoff among error-correction performance, energy efficiency and implementation area for different choices of message computation kernels. We identify the opportunities for energy savings that are enabled by the Self-Corrected Min-Sum (SCMS) kernel at three different levels: convergence speed, reduction on the number of active nodes and an efficient stopping criterion. At this level we also propose a technique to evaluate the syndrome of the code in *on-the-fly* fashion that offers a speedup on the decoding task.

At the architectural level we focus on the memory subsystem design of an LDPC decoder since this module is responsible for the majority of the implementation area and power consumption. We propose a methodology for data partitioning and allocation within a flexible memory subsystem that reconciles design cost, energy consumption and task latency. Furthermore, we study the impact of interleaving the memories for the posterior messages

in order to resolve conflicts. With the results from our studies at both the algorithmic and architectural levels we present the implementation of a multi-mode decoder for the quasi-cyclic LDPC codes defined in IEEE 802.11n/16e.

At the system level we propose dynamic power management strategies that rely upon iteration control and workload prediction. For iteration control we propose a control law that is aided by two decision metrics that follow the dynamics of the decoding task. Regarding workload prediction, we propose a control law that adjusts online a power manageable iterative decoder that guarantees a task deadline while minimizing energy expenditure.

Contents

List of Figures

List of Tables

Acronyms

These are the main acronyms used in this document. The meaning of an acronym is usually indicated once, when it first occurs in the text.

3GPP	Third Generation Partnership Project
ARQ	automatic repeat request
ASIC	application-specific integrated circuit
ASIP	application-specific instruction-set processor
AWGN	additive white Gaussian noise
BCJR	Bahl, Cocke, Jelinek and Raviv algorithm
BER	bit error rate
bps	bits per second
CMOS	complementary metal-oxide-semiconductor
DPM	dynamic power management
DVB	digital video broadcasting
DVFS	dynamic voltage and frequency scaling
FAR	false alarm rate
FEC	forward error correction
FER	frame error rate
HARQ	hybrid ARQ
HDA	hard-decision aided
IEEE	Institute of Electrical and Electronic Engineers
IFS	inter-frame space
IMT-Advanced	International Mobile Telecommunications - advanced
ITU-R	International Telecommunication Union - radiocommunication sector
LAN	local area network
LDPC	low-density parity-check
LLR	log-likelihood ratio
LTE	long term evolution (3GPP)
MAP	maximum a posteriori

MDR	missed detection rate
ML	maximum likelihood
MIMO	multiple-input multiple-output
MPEG	Moving Picture Experts Group
MS	min-sum
NMOS	N-type metal-oxide-semiconductor
NMS	normalized min-sum
NoC	network-on-chip
OMS	offset min-sum
PAN	personal area network
PMOS	P-type metal-oxide-semiconductor
QC	quasi-cyclic
QPP	quadratic polynomial permutation
QPSK	quadrature phase-shift keying
ROM	read-only memory
RTL	register transfer level
SCMS	self-corrected min-sum
SDA	soft-decision aided
SDR	software-defined radio
SISO	soft-input soft-output
SNR	signal-to-noise ratio
SoC	system-on-chip
SP	sum-product
SRAM	static random access memory
TDMP	turbo-decoding message-passing
UE	user equipment
UMTS	universal mobile telecommunications system (3GPP)
VLSI	very-large-scale integration
VNR	variable node reliability

Notations

Generally, boldface upper-case letters denote matrices and boldface lower-case letters denote vectors (unless stated otherwise). Calligraphic upper-case letters denote sets. The superscript ^{T} stands for transpose.

$\lvert \cdot \rvert$	absolute value
$\lceil \cdot \rceil$	ceil operation
$\lfloor \cdot \rfloor$	floor operation
$argmin$	argument function of the minimum
E_b	(information) bit energy
$\mathbb{F}_q$	finite (Galois) field with q elements
f_{clk}	clock frequency of a synchronous circuit
$\boldsymbol{G}$	generator matrix
$\boldsymbol{H}$	parity-check matrix
$\boldsymbol{I}_K$	$K \times K$ identity matrix
$L(\cdot)$	log-likelihood ratio
max	maximum of following expression
min	minimum of following expression
mod	modulo operation
N_0	one-sided noise power spectral density
$P(\cdot)$	probability of argument
$p(\cdot)$	probability density function
$sign(\cdot)$	sign of argument
t	continuous time
x_i	The ith element of vector $\mathbf{x}$, if the latter is defined

$\setminus$	set element exclusion
$\propto$	left side is proportional to the right side
$\subseteq$	subset
∞	infinity

$\chi(G)$	chromatic number of graph G
$\oplus$	sum in $\mathbb{F}_2$
$\boldsymbol{\delta}$	intrinsic message vector
$\boldsymbol{\gamma}$	posterior message vector
$\boldsymbol{\lambda}$	extrinsic message vector
$\boldsymbol{\Lambda}$	newly generated extrinsic message vector
$\boldsymbol{\rho}$	prior message vector
c_i	degree of row i in $\boldsymbol{H}$
$\boldsymbol{\kappa}$	*corrected* prior message vector
I	number of iterations
i_c	index of block-column
$\mathcal{I}_i$	set of c_i elements from row i in $\boldsymbol{H}$
m	horizontal interleaving degree
m_b	number of block-rows in a structured parity-check matrix
N	codeblock length
n_b	number of block-columns in a structured parity-check matrix
P	number of processing units in a semi-parallel architecture
R	code rate
s	vertical interleaving degree in Chapter 5, circulant shift value in Chapter 6
Z	expansion factor in a structured parity-check matrix

Chapter 1

Introduction

1.1 Motivation

Iterative decoding techniques for modern capacity-approaching codes are currently dominating the choices for forward error correction (FEC) in a plethora of applications. Turbo codes, proposed in 1993 [1], triggered the breakthrough in channel coding techniques as these codes approach the Shannon capacity limit. This was followed by the rediscovery of low-density parity-check (LDPC) codes in the 1990s, originally proposed by Gallager [2] in 1963.

Modern wireless communication standards have already adopted these types of codes for FEC and channel coding applications. For example, the Turbo codes are used in the 3GPP Universal Mobile Telecommunications System (UMTS) [3] and its Long Term Evolution (LTE) [4] system. On the other hand, LDPC codes can be found in applications ranging from wireless Local/Metropolitan Area Networks (LAN/MAN) (IEEE 802.11n [5] and 802.16e [6]) and high-speed wireless personal area networks (PAN) (IEEE 802.15.3c [7]) to Digital Video Broadcast (DVB-S2 [8]). Furthermore, these codes are currently being proposed for next generation cellular and mobile broadband systems as defined by the ITU-R to comply with the IMT-Advanced radio interface requirements: IEEE 802.16m [9] and 3GPP LTE-Advanced [10].

In Figure 1.1, we show the evolution and characteristics of various wire-

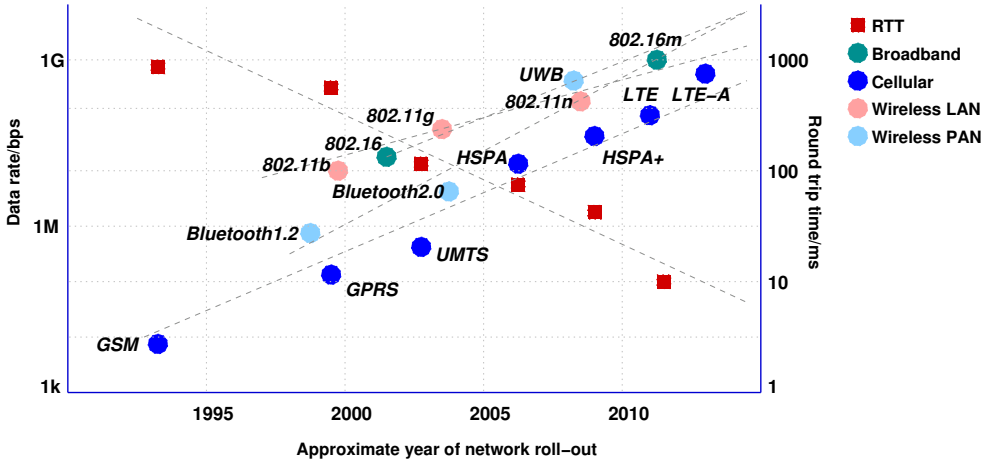


Figure 1.1: Evolution of wireless networks.

less communication standards for cellular networks, personal area networks and broadband networks. It can be observed how the data throughput and round-trip time requirements are pushed constantly to higher levels.

Nomadic devices targeted for mobile wireless communications present the challenge to use low-power computing architectures that provide high throughput and low latency performance. Moreover, these devices must provide a level of flexibility as they are set to support various communication standards in order to provide a seamless transition within different networks depending upon physical location, moving speed and required services. In a modern baseband computing device the use of these capacity-approaching codes represents one of the main sources for processing latency, power consumption and implementation area. Therefore, it is desirable to explore the available opportunities to enhance energy efficiency when targeting battery-powered mobile terminals.

Portable consumer electronics used within these technologies are battery-operated and the computation complexity performed by them is increased according to more robust signal processing techniques incorporated on each network technology. In Figure 1.2, we show the gap between battery technology enhancements and chip power demand for ASIC/SoC consumer portable products. This data is taken from [11] [12]. The maximum battery capacity shows an increase of 10-15% per year whereas the chip power requirement increases much faster at 35-40% per year. It is clear that current battery technology cannot cope with the demands for the ever in-

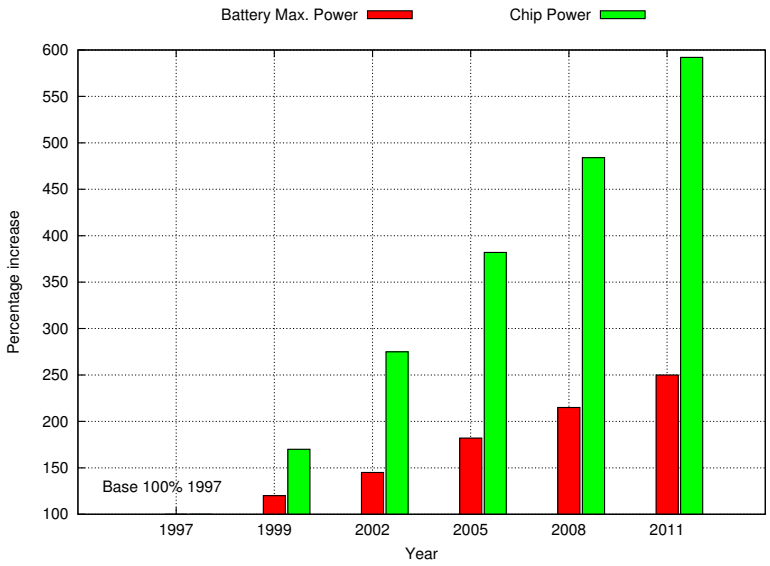


Figure 1.2: Battery power and chip power demand.

creasing complexity required for consumer electronics that execute complex signal processing tasks required by current and future network technologies. Therefore, there is a constant need to design for better energy efficiency. The design parameters and different components that have more impact on system power consumption must be identified and optimized in order to minimize their influence.

1.2 Research Contributions

In this dissertation, we focus on the design aspects for Very-Large-Scale Integration (VLSI) architectures of LDPC decoders with the goal to achieve high energy efficiency and flexibility in the sense that various codes may be supported. Our contributions are mainly distributed among three abstraction levels:

- Algorithmic level: We investigate the energy efficiency of VLSI decoders based upon the turbo-decoding message-passing (TDMP) [13] strategy using several message computation kernels. We perform a design-time exploration in order to assess the trade-offs between energy consumption, error-correction performance and implementation area. In particular, we concentrate on the Self-Corrected Min-Sum (SCMS) [14] kernel and show the advantages it brings from an energy efficiency

perspective. In addition, we present an optimization to the syndrome verification in order to speedup the decoding task.

- Architecture level: We consider the design issues of the memory subsystem for a flexible quasi-cyclic (QC) LDPC decoder. We present a flexible memory subsystem whose data organization and partition enable a low complexity shuffling network for message distribution. Furthermore, we investigate the impact of interleaving the memories in order to resolve access conflicts. The implementation of an energy efficient multi-mode decoder is presented along with comparisons to decoders from the previous art. By applying the proposed optimizations interesting gains in energy efficiency and implementation area are obtained.

- System level: We propose power management strategies for iterative decoders that rely upon laws that seek to control the iterative nature of the task and adjust the power level of a power manageable device. We apply the latter strategy to both LDPC and Turbo decoding and show notorious gains in energy efficiency.

As a final point and also to lay down the possibilities for future work within this research we consider the design of a unified decoder that supports both Turbo and LDPC codes. We focus on how to maximize the memory subsystem reuse and the selection aspects of the computation kernels.

The contributions of this research have been presented in the following publications:

- E. Amador, R. Pacalet and V. Rezard. "Optimum LDPC Decoder: A Memory Architecture Problem". In Proc. of the 46th Annual ACM/IEEE Design Automation Conference, pp. 891-896, July 2009.

- E. Amador, V. Rezard and R. Pacalet. "Energy Efficiency of SISO Algorithms for Turbo-Decoding Message-Passing LDPC Decoders". In Proc. of the 17th IFIP/IEEE International Conference on Very Large Scale of Integration, October 2009.

- E. Amador, R. Knopp, V. Rezard and R. Pacalet. "Dynamic Power Management on LDPC Decoders". In Proc. of IEEE Computer Society Annual Symposium on VLSI, pp. 416-421, July 2010.

- E. Amador, R. Knopp, R. Pacalet and V. Rezard. "On-the-fly Syndrome Check for LDPC Decoders". In Proc. of the 6th Interna-

tional Conference on Wireless and Mobile Communications, pp. 33-37, September 2010.

- E. Amador, R. Knopp, R. Pacalet and V. Rezard. "Hybrid Iteration Control on LDPC Decoders". In Proc. of the 6th International Conference on Wireless and Mobile Communications, pp. 102-106, September 2010.

- E. Amador. "Method and Device for Decoding Data", US Patent Application No. 12/961,645, filed on December 2010.

- E. Amador, R. Knopp, R. Pacalet and V. Rezard. "Dynamic Power Management for the Iterative Decoding of LDPC and Turbo Codes". Submitted to IEEE Transactions on Very Large Scale Integration, January 2011.

- E. Amador and D. Guenther. "Design of Interleaved Memory Architectures for QC-LDPC Decoders". Submitted to ACM Transactions on Design Automation of Electronic Systems, February 2011.

- E. Amador, R. Knopp, R. Pacalet and V. Rezard. "An Energy Efficient Multi-Mode LDPC Decoder for IEEE802.11n/16e Applications". Submitted to ACM Transactions on Embedded Computing Systems, February 2011.

- E. Amador, R. Knopp, R. Pacalet and V. Rezard. "High Throughput and Low Power Enhancements for LDPC Decoders". Accepted for publication on the International Journal on Advances in Telecommunications, April 2011.

1.3 Outline

This dissertation is organized as follows:

Chapter 2: Iterative Decoding

This chapter briefly introduces the topic of iterative decoding and the best-known codes that approach the limits set within information theory: Turbo and LDPC codes. The building blocks for this topic such as channel coding, optimal decoding and component codes are introduced serving as a background for the rest of the dissertation. A brief historic perspective on the developments of the field of channel coding and the evolution of wireless

networks is presented in order to support the quest for energy efficient and high throughput modems in general.

Chapter 3: Decoding of LDPC Codes

A specific instance of LDPC codes that are amenable for hardware implementations are presented in this chapter. Structured LDPC codes are addressed in the hardware architecture aspects of this work since these class of codes are preferred by the different standardization bodies as their hardware complexity is highly reduced. This is followed by an overview of the decoding algorithms and computation kernels for soft-decision decoding. This overview is not a comprehensive one since we focus our attention on the TDMP decoding proposed by Mansour et al [13] and low complexity computation kernels based upon variants of the min-sum (MS) algorithm.

Chapter 4: LDPC Decoder Architecture

An overview of the canonical decoding architecture for LDPC codes is presented in this chapter. This is followed by a comparison among several computation kernels in terms of performance, energy efficiency and implementation area. This comparison justifies the selection of the kernel used in the remaining chapters of this work. Our study reveals several properties of the SCMS kernel that can be exploited in order to achieve high error-correction performance, low implementation area and high energy efficiency. Finally, we propose an optimization in order to speedup the decoding task by modifying the syndrome verification. We provide insights into the impacts of this speedup at the levels of performance and implementation.

Chapter 5: Memory Architecture Design

This chapter outlines the design aspects for the memory subsystem of flexible QC-LDPC decoders. We formalize the design tasks of data partitioning and allocation with the goal of minimizing implementation area and providing sufficient flexibility. For the purpose of minimizing access latencies, we look into the topic of memory interleaving. Explorations of different memory configurations and decoding use cases serve to justify the relevance of this subsystem by comparing metrics like area per bit and average energy per iteration.

Chapter 6: Implementation of a Multi-mode LDPC Decoder

The work presented in Chapters 4 and 5 is used to implement a multi-mode decoder for IEEE 802.11n/16e applications. Architectural aspects are

detailed in this chapter including the memory subsystem, processing units and shuffling networks. Detailed breakdowns of the implementation area and energy consumption per use case are provided in order to justify the focus of our design approach. Furthermore, this implementation is compared with representative works from the previous art.

Chapter 7: Power Management for Iterative Decoders

In this chapter, we focus on two aspects of power management for these decoders: iteration control and workload prediction. For the former we propose a control law that relies upon two decision metrics in order to provide better energy gains on the average decoding task. The latter aspect is used to propose an online algorithm in order to adjust the power mode of a decoder such that a timing deadline is guaranteed for an energy efficient decoding task. Furthermore, we apply our work on dynamic power management to the decoding of Turbo codes. Implementation results of this work are provided for both LDPC and Turbo decoders.

Chapter 8: Unified Decoders and Future Work

Future work is aimed at the topic of unified decoding architectures for both Turbo and LDPC decoding. We present a brief state-of-the-art on such architectures and point out specific aspects that can be addressed for the purpose of designing energy efficient decoders based upon the different contributions presented in this dissertation.

Finally, in Chapter 9, concluding remarks are given.

Chapter 2

Iterative Decoding

In this chapter, we summarize a background for iterative channel decoding within digital communications. We present a brief historic perspective on error correcting codes that justifies the efforts on this research in order to achieve energy efficient and flexible VLSI decoders.

2.1 Digital Communications

The progress achieved in microelectronics allows the implementation of complex algorithms in an economic way. Essentially because of this reason today all major communication systems are used in the digital domain. Furthermore, the advances in circuit design, power management and fabrication technologies have allowed digital communication devices to achieve bit rates close to the limits defined by information theory.

A communication model is used to understand and disclose the challenges involved in the transmission of information between a source and one or several end points. Figure 2.1 shows the basic model used in information theory. Source symbols from an alphabet are mapped to a sequence of channel symbols $\mathbf{x} = [x_1, \ldots, x_n]$, this sequence goes through a channel that produces the sequence $\mathbf{y} = [y_1, \ldots, y_n]$. Each channel sequence is assigned to a particular continuous-time waveform, this is performed by the modulator. Prior to this a channel encoder introduces redundancy to a sequence that is used by a receiver in order to recover the sequence despite errors introduced

by the channel. At the receiver side the processing modules are applied in reverse fashion in order to estimate and recover the transmitted message.

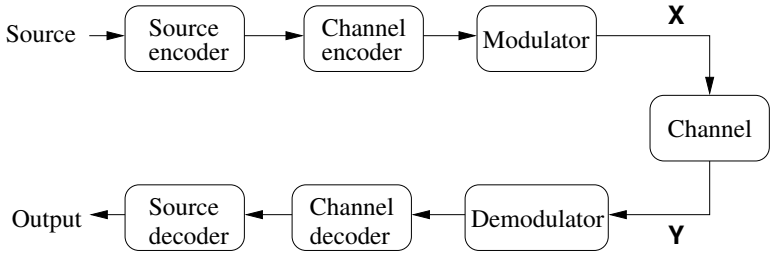


Figure 2.1: Basic communication model.

Information theory aims to answer two fundamental questions: which is the ultimate data compression? And, which is the ultimate transmission rate of information? Each module in the above figure contributes to the performance of the communication system as a whole. In this dissertation, we concentrate on the channel decoder, in particular to a type of high performance codes and the aspects involved in the implementation of VLSI architectures of high energy efficiency and low complexity.

2.2 Channel Coding

Channel coding refers to a set of techniques or signal transformations designed to achieve a reliable communication between two entities, namely a transmitter and a receiver. The communication between these entities is affected by various channel impairments like noise, interference and fading. These techniques offer a tradeoff between error performance and bandwidth.

Claude Shannon introduced the formalisms used to model and understand the communication problem in his celebrated paper in 1948 [15]. Indeed, information theory and coding were born with his seminal work. He showed how the communication problem can be separated into a source coding and a channel coding problem. Furthermore, he showed the limits at which error-free communication can be established over a noisy channel. More precisely, if the rate of transfered data is lower than a theoretical limit known as *channel capacity* it is possible to make the probability of error arbitrarily small at the receiver.

The "noisy channel coding theorem" asserts that the capacity C of a band-limited additive white Gaussian noise (AWGN) channel with bandwidth B is given by

$$C = B\log_2(1+SNR) = B\log_2\left(1+\frac{P_s}{P_n}\right)\ bit/s\ , \tag{2.1}$$

where the signal-to-noise ratio (SNR) is the ratio of the power at the transmitter P_s and the power of the noise P_n from the channel.

Shannon showed that for any transmission rate R less or equal to C there exists a coding scheme that accomplishes highly reliable communication or an arbritarily small probability of error. Equation (2.1) can be rewritten in different ways in order to provide bounds of different nature. The power at the transmitter is given by $P_s = E_b \cdot R$, where E_b is the energy per information bit. The power of the noise can be expressed by $P_n = N_0 \cdot B$, where N_0 is the one-sided noise power spectral density. The capacity can now be expressed by

$$C = B\log_2\left(1+\frac{R\cdot E_b}{B\cdot N_0}\right) = B\log_2\left(1+\eta\frac{E_b}{N_0}\right)\ bit/s\ , \tag{2.2}$$

where η is the spectral efficiency expressed in $bits/s/Hz$. From (2.2) for a given spectral efficiency, E_b/N_0 is lower bounded by

$$E_b/N_0 > \frac{2^\eta - 1}{\eta}\ . \tag{2.3}$$

In a similar way, the spectral efficiency is upper bounded by

$$\eta < \log_2(1+SNR)\ . \tag{2.4}$$

Shannon showed the existance of capacity-approaching codes by using random codes whose encoding/decoding complexity is intractable. Following his work, the coding research has focused on finding codes with simple structure but with enough "randomness" that allows near-capacity performance.

2.2.1 Error Correcting Codes

Error correcting codes typically add redundancy systematically to a message so that a receiver can determine the message with high probability despite the impairments of a noisy channel. These codes are used for Forward Error Correction (FEC), this gives the receiver the ability to correct errors without the need of a reverse channel to request retransmissions.

In coding theory two well-defined paradigms have dominated the design and construction of error correcting codes. *Algebraic* coding is concerned

with linear block codes that maximize the *minimum distance*, this is the minimum number of code symbols in which any two codewords differ. On the other hand, *probabilistic* coding is concerned with classes of codes that optimize average performance as a function of complexity. In Figure 2.2, we show the timeline[1]of some key progresses in coding theory and their applications.

Linear block codes and hard-decision decoding dominated the early decades after Shannon's seminal work. Even though these codes provide a good performance in the sense of coding gains they were not designed in the spirit of Shannon's idea of 'random-like' codes. It is with the use of soft-decision decoding and the introduction of Turbo codes by Berrou et al [1] that the limits set by Shannon became within reach. This breakthrough triggered a new paradigm for designing practical capacity-approaching codes. Moreover, the codes proposed by Gallager in 1963 [2], known as low-density parity-check (LDPC) codes, were rediscovered by MacKay [16] as he showed empirically that near-Shannon-limit performance was obtained with long block lengths and iterative decoding. The notion of iterative decoding from the work by Berrou et al was based upon the observation that a soft-input soft-output (SISO) decoder could be viewed as an SNR amplifier. A thorough account of the history of coding theory can be found in [17].

In this work, we focus on LDPC codes and their iterative decoding. Nevertheless, we include Turbo codes and their iterative decoding in some sections to illustrate the portions of this research that can be applied to iterative decoding in a broader sense. These codes are formed by the concatenation of simpler codes called *component* codes. In the following, we summarize some basic definitions and notations used throughout this work.

A code $\mathcal{C}$ of length N and cardinality $\mathcal{M}$ over a finite field $\mathbb{F}_q$ is a collection of $\mathcal{M}$ elements called *codewords* from $\mathbb{F}_q^N$. N is called the block length. Since a codeword is decodable by definition, in some sections we use the more general term *codeblock* when referring to instances of the decoding task that reach or not convergence.

An (N, K) binary linear block code is a set of codewords that forms a $K-$dimensional subspace in a $N-$dimensional vector space over $\mathbb{F}_2$. Since in this work we use binary codes, the field $\mathbb{F}_2$ is simply written as $\mathbb{F}$. The code can be represented by the row space of a generator matrix $\mathbf{G}$ that produces a codeword $\mathbf{x}$ from an information word $\mathbf{u} \in \mathbb{F}^N$:

[1]Main source taken from *Lecture Notes on Theory and Design of Turbo and Related Codes*, Forschungszentrum Telekommunikation Wien, 2004.

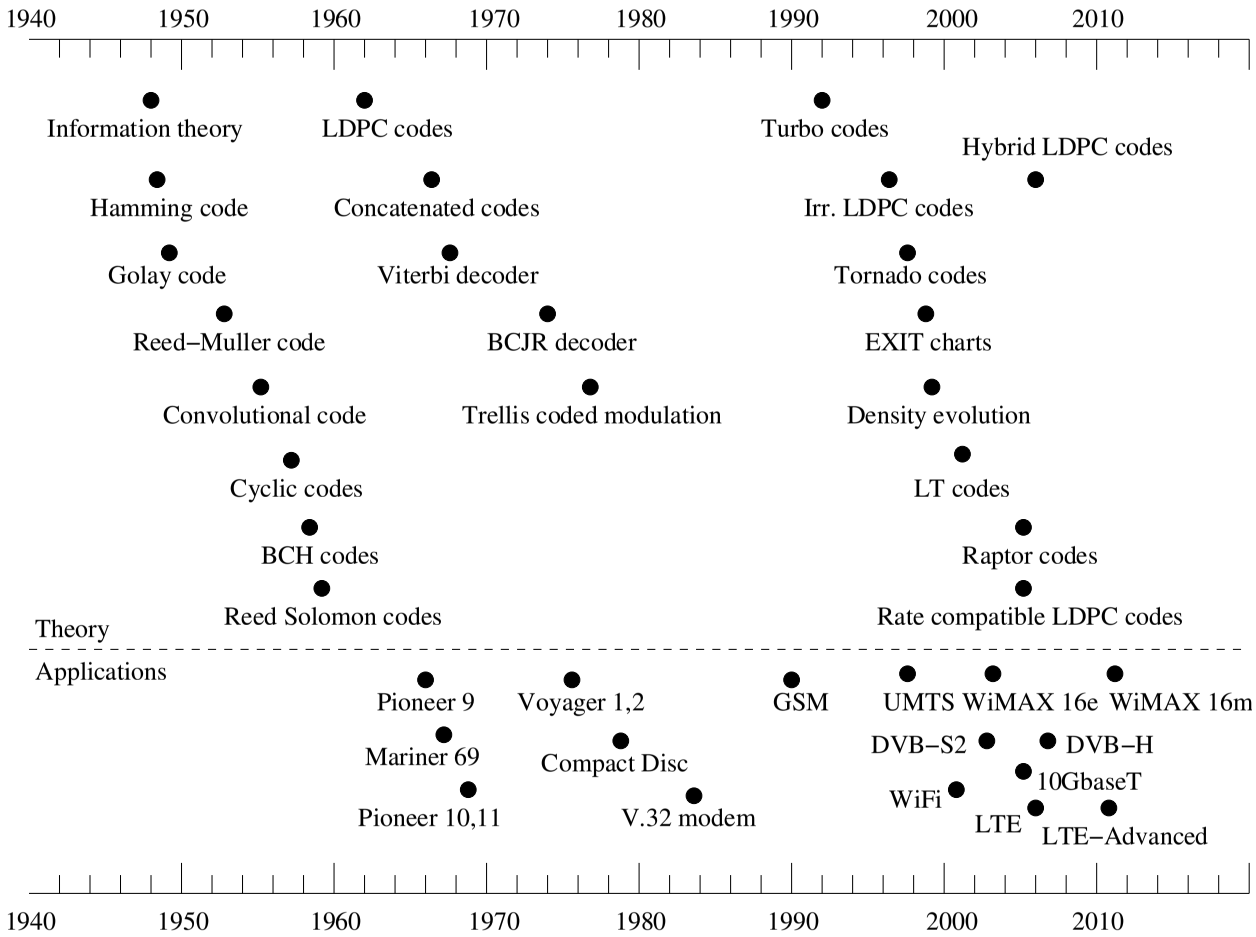


Figure 2.2: Evolution of coding theory and applications.

$$\mathcal{C} = \{\mathbf{x} : \mathbf{x} = \mathbf{uG}\} \ . \tag{2.5}$$

Alternately, the code can be represented as the null space of a parity-check matrix $\mathbf{H}$:

$$\mathcal{C} = \{\mathbf{x} : \mathbf{H}\mathbf{x}^T = \mathbf{0}\} \ . \tag{2.6}$$

The matrix $\mathbf{H}$ has dimensions $N \times M$ where $M = N - K$. The rate of the code is given by

$$R = 1 - \frac{M}{N} \ . \tag{2.7}$$

A *single parity-check code* of length N is defined by a parity-check matrix of dimensions $N \times 1$ of elements in $\mathbb{F}$.

A *convolutional code* is generated when passing an information sequence through a linear finite-state shift register. The shift register in general consists of k stages and n linear algebraic function generators. The input sequence is shifted k bits at a time generating n outputs, in this way the rate is given by $R = k/n$. The constraint length of the code is $K = k + 1$ and it describes the maximum number of information bits that contribute on the output of the encoder. The code may be described by a semi-infinite generator matrix or by the generating functions using a delay operator D.

Both Turbo and LDPC codes are constructed by a concatenation of simpler component codes like the ones mentioned above. In Section 2.3, we briefly introduce these codes.

2.2.2 Channel Abstraction and Optimal Decoding

The channel or link between transmitter and receiver is usually modeled as a non-deterministic mapping between an input $\mathbf{x}$ and an output $\mathbf{y}$ where a conditional probability density function $p(\mathbf{y}|\mathbf{x})$ characterizes this mapping. The inputs and outputs of the channel can be modeled as discrete values belonging to a specific alphabet and/or continuous values. Well-known channel models include the binary erasure channel, the binary symmetric channel and the binary input additive white Gaussian noise channel, [18].

The purpose of the decoder is to compute the most probable transmitted message $\mathbf{x}$ after it has been distorted by the channel. The decoded message $\hat{\mathbf{x}}$ can be estimated using two techniques depending upon the format of the messages taken from the channel output: *hard-decision* decoding uses

binary-decision values while *soft-decision* decoding uses non-binary-decision values.

The most probable codeword $\hat{\mathbf{x}}$ for a given received block $\mathbf{y}$ is estimated by maximizing the a-posteriori probability, this is known as the maximum a-posteriori (MAP) decoder:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}' \in \mathcal{C}}{\operatorname{argmin}} P(\mathbf{x} = \mathbf{x}' | \mathbf{y}) \; . \tag{2.8}$$

Provided all codewords are equally probable and using Bayes' rule the conditional probability in (2.8) can be expressed by

$$\hat{\mathbf{x}} = \underset{\mathbf{x}' \in \mathcal{C}}{\operatorname{argmin}} p(\mathbf{y} | \mathbf{x} = \mathbf{x}') \; , \tag{2.9}$$

this is known as the maximum likelihood (ML) decoder. Optimum MAP decoding involves testing each codeword from a total of 2^N for a code in the binary field $\mathbb{F}$. More precisely, a decoder chooses the codeword in $\mathcal{C}$ whose metric is closer to the received block $\mathbf{y}$. In the case of hard-decision decoding the metric is the *Hamming* distance whereas for soft-decision decoding the *Euclidean* distance is used. The Viterbi algorithm [19] provides an efficient block decoding, this minimizes the block or frame error rate (FER).

For the case of symbol by symbol (*bitwise*) decoding the optimum decoding rule to estimate each code symbol in $\hat{\mathbf{x}} = [x_1, \ldots, x_N]$ is

$$\hat{x}_n = \underset{x' \in \mathbb{F}}{\operatorname{argmin}} P(x_n = x' | \mathbf{y}) \; . \tag{2.10}$$

The BCJR algorithm [20] is an efficient symbol MAP decoder, indeed it minimizes the symbol or bit error rate (BER). Decoders for capacity-approaching codes typically use soft-decision kernels known as soft-input soft-output (SISO) decoders. The decoding kernels usually operate with soft-messages in the form of log-likelihood ratios (LLR). LLR messages are commonly used since the LLR-arithmetic [21] exhibits very low complexity (e.g., additions instead of multiplications). Furthermore, LLRs enable decoding strategies that are not restricted to a specific output alphabet of the channel model. For every received code symbol x the corresponding LLR is given by:

$$L(x) = \log \frac{P(x = 0)}{P(x = 1)} \tag{2.11}$$

where $P(A = y)$ defines the probability that A takes the value y. LLR values with a positive sign would imply the presence of a logic 0 whereas a

negative sign would imply a logic 1. The magnitude of the LLR provides a measure of reliability for the hypothesis regarding the presence of a logic 0 or 1. Considering the messages involved in the decoding process, the LLR of an information bit x is given by:

$$L(x) = L_c(x) + L_a(x) + L_e(x) \tag{2.12}$$

where $L_c(x)$ is the intrinsic message received from the channel, $L_a(x)$ is the a-priori value and $L_e(x)$ is the extrinsic value estimated using the code characteristics and constraints. $L(x)$ is the a-posteriori value and a hard-decision upon it, i.e., extraction of the mathematical sign, is used to deduce the binary decoded value.

2.3 Capacity-approaching Codes

The best-known codes that perform close to the Shannon limit are Turbo and LDPC codes. Both types of codes achieve such performance when iterative soft-decoding is used. Even though these codes have some fundamental differences, e.g., their component codes, it has been shown that these codes belong to a superset known as *codes on sparse graphs*. Wiberg showed in 1995 [22] that these codes are instances of codes defined over sparse graphs, and that their iterative decoding algorithms are instances of a general iterative algorithm called *sum-product*. Tanner introduced in 1981 [23] the bipartite graph used to model LDPC codes, his work along with the contributions by Wiberg are consolidated in what is known as *factor graphs* and the sum-product algorithm [24]. Indeed, modern coding theory relies upon graphical models and variants of the sum-product algorithm, [18].

Factor graphs essentially represent the factorization of a function. It has been shown how different applications can be naturally described within the framework of factor graphs, some of which include important applications in statistical inference like the Viterbi algorithm, the forward/backward algorithm and the Kalman filter, [24] [25].

The factor graph representation of a multivariate function enables the computation of marginals with respect to any variable by passing messages along the nodes of the graph. The sum-product algorithm describes how to compute marginals along a factor graph, these marginals can be shown to be exact provided the graph is cycle free, i.e., the graph is a tree. For the case where cycles are present the algorithm naturally iterates. Nevertheless, if cycles present in the graph are not too short the algorithm performs quite

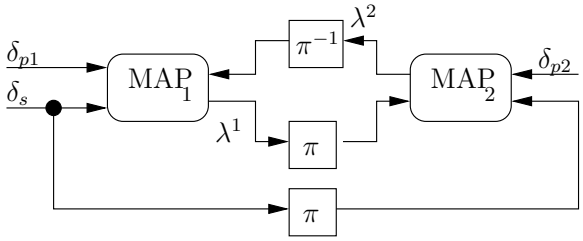


Figure 2.3: General structure of a Turbo decoder.

well. In coding theory this paradigm is used to model the a-posteriori and likelihood functions, equations (2.8) and (2.9) respectively.

2.3.1 Turbo Codes

Turbo codes consist of the parallel concatenation of two convolutional encoders separated by an interleaver. The codes are typically systematic and their error-correction performance depends in great part upon the characteristics of the interleaver. The decoding strategy for these codes consists of the decoding of the individual component codes and an iterative exchange of *extrinsic* information between the two decoders. SISO decoders are used and typically execute MAP decoding [20] in the logarithmic domain along the code *trellis*, a time-evolved graphical representation of the states of a convolutional encoder. The general structure of a Turbo decoder is shown in Figure 2.3. Intrinsic messages (δ_s for systematic bits and $\delta_{p1,p2}$ for parity bits) in the form of LLRs are distributed in non-interleaved/interleaved form to two MAP decoders that generate and exchange extrinsic information ($\lambda^{1,2}$) in an iterative fashion. Each decoding round performed by a MAP unit constitutes a *half-iteration*. Iterations are performed until convergence is achieved or a maximum number of iterations is completed.

2.3.2 Low-Density Parity-Check Codes

Binary LDPC codes are linear block codes described by a sparse parity-check matrix $\boldsymbol{H}_{M\times N}$ over $\mathbb{F}_2$. This matrix defines M parity constraints over N code symbols. The number of non-zero elements per row define the degree of the row. A codeword $\boldsymbol{c}$ satisfies the condition:

$$\boldsymbol{H}\boldsymbol{c}^T = \boldsymbol{S} = \boldsymbol{0}\,, \tag{2.13}$$

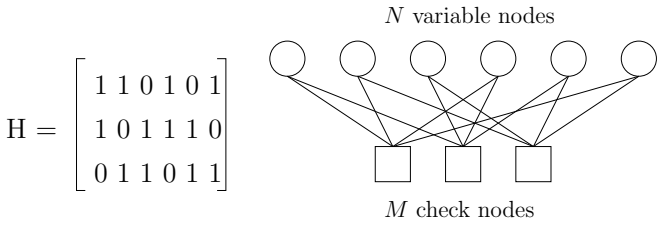


Figure 2.4: LDPC code graph example.

where $\boldsymbol{S}$ is referred to as the *syndrome* when this inner product is non-zero. Furthermore, the code can be represented by a bipartite graph in which rows of $\boldsymbol{H}$ are mapped to *check nodes* and columns to *variable nodes*. The non-zero elements in $\boldsymbol{H}$ define the connectivity between the nodes. Figure 2.4 shows an example matrix (non-sparse) and the corresponding code graph representation.

LDPC codes can be decoded iteratively by the sum-product algorithm, a so-called message-passing algorithm. This algorithm achieves MAP decoding performance provided the code graph is cycle-free. Even though there are hard-decision decoding algorithms based upon Gallager's work [2] and various bit-flipping algorithms used for different high-speed applications they are not treated in this work. The sum-product algorithm is applied to the code graph where check nodes and variable nodes exchange messages that essentially carry extrinsic reliability values associated with each code symbol.

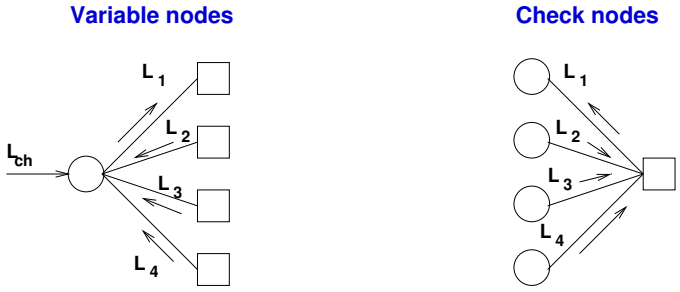


Figure 2.5: Example message exchange on the nodes of a code graph.

Figure 2.5 illustrates the message exchange between nodes. Each message L_i is a function of the incoming messages at a node but excluding the ith edge of the node. This guarantees that a newly generated message is independent from its previous value, this is the so-called *extrinsic principle*.

Message generation is performed at each node, for the case of a variable node a message is generated by

$$L_i = L_{ch} + \sum_{j \neq i} L_j \ , \tag{2.14}$$

where L_{ch} is an initial message taken from the channel observations. On the other hand, a check node generates a message by

$$L_i = 2 \cdot \tanh^{-1} \Big(\prod_{j \neq i} \tanh \frac{L_j}{2} \Big) \ . \tag{2.15}$$

Notice that the mentioned node operations correspond to the sum-product algorithm applied to the code graph of a binary LDPC code. In the succeeding chapters, we will concentrate on the alternatives for message computation that feature low complexity. The expression in (2.13) is evaluated to indicate when convergence has been achieved and is used to stop the decoding task. Otherwise, a maximum number of iterations is completed.

2.3.3 Performance of Iterative Decoding

Iterative decoding techniques on Turbo and LDPC codes show a characteristic behavior on the error rate as a function of SNR. Figure 2.6 shows the typical error rate curve exhibited on these codes. There is a clear distinction between three regions on this curve:

- Non convergence: the decoder shows low efficiency in the low SNR region, even an increase on the number of iterations would not improve much the performance.
- Waterfall region: the error rate improves substantially with small increments on the SNR at mid to high SNR values. The performance improves in this region with an increase of the number of iterations.
- Error floor region: in this region the error rate slope dramatically changes and the performance gains are limited despite the increase in signal energy.

The error floors occur typically at low error rates and in general do not present problems for applications that require moderately low bit error rates like wireless communications. However, for applications that require low bit error rates like magnetic storage and optical communications these floors are

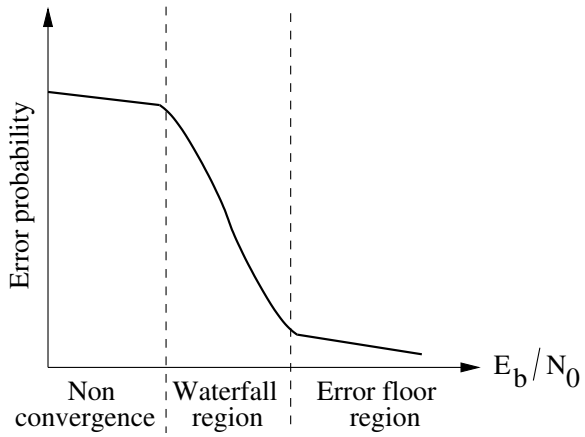


Figure 2.6: Typical behavior of error probability on iterative decoding algorithms.

problematic. In Section 6.1.3, we outline the reasons for this phenomenon for the LDPC case, even though in general the main cause being the distance properties of the code.

2.4 Conclusion

Shannon gave birth to the field of information theory and proved the existance of coding schemes that can achieve highly reliable communication below the channel capacity. In the search for such codes, coding theory has explored several paradigms including algebraic codes and codes described on sparse graphs along with their iterative decoding. Turbo codes and LDPC codes are formed by the concatenation of simpler codes that are combined in a random-like fashion, i.e., an interleaver in the Turbo case and the sparse nature of $\boldsymbol{H}$ in the LDPC case. Optimum decoding is realized by the MAP algorithm and is typically performed with soft-decision messages.

Chapter 3

Decoding of LDPC Codes

The random-like nature of LDPC codes makes their encoding/decoding task a complex one due to the connectivity within the code graph. Moreover, the sparse characteristic of the parity-check matrix increases the complexity of a decoder that may support various codes. In this chapter, we describe codes that enforce a particular structural property which greatly simplifies the encoding/decoding task and enables *semi-parallel* architectures. Furthermore, we review the decoding algorithm for these codes and various message computation kernels.

3.1 Structured Codes

LDPC codes meet outstanding performance for large block lengths [16], but this results in large parity-check and generator matrices. The generator matrix of an (N, K) systematic linear block code can be expressed in the form $\mathbf{G} = [\mathbf{I}_K | \mathbf{P}]$ provided the sparse parity-check matrix $\mathbf{H}$ is transformed to the form $[\mathbf{P}^T | \mathbf{I}]$. $\mathbf{I}_K$ is the $K \times K$ identity matrix and $\mathbf{P}$ is a $K \times (N-K)$ matrix. $\mathbf{P}$ is in general not sparse rendering the complexity of the encoding task quite high. From the side of the decoder, the random position of the non-zero elements in $\mathbf{H}$ translates into a complex interconnection network.

LDPC codes used in communication standards adopt some structural characteristics that circumvent the complexities just mentioned. *Architecture-aware* [13] codes and *quasi-cyclic* LDPC [26] codes achieve almost linear

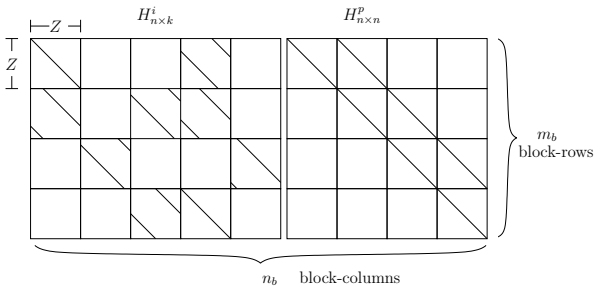
encoding complexity in the block length and enable a grouping of edges and nodes within the code graph. A quasi-cyclic (QC) LDPC code is obtained if $\boldsymbol{H}$ is formed by an array of sparse circulants of the same size, [26]. If $\boldsymbol{H}$ is a single sparse circulant or a column of sparse circulants this results in a cyclic LDPC code. These codes are composed of several layers of non-overlapping rows that enable the concurrent processing of subsets of rows without conflicts.

For example, the QC-LDPC codes defined in IEEE 802.11n [5] and IEEE 802.16e [6] consist of layers formed by $Z \times Z$ sub-matrices. Z is an expansion factor that shows the degree of available parallelism as Z non-overlapping rows can be processed concurrently. Each of these sub-matrices can be either an all-zeroes matrix or a circularly shifted identity matrix. Furthermore, these codes are constructed such that their encoding process exhibits linear complexity in the length of the code. For these codes (systematic) $\boldsymbol{H}$ is typically partitioned in two parts: a random part H^i for the systematic information and a prearranged part H^p for the parity information. Figure 3.1 shows the $\boldsymbol{H}$ matrix and the bipartite graph of an example structured code where $\boldsymbol{H}_{M\times N} = [H^i_{n\times k}|H^p_{n\times n}]$, with $M = n$ and $N = n + k$. Indeed, $\boldsymbol{H}$ consists of an array $m_b \times n_b$ of $Z \times Z$ blocks.

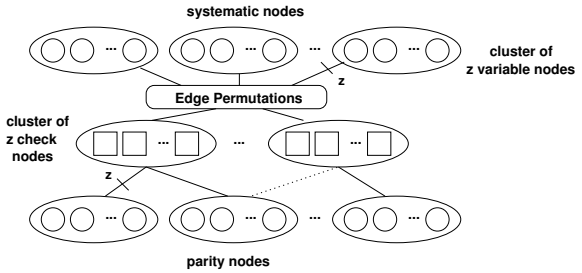
The graph representation in Figure 3.1b contains edges of width Z that group Z nodes into clusters. This grouping enables the possibility to instantiate a subset of the processing nodes. This represents a clear advantage in terms of flexibility and implementation area as well as a reduction of the complexity of the interconnection network (edge permutations). Such characteristics are indeed exploited under semi-parallel architectures.

3.2 Decoding Algorithms

As mentioned in Section 2.3.2, LDPC codes are typically decoded iteratively using a two-phase message-passing algorithm, the sum-product algorithm [24]. Each decoding iteration consists of two phases: variable nodes update and send messages to the neighboring check nodes, and check nodes update and send back their corresponding messages. Node operations are in general independent and may be executed in parallel. This allows the possibility to use different scheduling techniques that may impact the convergence speed of the code and the storage elements requirements. The algorithm initializes with intrinsic channel reliability values and iterates until hard-decisions upon the accumulated resulting posterior messages satisfy equation (2.13). Otherwise, a maximum number of iterations is completed.



(a) Parity-check matrix



(b) Bipartite graph

Figure 3.1: Structured LDPC code example.

The work in [27] presented a generalization for the decoding of sparse parity-check matrix codes. This work consolidated several concepts that have greatly optimized the decoding process, such as a merger of messages to save on memory requirements and a layered-scheduling that exploits architecture-aware codes. The decoding algorithm for sparse parity-check matrix codes is generalized in [27] in what is referred to as the turbo-decoding message-passing (TDMP) algorithm. In TDMP decoding the check nodes are evaluated sequentially updating and propagating more reliable messages along the code graph, consequently achieving convergence in a faster way (twice as fast as the two-phase schedule). At the core of TDMP decoding lies a SISO message computation kernel that corresponds to the check node operation in the code graph.

In the following, we summarize this algorithm from [27]. Let the vector $\boldsymbol{\delta} = [\delta_1, \ldots, \delta_N]$ denote the intrinsic channel observations per code symbol as log-likelihood ratios, and a vector $\boldsymbol{\gamma} = [\gamma_1, \ldots, \gamma_N]$ denote the sum of all messages generated in the rows of $\boldsymbol{H}$ for each code symbol (posterior messages). Let us define as well a vector $\boldsymbol{\lambda}^i = [\lambda^i_1, \ldots, \lambda^i_{c_i}]$ for each row i in $\boldsymbol{H}$ that contains the c_i extrinsic messages generated after each decoding

round where c_i is the degree of the row. Let $\mathcal{I}_i$ define the set of c_i non-zero values in row i such that the c_i elements of $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ that participate in row i are denoted by $\boldsymbol{\gamma}(\mathcal{I}_i)$ and $\boldsymbol{\delta}(\mathcal{I}_i)$ respectively. Furthermore, let the vector $\boldsymbol{\rho}$ define the prior messages. The decoding of the ith row is outlined in Algorithm 1.

Algorithm 1 TDMP Decoding

Initialization
$\boldsymbol{\lambda}^i \leftarrow \mathbf{0}$
$\boldsymbol{\gamma}(\mathcal{I}_i) \leftarrow \boldsymbol{\delta}(\mathcal{I}_i)$
For each iteration:

1. Read vectors $\boldsymbol{\gamma}(\mathcal{I}_i)$ and $\boldsymbol{\lambda}^i$
2. Generate prior messages: $\boldsymbol{\rho} = \boldsymbol{\gamma}(\mathcal{I}_i) - \boldsymbol{\lambda}^i$
3. Process $\boldsymbol{\rho}$ with a soft-input soft-output (SISO) algorithm: $\boldsymbol{\Lambda} = SISO(\boldsymbol{\rho})$
4. Writeback vectors:
   $\boldsymbol{\lambda}^i \leftarrow \boldsymbol{\Lambda}$
   $\boldsymbol{\gamma}(\mathcal{I}_i) \leftarrow \boldsymbol{\rho} + \boldsymbol{\Lambda}$

The process iterates until a stopping criterion is satisfied. The optimal stopping criterion corresponds to the syndrome check condition shown in equation (2.13). Nevertheless, there are several proposed stopping criteria (refer to Chapter 7) in order to detect early the possibility for an undecodable block. Hard-decisions are taken by slicing the vector $\boldsymbol{\gamma}$ to obtain the decoded message. TDMP decoding offers two advantages when compared to the traditional two-phase algorithm: a reduction of the memory requirements due to the merging of check and variable messages and a reduction in the number of iterations by up to 50%.

3.2.1 Message Computation Kernels

The processing complexity of the decoding task resides in the operations performed in the variable and check nodes of the code graph. Essentially the operation at the variable node is an addition of the incoming messages, whereas the operation at the check node involves more operations and it is where the tradeoff of performance and complexity takes place. In the context of TDMP the check node operation takes place on Step 3 of Algorithm

1. We refer to this step as *message computation* for generating the vector $\boldsymbol{\Lambda} = SISO(\boldsymbol{\rho})$ from the prior messages $\boldsymbol{\rho}$. The optimal message computation is performed by the Sum-Product (SP) algorithm by:

$$\Lambda_j = \psi^{-1}\Big(\sum_{n \in \mathcal{I}_i \setminus j} \psi(\rho_n) \Big) \,, \tag{3.1}$$

where (following the definitions in [13])

$$\psi(x) = -\frac{1}{2} \log(\tanh(\frac{x}{2})) = \psi^{-1}(x) \,. \tag{3.2}$$

Implementing (3.2) is highly complex mainly due to the effects of quantization and the non-linearity of the function. Along with the TDMP schedule, the authors in [13] proposed the computation of messages by using a simplified form of the BCJR algorithm [20] to process the 2-state trellis of each single parity-check constraint of the code. Indeed, this approach views an LDPC code as the parallel concatenation of single parity-check codes. A detailed analysis on this can be found in [13] [27]. The computation of messages is performed by:

$$\Lambda_j = Q_{[j]}(\ldots(Q(Q(\rho_1, \rho_2), \rho_3), \ldots), \rho_{c_i}) \,, \tag{3.3}$$

where

$$\begin{aligned} Q(x,y) \quad &= \max(x,y) + \max\left(\tfrac{5}{8} - \tfrac{|x-y|}{4}, 0\right) - \\ &\quad \max\left(\tfrac{5}{8} - \tfrac{|x+y|}{4}, 0\right) - \max(x+y, 0) \end{aligned} \tag{3.4}$$

is the so-called *max-quartet* function and the subscript $[j]$ denotes the index of the variable to exclude from the computation.

The Min-Sum (MS) algorithm [28] approximates the operation in (3.1) with less complexity but at the cost of error-correction performance. The MS operation computes messages by:

$$\Lambda_j = \Big(\prod_{n \in \mathcal{I}_i \setminus j} (sign(\rho_n)) \Big) \cdot \min_{n \in \mathcal{I}_i \setminus j} |\rho_n| \,. \tag{3.5}$$

Several correction methods have been proposed to recover the performance loss of the MS operation, such as the Normalized-MS (NMS) and Offset-MS (OMS) algorithms [28]. These correction methods essentially downscale the check node messages, which are overestimated in the first

place in MS. NMS computes messages by scaling equation (3.5) by a factor α:

$$\Lambda_j = \alpha \cdot \Big(\prod_{n \in \mathcal{I}_i \backslash j} (sign(\rho_n)) \Big) \cdot \min_{n \in \mathcal{I}_i \backslash j} |\rho_n| \,, \tag{3.6}$$

whereas OMS computes messages by:

$$\Lambda_j = \Big(\prod_{n \in \mathcal{I}_i \backslash j} (sign(\rho_n)) \Big) \cdot \max \left(\min_{n \in \mathcal{I}_i \backslash j} |\rho_n| - \beta, 0 \right) \,, \tag{3.7}$$

where β is an offset value.

It has been argued in [14] that the sub-optimality of MS decoding is not due to the overestimation of the check node messages, but instead to the loss of the symmetric Gaussian distribution of these messages. This symmetry can be recovered by eliminating unreliable variable node messages or *cleaning* the inputs of the check node operation. In [14] the Self-Corrected MS (SCMS) decoding is introduced. This kernel exhibits quasi-optimal error-correction performance with the same low-complexity found in the other MS-based kernels. An input to the check node operation is identified as *unreliable* if it has changed its sign with respect to the previous iteration. In Algorithm 2, we show how to integrate the SCMS kernel to the TDMP decoding of a row i.

The vector $\boldsymbol{\kappa} = [\kappa_1, \ldots, \kappa_{c_i}]$ corresponds to the *corrected* inputs for the MS operation. Steps 3 and 4 correspond to the main features of the SCMS algorithm where unreliable variable messages are identified and *erased*. In this way unreliable values are no longer propagated along the code graph.

The SCMS kernel is said to have quasi-optimal performance since it approaches the optimal SP kernel. Figure 3.2 shows the simulated bit error rate (BER) for the mentioned kernels for coding rate 1/2 over the additive white Gaussian noise (AWGN) channel with quadrature phase-shift keying (QPSK) modulation for the code of length 1944 in IEEE 802.11n [5]. A maximum of 60 decoding iterations were used, where the NMS kernel used a normalization factor of 0.8 and the OMS kernel used an offset factor of 0.35. The SP kernel is taken to be the optimal one, its behavior is closely followed by the BCJR kernel. The MS kernel exhibits a considerable performance loss that is somehow recovered by its variants (NMS and OMS). The interesting observation is the performance of the SCMS kernel. This kernel exhibits a lower error floor (similar results were obtained for other codes and use cases). This behavior is mainly due to the mitigation of the effects of cycles in the code when erasing unreliable messages.

Algorithm 2 TDMP-SCMS

Initialization
$\boldsymbol{\lambda}^i \leftarrow \mathbf{0}$
$\boldsymbol{\gamma}(\mathcal{I}_i) \leftarrow \boldsymbol{\delta}(\mathcal{I}_i)$
At iteration $k \neq 0$:

1. Read vectors $\boldsymbol{\gamma}(\mathcal{I}_i)$, $\boldsymbol{\lambda}^i$ and $\boldsymbol{\rho}^i_{old}$
2. Generate new prior messages: $\boldsymbol{\rho}^i_{new} = \boldsymbol{\gamma}(\mathcal{I}_i) - \boldsymbol{\lambda}^i$
3. Generate MS input $\boldsymbol{\kappa}$ such that:

for all $j \in c_i$ **do**
  if $sign(\rho^i_{new_j}) \neq sign(\rho^i_{old_j})$ **then**
    $\kappa_j = 0$
  else
    $\kappa_j = \rho^i_{new_j}$
  end if
end for

4. Generate MS output $\boldsymbol{\Lambda} = MS(\boldsymbol{\kappa})$:

$$\Lambda_j = \Big(\prod_{n \in \mathcal{I}_i \setminus j} (sign(\kappa_n)) \Big) \cdot \min_{n \in \mathcal{I}_i \setminus j} |\kappa_n| \tag{3.8}$$

5. Writeback vectors:
   $\boldsymbol{\lambda}^i \leftarrow \boldsymbol{\Lambda}$
   $\boldsymbol{\rho}^i_{old} \leftarrow \boldsymbol{\rho}^i_{new}$
   $\boldsymbol{\gamma}(\mathcal{I}_i) \leftarrow \boldsymbol{\rho}^i_{new} + \boldsymbol{\Lambda}$

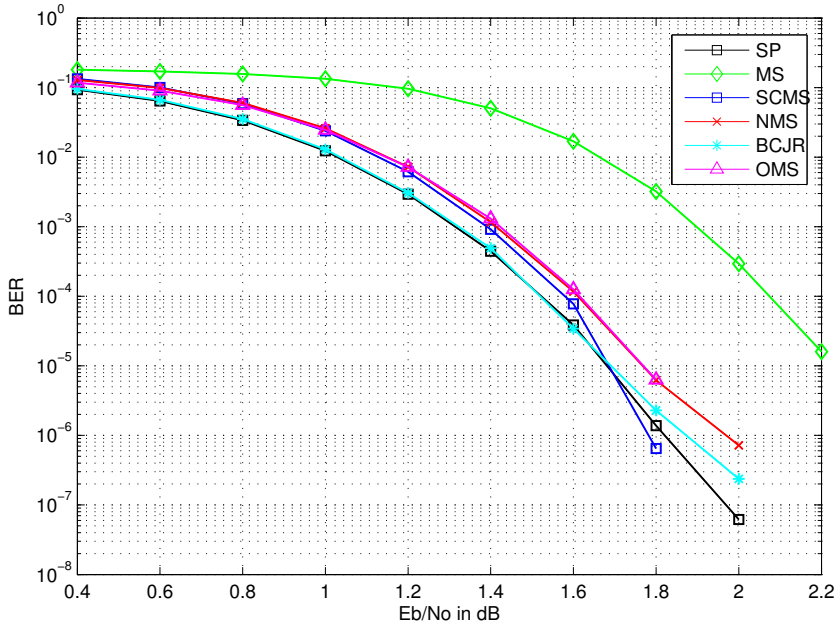


Figure 3.2: Bit error rate performance

3.3 Conclusion

Structured LDPC codes reduce the complexity of the encoding/decoding tasks. Furthermore, these codes enable the use of semi-parallel architectures that are inherently flexible. There are several kernels for message computation to be used for LDPC decoding. These kernels provide each a tradeoff between error-correction performance and computational complexity. In the succeeding chapter these kernels will be compared from the perspectives of energy efficiency and implementation area.

Chapter 4

LDPC Decoder Architecture

In this chapter, we focus on the architecture for LDPC decoding hardware based upon the TDMP decoding strategy presented in the previous chapter. The selection of a message computation kernel is based upon comparisons in performance, energy efficiency and implementation area. Finally, we present an optimization to speedup the decoding task by partitioning the code syndrome verification.

4.1 Decoding Architectures

The decoding algorithm for LDPC codes exhibits a high amount of parallelism that may be exploited in order to achieve high decoding throughput. Nevertheless, exploiting this parallelism has several challenges mainly due to the irregularity within the connectivity of the code. A fully parallel implementation like the one in [29] exhibits a high throughput advantage and a reduction in power consumption due to a rapid convergence, but suffers from a complex network of message wires and lacks flexibility. In the dissertation by Guilloud [30] a generic framework for analyzing LDPC decoding architectures is provided.

The codes shown in Section 3.1 exploit the regularity of the code structure and enable the instantiation of a subset of the processing nodes for so-called *partially-parallel* or *semi-parallel* architectures. Furthermore, the interconnection complexity between processing nodes is reduced since se-

veral edges of the code graph can join clusters of nodes. These architectures are inherently flexible and may support several types of codes. By flexibility we refer to the capability of an architecture to be used in different contexts, [31]. Such goal can be achieved by making an architecture *adaptable*, *reconfigurable* or *parameterizable*. In order to address flexibility software-based implementations for FEC processing have been proposed in the context of application-specific instruction-set processors (ASIP) with works in [32] [33] as the most prominent ones. This strategy nonetheless is not yet as efficient with respect to a dedicated hardware design in terms of power and energy expenditure due to the memory-intensive nature of this application.

Most of the literature on LDPC decoders shows a clear trend that favors semi-parallel architectures. Numerous works have proposed semi-parallel architectures that are optimized in different ways in order to conceive energy and area efficient designs. For example, for multi-mode decoders, authors in [34] proposed a reconfigurable network for message distribution and authors in [35] relied upon an early termination scheme and memory banking to reduce power consumption. The work presented in [36] divides the decoder operation in several tasks and arranges their order such that the challenges due to flexibility may be solved. Similarly, in [37] the authors proposed a memory-bypassing scheme where the order of the processing layers is altered such that energy consumption is minimized. The work in [38] addresses as well the topic of task rearrangement and optimizes the message quantization as well as their storage scheme. In Chapter 6, we compare some of these cited works with the implemented decoder presented in this dissertation.

4.2 Decoder Overview

The decoder architecture follows from the description in Algorithm 1. The required storage elements are: a posterior messages memory ($\boldsymbol{\gamma}$), which holds the N code symbol LLR messages; and an extrinsic messages memory ($\boldsymbol{\lambda}$), whose size corresponds to the number of edges of the code graph. It is important to note that as described in [27] with TDMP decoding the vector $\boldsymbol{\gamma}$ is initialized with the intrinsic channel observations $\boldsymbol{\delta}$. Consequently, $\boldsymbol{\delta}$ does not play a role in the iterative process other than during initialization. Besides these memories storage is required for the structure of the parity-check matrices, this involves the location of the non-zero sub-matrices and their corresponding shift value.

Figure 4.1 shows the top level view of the proposed decoder architecture.

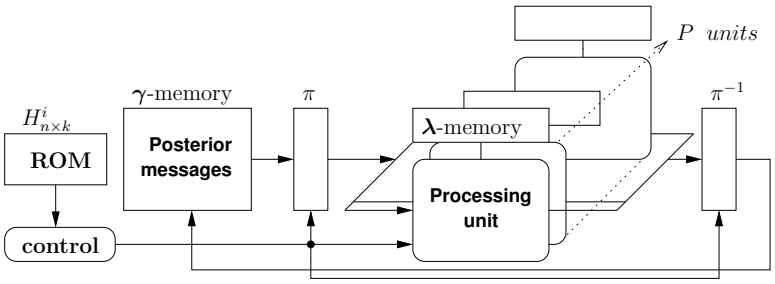


Figure 4.1: LDPC decoder architecture.

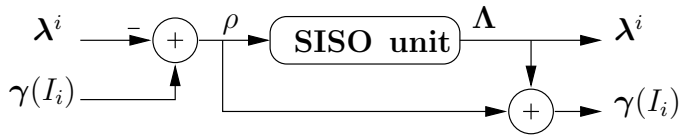


Figure 4.2: SISO processing unit.

A read-only memory (ROM) stores the control information from the structure of the section $H^i_{n\times k}$ in the parity-check matrix corresponding to the systematic part of the code. Shuffling units π and π^{-1} are used to distribute posterior messages to and from a set of P processing units. The processing units exploit the parallelism offered by the structure in $\boldsymbol{H}$ such that up to P rows can be decoded concurrently.

4.2.1 Processing Unit

The processing units execute the message computation kernel of choice, this could be any of the alternatives described in Section 3.2.1. Figure 4.2 shows the data flow for this unit.

The selection of the computation kernel impacts a portion of the memory subsystem (this is addressed in Chapter 5) but also the architecture of the processing unit could be impacted. Among the previously discussed kernels, only an SCMS-based processing unit would deviate from the one shown in Figure 4.2. The MS-based kernels fundamentally perform a running comparison of magnitudes and a sign calculation. The processing unit for SCMS requires a pre-computation block for performing Step 3 from Algorithm 2. Figure 4.3 shows the top level view of a processing unit for SCMS message computation.

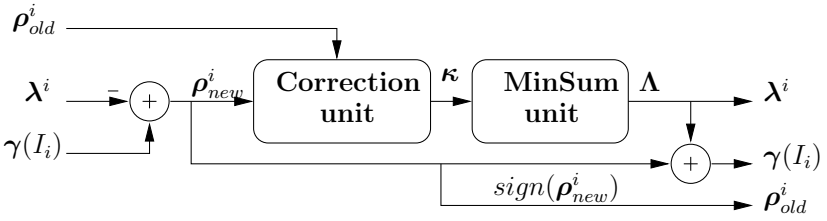


Figure 4.3: SCMS processing unit.

4.2.2 Shuffling Networks

Message distribution within LDPC decoding architectures is typically carried out by shuffling or interconnection networks. These components perform the operation of the code graph edges. It is straightforward to observe that their complexity is a function of the structure of the parity-check matrix, i.e., the position of the non-zero elements in $\boldsymbol{H}$. These networks in fact are equivalent to the interleavers used within Turbo decoding.

The structure of architecture-aware and quasi-cyclic codes reduces the complexity of these networks. Since these codes are processed in clusters or subsets of rows in $\boldsymbol{H}$ that belong to a circulant submatrix the data (posterior messages) within said circulant is rotated according to the shift value of the submatrix. Consequently, the architectures for these interconnection networks typically rely upon configurable barrel-shifters, crossbar switches or non-blocking switching networks like Banyan and Beneš networks [39]. In Section 6.3, we provide a brief description of representative state-of-the-art shuffling network architectures.

The shuffling networks act as links between the memories and the processing nodes within a semi-parallel architecture. Therefore, the structure of the networks depends upon the memory subsystem structure and the organization of the data. In Chapter 6, we present a low-complexity shuffling network that is the direct outcome of a particular memory organization and partition proposed in this dissertation.

4.3 Energy Efficient Computation Kernel

The message computation kernels described in Section 3.2.1 have different characteristics and impacts on a VLSI implementation. Energy expenditure and implementation area are the most critical issues for efficient mobile wireless modems. In this section, we look mainly into the energy efficiency

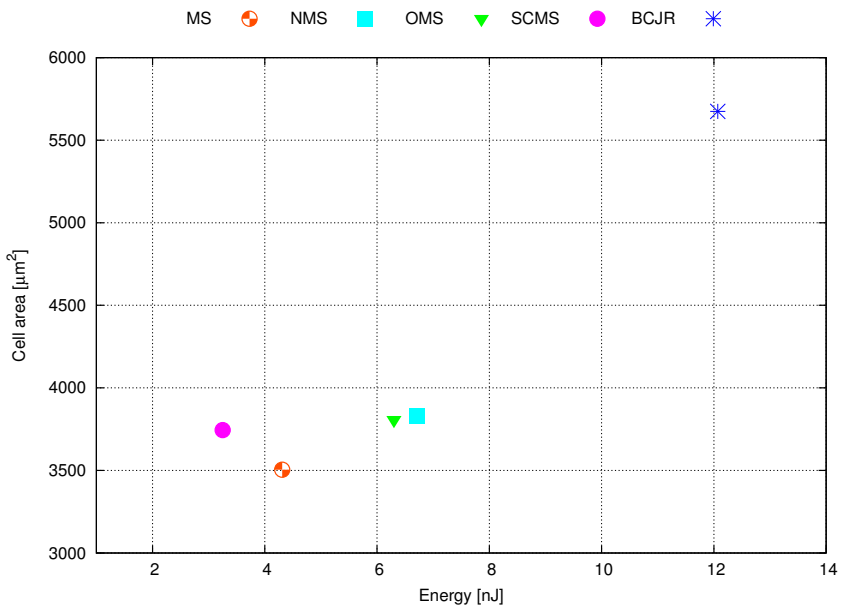


Figure 4.4: SISO kernels cost visualization.

of these kernels.

We implemented at the register-transfer level (RTL) the kernels for message computation in order to observe the costs previously mentioned. Postlayout netlists were used along with annotated switching activity in order to estimate the energy consumption. The tools used were Design Compiler® and PrimeTime PX® by Synopsys, Inc. In Figure 4.4, we show the cell area per kernel of a serial SISO unit with a message quantization of 6-bits on a CMOS technology process of $65nm$ and the average energy consumed per iteration per kernel when decoding the code in IEEE 802.11n of length 1944 and rate 1/2.

The BCJR kernel has the most complex data path and clearly its implementation area surpasses the other kernels, its cost on energy expenditure per iteration is as well the highest one. On the other hand, the MS kernel represents the simplest kernel with the smallest footprint. The variants of the MS kernel that resize the output messages raise both the area and energy cost. On the other hand, the SCMS kernel presents a 7% increase in area compared to the MS kernel and it slightly outperforms the MS kernel in energy consumption. In the following, we elaborate on the energy efficient characteristics offered by the SCMS kernel as an integral component for efficient decoders.

4.3.1 Convergence Rate

Iterative decoding comprises a dynamic task as the number of iterations depends upon factors like the SNR and channel quality. The message calculation clearly affects the convergence speed since reliable and unreliable messages are propagated along the code graph after each iteration. In this sense one kernel might converge faster or slower than others, nevertheless from an energy efficiency perspective it is compelling to compare the net consumption from each kernel. Figure 4.5 shows the comparison of message kernels[1]in terms of convergence speed and net energy consumption. These figures correspond to the simulation of the code in IEEE 802.11n of length 1944 and rate 1/2 with the same scenario from Figure 3.2. The energy values for Figure 4.5b are taken from the data in Figure 4.4. Notice that the SP kernel is not implemented as we focus on kernels with reduced complexity but we provide the SP convergence rate as it represents a performance bound.

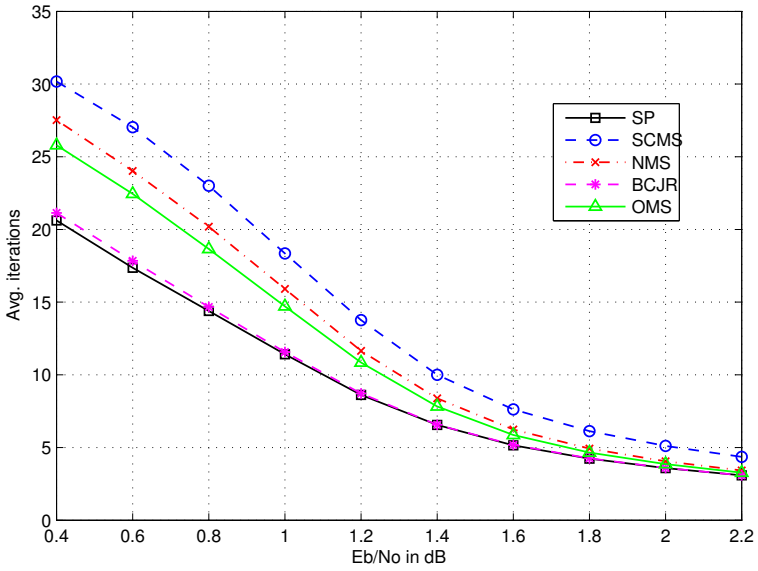
From these figures it is observed how SCMS is the slowest kernel to converge, this is expected as it provides the fewest number of messages per iteration. Nevertheless, it is for this same reason that it consumes less energy per iteration, rendering the overall decoding task more efficient.

4.3.2 Active Nodes Reduction

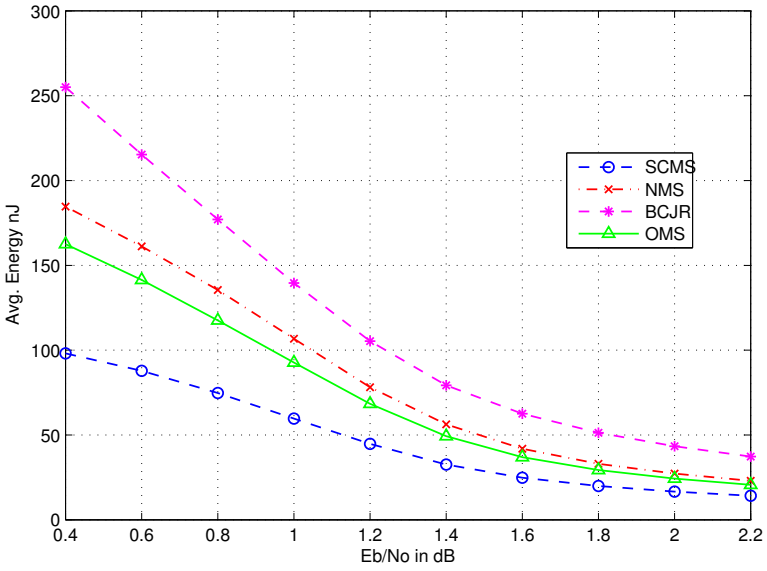
The number of nodes in the code graph that are active during each decoding iteration impacts the energy consumption. In [40], the authors proposed to deactivate the variable nodes that have converged to a strong belief after a few iterations. This condition is detected when the summation of all incoming messages surpasses a given threshold. The error-correction performance is affected by the value of this threshold. Furthermore, this criterion adds a compare operation per variable node.

The SCMS kernel enables a simple criterion to disable a check node on a given iteration. The concept of *erasing* messages avoids the propagation of unreliable messages along the code graph. If there are two or more erased messages per row (Step 3 in Algorithm 2) that particular check node is effectively rendered useless for the decoding task as all its output values would have magnitude zero. Although there is a similar argument for any MS-based kernel (two or more zero magnitude input messages) the SCMS kernel benefits from the fact that the minimum finders are not used. Detecting this condition allows to save the processing required along with the

[1]NMS with normalization factor of 0.8 and OMS with offset factor of 0.35.



(a) Average iterations



(b) Average energy

Figure 4.5: Decoding speed and energy consumption of computation kernels.

writeback of messages to the memories. Moreover, this criterion does not introduce any performance losses. Notice that this technique for power reduction can be viewed as an instance of the RTL-level optimization for low power known as *pre-computation* originally proposed in [41].

Figure 4.6 shows the percentage of disabled check nodes per iteration for two instances of a decoding task for a code in IEEE 802.11n (N=1944, R=1/2) and IEEE 802.16e (N=2304, R=1/2) at an SNR of $E_b/N_0 = 1dB$. In Chapter 6, we present a decoder implementation that with this optimization alone achieved an average of 10% of energy savings among various use cases.

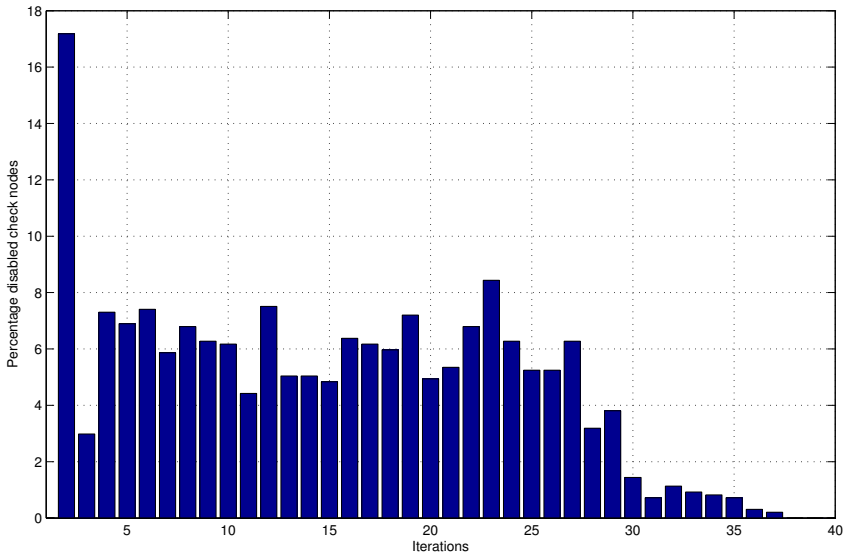
4.3.3 Stopping Criterion

Iteration control oversees that only the necessary number of iterations are executed for both successful and unsuccessful decoding. This indeed translates into energy savings since unnecessary decoder operation is avoided. In Chapter 7, we elaborate on a proposed control law for early stopping of the iterative decoding of LDPC codes aided by a decision metric extracted from the SCMS kernel.

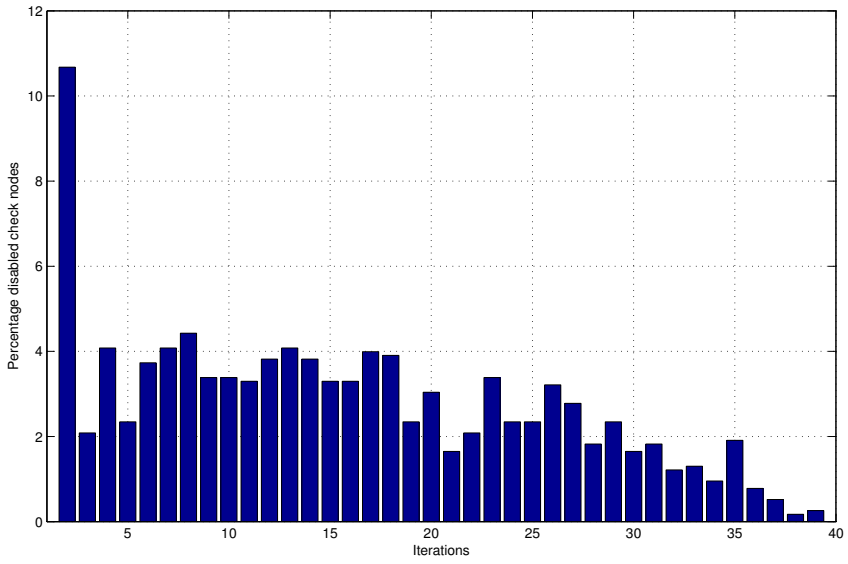
4.4 Decoders Comparison

We performed an estimation on the decoders implementation area and energy consumption of the individual components shown in the architecture of Figure 4.1, using three computation kernels for the test case of the code $N = 1944, R = 1/2$ in [5]. Beneš networks were used as shuffling units [42], and low-power dual-port RAMs for the required memories providing a bandwidth of 12 samples/cycle to three processing units in all decoders. Even though the design parameters of the decoders will become evident in Chapters 5 and 6 in terms of memory requirements we present in this section this comparison in order to provide a complete perspective on the kernels comparison.

In Figure 4.7, we show the implementation area and energy breakdown for decoders using the BCJR, SCMS and OMS (with offset value of 0.35) kernels with a message quantization of 6-bits in CMOS $65nm$ technology. The energy breakdown corresponds to the average energy expenditure per iteration for the previously mentioned test case at $E_b/N_0 = 1dB$. To have a fair comparison all decoders used the syndrome check as stopping criterion and all nodes were activated at run-time. On all decoders at least 70% of the energy is consumed on the memory subsystem, this shows the relevance of the optimization that can be exploited by the SCMS kernel mentioned



(a) 802.11n example



(b) 802.16e example

Figure 4.6: Inactive check nodes in SCMS decoding.

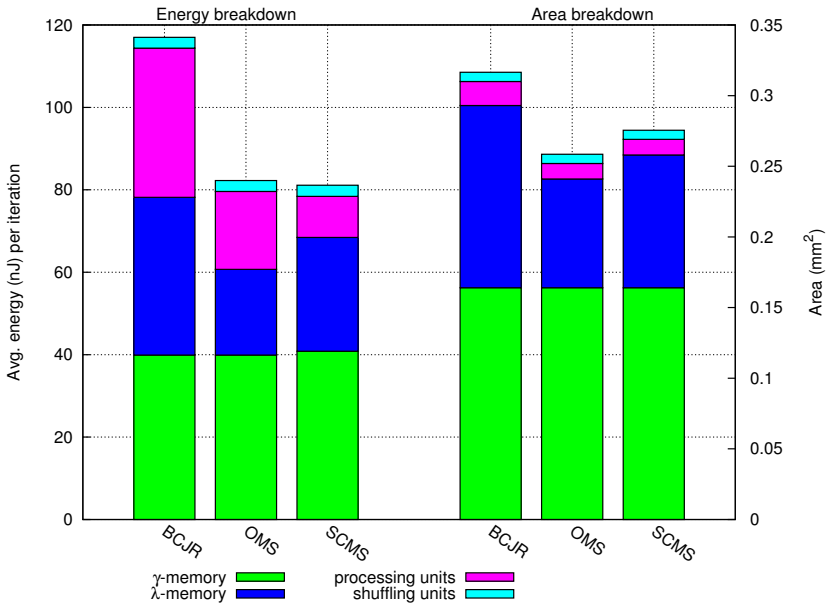


Figure 4.7: Comparison of decoders energy/area breakdown.

in Section 4.3.2 and the importance of the memory subsystem design presented in Chapter 5. The MS-based kernels consume less energy than the BCJR kernel on the extrinsic messages memories. Nonetheless, the SCMS kernel consumes more on this same memory when compared to the OMS kernel since it requires to store a reduced context from the previous iteration (this is further elaborated in Section 6.2.2). Furthermore, in this figure it can be observed how the SCMS kernel reduces the overall energy expenditure mainly due to the reduction in energy consumption at the level of the processing units.

In order to compare the decoders independently from use cases and iterations a frequently used figure of merit is the energy efficiency normalized to bits and iterations. In Table 4.1 we show the energy efficiency for the implemented decoders.

Table 4.1: Energy efficiency of SISO decoders.

Energy efficiency	BCJR	OMS	NMS	SCMS
$[pJ/bit/iteration]$	64.87	46.98	47.61	43.61

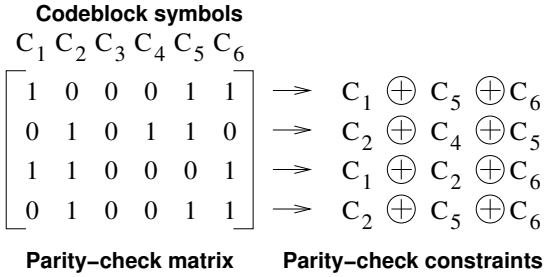


Figure 4.8: Example parity-check constraints.

4.5 Decoding Speedup

In this section, we present an optimization to the decoding algorithm for the purpose of speeding up the average decoding task. Syndrome check or verification is performed in order to confirm the validity of the obtained codeblock and hence decide whether to continue or halt the decoding process. This task corresponds to the evaluation of all the parity-check constraints imposed by the parity-check matrix. We propose to perform this check *on-the-fly* so that a partially unsatisfied parity-check constraint can disable a potential useless syndrome verification on the entire parity-check matrix. We identify as benefits from this technique the elimination of several hardware elements, a reduction on the overall task latency and an increase on system throughput.

Similar ideas of the proposed technique have been shown in [43] [44] [45] for the purpose of improving the energy efficiency of the decoder by reducing the average number of iterations. Nevertheless, these techniques are sub-optimal in the error-correcting sense as they introduce undetected codeblock errors. We analyze the performance of the proposed technique and elaborate on the recovery of the performance loss. Additionally, we quantify the impact of this proposal on a VLSI implementation.

4.5.1 On-the-fly Syndrome Check

A hard-decision vector on the posterior messages is required after each decoding iteration in order to calculate the syndrome. Syndrome calculation involves the product in equation (2.13), but this is equivalent to evaluate each parity-check constraint (each row in $\boldsymbol{H}$) with the corresponding code symbols. Figure 4.8 shows an example correspondence between the code symbols, the parity-check matrix and the parity-check constraints.

The parity-check constraints are of even parity and the $\oplus$ operation

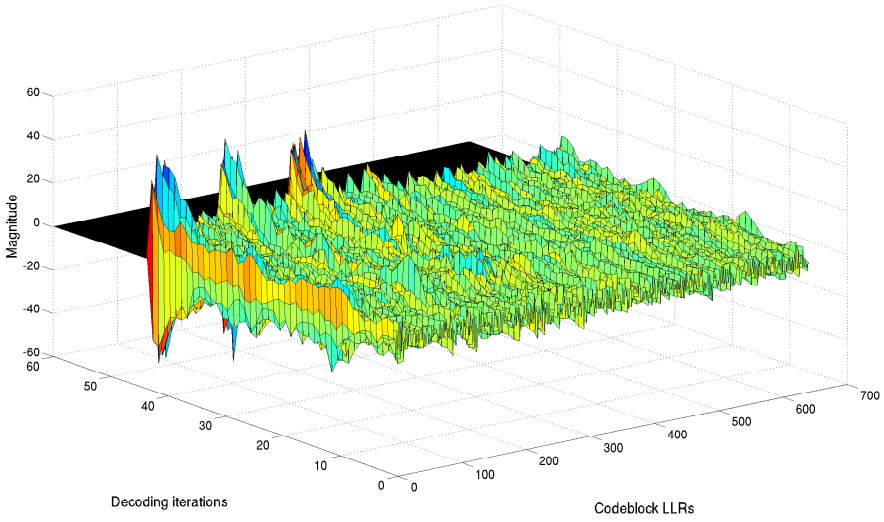


Figure 4.9: LLRs magnitude evolution as a function of decoding iterations.

corresponds to the modulo-2 addition. The arguments of each constraint correspond to the hard-decision of each LLR. A non-zero syndrome would correspond to any parity-check constraint resulting in odd parity. This condition suggests that a new decoding iteration must be triggered. The calculation of the syndrome in this way is synonymous to the verification of all parity-check constraints and indeed we refer to this as *syndrome check*.

The typical syndrome check requires a separate memory for the hard-decision symbols, a separate unit for the syndrome calculation (or verification of parity-check constraints) and indeed consumes time in which no decoding is involved. In this context, we use the word *typical* in two senses: one referring to the calculation of the syndrome with stable values and the other referring to the evaluation of the syndrome after the end of a decoding iteration.

In [46] [47] it has been shown how the LLR values evolve within the decoding process. Depending upon the operating SNR regime these values will initially fluctuate or enter right away a strictly monotonic behavior. Figure 4.9 shows the simulated LLRs magnitude evolution of an instance of decoding the QC-LDPC code defined in IEEE 802.11n for code length 648 and coding rate 1/2 over the AWGN channel at an SNR $E_b/N_0 = 1.5dB$.

Based upon the behavior of the LLRs we propose to perform the syn-

drome check *on-the-fly* in the following way: each parity-check constraint is verified right after each row is processed. Algorithm 3 outlines the proposed syndrome check within one decoding iteration for a parity-check matrix with M rows.

Algorithm 3 On-the-fly syndrome check

1. Decode each row i (or a plurality thereof for parallel architectures)
2. Evaluate each parity-check constraint PC_i by performing the $\oplus$ operation on the hard-decision values
3. Verification:

if $(PC_i = 1)$ **then**
  Disable further parity-checks verification
else
  if $(i = M)$ **then**
    Halt decoding: valid codeblock found
  end if
end if

Because of the structure of architecture-aware LDPC codes [13] and QC-LDPC codes it is possible to process several rows of $\boldsymbol{H}$ in parallel. For the proposed syndrome check there are two extreme cases regarding the latency between iterations. The worst-case scenario corresponds to the case when all individual parity-checks are satisfied but at least one from the last batch to process fails, in which case a new decoding iteration is triggered. The best-case scenario is when at least one of the first rows' parity-check fails, this disables further rows' parity-check verification and the next decoding iteration starts right after the end of the current one. The difference with the typical syndrome check is that it is always performed and it necessarily consumes more time as it involves the check of the entire $\boldsymbol{H}$. Figure 4.10 shows the timing visualization of these scenarios and the evident source for latency reduction of the decoding task.

The notion of *typical* syndrome check that we use might appear rather naive at first glance, but notice that among all the published works on decoder architectures the way the syndrome is verified is consistently neglected. It could be argued that the syndrome of an iteration can be verified concurrently with the decoding of the following iteration. This indeed would belittle our claim on task speedup but nevertheless the *on-the-fly* syndrome check hardware would still be of considerable lower complexity than said alternative mainly due to the lack of memories to save the hard decisions of the previous iteration.

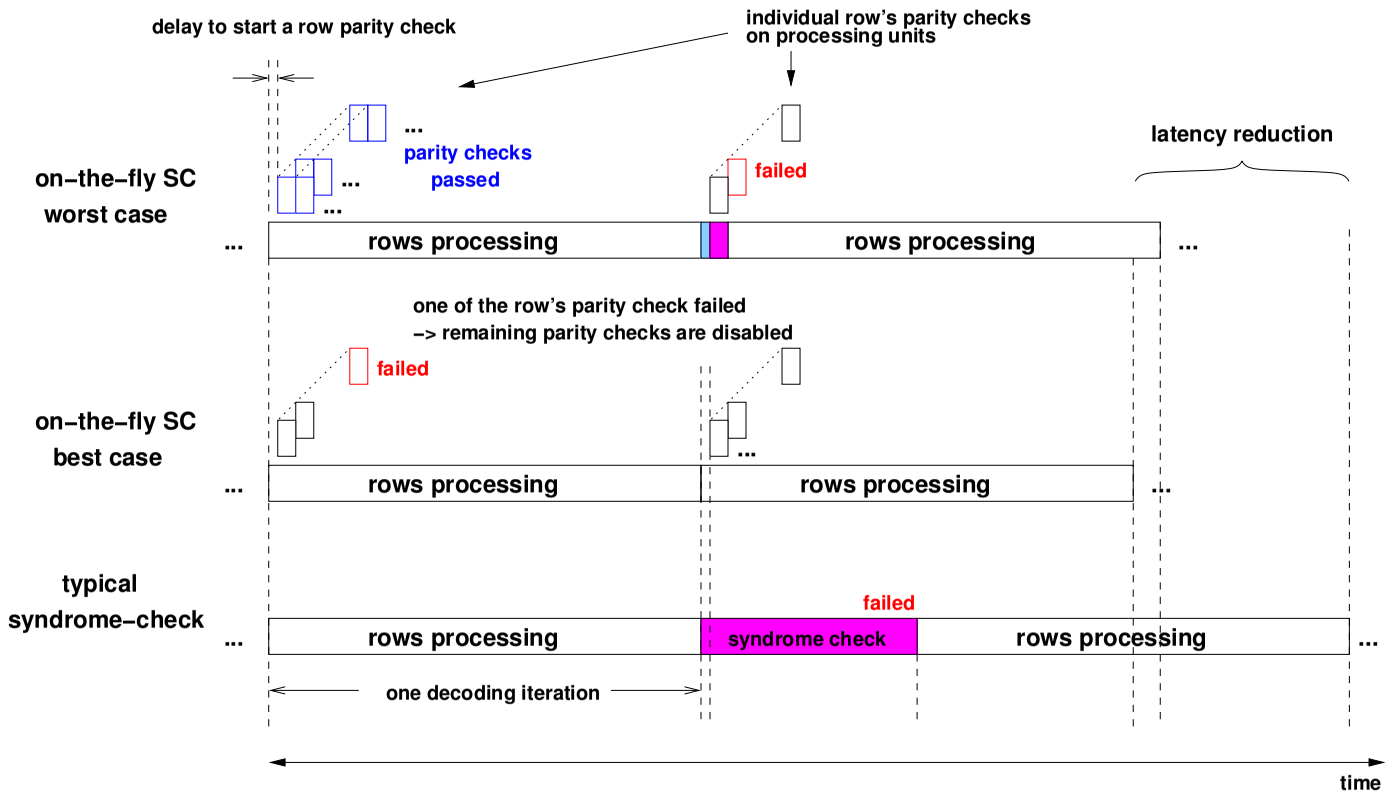


Figure 4.10: Timing visualization for two consecutive decoding iterations.

Table 4.2: Decision outcomes of the proposed syndrome check.

On-the-fly syndrome check	Typical syndrome check	Outcome decision
Pass	Pass	Hit
Pass	Fail	False Alarm
Fail	Pass	Miss
Fail	Fail	Hit

A closer examination of the proposed syndrome check reveals the possibility for special scenarios. Indeed, the proposed syndrome check does not correspond to equation (2.13) since the parity-check constraints are evaluated sequentially and their arguments (LLR sign) could change during the processing of the rows. Consequently, there is a possibility that the decision taken by the *on-the-fly* strategy might not be the correct one at the end of the decoding process. Table 4.2 shows the possible outcomes of the decision taken by the proposed strategy in contrast to the typical syndrome check. A *Pass* condition is synonymous to the condition $\boldsymbol{S} = \mathbf{0}$. A *false alarm* outcome corresponds to the case when all parity-check constraints were satisfied, indeed halting the decoding task during any iteration as a valid codeblock has been identified (when in fact a final typical syndrome check would fail). On the other hand, a *miss* outcome takes place when during the last iteration (maximum iteration limit) a single parity-check constraint fails rendering the codeblock as invalid (when in fact the typical syndrome check would pass). Both outcomes are the result of at least one LLR sign change right before the last row processing.

From this set of possible outcomes the probability P_H for the proposed syndrome check to be correct can be expressed by:

$$\begin{aligned} P_H &= 1 - (P_{FA} + P_M) \\ &= 1 - (P_P P_{CBE} + (1 - P_P)(1 - P_{CBE})) \,, \end{aligned} \tag{4.1}$$

where P_{FA} is the probability of a *false alarm*, P_M is the probability of a *miss* , P_{CBE} is the probability of a codeblock error and P_P is the probability of the proposed syndrome check to pass.

Based upon the analysis and observations in [46] [47] the LLRs monotonic behavior is guaranteed for the high SNR regime, in this regime the outcome decision would be a hit with probability 1. Nevertheless, as the

SNR degrades the inherent fluctuations of the LLRs at the beginning of the decoding process may cause the decision to be a miss or a false alarm with non-zero probability. In Figure 4.11, we show the outcome of the decoding of 10^5 codeblocks using the code length of 1944 and two code rates in IEEE 802.11n in order to observe the rate at which a miss and a false alarm may occur on the low SNR regime.

Even though the hit rate is shown to be empirically greater than a miss or a false alarm it is important to address the occurrence of such anomalies. A miss result would trigger an unnecessary retransmission in the presence of an ARQ protocol, while a false alarm result would introduce undetected codeblock errors. This indeed represents some concerns that must be analyzed on an application-specific context, as for example a wireless modem for [5] [6] is not likely to operate at such low SNR because of the required minimum packet-error rate performance.

The error-correction performance is affected by the false alarm outcomes. In Figure 4.12, we compare the simulated BER and FER of the typical syndrome check and the proposed method, this corresponds to the same simulation scenario from Figure 4.11a. The performance loss is evident, therefore we address the ways in which this situation can be circumvented.

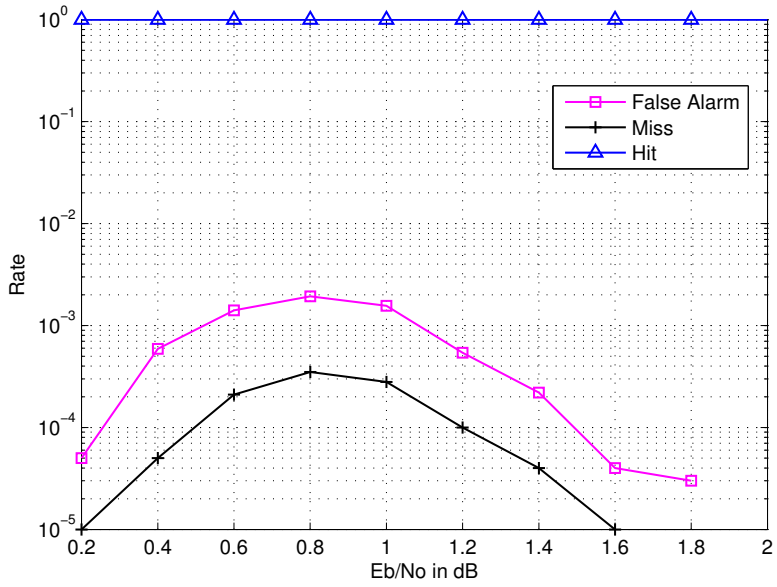
Detection of the miss and false alarm outcomes can be performed in two ways:

1. Validating the result provided by *on-the-fly* syndrome check by calculating the typical syndrome check.

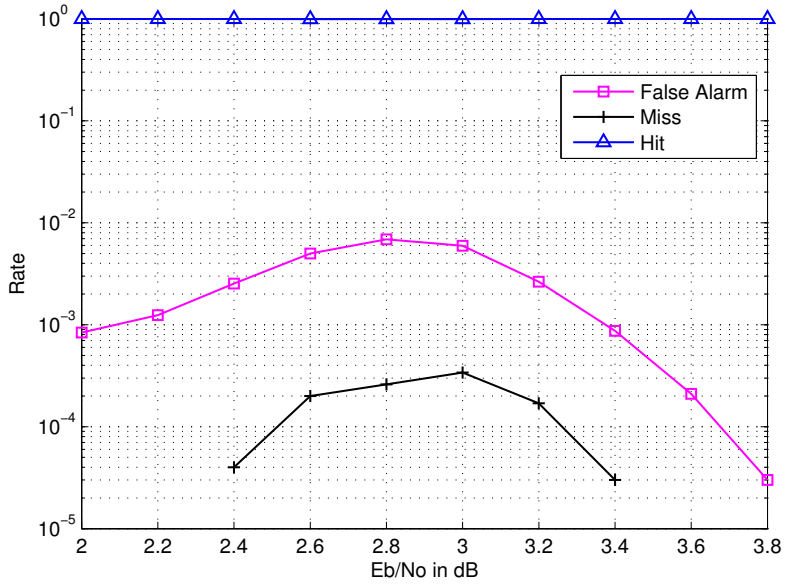
2. Allowing an outer coding scheme to detect such conditions, e.g., a cyclic redundancy check (CRC) that typically follows a codeblock decoding.

We propose to detect both miss and false alarm outcomes by validating the final calculated syndrome (in *on-the-fly* fashion) while executing the first iteration of the following codeblock. Figure 4.13 depicts both situations. The syndrome calculation for a codeblock CB[n] is validated during the first decoding iteration of codeblock CB[$n+1$]. In this way an ARQ protocol can react to a false alarm and also avoid an unnecessary retransmission under the presence of a miss outcome. The performance is fully recovered, shown in Figure 4.12 as *validated on-the-fly* syndrome check.

The implementation of the proposed syndrome check involves the addition of marginal components to each processing unit. Figure 4.14 shows the serial SISO processing unit from Figure 4.2 along with the added syndrome check capability. Synthesis results on CMOS 65nm technology showed that



(a) Rate 1/2



(b) Rate 5/6

Figure 4.11: Decision outcome rates from the proposed syndrome check for N=1944.

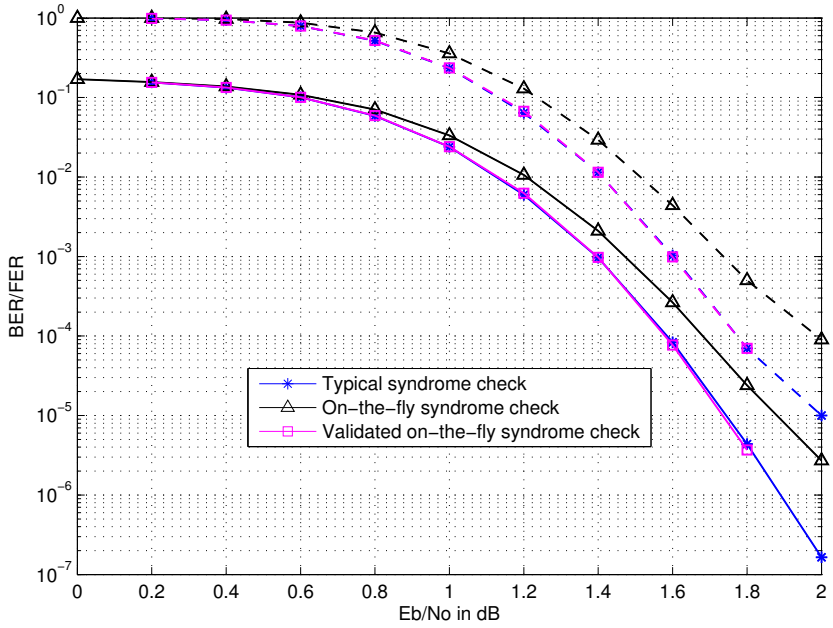


Figure 4.12: Error-correction performance of *on-the-fly* syndrome check.

the area overhead due to the syndrome check capability is only 0.65% for a BCJR-based processing unit (the SISO kernel is the modified BCJR algorithm described in [27] and Section 3.2.1).

The main benefit of the proposed syndrome check is the speedup of the overall decoding task. The processing latency per decoding iteration for P processing units is given in number of cycles by

$$\tau_c = m_b \times \frac{Z}{P} \times L_c \,, \tag{4.2}$$

where a quasi-cyclic LDPC code defines $\boldsymbol{H}$ as an array of m_b block-rows of Z rows. In this case P rows are processed concurrently. L_c is the number of cycles consumed during the decoding task where decoding and syndrome verification take place. This value depends upon the number of arguments to process per row, memory access latencies and syndrome verification duration. It is the latter time duration where our proposal exhibits advantages in terms of speedup. A reduction in the overall task latency improves as well the decoder throughput provided the frames input rate can guarantee a 100% decoder utilizaton. The throughput is given by

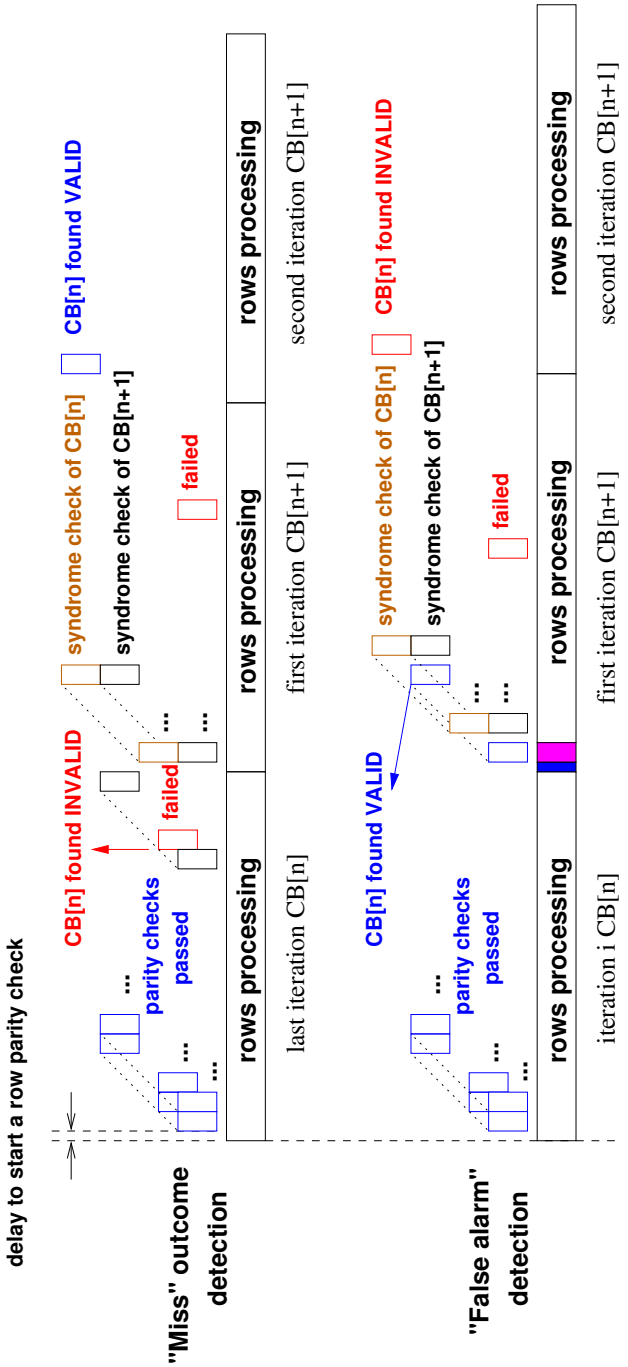


Figure 4.13: False alarm and miss outcomes detection example.

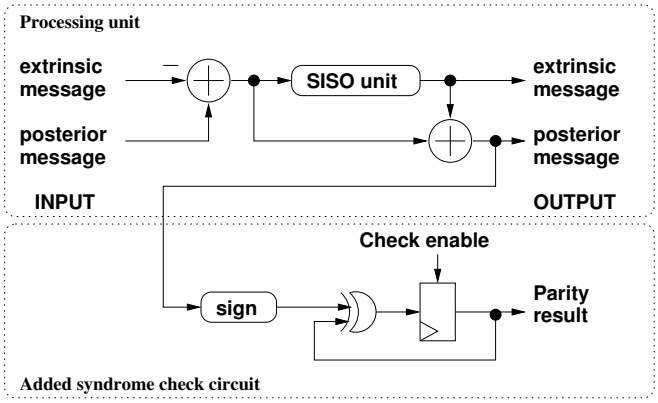


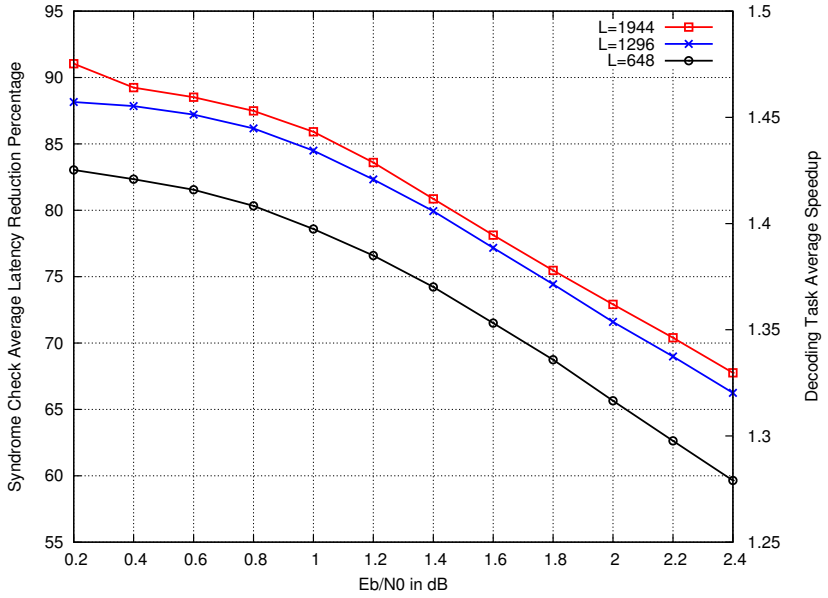
Figure 4.14: Processing unit with syndrome check option.

$$\begin{aligned} \Gamma &= \frac{N \times R \times f_{clk}}{I \times \tau_c} \\ &= \frac{(n_b - m_b) \times P}{I \times m_b \times L_c} \times f_{clk} \ , \end{aligned} \tag{4.3}$$

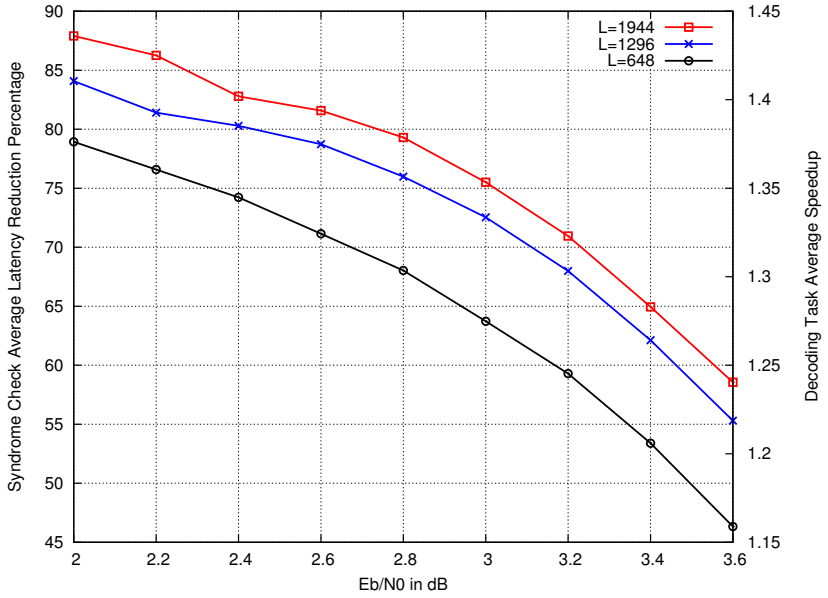
where I is the total number of iterations, R the coding rate, N the block length and f_{clk} the operating frequency.

The main benefit from the proposed strategy is the reduction in the time consumed during the syndrome check when the decoding process is far from reaching convergence. It could be argued that the syndrome check may very well be disabled during a preset number of initial iterations, but still this tuning must be done offline or shall depend upon extraneous variables as the SNR. Estimating these variables provides sensible overheads. Figure 4.15 shows the obtained average latency reduction of the syndrome check process compared to the typical one as a function of operating SNR. A total of three use cases with different code lengths are shown, for a code rate of 1/2 in Figure 4.15a and code rate 5/6 in Figure 4.15b. The low SNR region provides the best opportunities for syndrome check latency reduction since the LLRs fluctuate quite often in this region, i.e., a higher decoding effort renders useless the initial syndrome verification. Moreover, since in this region the latency is reduced around 90% the performance of the decoder could potentially be enhanced by increasing the number of iterations. Notice though that such optimization might be enabled only if channel state information is readily available.

Indeed, what this strategy is doing is speeding up a portion of the de-



(a) Rate 1/2



(b) Rate 5/6

Figure 4.15: Average latency reduction for the syndrome check process and overall decoding task speedup.

coding task. With the use of Amdahl's law [48] it is possible to observe the overall speedup of the decoding task based upon the obtained latency reduction of the syndrome check. The overall speedup is a function of the fraction $P_{enhanced}$ of the task that is enhanced and the speedup $S_{enhanced}$ of such fraction of the task:

$$S_{overall} = \frac{1}{(1 - P_{enhanced}) + \frac{P_{enhanced}}{S_{enhanced}}} \tag{4.4}$$

Figure 4.15 shows as well the average speedup obtained as a function of operating SNR for the same test cases, these results consider that the syndrome check process corresponds to 35% of the overall decoding task per iteration. Amdahl's law provides an upper bound for the achievable overall speedup, 1.53 for this setup. The average speedup is higher for the code rate 1/2 case since the parity-check matrix contains more rows than the code rate 5/6. For the former case the achieved speedup ranged from 84% to 96% of the maximum achievable bound, this corresponds to enhancing the decoder throughput by a factor of 1.28 and 1.48 respectively.

4.6 Conclusion

We presented top level views for the LDPC decoder and processing units used along our research. Low-complexity message computation kernels were studied at the level of energy efficiency performance and implementation area. The results of this study are used in Chapter 6 as we present an energy efficient decoder architecture. Finally, we presented an optimization that allows a speedup of the decoding task by partitioning the syndrome verification. Results from a decoder for the codes defined in IEEE 802.11n provided a speedup of up to a factor of 1.48 at a cost of less than 1% in logic area overhead for a $65nm$ CMOS process.

Chapter 5

Memory Architecture Design

Decoding of LDPC codes comprises a memory intensive kernel, this is a straight observation from the nature of the message-passing decoding algorithms and the relatively simple data-path involved in the processing nodes. The memory architecture for semi-parallel decoders represents both the bottleneck for performance and the main source of power consumption. Furthermore, memory partitioning is often used in semi-parallel architectures in order to provide the flexibility needed to support several codes. For such approach the alignment of data is important since it may cause conflicts and therefore introduce stall cycles during the decoding task execution. Works like [49] [50] have revealed the challenges of the memory subsystem design in terms of data alignment, partitioning and allocation. In [51] a memory optimization for FPGA implementations was proposed based upon vectorization and folding techniques in order to improve the decoder throughput. The authors in [37] proposed to reduce the power consumption of a decoder by reducing the number of memory accesses when altering the order of processing within the code parity-check matrix. The work in [52] addresses the issue of memory conflicts for pipelined architectures by reordering the parity-check matrix and by careful scheduling.

In the following, we show a design methodology for a flexible memory subsystem that reconciles design cost, energy consumption and required

latency. In addition, we utilize memory interleaving to avoid conflicts and investigate the conditions through which interleaving may allow a reduction in the memory read operation latency.

5.1 Memory Subsystem Overview

The structure of the memory subsystem is as well a consequence of the decoding algorithm. From Algorithm 1 in Chapter 3 it is clear that the following storage elements are required: posterior messages memory ($\boldsymbol{\gamma}$) and extrinsic messages memory ($\boldsymbol{\lambda}$). Extrinsic messages are generated after each decoding round per row, this is information that has no dependence on any a-priori knowledge of the same message. The posterior messages are generated by accumulating the information that is generated per iteration accounting for all rows and code symbols, these messages exhibit the reliability of each code symbol within the block. Both these types of messages arise from equation (2.12) in Chapter 2.

The structure within QC-LDPC codes impacts the posterior messages memory, this refers to the way in which the data is accessed and consequently how the data is allocated and partitioned. For the case of the extrinsic messages memory the use of MS-based kernels is advantageous since the size requirements are considerably less in comparison to the other processing kernels.

5.2 Posterior Messages Memory

This memory is initialized with the log-likelihood ratio values taken from the channel observations and updated after each row processing. This memory contains as many values as the codeword length. Hard decisions on the stored vector produce a valid codeword after successful decoding. One property provided by the structure of the codes is that when processing each block-row of $\boldsymbol{H}$ only a subset of the block-columns is used.

5.2.1 Data Mapping and Allocation

This memory is partitioned and dimensioned according to the structure of the codes. We define two types of data organization and placement based upon the structure of the QC-LDPC codes:

1. Micro-organization: This is provided by the shifted identity matrices in $\boldsymbol{H}$. Consecutive rows can be processed in parallel by distributing

adjacent posterior messages to parallel processing units. From this it follows that P messages may be grouped together to provide full bandwidth to P units.

2. Macro-organization: On each block-row processing up to c_i block-columns are used. From this observation we define the macro-organization as the mapping of block-columns to memory banks such that the required blocks are all accessible per read operation.

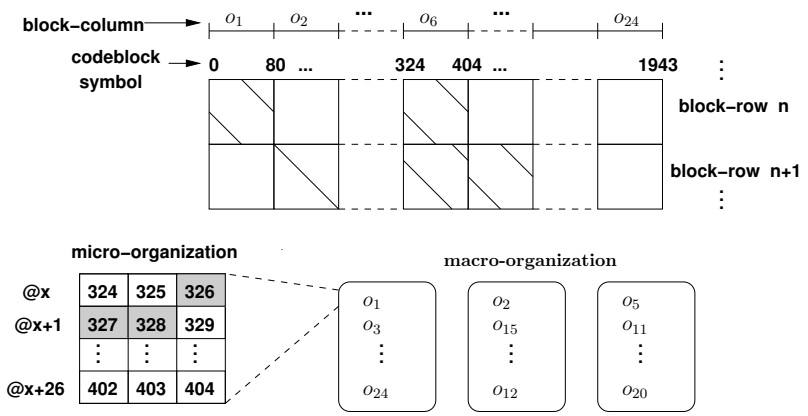


Figure 5.1: Posterior messages organization example.

Figure 5.1 shows these organization levels for $P = 3$ and $Z = 81$ for one block-column (symbols $\{324, 325, \ldots, 404\}$) of the use case $N = 1944$, $R = 1/2$ from IEEE 802.11n. The read messages are distributed by a shuffling unit to the processing units. The macro-organization corresponds to the mapping of the set of objects $\mathcal{O} = \{o_1, o_2, \ldots, o_{24}\}$ (block-columns) to memory banks.

The macro-organization constitutes a problem of allocating static objects (block-columns) to memory banks. As shown in [53], this problem is NP-complete and it maps naturally onto the well-known graph colouring problem. This introduces another dimension to the design criteria to organize the memory architecture since not only does the bandwidth have to be achieved but the data allocation required has to be possible. This situation is formalized as follows: a set of static objects $\mathcal{O} = \{o_1, o_2, \ldots, o_{24}\}$ is to be mapped to a set of memory banks B. Objects in $\mathcal{O}$ *conflict* when they need to be accessed at the same time. The memory allocation problem is solved by finding an allocation function $alloc : \mathcal{O} \rightarrow B$ that avoids as much conflicts as possible. A conflict represents a loss in execution speed, introducing stall cycles and bubbles into a pipelined implementation.

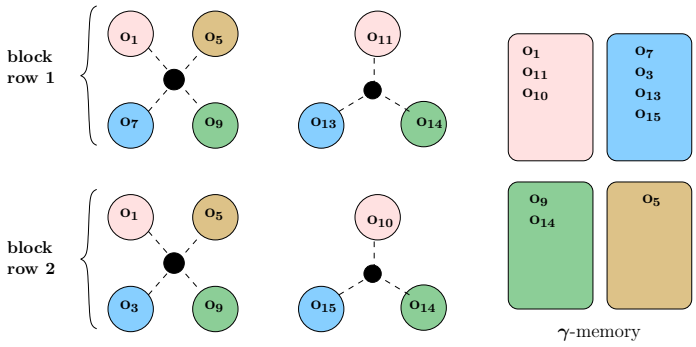


Figure 5.2: Macro-organization by conflict graphs.

Throughout this work we refer to a *use case* as a particular code to be supported by the decoder, these use cases are specified by a coding rate and a block length. For each use case of decoder operation we define a set of *conflict graphs* $G = (V, E)$, where V is the set of vertices representing the block-columns used on a given block-row and E is the set of edges between the nodes of V that incur in a conflict. The chromatic number $\chi(G)$ of a conflict graph determines the minimum number of colours needed such that all vertices of the graph are coloured and adjacent vertices do not share the same colour. In other words, the chromatic number determines the minimum number of memories needed to provide the required bandwidth. There should be as many conflict graphs as block-rows in $\boldsymbol{H}$ but the bandwidth requirement and the degree of the rows along with the cycle budget should be used to formulate the mapping problem. For example, in Figure 5.2 we show conflict graphs and colouring for the first block-rows of the use case $R = 1/2$ and $N = 1944$ from the codes in IEEE 802.11n for a memory bandwidth of 12 samples/cycle. In this example, 3 rows are processed every 2 cycles, this suggests that as many as 4 block-columns are accessed per read access. This produces 2 conflict graphs per block-row, where the chromatic number is bounded, $\chi(G) \leq 4$. In this way the partition of the posterior messages memory consists of $B = 4$ memory banks that provide a bandwidth of 12 samples/cycle to $P = 3$ processing units.

This mapping is performed offline and is solvable by the well-known graph colouring techniques. By analyzing the parity-check matrix the conflict graphs can be constructed and coloured such that the block-columns are allocated to the memory banks and the number of conflicts is reduced or even avoided. Notice that the macro-organization is required only when several block-columns are accessed at the same time. From this it follows that for serial processing units this level of organization would not be exploited.

5.2.2 Design Space Exploration

One important parameter for the cost of a memory architecture is the area per bit. By reducing the number of banks also interconnection links and control circuitry are reduced. This impacts not only cost but also power consumption, so it is fundamental to perform a study on the memory architecture candidates to achieve an efficient and economical design. From the area/bit point of view there are two extreme cases: using one single large memory or a maximum number of memories that guarantees the required number of parallel accesses. Compared to small memories, large memories consume more energy per access because of the longer word and bit lines. But distributing the data along several small memories leads also to a relatively high energy consumption due to the added interconnection lines and decoding circuitry overhead.

It is known that partitioning a memory into several banks in order to resolve conflicts or provide higher access bandwidth has costs. In Figure 5.3, we show the characteristics of a dual-port memory (clocked at 400MHz with 1.1V) of size [1024x64] cut into several banks, this corresponds to a CMOS technology of $40nm$. Area, leakage (static) power and average dynamic power per memory access operation[1]are shown. As this figure shows, it is relevant to consider an exploration of the possibilities to divide a physical memory space to assess both advantages and disadvantages. We prioritize then to minimize the number of memory cuts while providing the required bandwidth.

We performed an exploration of several memory partitions where the data mapping has been shown to be possible for a multi-mode decoder for the codes in IEEE 802.11n and IEEE 802.16e. The target technology is a $65nm$ CMOS process with $V_{dd} = 1.32V$. The exploration is carried out with a memory design tool provided by Intel Mobile Communications. We explored several configurations for bandwidths of 12 and 32 samples/cycle using low power dual-port SRAMs clocked at 648MHz and 243MHz respectively. The corner use cases explored are characterized by the number of memory accesses, which depends upon the degree of the block-rows. For example the case $R = 1/2$ and $N = 2304$ in 802.16e requires 76 accesses per decoding iteration to the 24 blocks of 96 samples each stored in this memory. Figure 5.4 shows the average energy consumed per iteration in this memory for several partitions. For both use cases shown we can observe that the configurations for 32 samples/cycle are more energy efficient than the 12 samples/cycle ones, but more costly in terms of area/bit.

[1]Average power on write and read operations with all data and addresses switching.

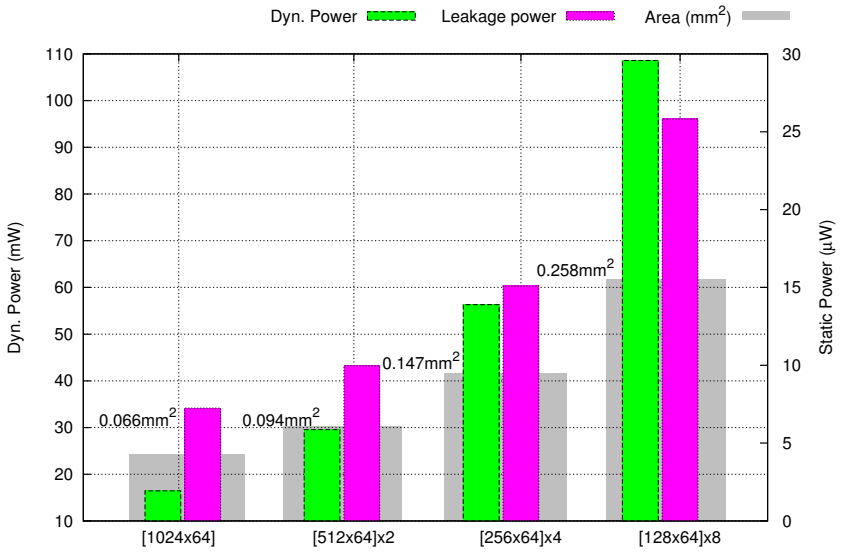


Figure 5.3: Area and power overhead associated with memory partitioning.

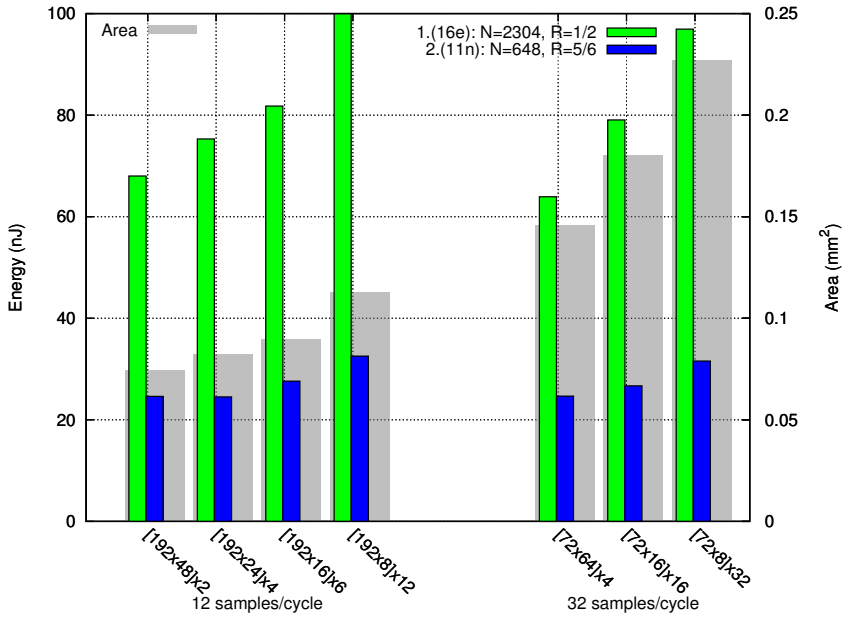


Figure 5.4: γ-memory exploration.

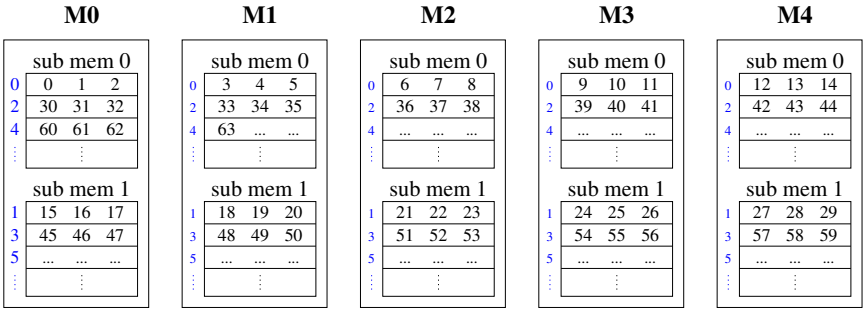


Figure 5.5: Interleaved memory with m=5, s=2 and k=3.

5.2.3 Memory Interleaving

In this section, we refer again to Figure 5.1 where the two levels of memory organization are shown. For this example, it is required to provide a full bandwidth of 3 samples per cycle to all P units when reading a block-matrix or circulant. As can be seen, this is only possible if there is a full alignment between the samples placement and the shift value of the block-matrix. For a shift value of 2 in the given figure, fetching samples $\{326, 327, 328\}$ would require 2 read cycles; whereas for a shift of 0 samples $\{324, 325, 326\}$ would require 1 read cycle. Due to the structure in $\boldsymbol{H}$ each block-matrix is processed in $\lceil \frac{Z}{P} \rceil$ *processing steps*. Each of these steps ideally requires as many read operations with a bandwidth of P samples each. In order to ensure the latter we propose to further subdivide the memory space by interleaving the memories both horizontally and vertically. By horizontal interleaving we mean distributing each object $o_i \in \mathcal{O}$ among m banks, and by vertical interleaving we further partition each bank into s sub-banks. Each memory word has a width of $k = P/m$. Figure 5.5 shows an example of an interleaved memory architecture with $m = 5$, $k = 3$ and $s = 2$.

There are two different kinds of conflicts when accessing the posterior messages memory. The first can occur during the initial block-matrix processing step. In the following, $shift$ refers to the shift value defined for each circulant in a structured parity-check matrix. If $shift \bmod k = 0$ reading starts at the leftmost code symbol within a memory word and then continues rightwards. As P/m is an integer, reading stops at the rightmost code symbol after reading m memory words from m memory banks. However, if $shift \bmod k \neq 0$ reading does not start at a leftmost code symbol and therefore $m + 1$ read operations are required to read all P code symbols.

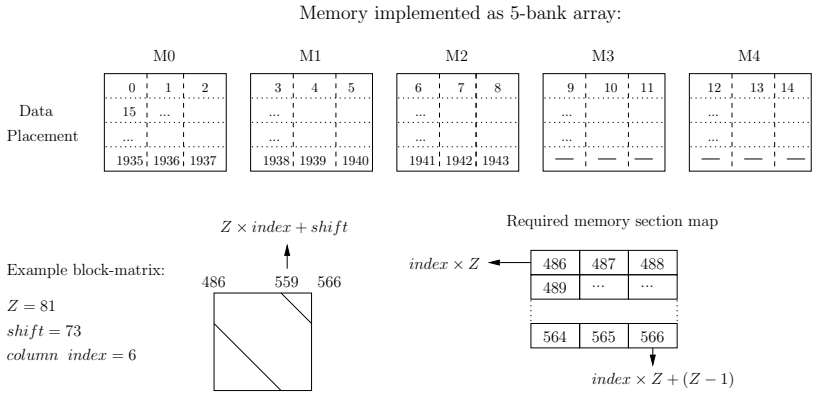


Figure 5.6: Posterior messages memory map and data allocation.

This causes a conflict when re-accessing the memory bank holding also the leftmost symbol. The symbols from the $(m+1)$th read operation that were not used in the current processing step can be stored in a register bank or FIFO for later usage, similar to the proposals in [49] [52], so that this kind of conflict does not occur again. Still, it is unavoidable in the first processing step. The second type of conflict occurs at a block-matrix *line break*, the point in which the matrix wraps around due to the shift value. There is such line break once per every block-matrix with $shift \neq 0$.

In Figure 5.6, we show an exemplary memory partition and data allocation (each sample number corresponds to a code symbol) for a decoder with $P = 15$, $m = 5$ and $k = 3$ with no further sub-banking ($s = 1$). The memory map refers to a virtual ordering of all the samples depending upon the use case of code length N. We show as well an example block-matrix to be read from the memory. In this case 81 samples are required by 15 processing units, ideally 15 samples should be read at each cycle. With the shift value of 73 the matrix line break presents a conflict of the second type previously mentioned. Figure 5.7 shows that when reading the first 15 samples $\{559-566, 486-492\}$ the memory banks M2 and M3 must be accessed twice. Notice though, that bank addresses before the matrix line break are odd while the ones after are even. If memory banks were sub-divided with $s = 2$ (as shown in Figure 5.5) memory accesses would be conflict-free for this example.

Conflicts occur either in the first processing step or at a matrix line break. The first conflict type is resolved by using an arbitrary number of memory banks with interleaving degree $s \geq 2$. This is justified by the data allocation

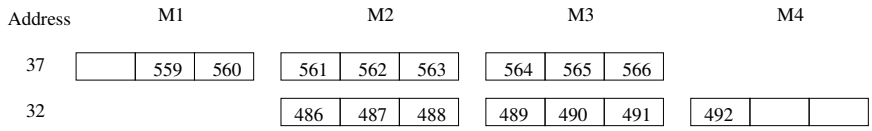


Figure 5.7: Access conflict example.

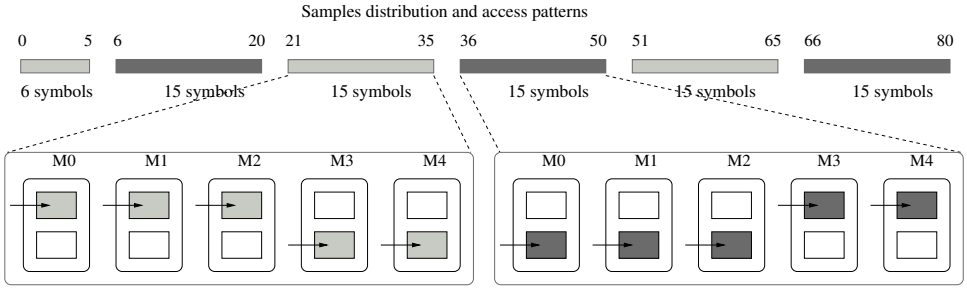


Figure 5.8: Access patterns example.

among the memory banks. During the processing step of a block-row with $shift \bmod k \neq 0$ the $(m+1)$th memory access re-accesses a certain memory bank, but the data request goes to a bank address which is incremented by one compared to the previous access and therefore is located in a different sub-bank.

To explain the mitigation of matrix line break conflicts the concept of *access patterns* is introduced. The rightmost m memory accesses delivering the last P symbols before a matrix line break will access certain sub-banks of the m memory banks. The subset of sub-banks accessed is called an *access pattern*. Due to the data allocation the m memory accesses to the left will use a different sub-set of sub-banks forming another access pattern and so forth, until s access patterns have been performed and the first access pattern occurs again (this happens every $s \times P$ symbols). Figure 5.8 shows an example of access patterns for the configuration shown in Figure 5.5.

If Z is a multiple of P a memory access is always conflict-free provided $shift \bmod k = 0$. Because of the data allocation, going back a multiple of P code symbols at a matrix line break will access the same memory bank as when going one symbol forward, which is conflict-free. If Z is no multiple of P the leftmost access pattern has a reduced width of $r = Z \bmod P$, whose size has an impact on memory conflicts.

With s alternating access patterns in a block-matrix the first condition for a conflict-free memory access is that there are different access patterns at

the beginning and the end of a block-matrix. This translates to the following condition for s for a given Z and P:

$$\left\lceil \frac{Z}{P} \right\rceil \bmod s \neq 1 \tag{5.1}$$

However, conflicts can still occur if reading from the beginning of the block-matrix comes across the same access pattern that was accessed before the matrix line break (access pattern overlap). This is indicated by:

$$\left\lceil \frac{Z}{P} \right\rceil \bmod s = s - 1 \tag{5.2}$$

Figure 5.9a illustrates the general situation where reading starts from an arbitrary position a within an access pattern. The symbols numbering is relative to the start of an access pattern before the matrix line break. The shaded areas correspond to two different access patterns as shown in Figure 5.8. The last symbol to be accessed during the current step has the index $a + P - 1$. The memory bank accessed for a is identified by $\left\lfloor \frac{a}{k} \right\rfloor$. To avoid that the same bank is accessed twice the last bank that may be accessed after the line break should have the index $\left\lfloor \frac{a}{k} \right\rfloor - 1$. This is to say that the re-accessed region must not reach the scope of the bank accessed prior to the line break. The following condition ensures the latter observation:

$$a + P - 1 \quad < \quad P + r + \left(\frac{a}{k} - 1\right) \cdot k \tag{5.3}$$

Reducing such expression to r and k and substituting their values with the architecture parameters leads to:

$$\frac{P}{m} \leq Z \bmod P \tag{5.4}$$

This suggests that the P symbols of the re-accessed pattern must be divided into sufficiently small partitions by m.

In Figure 5.9b, an example for a conflict-free setup is shown. M4 has already been accessed before the line break and the value of k guarantees that the same access pattern does not access twice the same bank M4. On the other hand, in Figure 5.9c a conflict occurs at M2 since P is not partitioned sufficiently by m to satisfy (5.4).

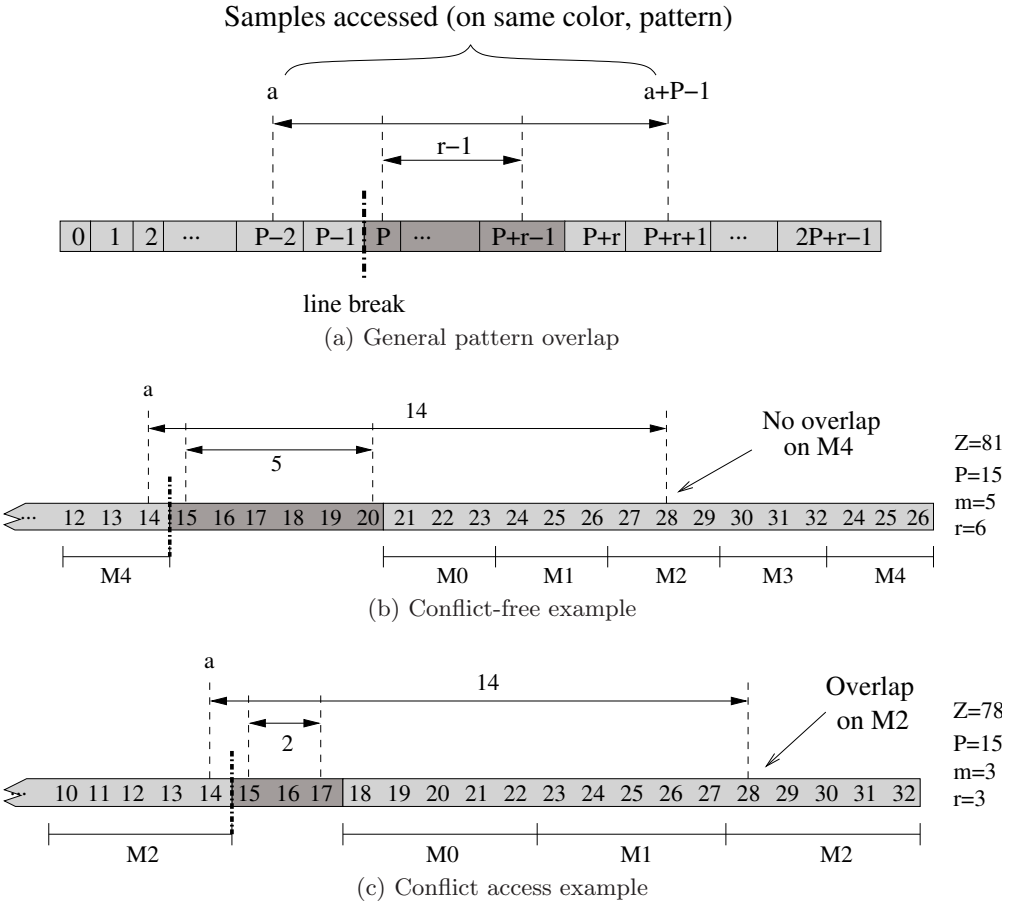


Figure 5.9: Access pattern overlap situations.

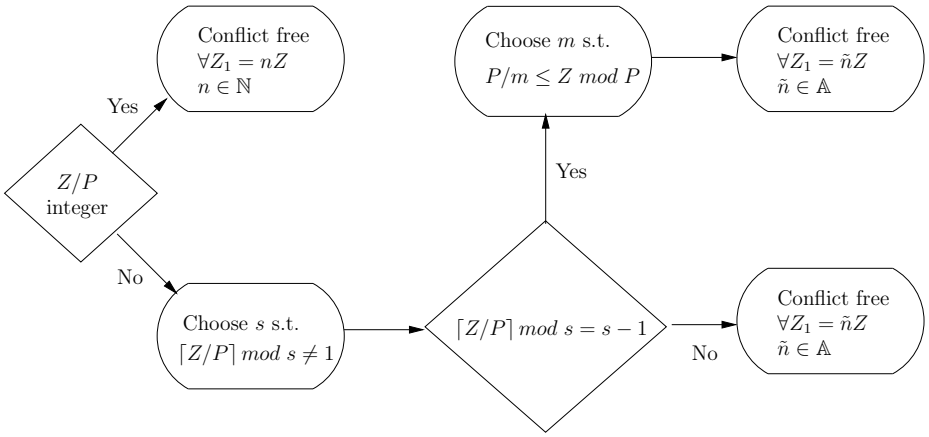


Figure 5.10: Conflict verification flow.

The memory subsystem for these decoders should be flexible in order to support various values for Z. If Z/P is an integer all expansion factors $Z_1 = nZ$ with $n \in \mathbb{N}$ are conflict-free. Otherwise, a sub-set $\mathbb{A} \subseteq \mathbb{N}$ has to be found such that for all $\tilde{n} \in \mathbb{A}$ the architecture is still conflict-free for expansion factors $Z_1 = \tilde{n}Z$. The following shows the conditions all elements $\tilde{n} \in \mathbb{A}$ have to fulfill.

It can be shown that for a ceil operation the following holds:

$$\lceil \tilde{n} \cdot x \rceil = \tilde{n} \cdot \lceil x \rceil - \lfloor \tilde{n} \left(\lceil x \rceil - x \right) \rfloor \tag{5.5}$$

Using $x = Z/P$ in (5.5) in order to rewrite (5.1) leads to this condition for $\tilde{n}$:

$$\left\lfloor \tilde{n} \left(\left\lceil \frac{Z}{P} \right\rceil - \frac{Z}{P} \right) \right\rfloor \bmod s \neq 1 \tag{5.6}$$

Furthermore, all elements in $\mathbb{A}$ have to fulfill (5.4) for $Z_1 = \tilde{n}Z$. In Figure 5.10, we generalize these conditions in order to provide a flow to verify that a design is able to provide conflict-free access.

From the previous discussion it is evident that an interleaved memory architecture requires a fewer number of cycles to provide the same bandwidth than its non-interleaved counterpart. This corresponds to a potential speedup when applying the interleaving previously described, or the possibility to reduce the frequency of operation for the interleaved design.

In Figure 5.11, we show the frequencies of operation for a non-interleaved and an interleaved design for several use cases (expansion factors Z and co-

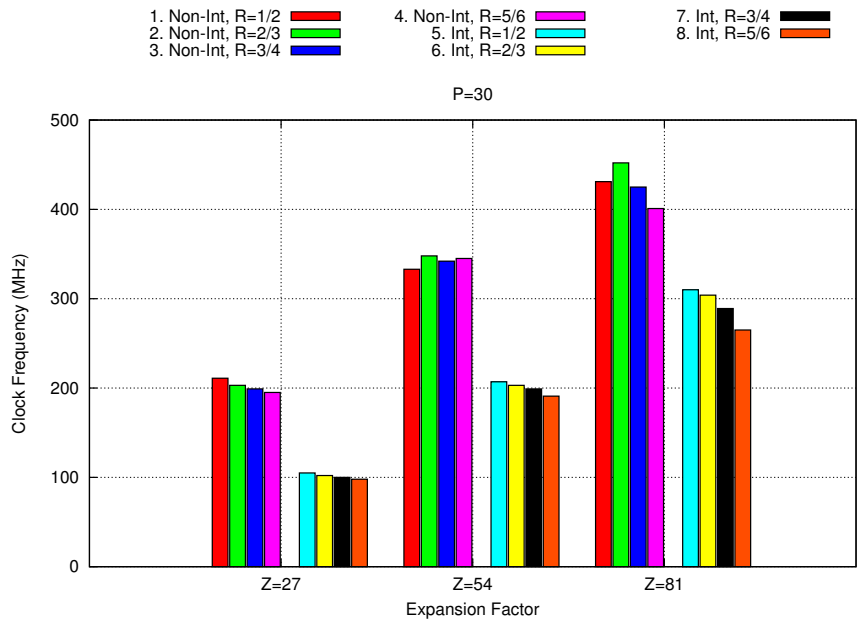


Figure 5.11: Clock frequencies for interleaved/non-interleaved architectures.

ding rates R) from the codes in 802.11n for a decoder with $P = 30$ processing units on a $65nm$ technology using dual-port SRAMs. The interleaved design has the configuration $m = 3$ and $s = 3$. By using interleaving a reduction of the clock frequency in the range of 25%-50% was obtained depending upon the use case. Notice that alternatively the throughput can be enhanced by this same range if the non-interleaved frequency is maintained.

5.3 Extrinsic Messages Memory

This memory stores the messages that are obtained after each decoding round per row, and are used as input prior messages to decode subsequent rows. The dimension of this memory corresponds to the number of non-zero elements in $\boldsymbol{H}$, or equivalently to the number of edges in the code graph. While this is the largest of the two memories it is easier to design. The data allocation is straightforward from Algorithm 1: the processing of each row uses always the same values from the same locations. The data does not have to be neither shifted nor distributed to different processing units if each row decoding is bound to a specific processing unit.

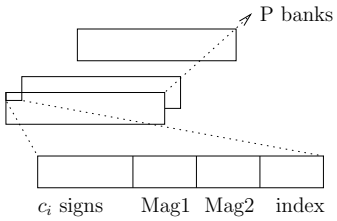


Figure 5.12: Extrinsic memory MS-based structure.

5.3.1 MS-based Kernels Data Allocation

The choice of an MS-based SISO kernel has further consequences on this memory. These kernels take c_i (row degree) inputs and produce c_i outputs, with each having a sign and one of two possible magnitudes: the first or second minimum value out of the c_i inputs. So instead of storing c_i messages it is only necessary to store c_i signs, two magnitudes and the index where the first minimum is to be found. Taking as a base the codes in IEEE 802.11n and IEEE 802.16e this represents a reduction in memory size of up to 65% with an 8-bits message quantization. Figure 5.12 shows the storage for these values.

5.3.2 Design Space Exploration

Just like the case for the posterior messages memory we perform an exploration in order to efficiently choose an optimum memory partition for the extrinsic messages memory in the sense of minimizing energy consumption per use case and implementation area. The exploration shown in this section follows the same case for a multi-mode decoder for IEEE802.11n/16e applications on CMOS 65nm technology.

The exploration of the extrinsic messages memory involves inquiring about the possibility of partitioning each of the memories that are bound to each processing unit. This memory stores the compressed row vector shown in Figure 5.12. Considering both standardized codes that are supported a total of 31 use cases originate. The corner cases are defined by the number of edges in the code graph and the degree of the rows. These requirements are considered in detail in Chapter 6. In Figure 5.13, we explored the area and average energy per iteration consumed in the extrinsic messages memory for the two corner cases (both from 802.16e) and two intermediate cases with several memory configurations using low power dual-port SRAMs clocked at 648MHz. For the least demanding cases the unused partitions are powered off. This simple power management strategy allows to observe the energy

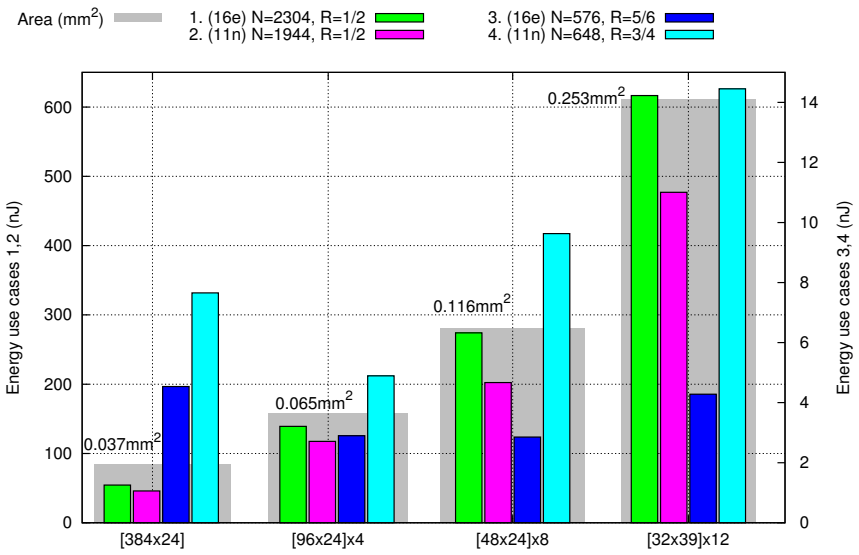


Figure 5.13: λ-memory exploration.

cost behavior according to the memory partitions where an optimal configuration exists for a particular use case. Once an optimal choice is taken more elaborated power management techniques (refer to Chapter 7) can further improve the energy efficiency.

5.3.3 SCMS Kernel Data Allocation

In the previous section, we have seen how the MS-based kernels have an advantage from the extrinsic messages perspective as this memory size can be reduced. Nonetheless, the SCMS kernel requires an additional storage flag that indicates whether a prior message was deleted or not during the previous decoding round (erasure flag) or the previous sign of the prior message. This is observed in Algorithm 2 in Chapter 3. In Chapter 6, we elaborate on the overheads that arise due to the implementation of the SCMS kernel.

5.4 Conclusion

The memory subsystem accounts for the majority of implementation area and power consumption within a VLSI decoder. Designing this subsystem carefully is hence of great importance for achieving economical and efficient

decoders. The design of this subsystem entails the allocation and partitioning of data as a function of the required data distribution and data bandwidth requirement. The memory subsystem provides storage for two types of messages: posterior and extrinsic messages. The structure embedded in QC-LDPC codes enables two levels of memory organization for the posterior messages memory. The allocation of the messages within the physical memory space has consequences on the routing structures used to distribute the messages to the processing units. Furthermore, the partitioning of the data and the interleaving of memory banks can be designed in order to provide the required bandwidth within a minimal number of access cycles. MS-based kernels reduce considerably the size requirements for storing the extrinsic messages. For flexible decoders it is of interest to explore the possible memory architecture configurations in order to assess the impact on implementation area and energy consumption as a function of the use cases to be supported.

Chapter 6

Implementation of a Multi-mode LDPC Decoder

Numerous works have proposed semi-parallel architectures that are optimized in different ways in order to conceive energy and area efficient designs. For example, for multi-mode decoders, authors in [34] proposed a reconfigurable network for message distribution and authors in [35] relied upon an early termination scheme and memory banking to reduce power consumption. The work presented in [36] divides the decoder operation in several tasks and arranges their order such that the challenges due to flexibility may be solved. Similarly, in [37] the authors proposed a memory-bypassing scheme where the order of the processing layers is altered such that energy consumption is minimized. The work in [38] addresses as well the topic of task rearrangement and optimizes the message quantization as well as their storage scheme.

We present an implementation of an energy efficient multi-mode LDPC decoder for the codes in IEEE 802.11n/16e driven by the SCMS message computation kernel. A flexible memory subsystem is outlined following the design issues presented in Chapter 5. The used data allocation and partition enables a low complexity shuffling network. A VLSI implementation on CMOS $65nm$ technology resulted in a $0.95mm^2$ decoder with an energy efficiency of $47pJ/bit/iter$. Comparisons among representative work reveal the benefits in energy efficiency for the proposed architecture.

6.1 System Design

The multi-mode decoder design must consider the timing deadlines imposed by the targeted standards. The system must be dimensioned in a way that guarantees these deadlines and offers the necessary flexibility to support a variety of use cases. The goal of our approach is to meet these requirements with a minimal implementation area and an energy efficient design.

6.1.1 Decoder Requirements

The starting point for designing the multi-mode decoder is to analyze the system requirements along with the characteristics of the codes. The IEEE 802.11n [5] and 802.16e [6] standards define different parity-check matrices based upon systematic linear block codes for different use cases; a use case is defined by a codeblock length and a coding rate. In 802.11n there are a total of 12 matrices for 12 use cases and in 802.16e there are 6 matrices for 19 use cases. The structure of each of these matrices follows the one shown in Figure 3.1a. $\boldsymbol{H}_{M\times N}$ consists of an array $m_b \times n_b$ of $Z \times Z$ blocks where for both standards $n_b = 24$ and $m_b \in \{4, 6, 8, 12\}$. In [5] [6] the parity-check matrix is generated from a base model matrix $\boldsymbol{H}_{bm}$ that indicates the position of the non-zero blocks and the corresponding circular shift value s. The shift values for the matrices in 802.11n are taken directly from $\boldsymbol{H}_{bm}$ while the shift values for 802.16e follow a precalculation dependent upon the use case. For all cases the shift value $s \leq Z$ where $Z_{11n} \in \{27, 54, 81\}$ and $Z_{16e} \in \{24, 28, 32, \ldots, 96\}$.

The decoding task in both standards exhibits demanding timing requirements. Table 6.1 shows the main characteristics defined in the standards for both the most and least demanding use cases. The decoding latency corresponds to the maximum permissible time in order to complete the decoding task such that the baseband processing respects the time allotted for generating negative or positive acknowledgement messages.

6.1.2 Decoder Overview

The decoder top level view corresponds to the one already presented in Section 4.2. The use case with the lowest code rate is used to dimension the decoder since it contains the highest workload (dimensions of $\boldsymbol{H}$). We refer to *dimensioning* to the selection of P and the partitioning of the memory subsystem. Figure 6.1 shows the number of serial processing units required to complete the decoding task with a maximum number of $I = 8$ iterations

Table 6.1: Specifications for supported LDPC codes.

Use case	Most demanding		Least demanding	
Standard 802.xx	11n	16e	11n	16e
Codeblock length	1944	2304	648	576
Coding rate	1/2		5/6	
$\boldsymbol{H}$ dimensions ($m_b \times n_b$)	12x24	12x24	4x24	4x24
Z value	81	96	27	24
Latency	8μs	0.25ms	8μs	0.25ms
Row degree c_i	7,8	6,7	22	20
Rows in $\boldsymbol{H}$	972	1152	108	96
Graph edges	6966	7296	2376	1920

at a given operating frequency for several use cases. We show the use cases with $Z = 81$ from the 802.11n standard and the use case $Z = 96$ with $R = 1/2$ from 802.16e. Serial processing units offer the most flexibility since the number of inputs (c_i) to process vary according to the use case and mode of operation of the decoder. The processing units used for this figure are rather naive in the architectural sense, in the sequel we present an optimized decoder that achieved a lower frequency than the one suggested from the figure.

The timing requirement for the decoding task with a deadline d is given as a function of the code parameters by:

$$\frac{m_b \times \frac{Z}{P} \times L_c \times I}{f_{clk}} \leq d \, . \tag{6.1}$$

6.1.3 Message Quantization

The finite quantization of messages impacts the error-correction performance of the system as well as the implementation area and power consumption. Following the quantization scheme outlined in [54], we use a uniform quantization (q, Δ) of q bits (sign and magnitude) with a quantization step of Δ to assess the performance loss. The dynamic range of the messages is very large as the posterior messages would ideally exhibit an increasing monotonic behavior. A finite quantization on the messages would imply a message clipping that affects the propagation of reliable messages along the code graph. This impacts error-correction performance and the convergence rate of the code.

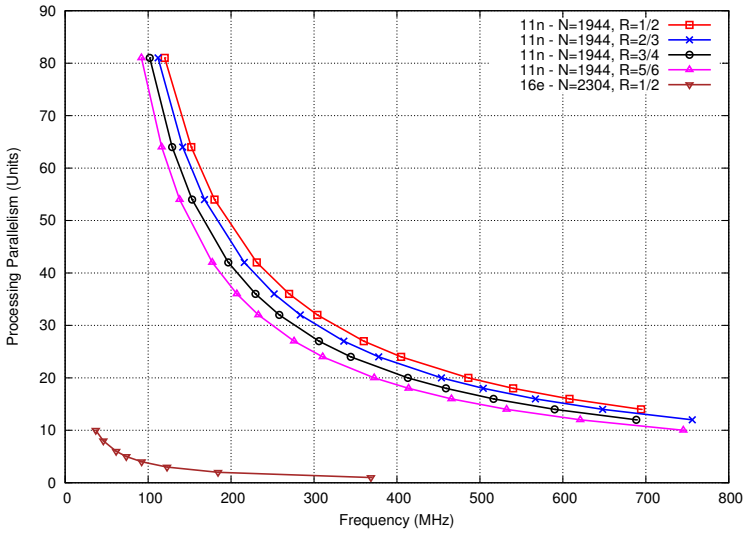
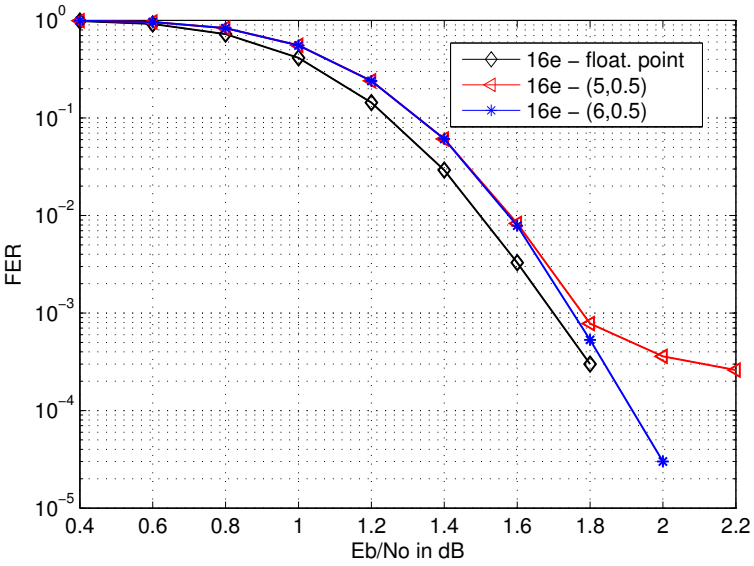


Figure 6.1: Required number of processing units and operating frequency per use case.

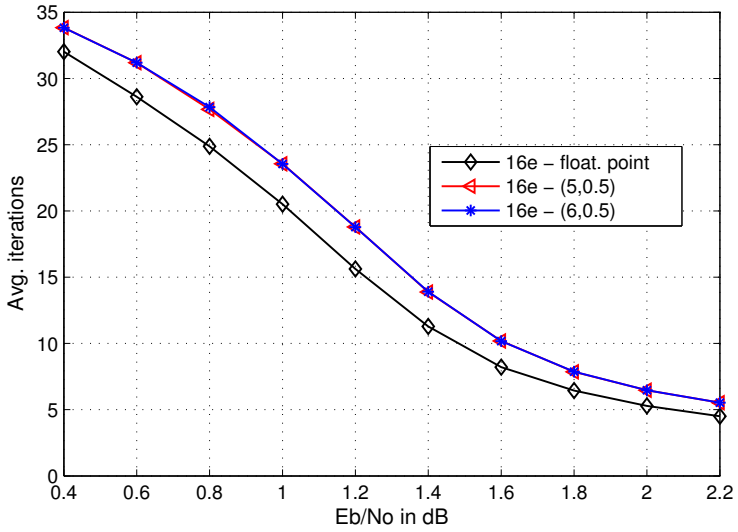
Message quantization on these particular codes has been extensively studied in the past. Besides the obvious loss in performance due to the finite precision of the data the most interesting phenomenon is the ocurrence of an error floor. This floor is typically attributed to the sub-optimality of iterative decoding techniques on graphs with cycles (refer to Chapter 2) and to specific pathologies within a code, some of which are:

- *Near-codewords* [55] or *pseudo-codewords* [56]. These are sequences that satisfy almost all of the check equations.
- *Trapping-sets* [57]. A trapping-set is the set of connected nodes in the code graph whose variable nodes fail to converge.
- *Stopping-sets* [58]. A stopping-set is a set of variable nodes whose adjacent check nodes are connected to at least two different nodes in that set, these sets are of relevance in the binary erasure channel.
- *Absorbing-sets* [59]. An absorbing-set is a set of variable nodes where every variable node has more satisfied than unsatisfied checks.

Figure 6.2a shows the frame-error rate (FER) for floating-point and fixed-point simulations for the use case of code length 2304 with rate 1/2 over the AWGN channel with QPSK modulation for 802.16e mode, with a maximum of 60 iterations. Figure 6.2b shows the convergence rate for the



(a) Frame error rate



(b) Convergence rate

Figure 6.2: Effects of message quantization.

same scenario. A message quantization of 6-bits showed a performance loss of 0.05dB at a FER of 10^{-3}, a low error floor and an average decrease of 22% on the convergence speed. Similar results were obtained for the 802.11n mode. Hereafter, 6-bit messages are used.

6.2 Decoder Architecture

In this section, we describe the architecture of the multi-mode decoder shown in Figure 4.1. The control structure is provided by a compressed storage of the parity-check matrices and a configuration register that governs the decoder operation. Storage elements comprise the posterior messages memory and the extrinsic messages memory per processing unit. Shuffling networks are used to distribute posterior messages from the memory to the processing units and viceversa.

6.2.1 Control Structure

The operation of the decoder is governed by the structure of the parity-check matrix and the variables that define a use case, namely the expansion factor Z and the shift value s of the block-column i_c in $\boldsymbol{H}$ to be processed. As described in Section 3.2, for decoding the ith row it is necessary to retrieve the set $\mathcal{I}_i$ of c_i elements from $\boldsymbol{\gamma}$. Storing $\boldsymbol{H}$ involves saving the shift value of the non-zero blocks and the block-column position. Since for Z rows c_i remains constant, the decoding of Z rows involves retrieving c_i shift values from $\boldsymbol{H}$. These values can be encoded in a read-only memory by concatenating in one word the shift value and column index. A 12-bit word can encode both values using 7-bits for the shift and 5-bits for the column index of each block in $\boldsymbol{H}$. Figure 6.3 shows the data allocation map for the parity-check matrices contained in 802.11n and 802.16e for a [512x48] ROM. Each line in the ROM contains four 12-bit words that indicate a shift value for a block-column index. The first word of every block-row contains an all-ones marker in the shift field indicating that the index field contains the degree of the block-row, in this way all the parity-check information is properly stored in an efficient compact format. Furthermore, the matrices can be grouped to better use the memory space and simplify the addressing. In Figure 6.3, the grouping is done by code rate.

A configuration register, shown in Figure 6.4, contains the specification of the operating mode and use case for the decoder. The code rate (A/B types are used in the 802.16e mode), the expansion factor Z and the mode

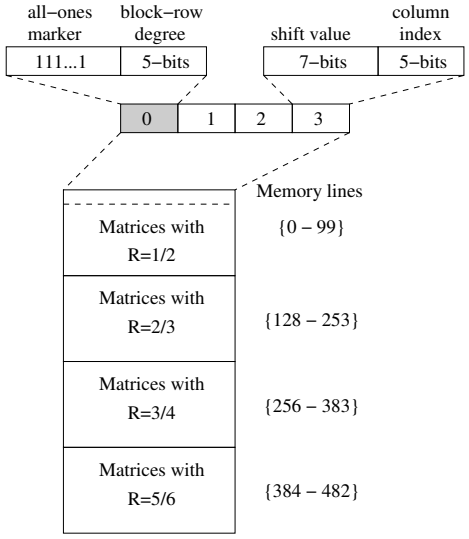


Figure 6.3: Parity-check matrices ROM storage.

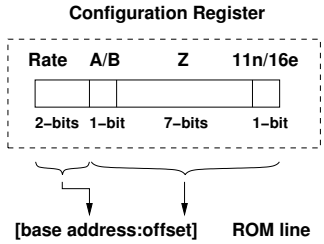


Figure 6.4: Configuration register.

(11n/16e) define a use case. This information is used to localize the corresponding parity-check matrix in the ROM by generating a base address from the code rate and an offset by the remaining bits.

6.2.2 Memory Subsystem

We target a decoder with 15 processing units and design a posterior messages memory that can provide the required bandwidth depending upon the operation mode. Figure 6.5 shows the memory map for the use cases in 802.11n and 802.16e modes as a function of the codeblock length N. This memory is implemented as an array of 5 dual-port SRAM modules; the data allocation shown in Figure 6.5 refers to the sample number corresponding to the N code symbols. In the 802.11n mode three messages are stored per memory line, whereas for 802.16e four messages are stored instead. In

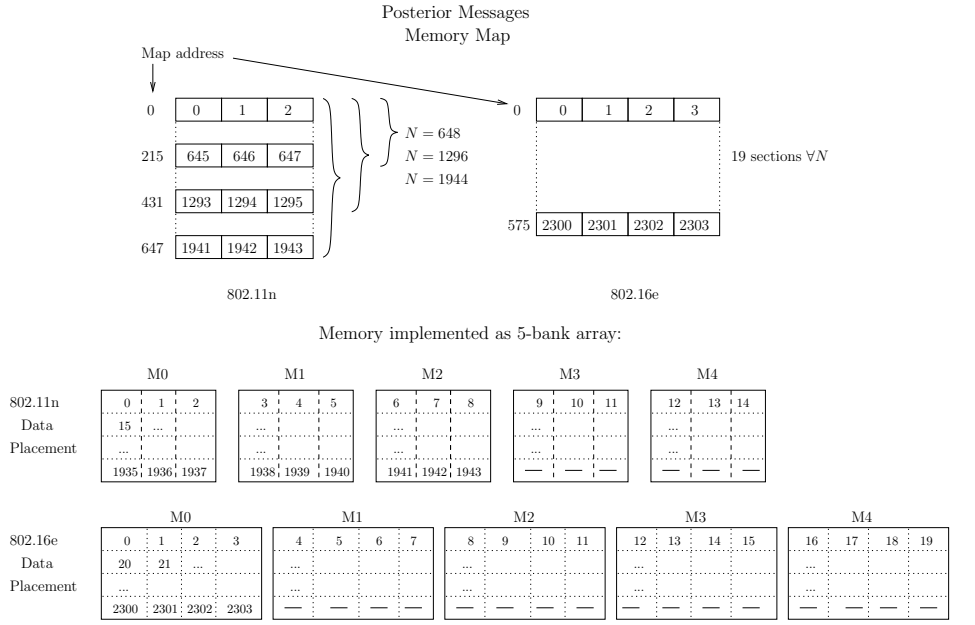


Figure 6.5: Posterior messages memory map and data allocation.

the case of 802.11n operation 15 processing units are used, whereas for the 802.16e case only one unit is used.

The extrinsic messages memory is partitioned and bound to the processing units following a static allocation of rows to processing units. In Figure 6.6, we show the map and format for this memory. The ρ_{old} portion of the memory consists of two bits where one denotes the erasure flag and the remaining one denotes the sign of the corresponding message (both from the previous iteration).

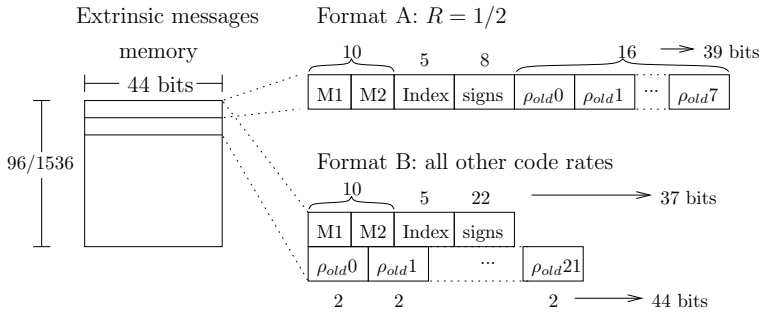


Figure 6.6: Extrinsic messages memory format.

Table 6.2: Row degrees in $\boldsymbol{H}$ according to use case.

Rate	1/2	2/3	3/4	5/6
$Z_{11n} = 27$	$\{7, 8\}$	$\{11\}$	$\{14, 15\}$	$\{22\}$
$Z_{11n} = 54$	$\{7, 8\}$	$\{11\}$	$\{14, 15\}$	$\{21, 22\}$
$Z_{11n} = 81$	$\{7, 8\}$	$\{11\}$	$\{14, 15\}$	$\{19, 20\}$
$\forall Z_{16e}$	$\{6, 7\}$	$\{10\}$	$\{14, 15\}$	$\{20\}$

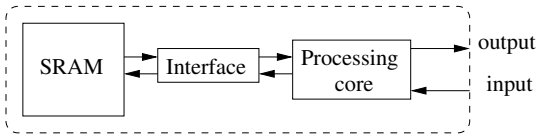


Figure 6.7: Processing unit architecture overview.

The c_i elements of each row i in $\boldsymbol{H}$ are stored in this memory following one of two possible formats (A and B), shown in Figure 6.6, depending upon the use case code rate. In Table 6.2 the row degrees in $\boldsymbol{H}$ are shown for each use case code rate, this determines the size requirements for the extrinsic messages memory. Since the use cases for code rate 1/2 are the most demanding in the sense that they use the biggest $\boldsymbol{H}$, the extrinsic memory format A is used as it guarantees that all the data required to process one row is accessible within one read operation. For the remaining use cases format B is used since there is allowable time to access the row data in two consecutive read operations. These formats represent an efficient reuse of memory space depending upon the use case and operation mode of the decoder. All processing units but one contain a [96x44] SRAM for this memory, the remaining unit uses an SRAM of depth 1536 and width 44 as only one unit is used for 802.16e operation.

6.2.3 Processing Unit

The processing unit consists of an extrinsic messages memory, an interface logic and the processing core for the SCMS kernel. This unit is shown in Figure 6.7, the outputs and inputs correspond to posterior messages coming from and going to the shuffling networks. The memory/processing core interface translates and follows the description of the memory map and formats outlined in the previous section.

The interface, shown in Figure 6.8, operates according to two signals: a *case_select* signal indicating the special case of code rate 1/2 or the re-

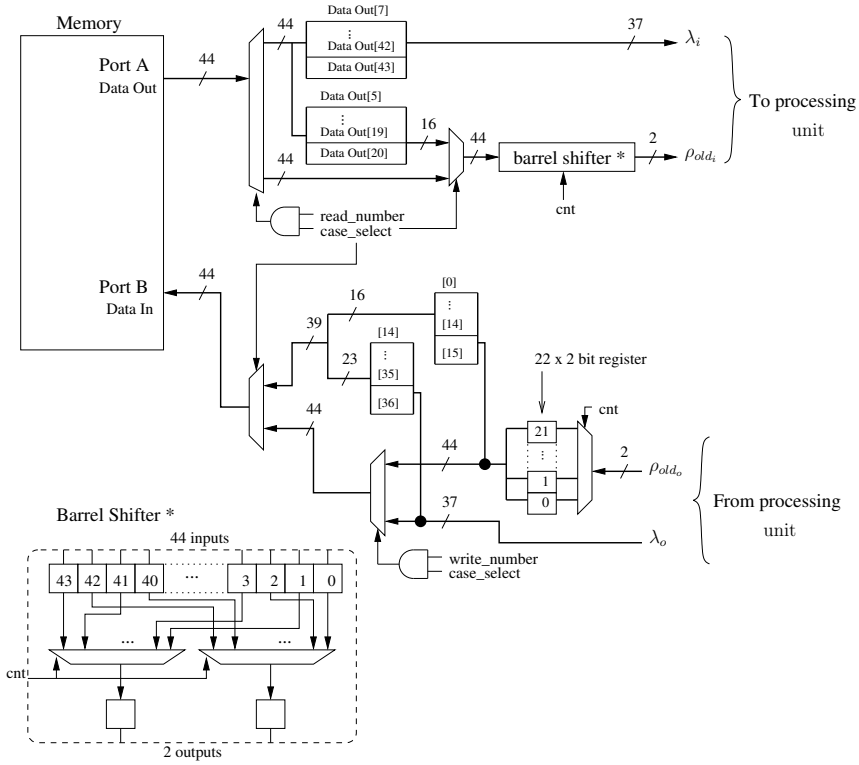


Figure 6.8: Extrinsic messages memory interface with processing core.

maining ones and a *read/write_number* signal that describes the format to read/write a line in the extrinsic messages memory. For the latter signal the first read/write operation corresponds to the format $(M1|M2|Index|signs)$ and the second one to the (ρ_{olds}) format. Reading from two locations (format B) is used only for the use cases not involving code rate 1/2, it follows from this and the architecture of the interface that $case\_select = 0$ for code rate 1/2.

The interface consists of a systematic bit selection and multiplexing that provides the required extrinsic messages λ_i and erasure status/sign ρ_{old} of the previous iteration of row i. A counter signal *cnt* indicates the current sample (out of a total of c_i samples) being read. This signal controls the selection of the proper bit positions for both read and write operations in the extrinsic messages memory.

The processing of a row consists of executing the SCMS kernel on the c_i prior messages, as described in Algorithm 2 in Chapter 3, in order to generate the corrected prior message $\boldsymbol{\kappa}$. This is performed by the SCMS

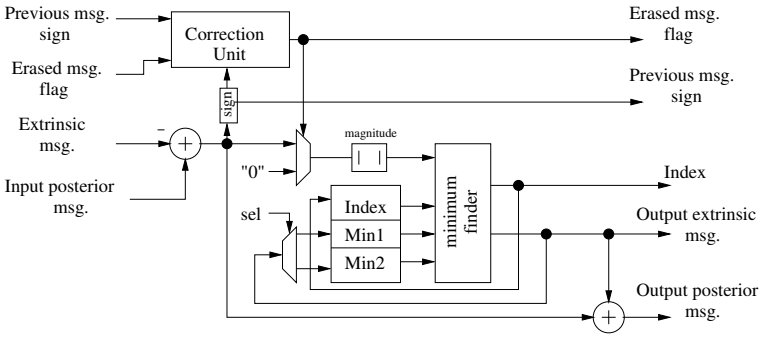


Figure 6.9: Architecture of processing core.

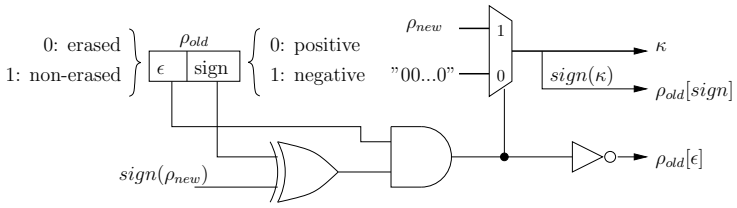


Figure 6.10: Correction unit architecture.

processing core shown in Figure 6.9. A correction unit is used to generate the inputs for a MS unit. This unit is shown in Figure 6.10 along with the format of the ρ_{old} information, which stores the sign of the previous prior message and the erasure status. This information is compared with the newly generated prior message ρ_{new} sign in order to introduce or not an all-zeroes message.

The standard MS kernel is applied on $\boldsymbol{\kappa}$ by means of a serial minimum finder unit, depicted in Figure 6.11. The registers in this unit are initialized to infinity and are updated according to the comparators signal. The AND gates in this figure correspond to the *ANDing* of a vector with a control signal coming from a comparator. The index or position of the first minimum is stored once the first comparator asserts that the value has been updated. The unit shown only describes the message magnitude calculation within the MS kernel. The message sign calculation is performed by an XOR operation between the input message sign and a global sign obtained from the XOR operation of all input signs.

Implementing the SCMS kernel produces some overheads when compared to the sub-optimal MS kernel that is found in various LDPC decoder implementations. Firstly, the memory requirements for the extrinsic memory are augmented by the erasure status and the sign of the prior messa-

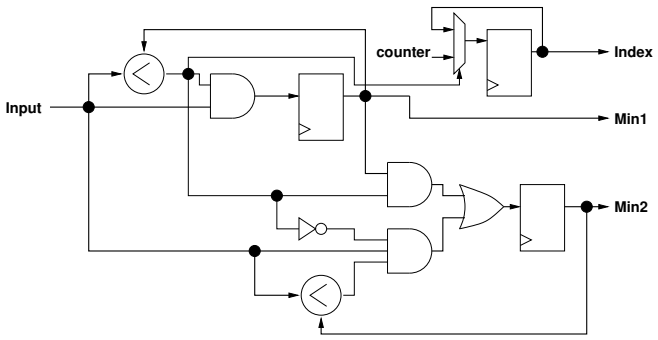


Figure 6.11: Minimum finder unit architecture.

ges of the previous iteration. This amounts to an increase of up to 35% in extrinsic memory requirements when compared to a MS-based implementation when using the 2-bit format per non-zero element in $\boldsymbol{H}$ shown in Figure 6.10. The second overhead corresponds to the correction unit used to *clean* the prior messages for the minimum finders. This unit is relatively small, for a $65nm$ CMOS process it represented 2.5% of the cell area of the minimum finders (an MS-based implementation would only require the minimum finders). These overheads are justified by the gain in error-correction performance and the energy reduction techniques enabled by this kernel outlined in Chapter 4.

The processing unit operates in serial fashion with respect to the c_i inputs when processing row i. As can be seen from the minimum finder macro, the processing unit is pipelined and is able to process a row of degree c_i in $2 \times c_i + 3$ cycles, accounting for both write and read stages. This corresponds to the value of L_c from the expression in equation (4.2) in Chapter 4.

6.2.4 Shuffling Networks

In a semi-parallel architecture the connectivity of the code graph is typically implemented by the reading and writing of the posterior messages that are distributed to and generated in the processing units. These tasks are performed by shuffling networks. Evidently, the complexity of the address generation is affected by the way in which the data is allocated. The data allocation described in Section 5.2.1 provides a simple address generation and avoids complex routing networks used in similar works.

The operation of the decoder is block-column-wise with respect to $\boldsymbol{H}$. The address generation of the required posterior messages for a block-column of index i_c is given with respect to the memory map and later translated to

the physical memory banks. These addresses are a function of Z, s and a value K that defines the use case between 802.11n and 802.16e operation. Relative to the memory map, the first sample required for block-column i_c is given by:

$$addr_{start} = \frac{Z}{K} \times i_c + \lfloor s/K \rfloor \,, \quad K = \begin{cases} 3 & \text{for 802.11n} \\ 4 & \text{for 802.16e} \end{cases} \tag{6.2}$$

If the shift value s is a multiple of K then five read operations are required to process the remaining part of the block-column, otherwise six read operations are necessary. In any case, subsequent addresses are given by:

$$addr = (addr_{start} + 1) \; mod \; \frac{Z}{K} + \frac{Z}{K} \times i_c \,. \tag{6.3}$$

These addresses are translated to the physical memory by selecting the memory module by $addr \; mod \; 5$ and memory row of $\lfloor addr/5 \rfloor$. The read posterior messages are distributed to the processing units by means of a shuffling network that consists of two multiplexing stages and a register for holding values before being redirected. The shuffling network is configurable depending upon the operation mode of the decoder. In Figure 6.12 the shuffling network is shown for the 802.11n case where the data-path is 3 messages wide.

The operation of this network is described in Figure 6.13 for a particular example: processing the block-column of index 4 for the use case of $Z = 81$ and shift $s = 79$. The requirement is to provide 81 samples (324 to 404) to 15 processing units. The first required sample (under the assumption that the block is processed in top-bottom fashion) corresponds to sample 403, located on the address 134 of the memory map. All required samples can be provided in six read operations by reading addresses 108 up to 112 after the first one already mentioned. In Figure 6.13 the hold register is shown with the values loaded from the memory modules once they have been properly addressed by translating the memory map addresses. The second multiplexing stage is termed *hold value select* and is configured according to the value of $shift \; mod \; 3$, this selection forwards the correct sample to a corresponding processing unit. A *boundary register* is used to avoid reading twice from a memory map address by storing the unused samples and reusing them in subsequent read operations. These values are highlighted in the example. There are up to c_i (22, maximum row degree in 11n mode) individual registers each holding 2 messages within the boundary register, this corresponds to the degree of the block-row currently being processed. The hold value

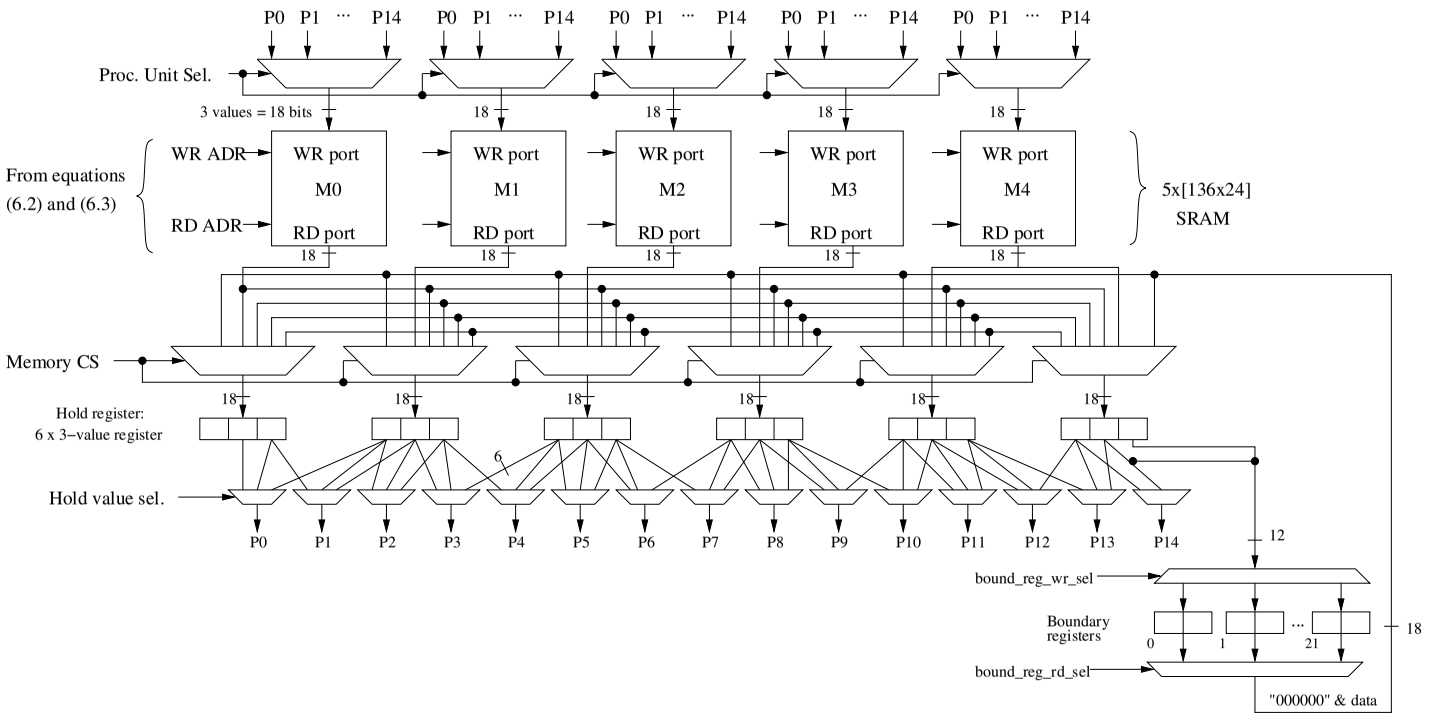


Figure 6.12: Shuffling network configured for 802.11n mode.

register contents are shown after each read operation to illustrate how the required samples are accessed and forwarded to the processing units. For the case where the shift value is a multiple of 3 the boundary registers are not used, this follows from the full alignment of the memory organization for such case.

When operating on the 802.16e mode the decoder uses only one processing unit, this reduces the active portion of the shuffling network. In Figure 6.14 the shuffling network is shown configured for this mode of operation. The data-path is now 4 messages wide (24-bits) and only one hold register is used. The boundary register contains 20 individual registers (maximum row degree in 16e mode) each holding 3 messages.

The operation of the network for this mode is shown in Figure 6.15 for an example case: processing the block-column of index 5 with $Z = 24$ and shift $s = 14$. The samples required are $\{134, 135, \ldots, 147, 120, 121, \ldots, 133\}$ in an ordered sequence. These samples are allocated along addresses 30 to 36 in the memory map. One difference from the 11n operation mode is that the hold value select signal is initialized at position $shift \ mod \ 4$ but is cycled to the right after each read cycle. As shown in the example, one sample is forwarded to the processing unit P14 on each cycle, the $MemoryCS$ signal is shown to indicate where the samples that are loaded into the hold register are coming from. For the case where the shift value is a multiple of 4 the boundary register is not used, again this is due to the full alignment of the memory organization.

6.3 Results and Comparisons

The multi-mode decoder has been implemented in a CMOS $65nm$ technology process using the quantization scheme $(6, 0.5)$. The posterior messages memory consists of an array of 5 [136x24] SRAM modules, while 14 processing units contain each an extrinsic messages memory of size [96x44]. The remaining processing unit contains an extrinsic messages memory of size [1536x44]. All SRAM modules are low-power high-density optimized dual-port memories. The operating conditions for the decoder are 1.32V at $450MHz$.

The typical paradigm to design a flexible shuffling network (also known as message-passing switch network) for LDPC decoding is to assume the availability of Z messages that have to be shifted and then forwarded to Z processing units. As discussed in Chapter 5, such assumption usually implies costly designs containing a high number of small memories since fewer

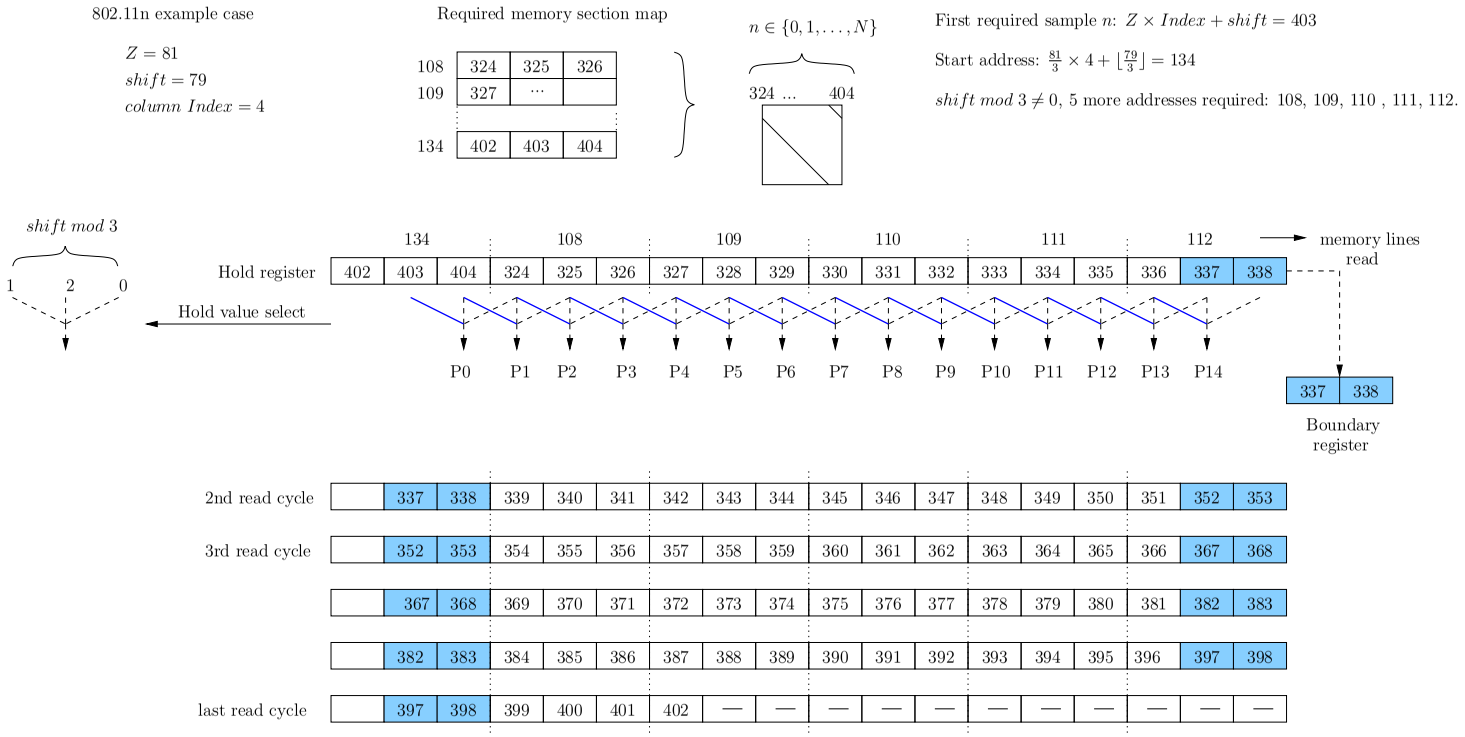


Figure 6.13: Example case for fetching samples on a block-column processing in 802.11n mode.

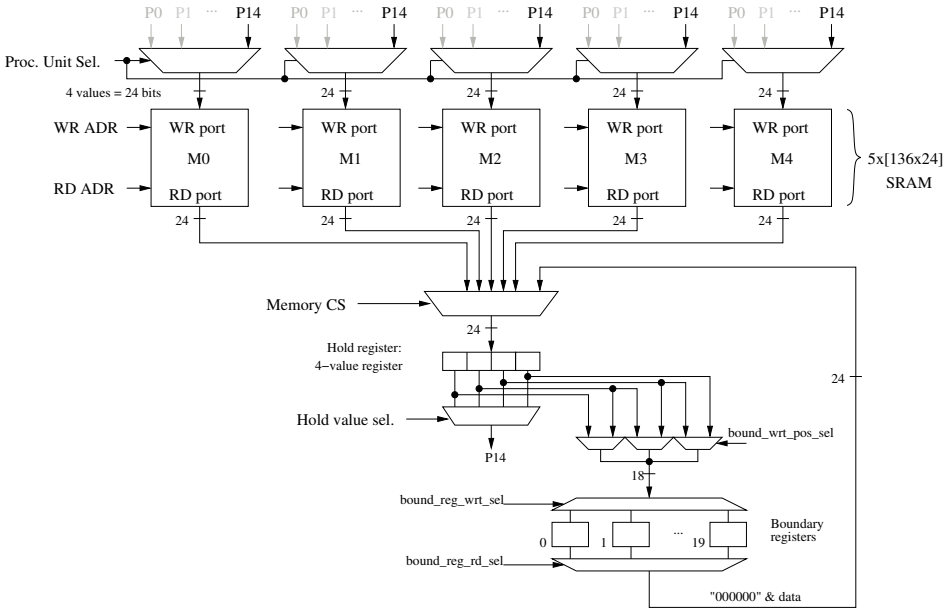


Figure 6.14: Shuffling network configured for 802.16e mode.

bigger memories are preferable to several small memories for on-chip SRAM. The work in [42] introduced multi-size circular shifting networks that can arbitrarily shift the data according to the code structure. The authors in [60] described a self-routing switch network that supports the codes in both 802.11n and 802.16e. In [61] the generation of control signals for reconfigurable barrel shifters is addressed by utilizing small lookup tables. The work in [62] optimizes a Beneš network for the required shift values. While these works contributed with novel architectures to solve the problem of flexibility in order to support several values of Z, the memory allocation and level of parallelism used within the system were not explicitly considered. In this work, we follow a holistic approach to design the required shuffling network by considering the memory allocation and placement of data along with the number of processing units to serve. In Table 6.3, we compare the shuffling network in this work and the previously cited ones. The proposed shuffling network requires no complex routing since the memory allocation succesfully groups consecutive samples that only go through two multiplexing stages to find their destination. A further consequence of this grouping is the reduced complexity of the multiplexing stages. Compared to the previous art the proposed shuffling network requires a fewer number of shift values for the data, 3 for the 802.11n mode and 4 for 802.16e.

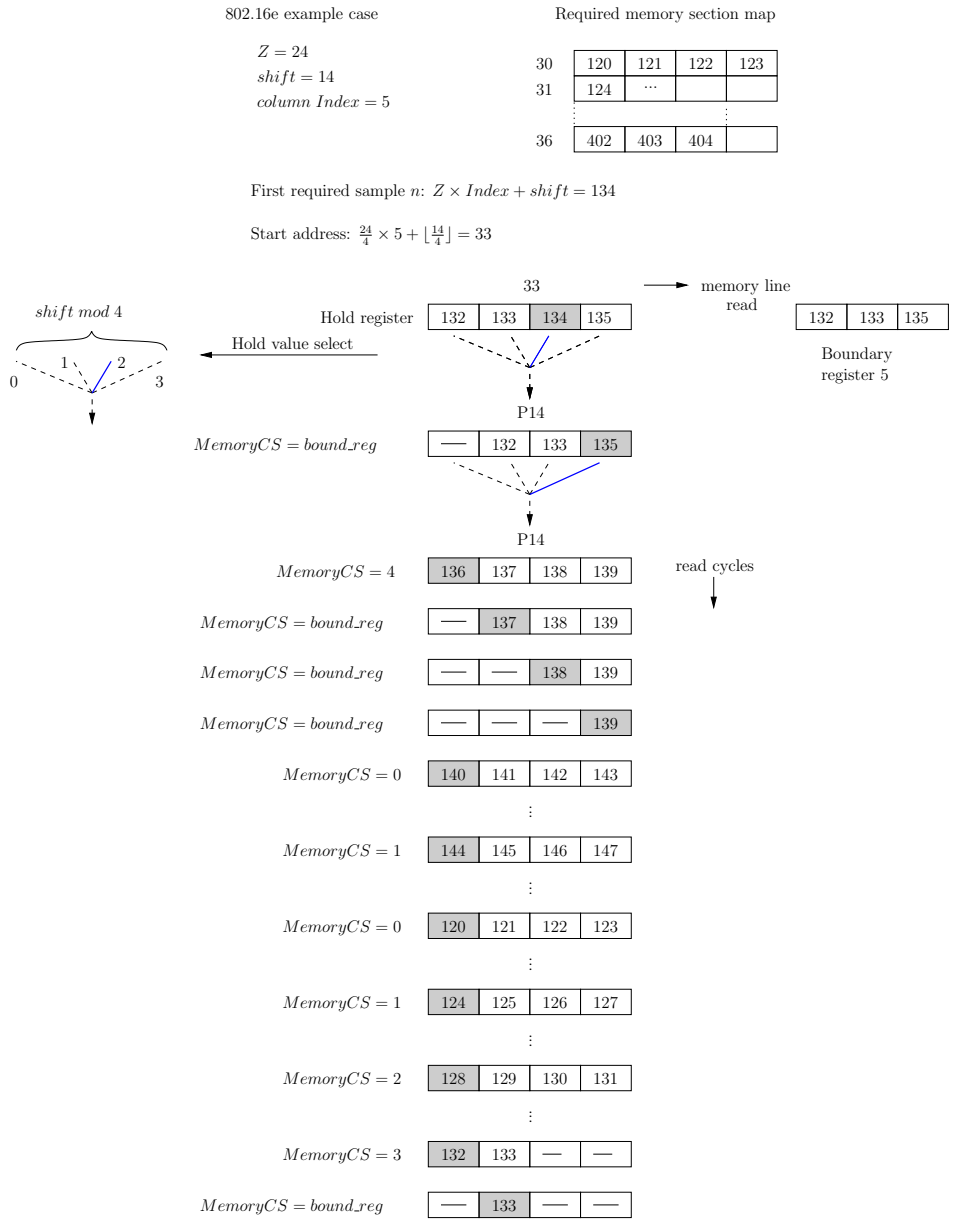


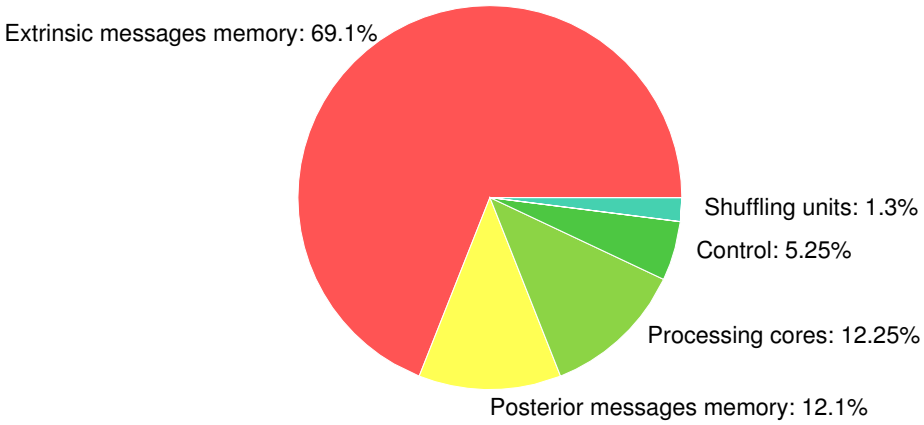
Figure 6.15: Example case on block-column processing in 802.16e mode.

Table 6.3: Shuffling networks comparison.

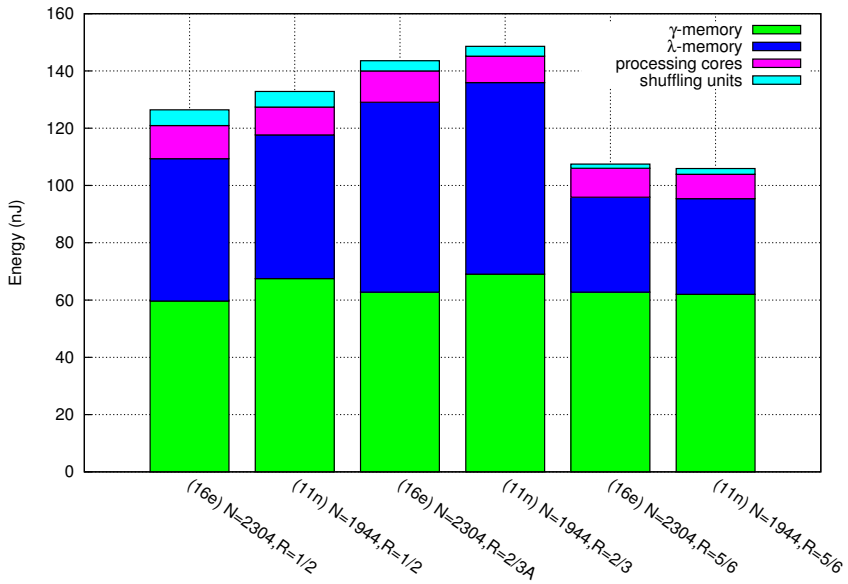
Work	Proposed	[42]	[60]	[61]	[62]
CMOS technology	65nm	65nm	130nm	180nm	180nm
Message length	6-bit	6-bit	6-bit	8-bit	6-bit
Area $[mm^2]$	0.0123	0.0226	0.1095	0.722	0.160
Supported modes	802.11n 802.16e	802.11n	802.11n 802.16e	802.11n 802.16e	802.16e

In Table 6.4, we compare the proposed decoder with representative state-of-the-art works. The work in [34] is based upon the sub-optimal MS kernel and exploits the concept of shift-routing networks to offer the flexibility required to support various codes, while the work in [35] targets very high throughput and focuses on the implementation of the SP kernel. For single mode platforms we show the works from [36] [37] [38]. Even though the cited works and the proposed one are difficult to compare due to the differences in CMOS process and message quantization a few conclusions may be drawn. By applying technology scaling theory [63] it is possible to identify an advantage in the proposed work with respect to implementation area and power consumption. The proposed decoder achieves a high energy efficiency and a small implementation area mainly due to the low-complexity message kernel and the compact memory design.

The energy consumption per iteration was investigated per component per use case in order to reveal whether power management techniques may be applied due to the difference in workload among them. Figure 6.16 shows the breakdown of implementation area and average energy consumption per iteration for several use cases. It is evident how LDPC decoding is a memory intensive application as for all cases at least 80% of the energy is consumed in the memory subsystem. The processing data-path accounting for the actual message computation kernel represents only 12.25% of the total implementation area, whereas the overall memory subsystem comprises no less than 80%. The only power management technique used in this implementation consisted of power gating the unused processing units between the operation modes of 11n and 16e. Chapter 7 discusses the topic of power management in detail. The energy per iteration for the use cases shown in Figure 6.16b is similar between the cases that use the same coding rate. This is expected as the dimensions of $\boldsymbol{H}$ for both modes are similar. Ne-



(a) Implementation area



(b) Average energy per iteration

Figure 6.16: Implementation area and energy breakdown.

vertheless, the energy consumption for the cases in 11n is slightly higher due to the higher consumption on the extrinsic messages memory, which is distributed along the processing units. In all cases the control section of the decoder (5.25% of the area) consumed less than 1% of the average energy expenditure per iteration.

6.4 Conclusion

We presented the design and architecture of a $0.95mm^2$ multi-mode LDPC decoder that supports the quasi-cyclic codes described in IEEE 802.11n and 802.16e. The implementation of the high performance SCMS kernel enabled an energy efficient operation. Furthermore, the design of a compact and flexible memory subsystem enabled the use of a reduced-complexity shuffling network. We provided architectural details that revealed an efficient reuse of hardware components for the supported modes of operation. Comparisons among representative state-of-the-art revealed the benefits of the proposed system in terms of implementation area and energy efficiency.

Table 6.4: LDPC decoders comparison.

Work	Proposed	[34]	[35]	[36]	[37]	[38]
CMOS technology	65nm	90nm	90nm	90nm	180nm	180nm
Message quantization	6-bit	9-bit	N/A	7-bit	6-bit	8-bit
Computation kernel	SCMS	MS	SP	OMS	MS	NMS
Supported codes	802.11n/16e	802.11n/16e	802.11n/16e	802.16e	802.11n	DTMB
Maximum Throughput	352 Mbps (450MHz)	212 Mbps (300MHz)	1 Gbps (450MHz)	200 Mbps (400MHz)	503 Mbps (250MHz)	104.5 Mbps (125MHz)
Maximum iterations	8	20	10	12	10	15
Area $[mm^2]$	0.95	6.22	3.5	0.679	2.67	9.76
Power $[mW]$	110	528	410	124	463	486
Energy Efficiency $[pJ/bit/iter]$	47	126	N/A	135	92	310

Chapter 7

Power Management for Iterative Decoders

So far in this dissertation we have focused on design-time aspects for realizing energy efficient LDPC decoders. In this chapter, we address the aspects involved in the low power operation of the decoders, namely power management techniques to be executed at run-time. LDPC codes are typically decoded by an iterative message-passing algorithm. Iterations are executed until a stopping criterion is satisfied or a preset maximum number of iterations is reached. Because of the iterative nature of the decoding algorithms for Turbo and LDPC codes, *iteration control* has been a recurrent topic for reducing the power consumption of these decoders. Iteration control techniques, also known as *early stopping* or *early termination criteria*, attempt to reduce the operational complexity of the decoder by an early detection of codeblock convergence or lack thereof before the maximum number of iterations is reached. We first address iteration control and propose a control policy that is driven by the combination of two decision metrics. Furthermore, we show empirically how stopping criteria should be tuned as a function of false alarm and missed detection rates.

As a second point, we address the problem of reducing the decoder power consumption from a different perspective. Our approach is based upon the following observations:

- Practical decoders are dimensioned and designed in order to execute

a maximum number of iterations so that a timing deadline is fulfilled.

- It is well-known that error-free decoding can be achieved with a few iterations under good channel conditions.
- For bad channel conditions a codeblock might not even be decoded with the maximum number of iterations.

We argue that by monitoring the dynamics of the iterative decoding process it should be possible to control a power manageable decoder such that energy efficiency is improved. We propose a dynamic power management policy that looks for opportunities to slowdown the system in order to reclaim the timing slack due to a codeblock that converges before the task deadline. Based upon a prediction of the workload of the decoding task, we take advantage of the fact that the decoder is designed for a maximum number of iterations that should take place within a particular timing deadline that is usually defined by a latency or a throughput constraint. We formulate an online algorithm that adjusts the operation mode of a decoder based upon the characteristic behavior of a convergence metric. Furthermore, we analyze the feasibility of a VLSI implementation for such algorithm and control law in a CMOS technology of $65nm$. We apply the proposed technique to both a Turbo (LTE [4] code) and an LDPC (IEEE 802.11n [5] code) decoder.

7.1 LDPC Decoder Iteration Control

Iterative decoding algorithms are inherently dynamic since the number of iterations depends upon several factors. Proper iteration control policies should identify decodable and undecodable blocks in order to improve on energy expenditure and overall task latency. Convergence of a codeword is detected by verifying equation (2.13) (syndrome verification) while non-convergence is usually detected by completing a preset maximum number of iterations.

In this section, we identify a decision metric provided by a specific decoding algorithm, the Self-Corrected Min-Sum algorithm [14]. Motivated by the quasi-optimal error-correction performance of this kernel along with its low complexity and energy efficiency properties outlined in Chapter 4 we look into the possibilities of this kernel to aid on an iteration control policy. We propose to combine two decision metrics in order to control the iterative decoding task. We perform comparisons among the previous art and the proposed control policy in terms of error-correction performance,

average number of iterations and false alarm rate. The main advantage our work shows is the energy efficiency of the proposed policy as it exhibits empirically very low missed detection rates. Furthermore, we argue that the tuning of parameters of a stopping rule should be done based upon the false alarm and missed detection rates performance.

7.1.1 Prior Art

Iteration control techniques attempt to detect or predict the convergence or not of a codeblock and decide whether to halt the decoding task. This decision is aided by so-called *hard* or *soft* decisions. Hard-decision aided (HDA) criteria are obtained as a function of binary-decision values from the decoding process; on the other hand the soft-decision aided (SDA) criteria use a non-binary-decision parameter from the decoding process that is compared against threshold values.

The authors in [64] proposed a termination criterion that detects so-called *soft-word cycles* where the decoder is trapped in a continuous repetition without concluding in a codeword. This is achieved by storing and comparing the soft-words generated after each decoding iteration. This is carried out by means of content-addressable memories. This criterion saves on average iterations but clearly introduces storage elements.

In [65] a stopping criterion was proposed based upon the monitoring of the variable node reliability (VNR), defined as the sum of the magnitudes of the variable node messages. This decision rule stops the decoding process if the VNR does not change or decreases within two successive iterations. This comes from the observation that a monotonic increasing behavior is expected from the VNR of a block achieving convergence. The criterion is switched off once the VNR passes a threshold value that is channel dependent.

The criterion proposed in [66] is similar to the one in [65], it monitors the convergence of the mean magnitude of the variable node reliabilities. The decision rule uses two parameters tuned by simulations that are claimed to be channel independent.

The authors in [67] proposed a criterion that uses the number of satisfied parity-check constraints as the decision metric. Given the syndrome $\boldsymbol{S} = [s_1, s_2, \ldots, s_M]^T$, the number of satisfied constraints at iteration l is:

$$N_{spc}^{l} = M - \sum_{m=1}^{M} s_m \ . \tag{7.1}$$

The decision rule monitors the behavior of this metric tracking the increments and their magnitudes as well as the persistence of such behavior.

In this rule three threshold values are used, all claimed to be channel independent.

A similar scheme was presented in [68]. This criterion monitors the summation of the checksums of all parity-checks given by:

$$S_p = \sum_{m=1}^{M} P_m \ , \tag{7.2}$$

where P_m is the checksum of row m as follows:

$$P_m = \bigoplus_{n \in \mathcal{I}_m} c(n) \, , \text{ with } c(n) = \begin{cases} 0 & \text{if } sign(n) > 0 \\ 1 & \text{otherwise} \end{cases} \tag{7.3}$$

where $c(n)$ is the hard-decision mapping of a soft-input of a row. This is indeed the complement of the decision metric used in [67]. The decision rule monitors this metric and uses two threshold values that are dependent upon signal-to-noise ratio to make a decision.

In [69] a channel-adaptive criterion was proposed by monitoring the *sign-changing rate* of the LLRs per iteration. The control rule uses two threshold values that are claimed to be channel independent.

The above control policies have been derived based upon the observation of the characteristic behavior shown by a particular decision metric within the decoding task. The decision metrics used by these control policies are characterized by their dependence or not upon extraneous variables. Estimating these variables (e.g., SNR) raises the implementation effort.

7.1.2 Hybrid Control Policy

SCMS decoding introduces the concept of *erased messages*, messages which are deemed useless and are discarded after each decoding iteration. A formal treatment behind the concept of erased messages can be found in [14], but intuitively the number of messages erased per iteration provides some measure of the reliability (convergence) of the decoding task. For example, the fewer messages erased, the more reliable the decoding task is. Through simulations we observed the total number of erased messages per iteration to identify the possibility to detect earlier an unsuccesful decoding task and also convergence. In the case of an undecodable block the number of erased messages fluctuates around a mean value (dependent upon the SNR), whereas for a decodable block this metric approaches zero relatively fast. In Figure 7.1, we show how the percentage of erased messages evolves with each decoding iteration for an instance of a decodable and an undecodable block.

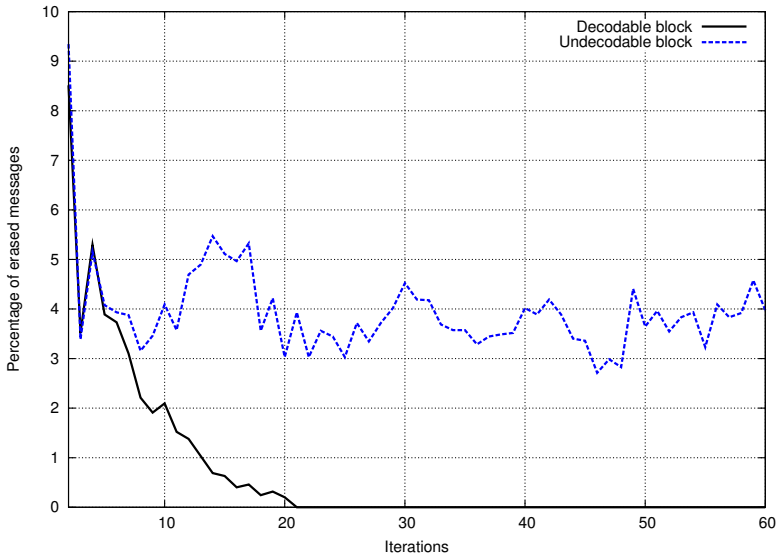


Figure 7.1: Percentage of erased messages.

This corresponds to the decoding of the code defined in [5] with block length 1944 and coding rate 1/2 over the AWGN channel with QPSK modulation, with a maximum of 60 decoding iterations at $E_b/N_0 = 1dB$.

By detecting the characteristic monotonic decreasing behavior of the total number of erased messages when the decoder enters a convergence state, it is possible to save energy on potential undecodable blocks. The *erased messages* metric follows the cumulative quality of the arguments for the parity-check constraints, allowing in fact to observe the dynamics and evolution of the decoding process with fine granularity.

In Figure 7.2, we show the average number of decoding iterations as a function of SNR for the same simulation scenario of Figure 7.1 for several stopping rules:

1. Syndrome check verification, this corresponds to equation (2.13).

2. Erased messages metric. Decoding is halted when either the number of erased messages equals zero or a non-convergence condition is satisfied. For non-convergence detection we allow only a fixed number of increments of this metric.

3. Genie. An ideal stopping rule with foreknowledge of the transmitted block, in this case decoding would not even start on an undecodable block.

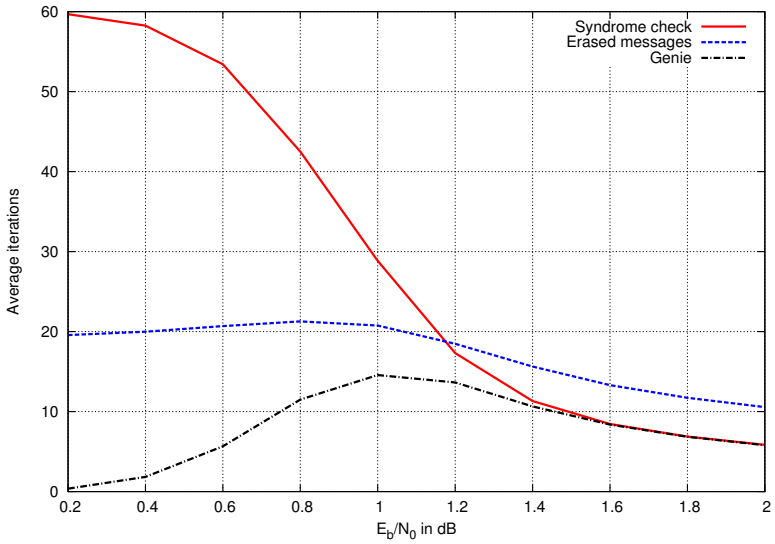


Figure 7.2: Average iterations for stopping rules.

The syndrome check and the genie criteria correspond to the empirical bounds of any valid stopping rule. From Figure 7.2 we observe that the number of erased messages may be used as a decision metric to detect earlier undecodable blocks, but it is not efficient for the detection of early convergence since the absence of erased messages within an iteration is not a necessary condition for convergence, [14].

From these observations we propose to use the erased messages metric to detect an undecodable block and the syndrome check for decodable blocks. We devise a stopping rule that follows the evolution of the total number of erased messages by counting the increments of this metric and halting the decoding task once the number of increments exceeds a given threshold T. This threshold is a static parameter that essentially trades error-correction performance and the average number of iterations. Algorithm 4 outlines the proposed decision rule. After the decoding of a row m the number of erased messages ϵ_m is accumulated per iteration in S_ϵ^l. This sum is compared with the one from the previous iteration in order to detect the behavior of the metric as illustrated in Figure 7.1.

The objective of a stopping criterion can be formulated as the detection of an undecodable block. Thus the possible outcomes of such criterion may be a hit, a false alarm or a missed detection. A false alarm corresponds to the halting of the decoding task that would have been successful in the absence of such stopping rule. This indeed generates unnecessary retransmissions in

Algorithm 4 Stopping Criterion - SCMS

ϵ_m: number of erased messages in row m
$\mathcal{M}$: set of check nodes
f_s: boolean function for syndrome check, equation (2.13)
$count \leftarrow 0$; $S_\epsilon^l \leftarrow 0$
for all $iterations$ $1 < l \leq iterations_{max}$ **do**
  for all rows $m \in \mathcal{M}$ **do**
    Decode row m
    $S_\epsilon^l \leftarrow S_\epsilon^l + \epsilon_m$
  end for
  if (f_s) **then**
    Halt decoding (convergence)
  end if
  if $(S_\epsilon^l > S_\epsilon^{l-1})$ **then**
    $count \leftarrow count + 1$
  end if
  if $(count > T)$ **then**
    Halt decoding (non-convergence)
  end if
end for

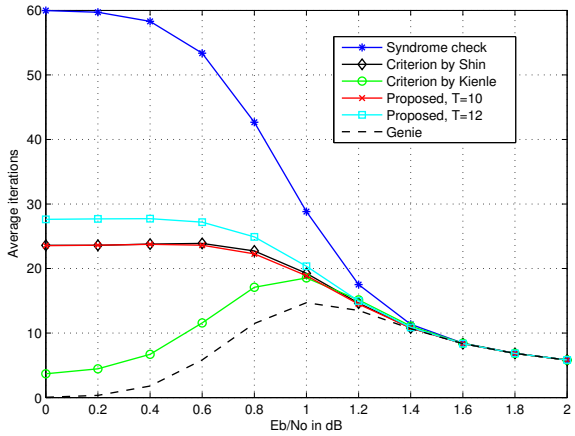
ARQ protocols. On the other hand, a missed detection represents useless energy expenditure and an unnecessary delay to request a retransmission. Even though any stopping criteria can be tuned to make arbitrarily small the average number of iterations this has an impact on the false alarm rate.

In [67] the authors showed empirically how the average number of iterations and the false alarm rate are complementary. We investigated further by looking at the missed detection rate since this indeed can provide hints into a criterion's efficiency. We compare the proposed criterion in Algorithm 4 to the works in [67] (Shin) and [65] (Kienle). In Figure 7.3, we show the performance comparison in terms of average iterations, false alarm and missed detection rates. We observed that when tuning the stopping criteria to have a similar false alarm rate, as shown in Figure 7.3b, the missed detection rates exhibit different behaviors. In fact the proposed criterion showed missed detection rates of several orders of magnitude smaller than the other criteria. The curves for $T = 10$ and $T = 12$ of the proposed criterion are below the value of 10^{-6}, not shown in the figure. These figures pertain to the simulation scenario used in Figure 7.1.

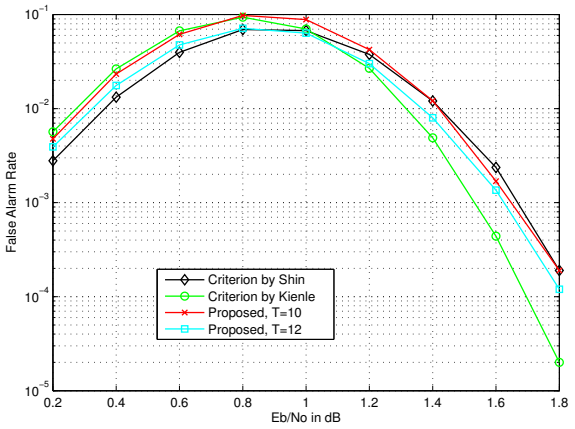
Since it is possible to monitor several decision metrics we investigated how a particular combination may impact the tuning and performance of the resulting *hybrid* control rule. By assisting the decision process with several metrics it is possible to tune the control policy to reach better performance in terms of false alarms, missed detections and average number of iterations. Nevertheless, it was possible to reduce the missed detections only by using the rule in Algorithm 4. For this reason we propose to enhance the performance of the previous rules by adding the number of erased messages per iteration as another decision metric on a SCMS-based LDPC decoder. Figure 7.4 shows the proposed hybrid iteration control system.

We selected the number of parity-check constraints metric [67] as it offers less computational complexity than the VNR metric [65]. In Table 7.1, we compare the cited stopping rules and the one proposed in [69] along with Algorithm 4. The number of operations is given as a function of the dimensions of the parity-check matrix. N is usually much larger than M (e.g., twice for a rate 1/2 code), this means that on the number of calculations alone the criterion by Kienle is the most complex one. Furthermore, the type of data used by this criterion requires full resolution real quantities, this indeed imposes a more complex datapath (within a VLSI implementation) when compared to the other listed criteria.

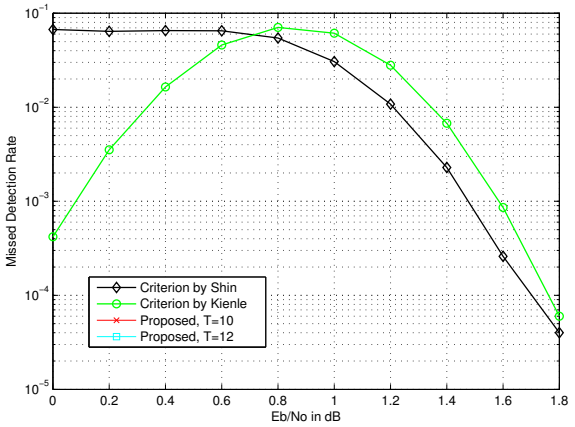
Therefore, by observing the performance (error-correction, average iterations, false alarm and missed detection rates) of the mentioned stopping criteria we propose the hybrid iteration control policy for SCMS-based LDPC



(a) Average iterations



(b) False alarm rates



(c) Missed detection rates

Figure 7.3: Performance of stopping criteria.

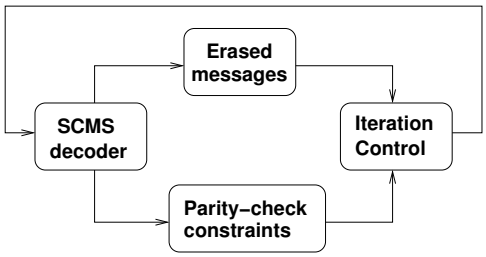


Figure 7.4: Hybrid iteration control system.

Table 7.1: Complexity of decision rules.

Criterion	Operations		Tuning Parameters	Data Type
	Compare	Add		
Shin [67]	3	M+3	3	Integer
Kienle [65]	1	N	1	Real
Chen [69]	3	N	2	Integer
Algorithm 4	2	M+2	1	Integer

decoders such that two decision metrics are monitored in order to detect decodable and undecodable blocks. Even though it is possible to monitor all previously mentioned decision metrics we found out that the erased messages metric provides the most effective detection for undecodable blocks (in the sense of exhibiting the lowest missed detection rate). In the following, we provide results when utilizing the hybrid technique by using both Algorithm 4 and the criterion in [67] embodied as shown in Figure 7.4.

7.1.3 Stopping Criteria Comparison

All stopping criteria can reduce the average number of iterations depending upon the tuning of the decision parameters used within their control policy. This has consequences of different aspects that are worth investigating. In the following, we tune the stopping criteria in [65] [67] [69] along with the proposed hybrid control to be used in the SCMS decoding within the simulation scenario described in the previous section.

Figure 7.5 shows the simulated BER performance for the tested criteria. The stopping criteria can be tuned to be close in performance, for the case of the criterion in [67] (Shin) the parameters used were $\theta_d = 6, \theta_{max} = 4$ and $\theta_{spc} = 825$; for the criterion in [65] (Kienle) $MB = 16$ was used; and for the criterion in [69] (Chen) $lte = 6, thr = 9\%$ were used. The proposed

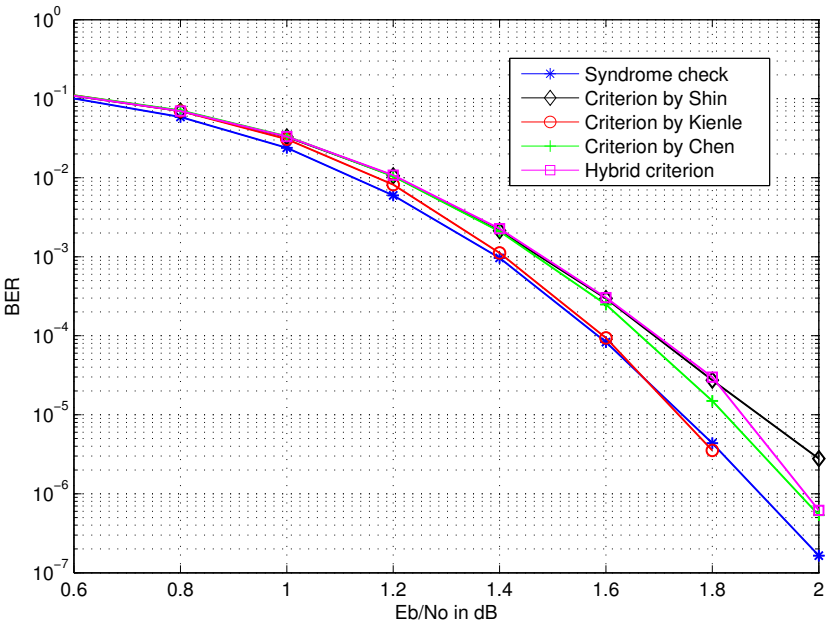


Figure 7.5: Error-correction performance for stopping criteria.

hybrid criterion uses $T = 22$ and the same setup just mentioned for [67].

Figure 7.6 shows the average number of iterations for the stopping criteria. The syndrome check and the genie are once again provided to observe the achievable empirical bounds. Here the tradeoff between average iterations and performance loss (Figure 7.5) is evident. From these figures the criterion by Kienle shows an advantage for a fewer number of iterations in the low SNR region with the smallest performance loss, but this criterion shows the highest false alarm rate (FAR) on the same SNR region.

In Figure 7.7, we show the FAR of the simulated stopping criteria. This is a relevant figure of merit since the stopping mechanism on its own can be responsible for unnecessary retransmissions. We can observe how the criterion by Kienle shows a smaller number of false alarms on the high SNR region, this is due to the inherent threshold that is used within this criterion to disable the stopping rule, but on the other hand this criterion shows the highest false alarm rate for the low SNR region. The comparison between the proposed criterion and the one by Shin and Chen is much closer and indeed can be tuned to have a similar performance.

So far we can observe that the criterion by Kienle in the low SNR region exhibits the lowest average number of iterations but leads to the highest

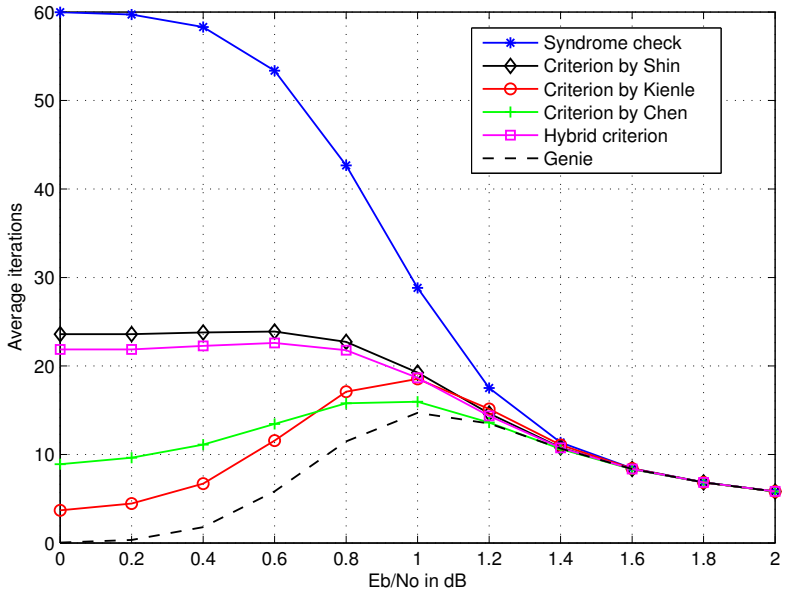


Figure 7.6: Average iterations for stopping criteria.

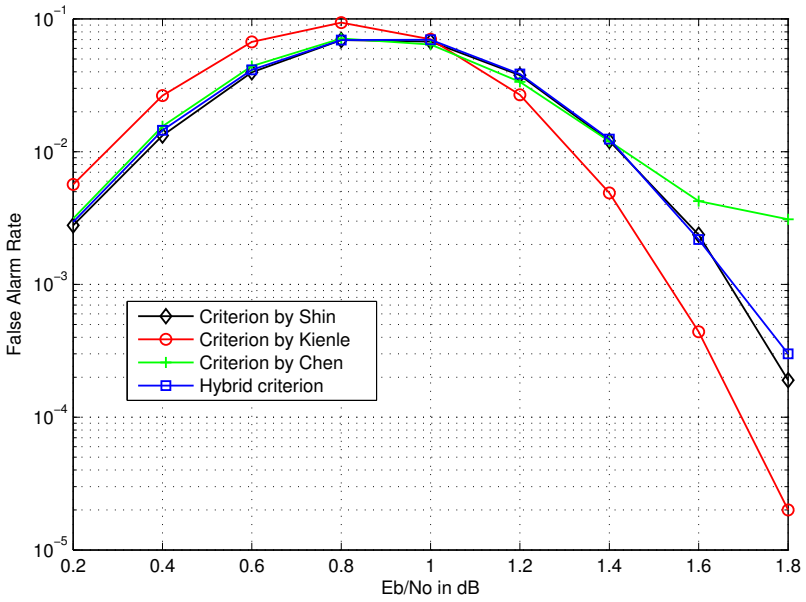


Figure 7.7: False alarm rate of stopping criteria.

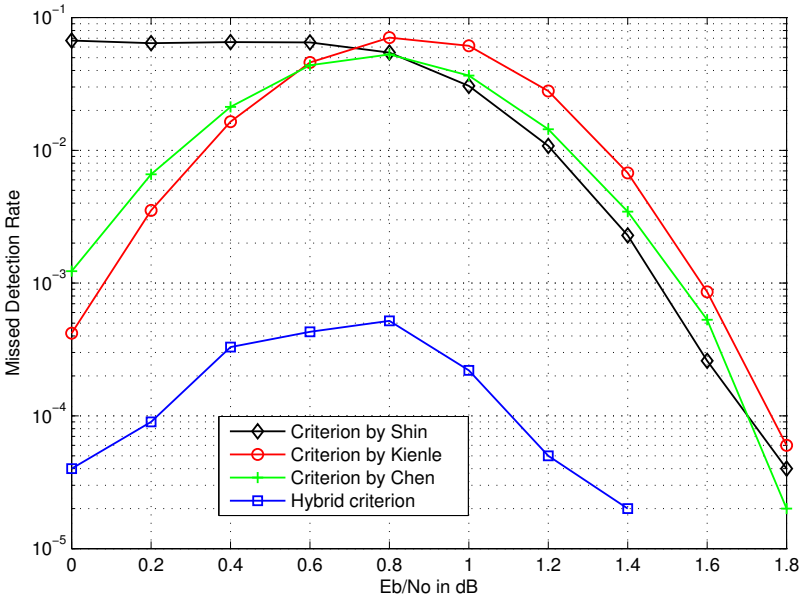


Figure 7.8: Miss rate of stopping criteria.

number of retransmissions. In general, the FAR of these criteria is relatively close, so we proceed to investigate their missed detection performance. Indeed, the missed detection rate (MDR) can provide further insights into which criterion is actually saving energy without incurring into any penalties. Figure 7.8 shows the MDR for the investigated criteria. The criterion by Kienle performs better than Shin for the low SNR region, but this no longer holds as the SNR increases. The criterion by Chen follows similarly the criterion by Shin. The most relevant result is that the proposed hybrid criterion achieved a MDR at least one order of magnitude below the best of the other ones. Notice that it is patently obvious that stopping criteria are useful in the range of low to mid SNR values. Their application should be introduced once the operating regions of a receiver have been properly defined.

The performance for each stopping criterion depends upon the tuning of the decision-making parameters. In Figure 7.9, we show the FAR and MDR for different choices of tuning parameters that result in different average number of iterations. These results are from the same simulated scenario for $E_b/N_0 = 1dB$. From this we can observe the tradeoff involving FAR and the average number of iterations for all criteria. In general the criteria can reduce the average number of iterations but this would result in a higher FAR,

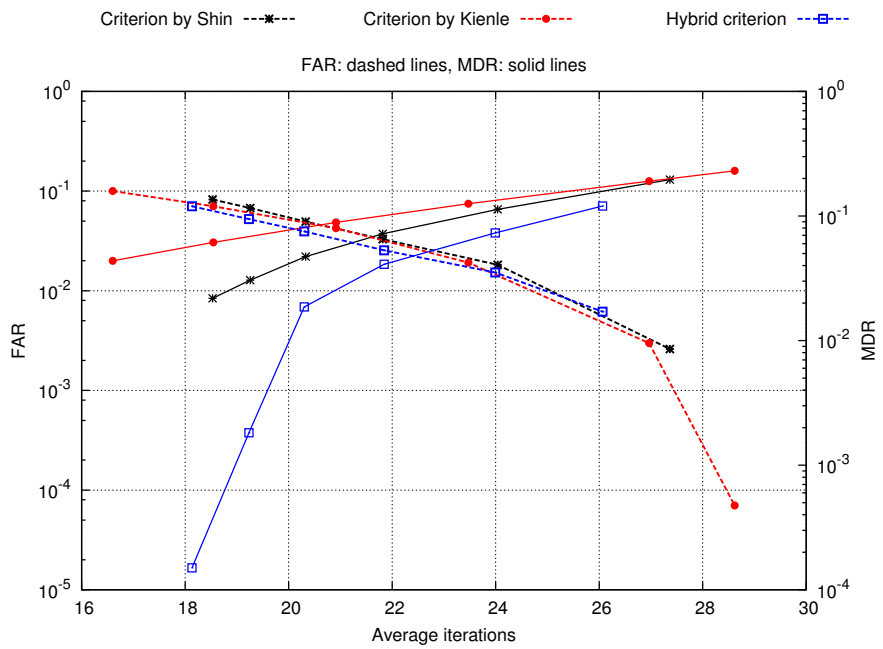


Figure 7.9: FAR and MDR for stopping rules with different tuning of parameters.

this tradeoff must be selected based upon the particular target application (required throughput and allowable retransmissions). Furthermore, we can observe the relationship between MDR and average number of iterations. In this respect the proposed criterion exhibits the best performance. From this figure we can see how a proper tuning of the parameters for a decision rule must consider the relationship between FAR and MDR. FAR refers to the penalty risk introduced by the stopping rule, whereas MDR refers to how effective the stopping rule is for detecting undecodable blocks.

7.2 Dynamic Power Management

In this section, we refer to both LDPC and Turbo decoders. The iterative nature of these decoders represents a dynamic workload within the decoding task: the number of iterations varies according to external factors. Moreover, the decoders are usually designed and dimensioned to complete a preset maximum number of iterations within a hard timing deadline. This design paradigm is a pessimistic one as typically codeblocks converge well before the maximum number of iterations is completed. We present an *online* dy-

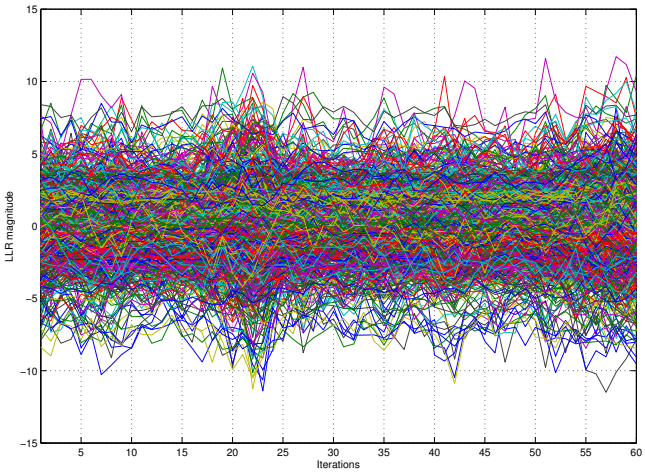
namic power management strategy for iterative decoders. An algorithm is proposed to tune online a power manageable decoder according to a prediction of the workload involved within the decoding task. By reclaiming the timing slack left when operating the decoder at a high power mode, the proposed algorithm continuously looks for opportunities to switch to a lower power mode that guarantees the task completion within the timing deadline. We apply this technique to the iterative decoding of LDPC and Turbo codes and explore the feasibility of a VLSI implementation on a CMOS technology of $65nm$. Energy savings of up to 54% were achieved with a relatively low loss in error-correction performance.

7.2.1 Dynamics of the Decoding Task

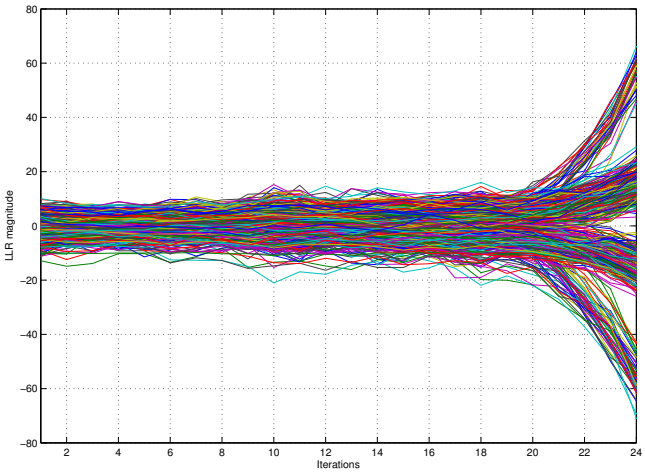
As discussed in the previous section, it is beneficial to detect early the convergent or divergent behavior of a decoding task for the interest of achieving energy efficiency. In other words, it is of interest to observe the dynamics of the decoding process in order to look for opportunities to reduce operational complexity and power consumption. Figures 7.10 and 7.11 show the evolution of the posterior messages LLRs for both a converging and a non-converging codeblock for both types of codes as a function of the decoding iterations. Figures 7.10a and 7.10b correspond to instances of decoding the LDPC code defined in [5] with codeblock length of 648 and code rate 1/2 over the AWGN channel with QPSK modulation at an SNR of $E_b/N_0 = 1dB$ with 60 maximum iterations. Figures 7.11a and 7.11b show two instances of decoding the Turbo code defined in [4] with codeblock length of 6144 and code rate 2/3, as well through the AWGN channel with QPSK modulation at $E_b/N_0 = 1dB$ with 16 maximum half-iterations.

In the following, we argue that by monitoring the dynamics shown in Figures 7.10 and 7.11 it should be possible to make predictions on the required decoder workload. We use the term workload to refer to either iterations or half-iterations depending upon the code at hand.

Dynamic power management (DPM) refers to a set of techniques used to achieve energy-efficient operation of a system. This is performed by judiciously adjusting or reconfiguring the system to provide a requested service and performance level with a minimum energy expenditure based upon runtime observations. Several techniques exist to achieve this at different levels (e.g., system and gate levels) such as sleep, slowdown modes and clock gating. In order to apply DPM usually two premises are considered, [70]: the system experiences a non-uniform workload during operation time and it is possible to certain degree to predict the fluctuations of the workload. Usu-

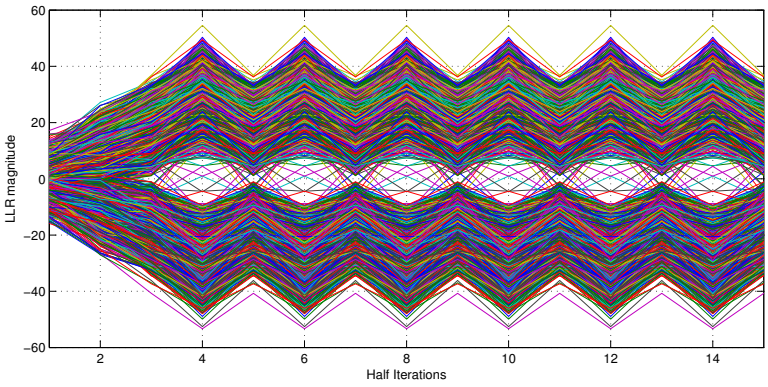


(a) LDPC non-converging codeblock

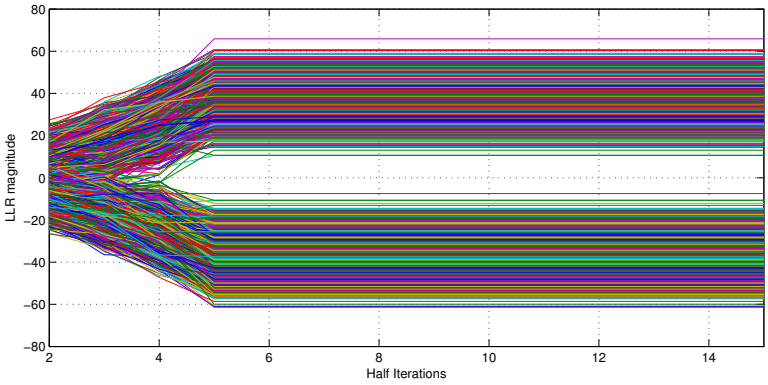


(b) LDPC converging codeblock

Figure 7.10: Posterior messages LLRs magnitude evolution on an LDPC code.



(a) Turbo non-converging codeblock



(b) Turbo converging codeblock

Figure 7.11: Posterior messages LLRs magnitude evolution on a Turbo code.

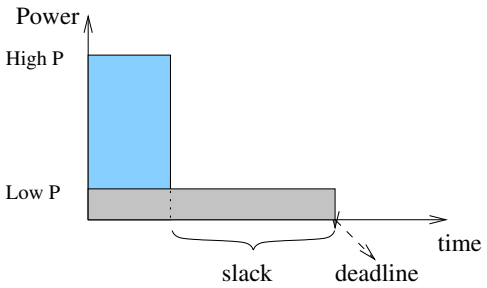


Figure 7.12: Example power/slowdown scenario.

ally, a *power manager* executes a control procedure known as a *policy* or *law* that is based upon observations of the task workload.

The iterative decoding of LDPC and Turbo codes is inherently dynamic in the sense that the number of iterations depends upon the reliability of the decoding process. This is basically determined by the level of noise present in the received codeblock. This is therefore a task with variable workload. Typically, these decoders are dimensioned to operate at a high performance mode in order to complete a maximum number of iterations within a given timing deadline. Nevertheless, this design paradigm is strictly pessimistic since a codeblock would typically reach convergence in fewer iterations than the preset maximum. This fact could be exploited in order to reclaim the timing slack and slowdown the decoder to a low-power mode.

Figure 7.12 shows a decoding task performed under both a high- and low-power mode. Under the high-power mode the task is completed before a timing deadline, while under the low-power mode the task is also completed by the deadline but utilizes the full slack that remains from the high-power mode. The area under each curve indicates the total energy spent for each task. Depending upon the relationship among the power levels and the slowdown factor energy efficiency may be improved by reclaiming the slack left when running at high-power mode. Notice that the task deadline is typically defined in order to comply with performance requirements like latency and/or throughput.

7.2.2 Prior Art

As mentioned in Section 7.1.1, previous works addressing power management on LDPC decoders focus on reducing the number of iterations to avoid unnecessary decoder operation. For the case of Turbo decoding the authors in [71] proposed to monitor the sign changes of the LLRs in order to detect

the codeblock convergence. Well-known methods for SDA criteria monitor the *cross-entropy value* [21] and the *mean-reliability value* [72].

In [73] [74] the authors proposed a preprocessing stage for LDPC decoding that estimates the required decoding effort and proceed to adjust the system power mode (voltage and frequency) in order to have a constant decoding-time. To the best of our knowledge, no other work within the prior art has attempted to follow online the iterative decoding process in order to make predictions about the codeblock convergence and look for opportunities to apply dynamic power management strategies.

For iterative decoding in general, the work in [75] proposes a method for reducing the average power consumed by a decoder by reducing the average number of iterations. The authors of this patent propose a control loop that adaptively restricts the maximum number of iterations performed on highly corrupted codeblocks. This work effectively comprises an SDA stopping criterion.

DPM techniques based upon workload prediction have been previously proposed in different contexts. For example, the work in [76] predicts the MPEG frame decoding time and applies dynamic voltage scaling. The prediction is based upon the block level statistics where the premise for workload variation is the difference on block processing time, this depends upon the block type. In [77], the authors target embedded system applications and propose a software-based power manager that profiles the workload characteristics through a queuing model. By means of an *initial value problem* the workload is predicted and the system frequency of operation is adjusted. Both works assume continuous frequency/voltage scaling capabilities.

DPM has been a topic of intense research, comprehensive surveys can be found in [70] [78]. As revealed by these works, DPM has been mostly investigated in the context of operating systems for general purpose and embedded computing. The main problem studied has been to find the optimal transition times to low-power or idle modes. In this work, we target a real-time kernel for mobile computing devices that must rely upon control policies of very low complexity in order to enable DPM. Following the taxonomy introduced in [70], we present an *adaptive predictive* control scheme for the iterative decoding of LDPC and Turbo codes.

7.2.3 Problem Definition

We consider an iterative decoder to be a power manageable CMOS component governed by a power manager. The set $\mathcal{P} = \{P_0, P_1, \ldots, P_{n-1}\}$ defines n power modes where, [73]:

$$\begin{aligned} P_k &= P_k^{sw} + P_k^{sc} + P_k^{leak} \\ &= E_{sw} C_L V_k^2 f_k + I_{SC} V_k + I_{leak} V_k \, . \end{aligned} \tag{7.4}$$

P_k^{sw} is the power due to the switching activity when charging and discharging the load capacitance C_L with switching activity factor E_{sw}. P_k^{sc} is the power due to a short-circuit current when both NMOS and PMOS sections of the circuit are switched. P_k^{leak} is the power due to the leakage current I_{leak} (subthreshold plus reverse bias junction current), a technology dependent parameter. The *threshold* voltage V_t can also be incorporated in (7.4), [73]. Each power mode P_k operates at a particular voltage V_k and frequency f_k. In the following, we assume that the first state P_0 consumes the most power, subsequent states consume each less power than the previous one. For each power mode P_k there is a corresponding slowdown factor α_k where for the fastest mode $\alpha_0 = 1$. Each power mode can also be described as a fraction of the highest power mode P_0 by a factor β_k, e.g., $P_k = \beta_k P_0$. Given the quadratic relation between power and voltage and the linear relation between power and frequency, it is possible to slowdown the system (consequently augmenting processing time) such that the total energy expenditure is reduced. This is the principle behind the well-known concept of dynamic voltage and frequency scaling (DVFS), [79].

Given the model of the power function $P \propto V^2$ and $f \propto (V - V_t^2)/V$, [80], it is in the best interest of the power manager to run a task as slowly as possible due to the convexity of the power function, [78] [81].

Given a workload of I iterations to be executed before a timing deadline d, we wish to find a subset of power modes $\mathcal{P}' \subseteq \mathcal{P}$ such that the total energy is minimized. If an iteration is executed in time t_k through a power mode P_k, the problem is stated as finding the optimal $\mathcal{P}'$ that minimizes the total energy spent:

$$\begin{aligned} &minimize \ \sum_{i}^{I} P_k^{(i)} t_k^{(i)} \, , \ \ P_k \in \mathcal{P}' \\ &subject \ to \ \sum_{i}^{I} t_k^{(i)} \leq d \, , \end{aligned} \tag{7.5}$$

where $P_k^{(i)}$ indicates the power mode used during the ith iteration. This problem can be formulated as well as a linear program that minimizes an energy function by finding a power $P \in \mathbb{R}$ that guarantees the constraint d.

Such formulation, however, would not capture the adaptive online characteristic of choosing a power mode from a finite set under the uncertainty of the workload. In the following, we propose a heuristic to solve (7.5) based upon workload predictions.

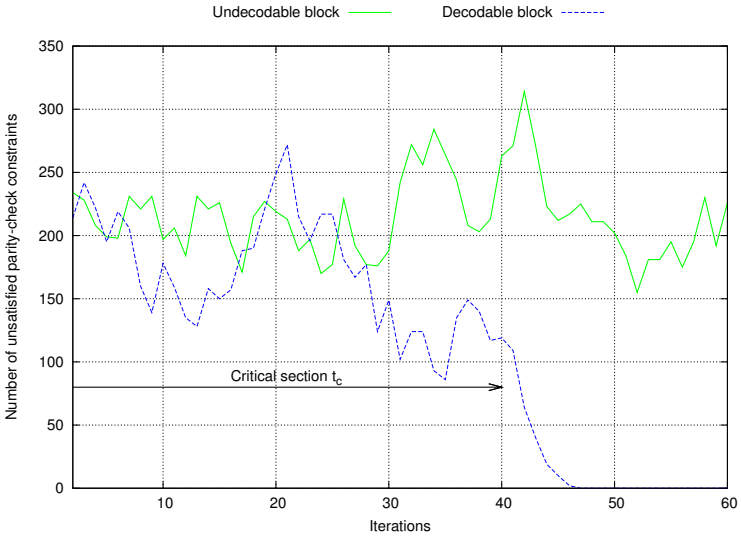
7.2.4 Control Policy

DPM is at its core an *online* problem since a power manager must make decisions about the system operation mode before all of the input to the system is available. The input here refers to the total required number of iterations to achieve a codeblock convergence. Indeed, an online algorithm attempts to find an optimal power mode based upon information available only at runtime. On the other hand, an *offline* algorithm finds the optimal power mode to satisfy (7.5) assuming the total required number of decoding iterations is known.

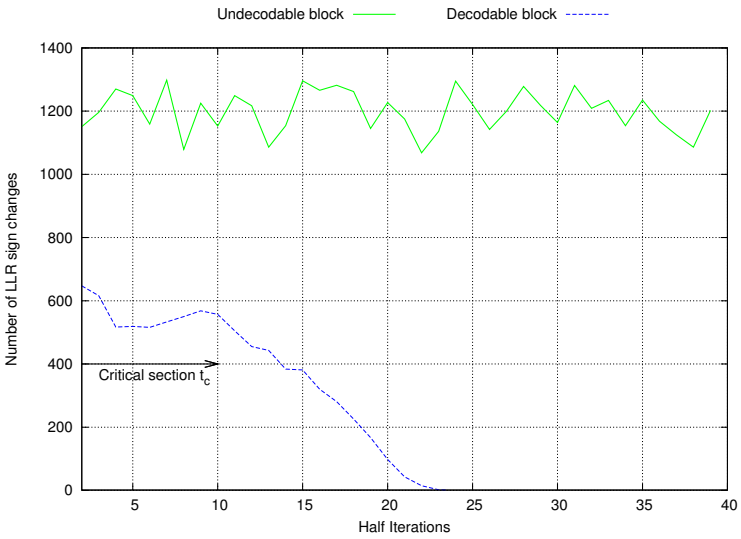
In order to formulate the online strategy it is necessary to look into the dynamics of the decoding process. For the case of LDPC codes, in [67] [82] it was proposed to use the number of satisfied parity-check constraints as a decision metric for iteration and power control. The complement of this metric (number of unsatisfied constraints) follows a monotonic decreasing behavior when the decoder enters a *convergence state*. For the case of Turbo codes, in this work we propose to use the number of hard-decision changes upon the posterior messages after each half-iteration, i.e., at the output of each component SISO decoder.

Figure 7.13a shows the number of unsatisfied parity-check constraints per iteration for an instance of an undecodable and a decodable LDPC codeblock. This corresponds to the decoding of the code of length 1944 and rate 1/2 defined in [5], simulated through the AWGN channel ($E_b/N_0 = 1dB$) with QPSK modulation and a maximum of 60 iterations. Figure 7.13b shows the number of sign changes in the posterior LLRs after each half-iteration for an instance of an undecodable and a decodable Turbo codeblock. This corresponds to the Turbo code defined in [4] with codeblock length 6144 and rate 2/3 through the AWGN channel ($E_b/N_0 = 1dB$) with QPSK modulation and 20 maximum full-iterations. It is observed for both codes that the corresponding metric fluctuates around a mean value for undecodable blocks, but for decodable blocks it fluctuates for a period of time t_c and later enters a *convergence* mode characterized by a monotonic decreasing behavior. We refer to the period t_c as a *critical* period since no decision can be made regarding the convergence of the code.

The shown metrics reveal a characteristic behavior that can be exploited



(a) LDPC code



(b) Turbo code

Figure 7.13: Convergence metric example behavior.

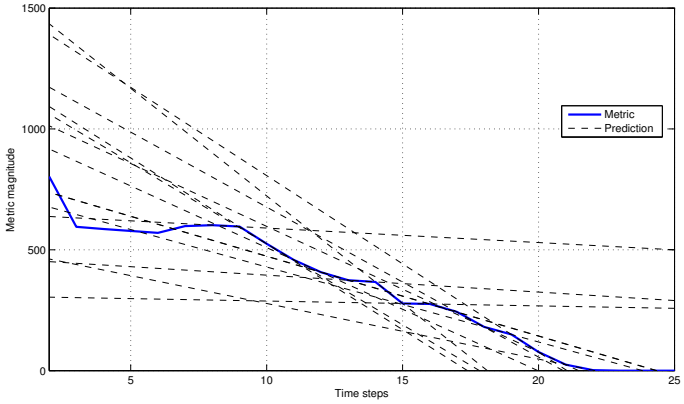


Figure 7.14: Convergence metric example and first order predictions.

to make predictions about the possible outcome of the decoding task. For both types of codes the respective metric approaches zero once convergence is achieved. By predicting this time instant a power manager could calculate the remaining number of iterations and adjust the system operation so that minimal power is consumed and the task deadline is satisfied. Figure 7.14 shows the behavior of a decision metric and different predictions performed at different time instants. These predictions are based upon a simple approximation, a first order derivative using two samples from the metric history.

As shown in Figure 7.13, no predictions can actually be valid during the critical time section due to the repeated and irregular fluctuations of the metric. But indeed once the convergence mode is entered predictions can be refined at each time step in order to approximate with higher accuracy the probable end of the task. Based upon the assumption that slowing the task speed is the optimal decision in terms of energy consumption, we propose to operate the decoder at a high-power mode (high speed) during the critical period and look for opportunities to slow the system down (low-power modes) based upon the metric predictions. Figure 7.15 illustrates this idea, a typical convergence metric behavior is shown along with the set of power modes that a power manager may select based upon the metric predictions. Another possibility would be to slow the system down in the critical section and speedup at the end once convergence has been detected. However, we do not pursue such approach since this could potentially compromise the deadline fulfillment.

We formulate the proposed control policy based upon the observations of

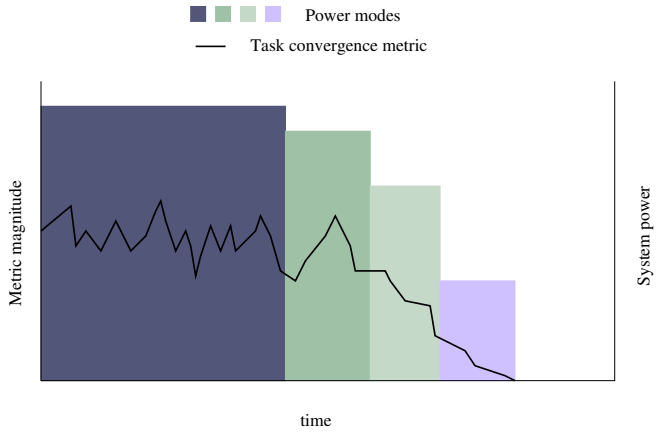


Figure 7.15: Metric-driven DPM example.

the selected convergence metrics for both types of codes. Figure 7.16 shows the decision flow of the control policy. Decision making is based upon the behavior of the monitored metric that reveals whether a convergence mode is entered or not and whether further decoding iterations may be triggered or not. The latter point in fact refers to an early stopping criterion just like the ones mentioned in the prior art for both types of codes. In Section 7.2.6, we will show how the performance loss in the error-correction sense is due to the stopping criteria used within the power control policy.

The proposed decision flow is used to design the online DPM strategy for iterative decoding, outlined in Algorithm 5. The decision metric ϵ is monitored on an iteration/half-iteration basis depending upon the type of code.

The algorithm essentially starts the system on the highest power mode since it tries to exit the critical period as fast as possible. During this period there is uncertainty with respect to the convergence of the block and indeed the energy cap E_c represents a stopping criterion. Stopping criteria suffer from false alarms, i.e., codeblocks that would have been successfully decoded in the absence of such rule. This translates into a loss on error-correction performance. Convergence is detected when the last q values of the metric ϵ are strictly decreasing. If the consumed energy is below the energy cap the convergence metric is used to estimate the remaining decoding effort. Equation (7.6) shows a function based upon the last w values of the metric where prediction functions of different degrees of complexity may be used. A first order approximation based upon two metric samples provides the simplest prediction function. Figure 7.17 shows how two values of the metric are used to approximate the convergence region to a line segment.

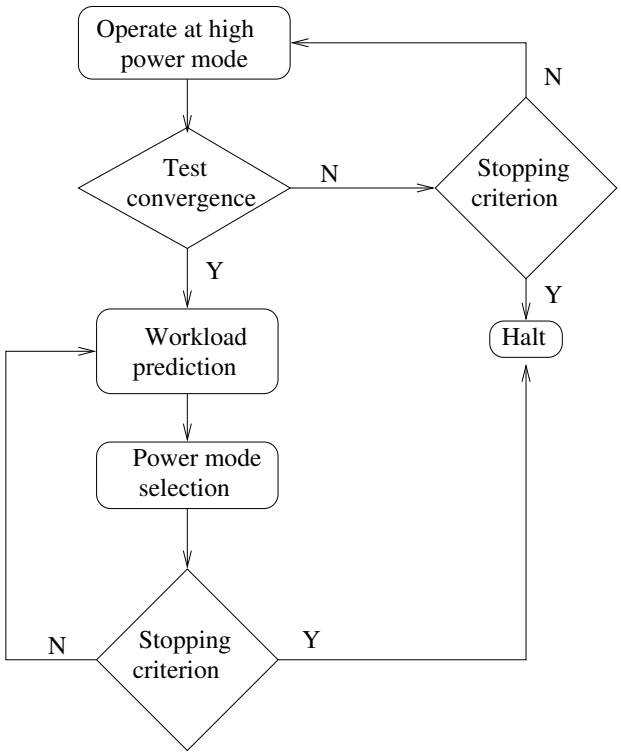


Figure 7.16: Proposed DPM policy flow.

Algorithm 5 Online Power Management Policy

i: current iteration/half-iteration, $i \in \{1, 2, \ldots, I_{max}\}$
ϵ_i: convergence metric value at iteration i
1. Decoding starts, $i = 1$
Set power mode P_0
2. Critical section
for $i = 1$ to I_{max} **do**
if $(P_0 t_0 i \geq E_c)$ **then**
Halt decoding
else
Check convergence state
if $(\epsilon_i < \epsilon_{i-1} < \ldots < \epsilon_{i-q})$ **then**
Go to 3.
end if
end if
end for
3. Convergence section
for $j = i$ to I_{max} **do**
Estimate required workload

$$\hat{I} = f(\epsilon_j, \epsilon_{j-1}, \ldots, \epsilon_{j-w}) \tag{7.6}$$

Switch to power mode P_k such that

$$\sum_{m=1}^{j} t_k^{(m)} + (\hat{I} - j) t_k \leq d \tag{7.7}$$

Stopping Criterion
if $(\epsilon_j = 0)$ **then**
Halt decoding
end if
end for

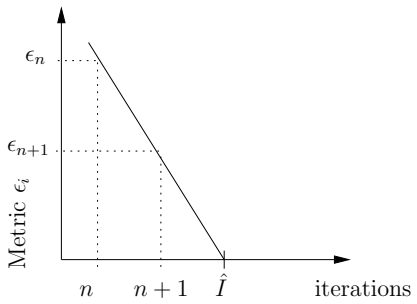


Figure 7.17: Estimation of decoding effort by first order approximation.

Given the two latest values of the convergence metric ϵ_n and ϵ_{n+1} (at iterations/half-iterations n and $n+1$ respectively) the total estimated decoding iterations/half-iterations $\hat{I}$ is given by:

$$\hat{I} = n - \frac{\epsilon_n}{\epsilon_{n+1} - \epsilon_n} \, . \tag{7.8}$$

Overheads associated with the transitions between power modes (e.g. transition times and energy cost) are not explicitly mentioned in Algorithm 5 for simplicity.

Competitive analysis has been used in theoretical computer science to design and analyze online algorithms; dynamic power management has been previously analyzed within this context in [78] [83]. The performance of an online algorithm operating on a continuous input stream can be compared to an offline algorithm which has access to the same input in advance. The *competitive ratio* is the ratio between the performances of the online and offline algorithms. In the context of the problem presented in this section, an offline algorithm can perform the decoding task with a minimum energy expenditure since it is able to find an optimal low-power mode that guarantees the task timing deadline. This is due to the foreknowledge of the task workload (required number of iterations). The performance of the online algorithm can be evaluated by the competitive ratio with respect to the cost of the offline alternative. The notion of cost in this case is taken as the total consumed energy by each strategy. The upper bound of this ratio is given by:

$$c = \frac{P_0}{P_{n-1} \cdot \alpha_{n-1}} = \frac{1}{\beta_{n-1} \cdot \alpha_{n-1}} \, . \tag{7.9}$$

This performance metric indicates that an online algorithm can find a solution with a cost less than c times the cost of the offline alternative.

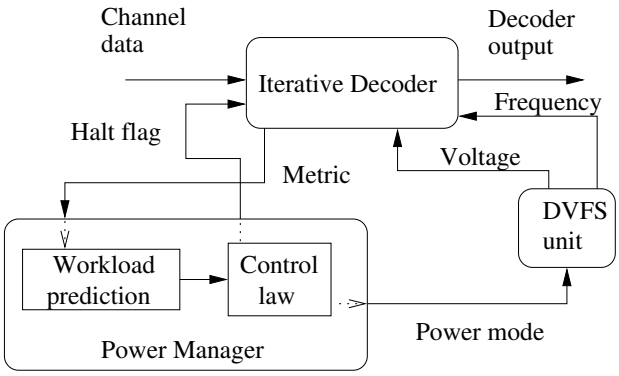


Figure 7.18: DPM system block diagram.

An online algorithm is said to be *c-competitive* if for any input the cost is bounded by c times the cost of the offline algorithm for that input. Nevertheless, it has been previously noted [83] that competitive analysis provides an overly pessimistic bound on the performance of online techniques, this will become evident with the results presented in Section 7.2.6. This is due to the worst-case analysis nature of the competitive ratio, nonetheless this analysis provides good insights for tuning an online strategy and improving performance.

7.2.5 DPM System Implementation

The proposed DPM system is shown in Figure 7.18. An iterative decoder with adjustable voltage and frequency operation is governed by a power manager. The decoder receives intrinsic channel values in the form of LLRs and produces hard-decision bits for the decoded message. By constantly monitoring a convergence metric from the decoder, the power manager executes the control policy outlined in Algorithm 5. The power manager sets the state of a DVFS unit that provides the operating conditions for the decoder.

Figure 7.19 shows the architecture of the power manager unit. This unit executes three main tasks duly represented in this figure: estimation of remaining decoding effort, power mode selection and convergence/energy cap detection. The estimation function shown implements the complement of equation (7.8) to predict the remaining workload, this function relies upon two samples of the metric ϵ. In the power mode selection block, the processing time t_k per iteration of each power mode k is used to calculate the cumulative decoding time. This value is constantly being added to the es-

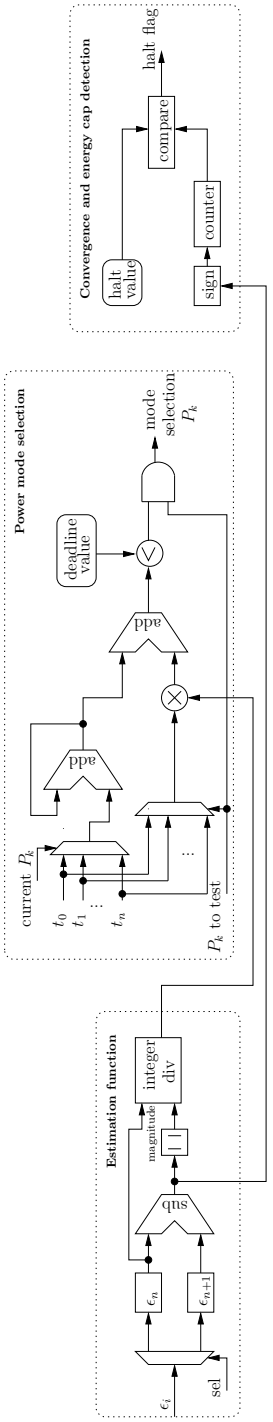


Figure 7.19: Architecture of power manager unit.

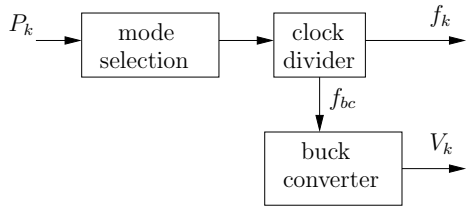


Figure 7.20: DVFS unit diagram.

timated remaining time provided by the estimation function block. The calculated predicted time is compared against the task deadline in order to validate a power mode selection. The power mode selection block essentially searches for a power mode that guarantees the task completion, and by starting the search with the lowest power mode it greedily looks for the highest energy savings available. The convergence/energy cap detection block implements the stopping criteria shown in the policy flow in Figure 7.16. This block uses as a decision metric the difference between consecutive samples of ϵ. The increments/decrements of this metric are compared against a precalculated threshold characteristic. Using two samples for ϵ in the estimation function block and the convergence/energy cap detection block corresponds to $q = w = 2$ in Algorithm 5.

The energy savings obtained by the proposed DPM policy depend upon the characteristics of the power modes used and the implementation of the DVFS block. There are numerous works on how to implement a DVFS unit, using different techniques where several tradeoffs take place: size and power overhead, mode switching speed and conversion efficiency. The work in [84] provides a study on on-chip regulators for DVFS implementation on a dedicated core. This and similar work in [85] show sufficiently fast switching regulators (voltage transition times on the order of tens of nanoseconds) for demanding applications such as LDPC and Turbo decoding.

Based upon [84] [85], we target an on-chip solution to implement the DVFS unit, shown in Figure 7.20. A mode selection signal selects the appropriate setting for a clock divider. The clock divider is based upon the architecture presented in [86] and generates a signal f_{bc} that is used by a buck converter with hysteretic control. These two last blocks generate the signals f_k and V_k that drive the decoder unit. Notice that the buck converter is the critical component in this block due to its conversion losses. We rely upon the works in [84] [85] to characterize this component in order to study the feasibility to implement the proposed control policy in a VLSI architecture.

Table 7.2: Characterization of decoder power modes.

Mode	Voltage	Frequency	Power	Slowdown factor
P_0	V_{max}	f_{max}	P_{max}	1.0
P_1	$0.9V_{max}$	$0.66f_{max}$	$0.5346P_{max}$	1.5
P_2	$0.75V_{max}$	$0.5f_{max}$	$0.2812P_{max}$	2.0
P_3	$0.68V_{max}$	$0.4f_{max}$	$0.1849P_{max}$	2.5

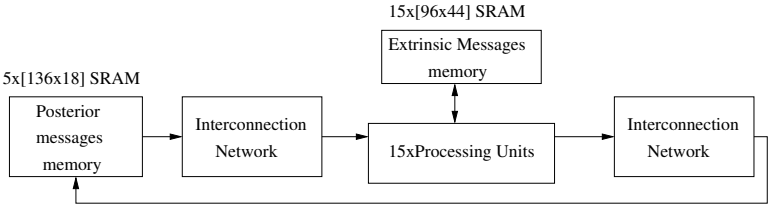


Figure 7.21: LDPC decoder architecture.

7.2.6 Results

In order to implement the proposed system several factors have to be taken into consideration: control policy tuning, workload characterization, workload prediction accuracy, prediction function complexity and available power modes that depend upon the technology being used. By control policy tuning we refer to the setting of the parameters within the policy like the energy cap E_c and the stopping criteria used. This is done in conjunction with the workload characterization, which refers to observations from the average number of iterations as a function of SNR. This provides insights into the required workload based upon the channel quality. By observing the average number of critical iterations the energy cap is adjusted to limit the time the decoder will operate at a high-power mode. Regarding the stopping criteria, these rules have to be tuned as a function of their performance in the sense of false alarms and missed detections, this tuning was addressed in Section 7.1.3.

In Table 7.2, we show the characterization of the power modes of the target system considered for LDPC decoding. The LDPC code to support is the one defined in IEEE 802.11n [5]. The target CMOS technology is of $65nm$; $V_{max} = 1.32V$ and $f_{max} = 400MHz$ are used in the following. In our simulations with this technology the static power was below 2% of the total power (refer to Figure 5.3). This is why the characterization of the power modes is dominated by the dynamic power.

Figure 7.21 shows the architecture of the LDPC decoder synthesized within the embodiment of the DPM system previously depicted in Figure 7.18. This decoder consists of 15 processing units that perform the SCMS decoding of a row in $\boldsymbol{H}$ in serial fashion as outlined in [14]. The decoder essentially updates posterior messages (quantized to 6-bits) that are distributed by an interconnection network (acting as the edges of the underlying code graph) to a set of processing units. These units process the messages along with the previous results (extrinsic messages) and write back the newly calculated posterior messages (again the messages are properly distributed by an interconnection network). This decoder achieved a throughput of 156Mbps.

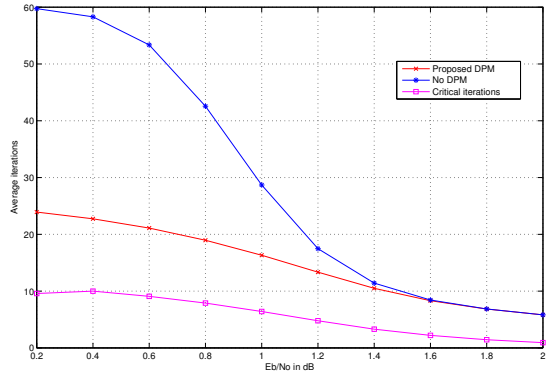
Simulations (same scenario as in Figure 7.13a) for the decoding of the code of length 1944 and rate 1/2 are shown in Figure 7.22a to depict the workload characterization for this use case and tune the DPM control policy. The average decoding iterations with no DPM are shown as well as the average critical iterations. The curve corresponding to *No DPM* implements only as stopping criterion the fulfillment of equation (2.13) or the maximum number of iterations.

By applying the DPM technique a reduction in the average iterations is observed, this is a consequence of the inherent stopping criteria within the flow of the policy. As previously noted, the stopping criteria can produce wrong decisions known as false alarms, i.e., instances of the decoding task that were halted when in fact they were able to converge within the maximum number of iterations. This represents a loss on error-correction performance. Figures 7.22b and 7.22c show the false alarm rate and the bit-error rate (BER)/frame-error rate (FER) obtained for this setup respectively. As shown as well in the Turbo decoding case that follows, the false alarms peak within a region of high workload, i.e., high critical iterations. Tuning the control policy implies making a judicious tradeoff between average performance loss and energy savings.

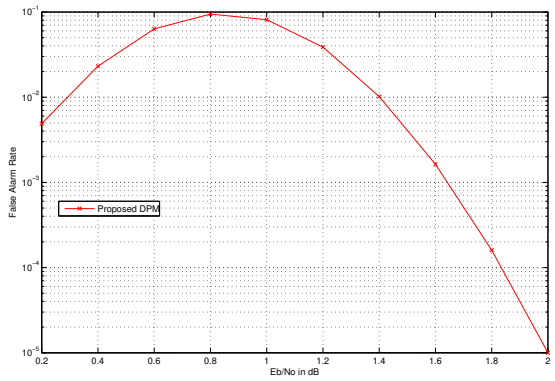
The average energy consumed by a decoder using the typical design paradigm of operating at a unique high-power mode P_0 disregarding the duration of the task is given by:

$$E_{TYP} = I_{avg} \cdot P_0 \ , \tag{7.10}$$

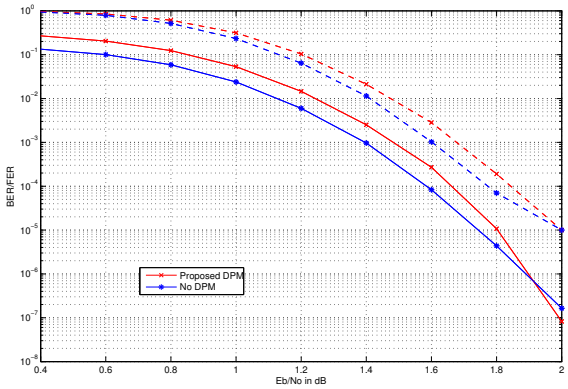
where I_{avg} is the average number of iterations for a given SNR operating point. On the other hand, the energy consumed when applying the DPM policy on a system with n power modes must incorporate the critical iterations and the possibility of switching to a low-power mode. This energy



(a) Average iterations and critical iterations



(b) False alarm rate



(c) BER/FER performance

Figure 7.22: DPM technique on LDPC code K=1944, R=1/2.

E_{DPM} has a lower bound when the lowest power mode can guarantee the completion of the task deadline, this is given by:

$$E_{DPM} = I_c \cdot P_0 + (I_{max} - I_c) \cdot \beta_{n-1} \cdot P_0 \cdot \alpha_{n-1} \ , \tag{7.11}$$

where I_c is the average number of critical iterations and I_{max} is the maximum number of iterations. The lowest power mode P_{n-1} is described by a slowdown factor α_{n-1} and a power factor β_{n-1} of the highest power mode. In this same manner, we can describe the lower bound of the energy consumed by an offline power management strategy that would operate at the lowest power mode provided the task deadline is achieved. This would be the unachievable theoretical bound for the DPM strategy. This bound is given by:

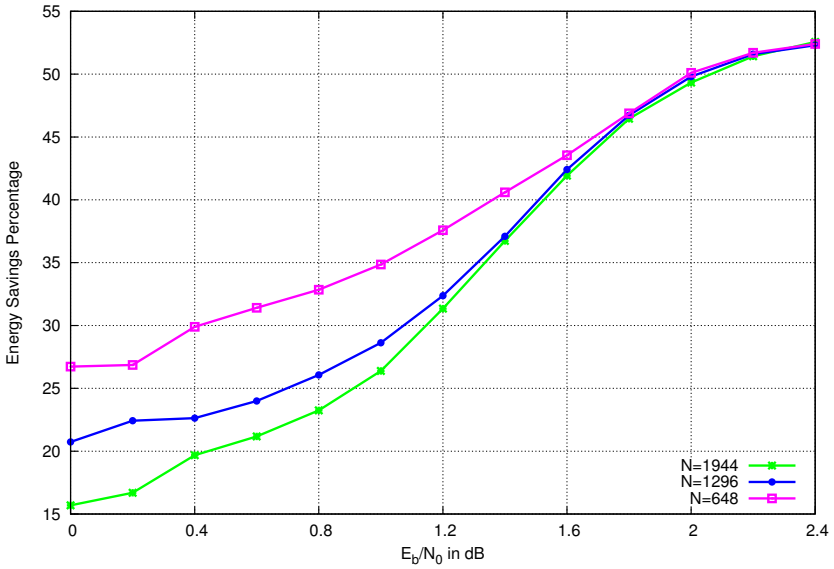
$$E_{offline} = I_{avg} \cdot \beta_{n-1} \cdot P_0 \cdot \alpha_{n-1} \ . \tag{7.12}$$

The design was synthesized with Synopsys Design Compiler and the power consumption was estimated with Synopsys PrimeTime using post-layout netlists. We combined the simulations of the proposed DPM policy using $I_{max} = 15$ with the characteristic per power mode to estimate the energy savings on several use cases. In Figure 7.23, we show the average energy savings obtained by the online algorithm (E_{DPM}) with respect to the absence of a power management strategy (E_{TYP}), i.e., constant operation at full power. Three code lengths N were used along with two coding rates in order to observe the behavior for several use cases. At a low SNR there are fewer opportunities for energy savings (20% with code rate 1/2 and 40% for rate 5/6) because of a higher decoding effort (longer critical time), but for the high SNR region up to 54% energy savings were obtained.

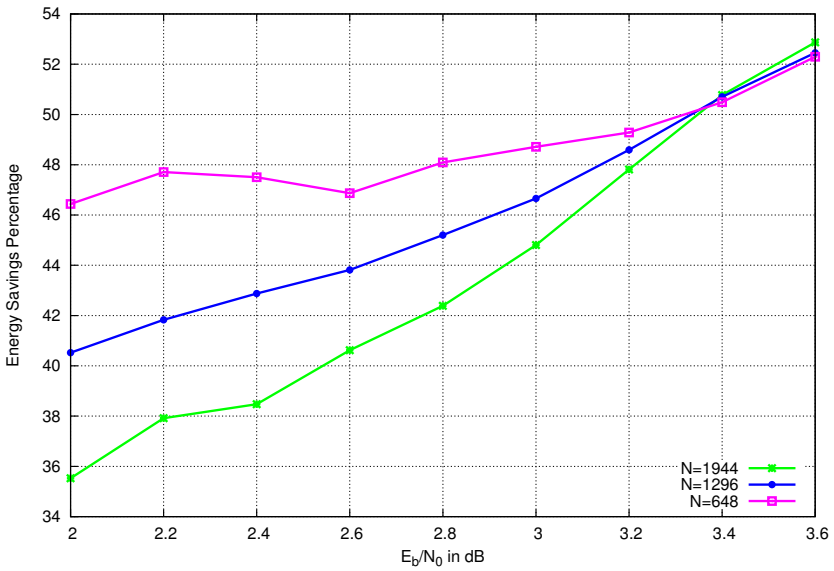
The energy savings from the proposed policy have two components: one due to the inherent stopping criteria and another one due to the system slowdown. The former dominates on the low SNR region and the latter on the mid to high SNR region. At low SNR the stopping criteria detect the potential codeblocks that are not likely to be decoded, whereas at mid to high SNR values the system takes advantage of the fast convergence in order to reduce power consumption.

The competitive ratio for this setup is $c = 2.17$, nevertheless, from the simulation results it was observed that the competitive ratio had an upper bound of $c = 0.97$. This is not surprising since competitive analysis often provides an overly pessimistic bound for the performance of algorithms.

We applied also the proposed DPM technique to an LTE Turbo decoder, depicted in Figure 7.24. This decoder uses 6 SISO radix-2 units with



(a) Rate=1/2



(b) Rate=5/6

Figure 7.23: DPM technique average energy savings on LDPC decoding.

a window length of 32 samples with a message quantization of 6-bits. It provides a throughput of 95Mbps and completes a decoding task of 8 full iterations in $65\mu s$. Because of the reduced number of iterations (compared to an LDPC decoder) only two power modes were used for the Turbo case: 1.2V at 266MHz and 0.9V at 160MHz. This setup provides a competitive ratio $c = 3.27$.

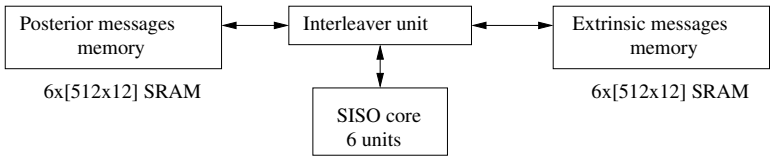


Figure 7.24: Turbo decoder architecture.

Three use cases were simulated with different code lengths K at the code rate 2/3. Figure 7.26 shows the workload characterization for these use cases as a function of SNR. The average half-iterations are shown for *No DPM*, *DPM* and the average critical half-iterations. The curve corresponding to *No DPM* uses as stopping criterion the HDA rule as outlined in [71]. Based upon these results and the error-correction performance loss, the DPM policy is tuned in order to provide the biggest gains in energy savings at the lowest performance loss. As mentioned before, these losses stem from the potential wrong decisions that a stopping criterion may take. In Figure 7.25, we show the false alarm rate for the simulated use cases. Again, the false alarms exhibit high values in the regions of intermediate SNR that correspond to the highest workloads for the decoder. Figure 7.28 shows the BER/FER loss for applying the DPM technique on each use case.

The achieved energy savings are depicted in Figure 7.27 for each use case, this would correspond to the difference percentage between E_{TYP} and E_{DPM}, equations (7.10) and (7.11), respectively. The energy savings achieved by an offline strategy $E_{offline}$ are shown as well to indicate how far the DPM policy behaves from its ideal performance. This difference is indeed an indirect measure of the prediction error from equation (7.8).

The use case of $K = 6144$ encompasses the most workload, in fact because of the characteristic of the power modes it is only after $E_b/N_0 = 0.8dB$ that the DPM technique starts delivering results. For all cases the offline policy produces around 54% of energy savings, this is asymptotically approached by the DPM technique on the high SNR region. This comes from the fact that at high SNR values the critical workload is very low, this suggests in fact that the blocks enter convergence relatively fast, exactly what would be expected at a good channel quality. The achieved energy savings

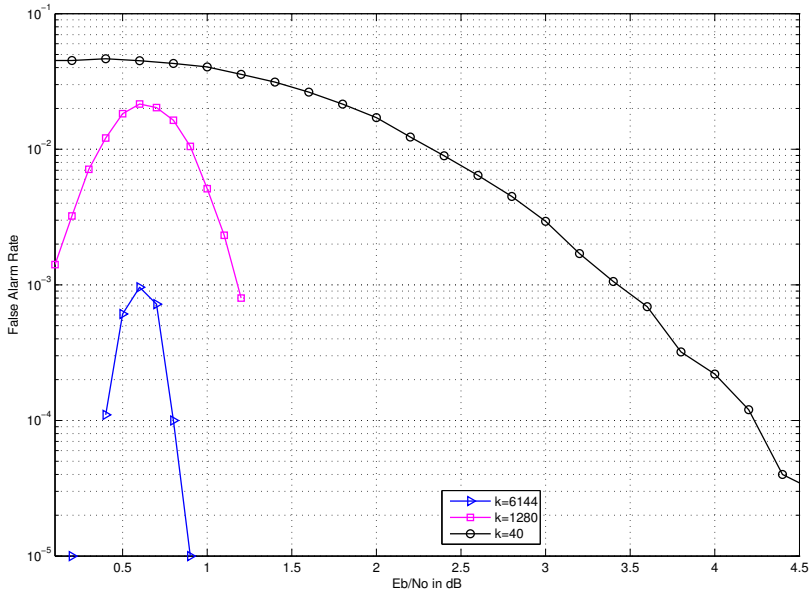


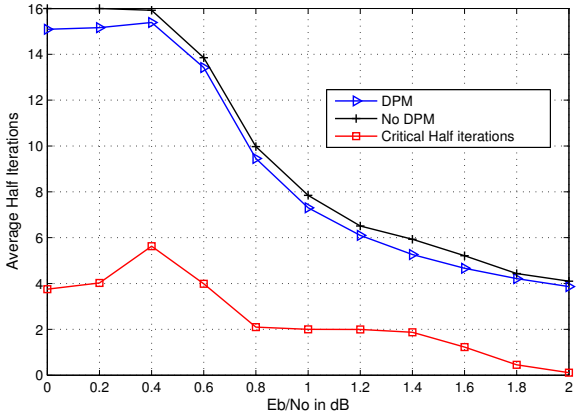
Figure 7.25: DPM technique on Turbo code: false alarm rate.

Table 7.3: Area and power comparison for DPM system.

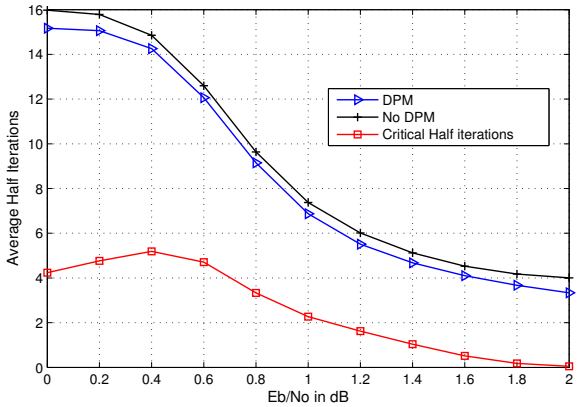
Component	Area $[mm^2]$	Power $[mW]$
LDPC decoder (No DPM)	0.85	70
Turbo decoder (No DPM)	0.62	180
Power manager	0.08	5
DVFS unit	0.12	15

vary between 34% to 54% for all use cases depending upon the channel quality.

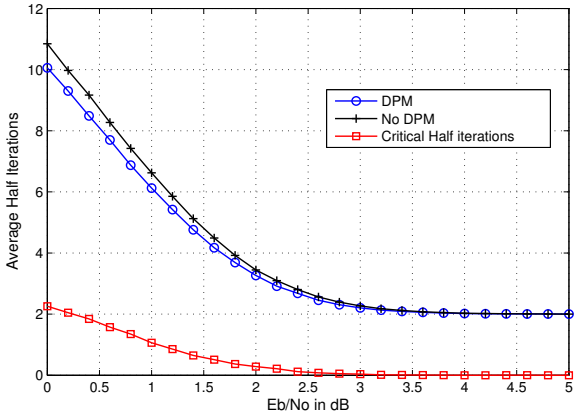
The area and power overheads for applying DPM on these decoders are revealed in Table 7.3. For both setups (LDPC and Turbo) the same DVFS unit was used. This unit is characterized from the results presented in [84] [85]. From those works we extract the data for a buck converter with a switching frequency of 100MHz and a conversion efficiency in the range of 75%-87% with an output voltage range of 0.9V-1.3V. The components involved are compared in order to assess the impact of applying DPM, this should be considered alongside the error-correction performance loss and the energy savings achieved for all cases.



(a) K=6144

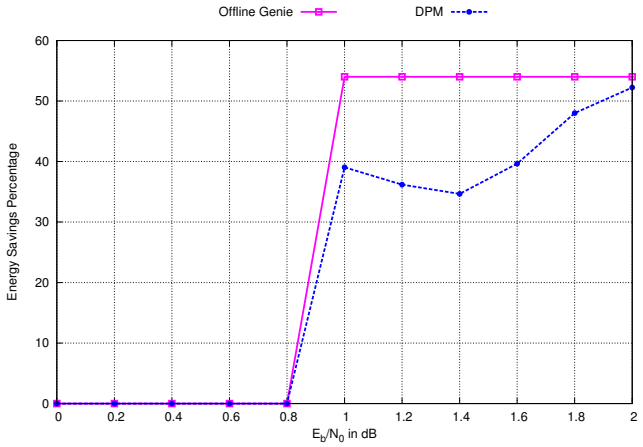


(b) K=1280

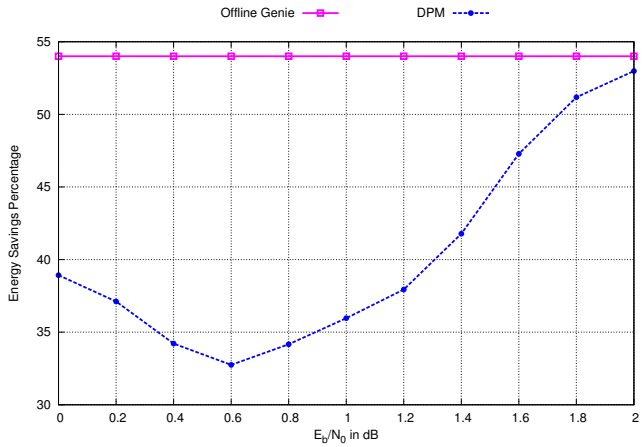


(c) K=40

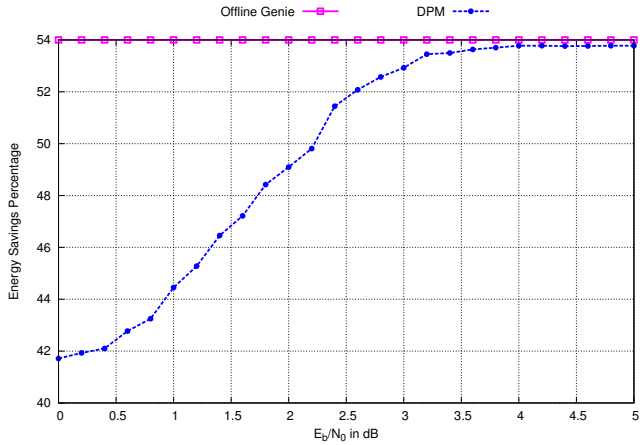
Figure 7.26: Turbo decoding workload visualization: average half-iterations and critical half-iterations.



(a) K=6144

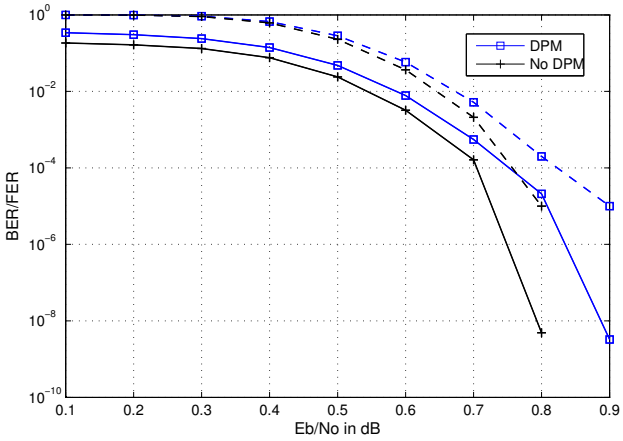


(b) K=1280

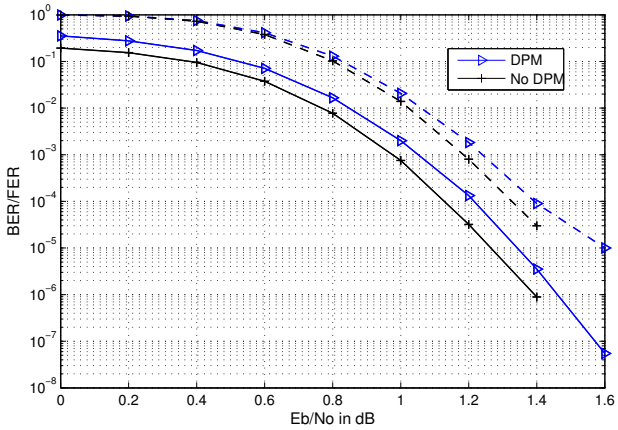


(c) K=40

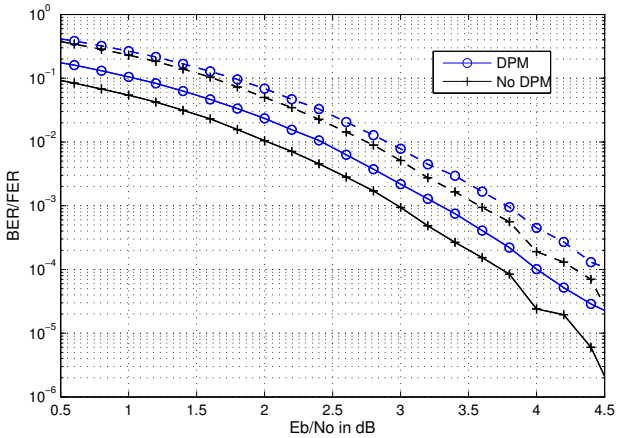
Figure 7.27: DPM technique average energy savings on Turbo decoding.



(a) K=6144



(b) K=1280



(c) K=40

Figure 7.28: DPM technique on Turbo code: BER/FER performance.

Table 7.4: Comparison of energy savings techniques.

	Work	Proposed	[73]	[74]	[87] (SC)	[87] (PR)
LDPC	Energy savings %	54	37	30	-	-
	SNR loss [dB]	0.07	N/A	0.05	-	-
Turbo	Energy savings %	54	-	-	17.5	24.5
	SNR loss [dB]	0.08	-	-	0.34	0.48

We compare the achieved average energy savings with works from the prior art in Table 7.4. The SNR loss reported in the table corresponds to the point at $BER = 10^{-5}$ for the corresponding code from each work. Even though the codes among the cited works are different, the SNR loss provides a measure on the impact in performance for each applied power savings technique. The work in [87] analyses individual techniques proposed for energy reduction in Turbo decoding. Among them we show the reported achieved energy savings for reduction in the number of paths (PR) and the LLR stopping criterion (SC). We acknowledge that in [87] several of the analyzed techniques therein were combined and savings of up to 66% were reported.

7.3 Conclusion

In this chapter, power management techniques for iterative decoders were presented. Iteration control for LDPC codes was discussed. Even though iteration control is relevant only for the low SNR region of the performance curves it is an important technique studied for the purpose of avoiding useless decoder operation. We provided insights into the performance of several control rules in terms of the detection of undecodable blocks as false alarms and missed detections. We proposed an iteration control law that monitors two decision metrics that allowed a notorious decrease in the missed detection rates found among the prior art. By decreasing the average number of missed detections we argued that an iteration control law is more energy efficient. Furthermore, by observing the tuning of the control rules the performance can be selected according to the target application.

In addition, we considered the dynamic power management of iterative decoders. We proposed an online policy that is aided by the dynamics of the decoding process that can be extracted from a particular convergence metric. By making a recurrent estimation on the required decoding effort the policy adjusts the performance of the system so that it minimizes the energy consumption while the task deadline is satisfied. A judicious selection of a power mode is carried out at runtime by a power manager that considers the decoding task deadline and the predicted remaining decoding time once the decoder has entered a convergence mode. The proposed technique has been applied to the decoding of LDPC and Turbo codes. For the case of LDPC codes the number of unsatisfied parity-check constraints was used as a decision parameter, while for the Turbo case the total number of hard-decision changes in the posterior messages was used. Both metrics were monitored per iteration/half-iteration in order to make decisions about the system operation. Several use cases were simulated in order to observe the achieved energy savings as a function of codeblock length and code rate. Energy savings of up to 54% were achieved with a relatively low impact on error-correction performance.

Chapter 8

Unified Decoders and Future Work

Due to the increasing number of mobile wireless communication standards to be supported by nomadic devices, there is a need to design highly reusable and flexible receivers. These devices must provide a seamless transition within different network technologies depending upon physical location, moving speed and required services. Moreover, should these devices be connected to more than one network, concurrent data streams should be processed.

In this chapter, we address the challenges to conceive a unified architecture for the decoding of both Turbo and LDPC codes. Due to the dissemination of these codes among current and future communication standards, a flexible and efficient multi-mode decoder is of interest. We briefly disclose the design challenges for a dual-mode Turbo/LDPC decoder with concurrent processing of data streams in order to describe how the work in this dissertation may be extended.

8.1 Prior Art

In this section, we outline in a compact way the relevant previous art that addresses unified and common architectures for the decoding of Turbo and LDPC codes. The major points to discuss include two architectural

paradigms that achieve the required flexibility and the formulation of decoding algorithms that may act upon both types of codes.

Application-specific instruction-set processors.
A common theme found in the previous works that address flexibility and multi-mode functionality has been the paradigm of application-specific instruction-set processors (ASIP). These devices consist of an optimized data path and a reduced instruction-set that offers programmability and good performance. These processing architectures are gaining interest to be used within programmable platforms for software defined radios (SDR). As suggested by its name, these platforms provide a fully programmable solution for radio baseband architectures. The works in [32] [33] describe a unified architecture for ASIPs that support Turbo, LDPC and convolutional codes. Authors in [88] consider a multi-user approach instead, where multiple decoder instances are used to support concurrent decoding tasks.

Application-specific integrated circuits.
Due to the computational complexity of the iterative decoding of these codes along with high throughput demands and high energy efficiency dedicated unified architectures have been proposed as well. These architectures offer high efficiency and hardware reuse at the expense of lower flexibility when compared to their ASIP counterparts. Works in [89] [90] [91] [92] describe similar architectural constructs based upon flexible functional units and reconfigurable permutation networks. The work in [92] goes beyond by implementing the distribution of messages among processing units by means of a network on chip (NoC).

Unified decoding algorithms.
Based upon the previous works we describe the decoding strategies that enable common architectures for both types of codes. As mentioned in Chapter 2, Turbo and LDPC codes can be described in general as codes on sparse graphs and their decoding can be explained as an instance of the sum-product algorithm applied to their respective graphs. Two main approaches have been followed to unify the decoding of these codes:

1. LDPC decoding transformed into a Turbo decoding problem: The work in [13] [27] showed how an LDPC code can be interpreted as the parallel concatenation of single parity-check codes. The MAP algorithm can be applied to the 2-state trellis representation of each parity-check where each state represents the value of the parity-check according to the variable

nodes that particular row in **H** is connected to. The works in [89] [91] [92] follow this approach to design flexible SISO units that essentially apply the BCJR algorithm on a trellis representation.

2. Turbo decoding transformed into an LDPC decoding problem: A Turbo code is typically represented by the concatenation of convolutional codes and the trellis diagram of each component code. However, recent work has shown the ways in which a Turbo code can be analyzed from a block and low-density parity-check code point of view. Works in [90] [93] have shown how to construct a parity-check matrix from a Turbo code. Indeed, such approach provides a better understanding of the relationship between these types of codes. After obtaining a parity-check matrix for a Turbo code the typical sum-product algorithm may be applied to the resulting code graph. This approach is adopted in [90] where performance losses are reported mainly due to transformations necessary on the resulting parity-check matrix.

The previous works have concentrated on maximum hardware reuse and in fact report attractive implementation area gains when compared to the separate instantiation of the individual decoders they support. In the following, we outline specific aspects to be considered in order to achieve high energy efficiency and the possibility to support concurrent coded streams.

8.2 Dual-mode Decoder Design

In this section, we target the design of a dual-mode decoder for the decoding of the LTE [4] Turbo code and the IEEE 802.11n [5] LDPC code. We assume there is a requirement to support concurrently one coded stream for each type of code. Such assumption may arise from a multi-mode device that is required to operate on both networks, one simple scenario may be the operation of a mobile device that is downloading data through a cellular LTE network and at the same time operating as an access point for a 802.11n local area network. Figure 8.1 illustrates the mentioned scenario.

Admittedly, any optimal decoder design should be competitive to the separate instantiation of the supported decoders, the support of concurrent coded streams may limit the level of hardware reuse. In the following, we formulate the design issues for this case study with the purpose to lay down the aspects of future work for this research.

Figure 8.1: Dual-mode receiver scenario.

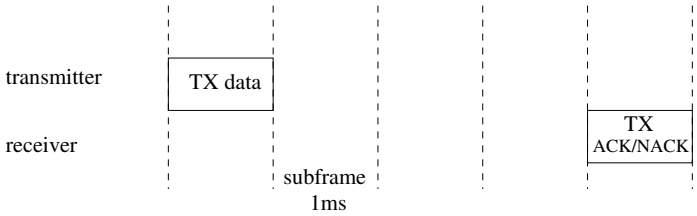


Figure 8.2: LTE ACK/NACK timing requirements.

8.2.1 Requirements

The Turbo coding scheme used in 3GPP-LTE [4] features the parallel concatenation of two 8-state convolutional codes separated by a quadratic permutation polynomial (QPP) interleaver. The native coding rate is 1/3 and 12 tail-bits are added to terminate the trellis. Puncturing is used to change the coding rate up to 0.95. A total of 188 different block sizes N are defined, with $40 \leq N \leq 6144$.

An LTE User Equipment (UE) must comply with key performance constraints in order to complete hybrid automatic repeat request (HARQ) processes. LTE takes the advantages of adaptive coding and modulation and HARQ in order to maximize the link throughput under poor channel conditions. The latency requirement for a UE in order to provide an acknowledgement ACK/NACK upon a decoded frame is shown in Figure 8.2.

On the other hand, the UE must comply with a peak throughput requirement of 75Mbps per MIMO stream. Up to 4x4 MIMO configurations are supported in LTE.

The structured LDPC codes defined in IEEE 802.11n [5] contain several parity-check matrices for several use cases that are characterized by various codeblock lengths and coding rates. There are a total of 12 matrices for 12 use cases, each matrix $\boldsymbol{H}_{M\times N}$ consists of an array $m_b \times n_b$ of $Z \times Z$

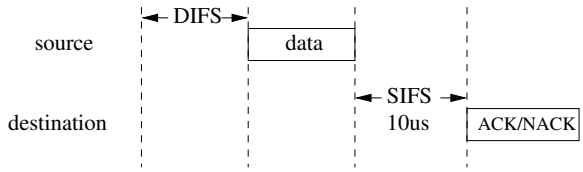


Figure 8.3: 802.11n ACK/NACK timing requirements.

blocks. The defined values in the code are $Z \in \{27, 54, 81\}$, $n_b = 24$ and $m_b \in \{4, 6, 8, 12\}$ and they depend upon the use case with codeblock lengths $N \in \{648, 1296, 1944\}$ and coding rate $R \in \{1/2, 2/3, 3/4, 5/6\}$.

The latency requirement for a high throughput station (HT-STA) in 802.11n is shown in Figure 8.3. The short inter-frame spacing (SIFS) defines the allotted time to generate an ACK/NACK message. The distributed IFS (DIFS) is the time a station uses to sense a clear radio before starting a new transmission sequence.

The HT-STA must comply with a peak throughput of 150Mbps per MIMO spatial stream (at a channel bandwidth of 40MHz), where up to 4x4 MIMO systems are supported.

For both systems the latency and throughput constraints impose the dimensioning specifications of a baseband processor. It is required to estimate the processing latency and throughput required per processing block within the baseband chain. This chain usually contains blocks like channel estimation and equalization, demodulation, channel decoding, rate matching and frame construction. Hereafter, we assume a 60% of the baseband processing time to be allocated to channel decoding.

8.2.2 Dimensioning

As mentioned in Chapter 6, we refer to *dimensioning* to the selection of the level of parallelism required for the target system. For a decoder operating at a frequency f_{clk} a latency constraint involves the completion of a decoding task under a deadline d:

$$\frac{D}{f_{clk}} \leq d \,, \tag{8.1}$$

where D cycles are consumed. On the other hand, a throughput Γ constraint for decoding a codeblock of length N is given by:

$$\frac{N \times f_{clk}}{D} \geq \Gamma \,. \tag{8.2}$$

We consider now the latency for each decoder assuming the use of the BCJR kernel for MAP [20] decoding on the Turbo case and the general description of latency for an LDPC decoder given in Chapter 4 on equation (4.2). A MAP decoder traverses the trellis of a code to estimate the possible state transitions of the decoder at each time stamp, this is done in both forward and backward directions. Values known as *state* and *branch* metrics are generated at each node of the trellis and are used to calculate the output posterior messages. Traversing the whole trellis for long codeblocks involves the storage of long chains of metrics, therefore typically a *sliding-window* approach is followed instead, [94]. In this approach, the trellis is processed in a fixed length that advances within time, saving on storage requirements but introducing delays that are required in order to estimate the metrics at the edges of the window. This is done by so-called *training* or *dummy* calculations.

MAP processing is typically performed in radix-2 architectures or doubling the throughput with radix-4 computations by compressing the trellis in time, [95] [96].

The decoding latency of a codeblock of length N in clock cycles for a decoding task of I iterations is given by:

$$D_{Turbo} = 2 \times I \times (\frac{N}{P} + 2W) \tag{8.3}$$

for P MAP radix-2 units using a window length W. For the case of a radix-4 unit, this latency is reduced by 50%.

On the other hand, structured LDPC codes with an expansion factor Z can be processed in parallel fashion, the decoding latency in clock cycles of a codeblock by P processing units performing I decoding iterations is given by:

$$D_{LDPC} = I \times m_b \times \frac{Z}{P} \times L_c \,, \tag{8.4}$$

where L_C is the number of cycles consumed for processing a row in $\boldsymbol{H}$.

An inspection on the values provided by each specification reveals that for LTE the dominating constraint is the throughput requirement whereas for 802.11n the latency is the dominating one. We use the term *dominating* in the sense that a performance constraint reveals the necessary processing units P when considering equations (8.3) and (8.4) in both constraints (8.1) and (8.2). In Figure 8.4, we show the required number of processing units for individual decoders in order to comply with their respective dominating constraint. For Turbo decoding a radix-2 MAP unit is used with window

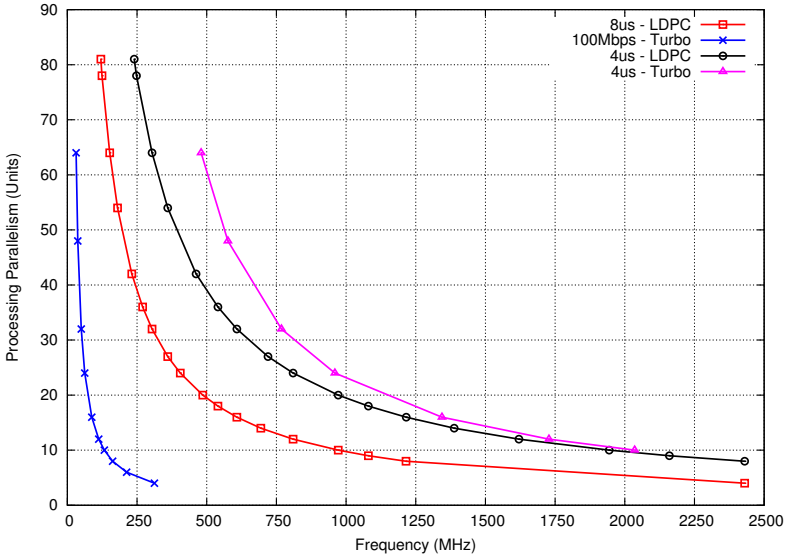
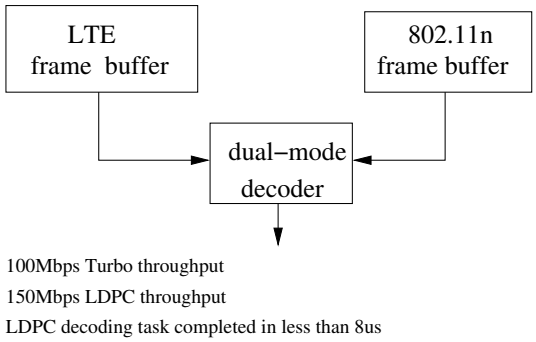


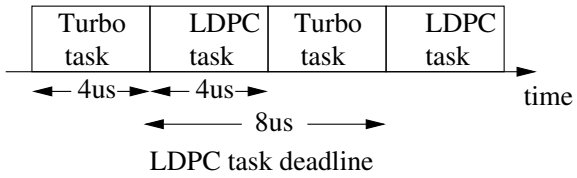
Figure 8.4: Processing units and required frequency per decoder mode constraint.

length $W = 64$ and 6 full iterations; for LDPC decoding a unit that processes a row of degree d_c in $d_c + 3$ cycles with 8 iterations is used. The use cases shown are: LTE with $N = 6144$ and $R = 1/3$; and 802.11n with $N = 1944$, $d_c = 8$ and $R = 1/2$. On each curve of the figure the used constraint is shown. Here the tradeoffs of area and power can be roughly estimated.

Nevertheless, for the case of a system that supports both coded streams in concurrent fashion the latency constraints would change. An LTE Turbo decoder that complies with the throughput requirement exhibits a latency between $40\mu s$ and $60\mu s$. Since the latency requirement for an LDPC decoder in 802.11n is in the order of less than $10\mu s$ this inevitably suggests that a Turbo decoding task must be preempted or halted in order to schedule a higher priority LDPC decoding task. This would lead to a duplication of the memory requirements as the context of the halted task must be saved. This of course eliminates any possibility to reuse hardware and hence achieve an optimum implementation area. Therefore it is compelling to explore the design space of the decoders after reducing their latency requirements so that both Turbo and LDPC decoding tasks may be executed and completed one after another and still respect their required specifications. Figure 8.5 shows the time-multiplexed operation of a dual-mode decoder with incoming dual frame buffers as input streams.



(a) Dual-mode decoder operation



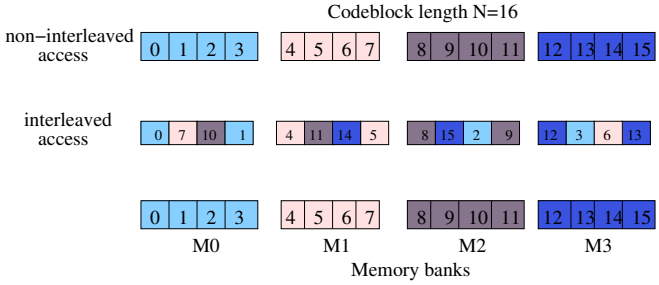
(b) Time allocation of tasks

Figure 8.5: Dual-mode decoder characteristics.

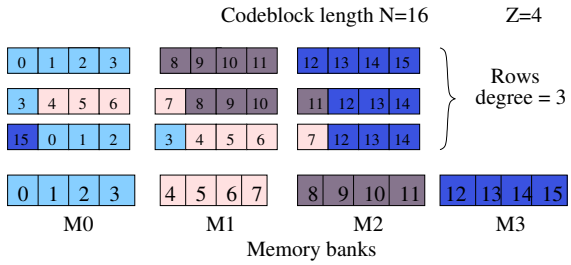
8.3 Future Work

The work presented in this thesis can be extended to address the design space exploration of a unified decoder architecture as shown in the previous section. Namely, the following relevant points can be addressed:

1. Memory architecture exploration: As argued in the previous chapters, the memory subsystem of decoders for Turbo and LDPC codes is critical in terms of implementation area and power consumption, [97] [98] [99]. For a dual-mode decoder reusing this subsytem represents the highest opportunities to improve on implementation area. Therefore, it is of interest to explore the ways in which the memory may be partitioned and how the data can be allocated to enable a high reuse factor. The particular memory access requirements of each code impact the previously mentioned design choices since different access patterns may cause read/write conflicts between competing processing units. For the case study considered in the previous section, the LTE QPP interleaver guarantees contention-free access of P processing units to P memory banks when P is a factor of the interleaver length, [100]. For the case of structured LDPC codes the memory access



(a) Turbo code



(b) LDPC code

Figure 8.6: Memory access characteristic for a QPP Turbo code and a structured LDPC code.

configurations have been studied in Chapter 5. Figure 8.6 shows examples of the required memory access characteristic for a QPP Turbo code and a structured LDPC code.

In Figure 8.6a a Turbo code of length $N = 16$ is processed in 4 windows and the data is distributed among 4 memory banks. For the first decoding half-iteration the non-interleaved access is used, then for the second half-iteration the interleaved data is fetched in a contention-free fashion. For the case of LDPC codes, Figure 8.6b shows a codeblock of length $N = 16$ distributed among 4 memory banks and an expansion factor of $Z = 4$, which means that up to 4 rows can be processed concurrently. With a row degree $d_c = 3$ example access patterns are shown, contention-free access is provided by the micro and macro placement of data described in Chapter 5. What is then the optimal memory partition and data allocation that can maximize the reuse of memory between each decoding mode?

2. Decoding kernels: Optimal MAP decoding can be performed on both types of codes with the BCJR algorithm as described in the previous

art in Section 8.1. As discussed in Section 4.3 for the LDPC code case, a BCJR processing unit provides excellent error-correction performance but at the cost of higher implementation area and energy expenditure. Nonetheless, a plurality of processing units in a parallel architecture constitute just a fraction of the memory footprint. Therefore, it is of interest to disclose the impact on power consumption when combining BCJR and SCMS (or any LDPC decoding kernel of similar performance/energy cost) processing units in a dual-mode decoder. For instance, it should be possible to operate the decoder on lower frequencies during LDPC mode given the lower workload an MS kernel exhibits compared to the BCJR kernel. What quantitative advantages would this design choice bring about?

3. Power management: Given the iterative nature of both Turbo and LDPC decoding and the joint processing of both types of decoding instances, it is of interest to study the possibilities to operate the system at low power modes depending upon different factors such as: input frames arrival times, workload estimation, use case (determined by code length and rate) and required throughput among others. Which power management techniques can be formulated for this type of decoding architectures?

Chapter 9

Concluding Remarks

In this dissertation, we have addressed important aspects for the realization of energy efficient LDPC decoders. We have provided insights and contributions at different levels that are of interest for VLSI designers skilled in the art of channel decoders.

Firstly, we have considered the decoding kernels that offer different tradeoffs in terms of performance, complexity and energy consumption. We observed how a particular kernel choice can impact design parameters as memory requirements, performance metrics like convergence speed and error-correction as well as overall task energy consumption. A thorough exploration of these kernels is therefore compelling for optimal designs. Furthermore, we described a proposed method for optimizing the syndrome verification in order to speedup the decoding task.

Secondly, at the architectural level, much discussion was focused on the memory subsystem. Indeed, by means of different results within this dissertation we showed the relevance of this subsystem in terms of implementation area and energy consumption. We described several characteristics of this subsystem that are important for achieving both flexibility and energy efficiency besides a reduced footprint. Particularly, the topics of memory partition and data allocation play an important role for achieving these mentioned goals. In addition, we elaborated on architectural descriptions of a multi-mode decoder that features a compact and flexible memory design that enables a low complexity interconnection network, a high performance

decoding kernel and high energy efficiency when compared to the state-of-the-art.

Finally, at the system level, we considered the topic of power management. We analyzed the performance of stopping criteria and proposed a control policy that enhances the energy efficiency of the previous art. We showed how these criteria are only relevant in the low SNR region of a decoder. Our analysis on the tuning of these decision rules has shown the tradeoffs between false alarms (penalties of a rule usually materialized as retransmissions) and missed detections (futile energy spent on an undecodable block). In summary, for applications that include power-limited devices and bad channel conditions a tuning of a stopping criterion should consider the previous tradeoff.

As a second point on power management, we proposed an online policy that adjusts a power manageable iterative decoder so that a timing deadline is guaranteed at a low energy expenditure. We provided implementation details for such policy on both LDPC and Turbo decoders. Throughout this work we have provided implementation results on relevant technologies along with comparisons with representative work in order to support the quantitative advantages of our contributions.

Finally, we addressed the topic of unified decoder architectures in order to introduce the directions in which this research can be extended. We specified at least three points of this topic that can benefit from the ideas discussed throughout this dissertation.

Appendices

Appendix A

Résumé étendu

This is a French extended summary of this dissertation required by the publisher.

A.1 Introduction

Les techniques de décodage itératif pour les codes modernes dominent actuellement le choix pour la correction des erreurs dans une pléthore d'applications. Les Turbo codes, présentés en 1993 [1], ont déclenché une révolution dans le domaine du codage de canal parce qu'ils permettent de s'approcher de la limite de Shannon. Ensuite, les codes LDPC (low-density parity-check) [2] ont été redécouverts.

Ces codes sont actuellement omniprésents dans le contexte des communications mobiles sans fil, entre autres domaines d'application. Par exemple, les Turbo codes sont utilisés dans le standard de téléphonie cellulaire de troisième génération 3GPP Universal Mobile Telecommunications System (UMTS) [3] et son évolution Long Term Evolution (LTE) [4]. D'un autre côté, nous pouvons trouver les codes LDPC dans les standards de Wireless Local/Metropolitan Area Networks (LAN/MAN) (IEEE 802.11n [5] et 802.16e [6]) ainsi que la la Wireless Personal Area Networks (PAN) (IEEE 802.15.3c [7]). D'autres applications comprennent la transmission de vidéo par satellite de seconde génération Digital Video Broadcast (DVB-S2 [8]).

Les dispositifs nomades pour les communications mobiles sont générale-

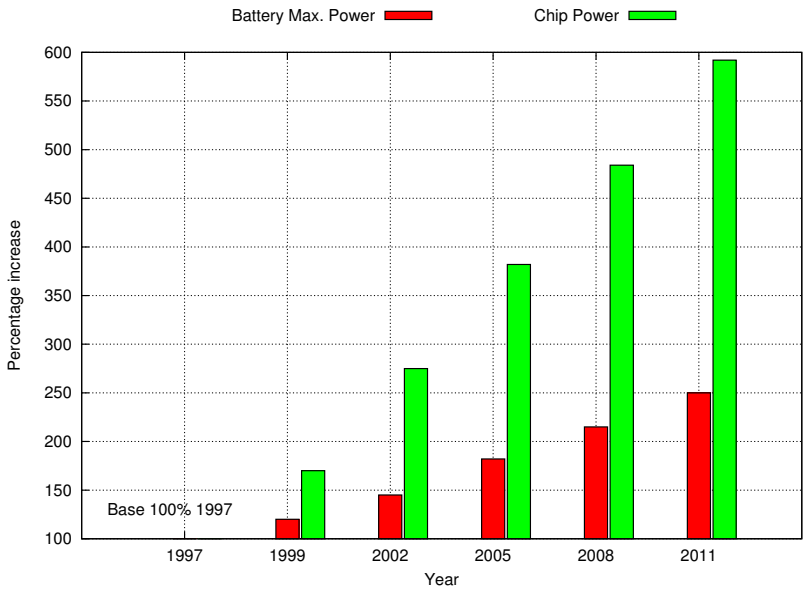


Figure A.1: Consommation d'énergie.

ment alimentés par des batteries et ils ont besoin d'une haute efficacité énergéti-que et d'une haute performance dans une surface optimale. De plus, ces dispositifs doivent être flexibles pour utiliser plusieurs standards et assurer une transition harmonieuse entre les réseaux en fonction de l'emplacement physique, la vitesse de déplacement et les services requis. La complexité des calculs effectués par ces dispositifs est augmentée en fonction de techniques plus robustes de traitement du signal intégrées sur chaque technologie de réseau. L'écart entre les améliorations de la technologie des batteries et la puissance des puces électroniques spécifique (ASIC) et System-on-Chip (SoC) est illustré dans la Figure A.1 (source de données [11] [12]). La capacité maximale des batteries affiche une augmentation de 10-15% par année mais la demande de puissance de la puce augmente beaucoup plus vite, 35-40% par année.

Il est clair que la technologie des batteries actuelles ne peut pas faire face aux exigences pour le traitement du signal requis par les technologies de réseau futurs. Par conséquent, il existe un besoin constant pour une meilleure efficacité énergétique. En outre, les décodeurs itératifs (Turbo et LDPC) sont généralement responsables d'une part importante de la consommation d'énergie dans la chaîne de traitement en bande de base d'un récepteur sans fil.

A.1.1 Contributions

Dans cette thèse, l'accent est mis sur les aspects et défis pour la conception des décodeurs VLSI à basse consommation destinés à la communication sans fil. Nous considérons les aspects des décodeurs LDPC. Nos contributions sont principalement réparties entre trois niveaux d'abstraction:

Niveau algorithmique
Au niveau algorithmique nous étudions les compromis entre la performance, l'efficacité énergétique et la surface pour les différents algorithmes de traitement de la contrainte de parité. Ceci correspond aux Chapitres 3 et 4.

Niveau architecture
Au niveau de l'architecture nous considérons la conception des mémoires. Ce point est particulièrement important pour la consommation et la surface finale du décodeur. Nous présentons l'implémentation d'un décodeur multi-mode efficace en termes d'énergie consommée. Ceci correspond aux Chapitre 4, 5 et 6.

Niveau système
Nous proposons des stratégies pour la gestion dynamique de la puissance pour les décodeurs Turbo et LDPC. Ces stratégies sont basées sur le contrôle au niveau de l'itération et de la prédiction du nombre d'itérations. Ce sujet est développé dans le Chapitre 7.

Le Chapitre 1 introduit la problématique de la thèse et sert d'introduction. Le deuxième chapitre introduit les notions de capacité du canal et du codage de canal pour comprendre le concept de décodage itératif. Le Chapitre 8 propose une ouverture vers des décodeurs unifiés pouvant traiter à la fois les codes Turbo et LDPC. Le Chapitre 9 conclut la thèse.

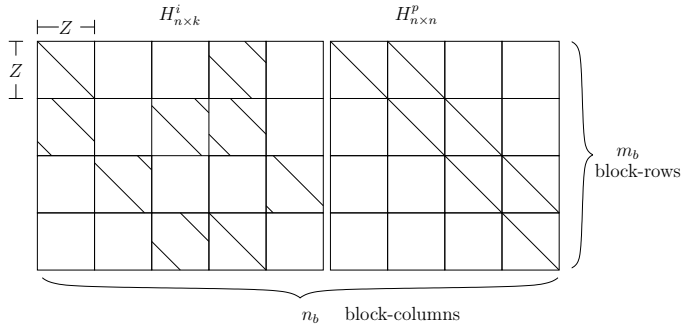
A.2 Décodage des codes LDPC

Les codes LDPC, inventés par Gallager [2], sont des codes linéaires définis par une matrice de parité creuse $\boldsymbol{H}_{M\times N}$ sur $\mathbb{F}_2$. Cette matrice définit M contraintes de parité sur N symboles de code. Un mot de code satisfait:

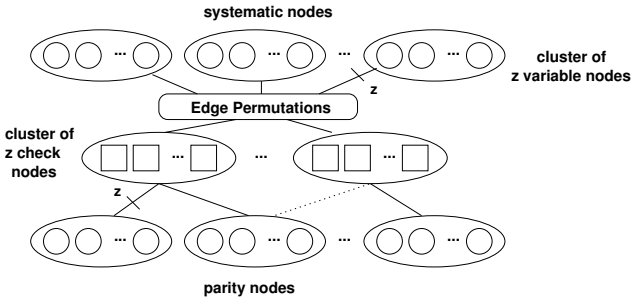
$$\boldsymbol{H}\boldsymbol{c}^T = \boldsymbol{S} = \boldsymbol{0}\,, \tag{A.1}$$

où $\boldsymbol{S}$ est le *syndrome*.

Le code peut être défini par un graphe de Tanner. Dans un graphe de Tanner chaque symbole du mot de code est représentée par un nœud *de variable* et chaque contrainte de parité est représenté par un nœud *de parité*. Les éléments non nuls dans $\boldsymbol{H}$ déterminent la connexion (arêtes) entre les nœuds. Les codes utilisés dans les standards de communication adoptent une caractéristique structurelle pour réduire la complexité introduite par le caractère aléatoire de la matrice de parité. Les codes *architecture-aware* [13] et les codes LDPC quasi-cycliques [26] (QC-LDPC) sont constitués de couches formées par des sous-matrices de dimensions $Z \times Z$. Z est un facteur d'expansion qui montre le degré de parallélisme disponible. La sous-matrice peut être une matrice nulle ou une matrice identité décalée. La Figure A.2 montre une matrice de parité structurée et le graphe correspondant. Chaque arête dans le graphe regroupe Z nœuds. Cette caractéristique permet l'utilisation d'architectures semi-parallèles.



(a) Matrice de parité



(b) Graphe de Tanner

Figure A.2: Exemple de code LDPC structuré.

A.2.1 Algorithmes de décodage

Les codes LDPC sont généralement décodés de manière itérative par l'algorithme *sum-product* [24]. Pendant le processus de décodage, des messages sont échangés entre les nœuds de variable et de parité. Cet échange est illustré dans la Figure A.3.

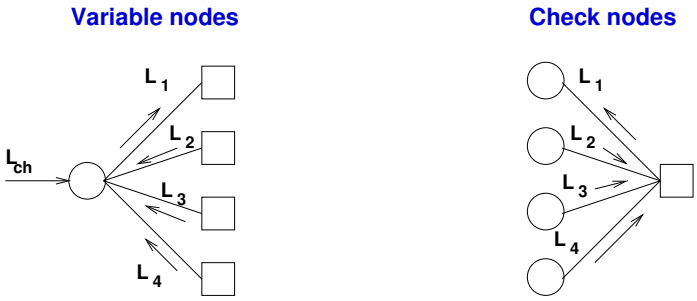


Figure A.3: Exemple d'échange de messages sur les noeuds d'un graphe.

Chaque nœud effectue un calcul et génère un nouveau message. En général les nœuds de variable font une somme de messages. D'autre part, les nœuds de parité font le traitement des contraintes de parité. Il y a plusieurs algorithmes de différents niveaux de complexité: *sum-product* (SP) [24], *min-sum* (MS), *offset-MS* (OMS) et *normalized-MS* (NMS) [28]. L'algorithme *self-corrected-MS* (SCMS) [14] présente un bon compromis entre la complexité et la performance. La Figure A.4 montre la performance (taux d'erreur ou bit-error-rate BER) de chaque algorithme pour le décodage des codes LDPC dans le standard IEEE 802.11n à travers le canal AWGN avec une modulation QPSK; la taille des mots utilisée est 1944 avec un taux de codage 1/2 et 60 itérations maximum.

La Figure A.5 montre les résultats d'une implémentation d'intégration à très grande échelle (VLSI) des différents algorithmes de faible complexité dans une technologie CMOS de $65nm$. L'implémentation correspond à un processeur série avec une quantification de message sur 6-bits. Les résultats incluent la surface et la consommation moyenne d'énergie par itération. L'algorithme avec la plus grande surface et consommation d'énergie est le BCJR [20] [13] [27]. Cet algorithme présente des calculs de haute complexité. D'autre part, l'algorithme MS présente la surface la plus faible en raison de calculs de faible complexité. Toutefois, l'algorithme SCMS présente une réduction de la consommation d'énergie en raison d'une réduction du nombre de messages échangés à chaque itération. Effectivement l'algorithme

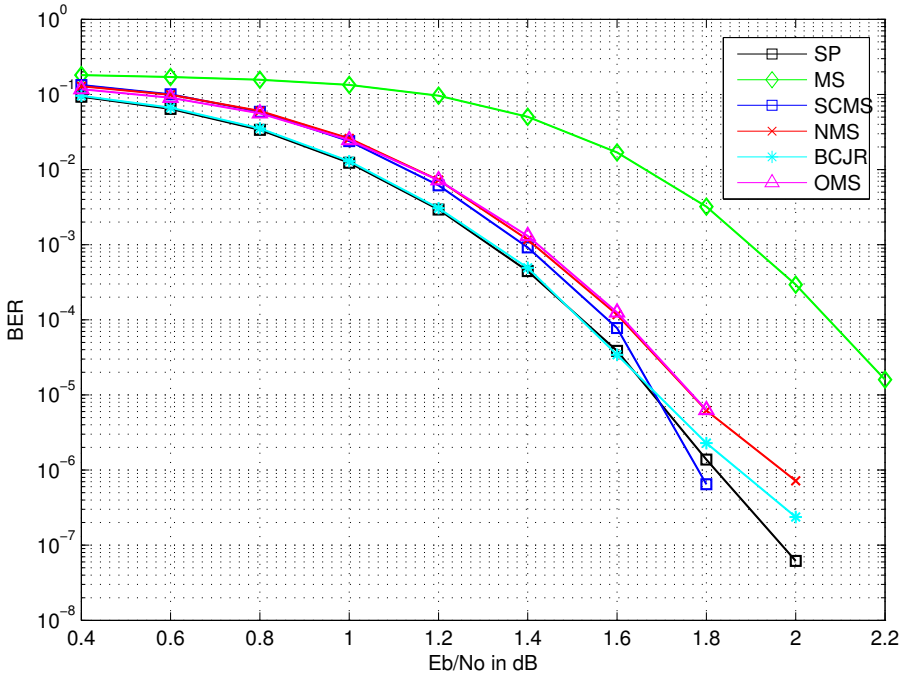


Figure A.4: Taux d'erreur des algorithmes.

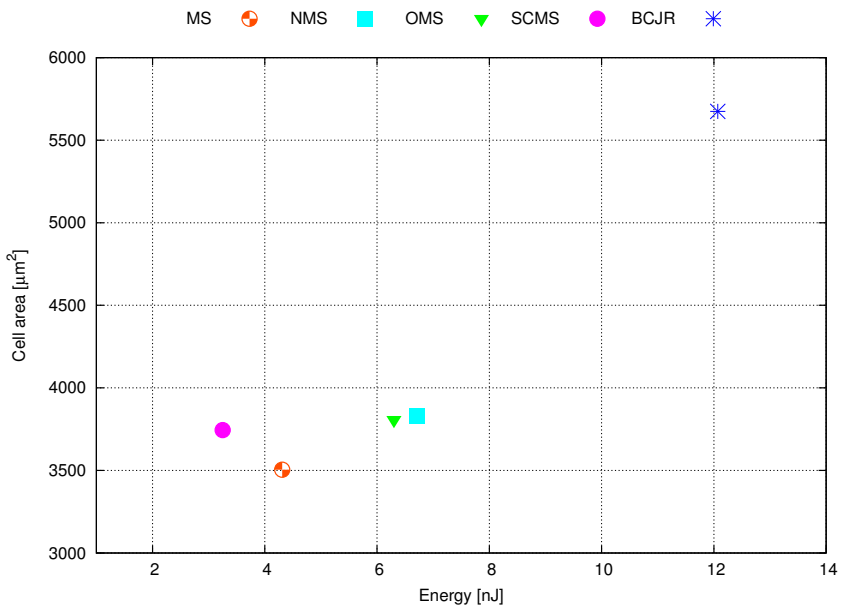
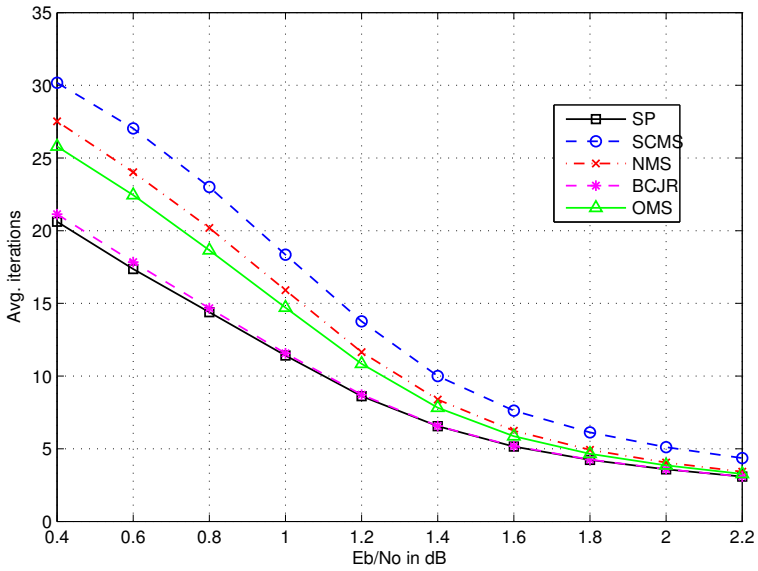


Figure A.5: Comparaison des algorithmes dans une implémentation VLSI.

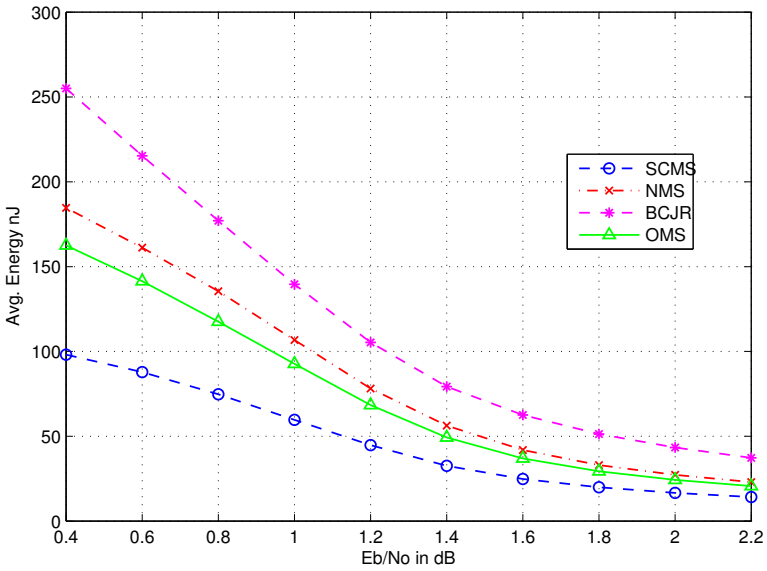
SCMS présente un compromis intéressant entre consommation d'énergie, surface et performance. Nous avons proposé de geler le processeur SCMS de parité dès que deux, ou plus, de ses entrées sont effacées. Un gain de consommation moyenne de 10% est realisé.

Nous examinons le taux de convergence des algorithmes. Ce taux est affecté par le nombre de messages par itération et leur précision. La Figure A.6a montre le taux de convergence pour les algorithmes de la Figure A.4. L'algorithme SCMS a une vitesse de convergence lente en raison de la réduction du nombre de messages, mais en général cet algorithme a la consommation d'énergie la plus faible. La Figure A.6b montre les données de la Figure A.5 (l'énergie) combinées avec les taux de convergence.

Après cette analyse, nous pouvons choisir l'algorithme de décodage optimal. Jusqu'à présent nous avons considéré que les processeurs de parité. Nous avons fait l'implémentation complète des décodeurs LDPC pour la norme IEEE 802.11n [5] avec plusieurs algorithmes. La Figure A.7 montre la surface d'implémentation et consommation moyenne par itération (SNR=1dB) de chaque module du décodeur (processeurs, mémoires et entrelaceur). Le décodeur SCMS présente la meilleure performance en termes d'efficacité énergétique.



(a) Nombre moyen d'itérations



(b) Consommation moyenne d'énergie

Figure A.6: Taux de convergence et consommation d'énergie des algorithmes.

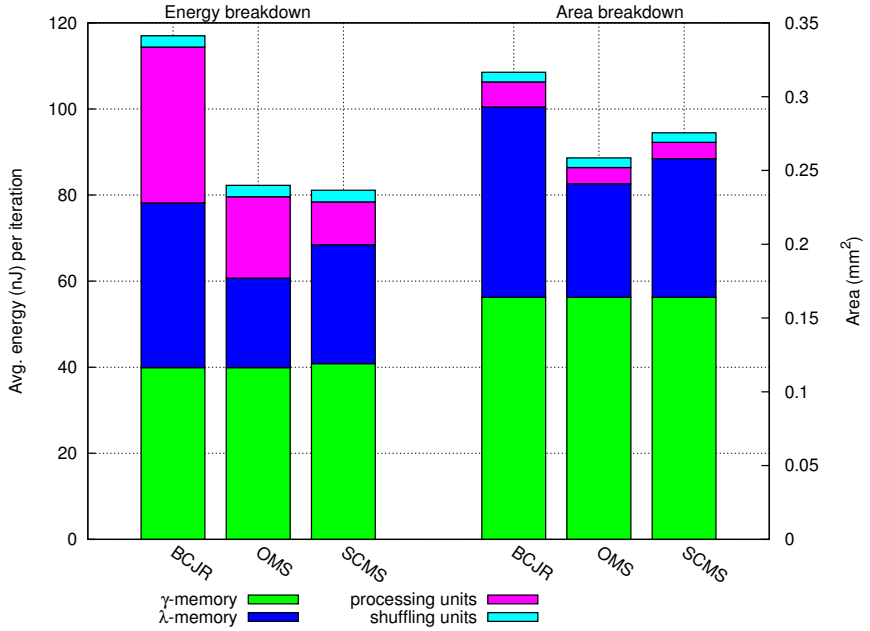


Figure A.7: Surface et consommation d'énergie des décodeurs.

A.3 Architectures de décodeurs

Le décodage des codes LDPC présente de grandes possibilités de parallélisme. En général le décodage consiste en l'échange de messages entre les processeurs de parité. Basé sur le décodage TDMP [13] [27], il y a deux types de messages: l'information *extrinsèque* et l'infomation *a posteriori*. Ces messages sont échangés par un entrelaceur. L'architecture générale d'un décodeur LDPC est illustrée par la Figure A.8. L'information de contrôle est stockée dans une mémoire *read-only* (ROM) et provient de la matrice de parité $\boldsymbol{H}$. Chaque processeur dispose d'une mémoire pour les messages extrinsèques selon une allocation statique entre processeurs et lignes de la matrice. En outre, il y a une mémoire pour les messages a posteriori. Les modules π et π^{-1} correspondent à la connectivité du graphe, typiquement ils forment un réseau de permutation.

A.3.1 Calcul du syndrome

Le message décodé à la fin du processus de décodage provient des messages a posteriori. Après chaque itération il faut faire le calcul du syndrome (A.1)

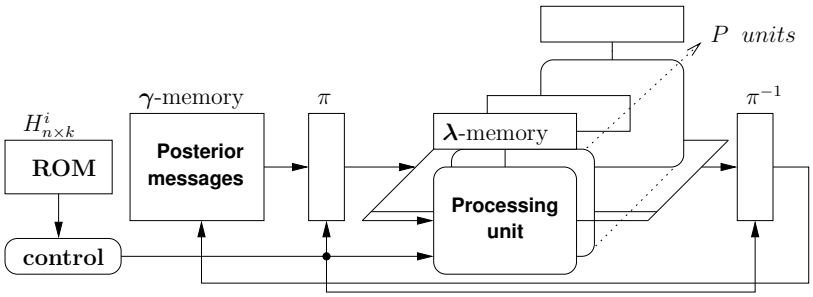


Figure A.8: Architecture de décodeur LDPC.

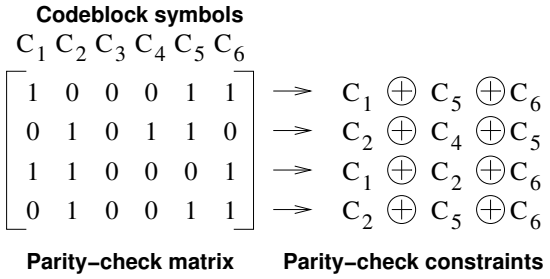


Figure A.9: Exemple de contraintes de parité.

afin de décider s'il faut arrêter le processus de décodage. Chaque ligne de la matrice $\boldsymbol{H}$ correspond à une contrainte de parité. Le calcul du syndrome est équivalent à l'évaluation de chaque contrainte. La Figure A.9 montre un exemple de contraintes de parité basé sur la matrice de parité.

L'opération $\oplus$ correspond à l'addition sur $\mathbb{F}_2$. Un syndrome non nul correspond à au moins une contrainte avec une parité impaire. Cette condition suggère qu'une nouvelle itération doit être déclenchée. Le calcul typique du syndrome a besoin d'une mémoire supplémentaire et il est effectué après chaque itération. Nous avons proposé une méthode de calcul du syndrome *à la volée* (*on-the-fly*): chaque contrainte est evaluée après le traitement de chaque ligne. Cette méthode n'est pas équivalente à l'équation (A.1) en raison des fluctuations des arguments des contraintes de parité. Les messages a posteriori sont les arguments des contraintes. Ces messages présentent un comportement très dynamique. La Figure A.10 montre l'évolution des messages a posteriori pour une instance de décodage des codes dans la norme IEEE 802.11n [5] avec une taille de mot de 648 et un taux de codage de 1/2 à travers un canal AWGN (SNR $E_b/N_0 = 1.5dB$).

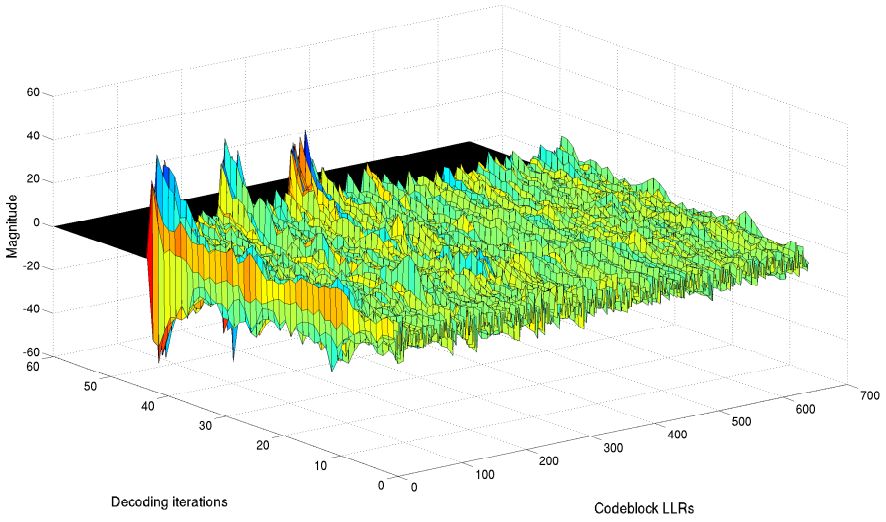


Figure A.10: L'évolution des messages en fonction des itérations.

La méthode proposée peut avoir des résultats incorrects à la fin du processus de décodage. La Table A.1 montre les résultats possibles par rapport au calcul typique. Si les deux résultats sont les mêmes la décision est un *vrai positif* (Pass), sinon la décision est un *faux positif* (False Alarm) ou un *faux negatif* (Miss).

Nous examinons le taux des résultats des décisions de la méthode proposée. La Figure A.11 montre les résultats du décodage de 10^5 mots de la norme IEEE 802.11n avec une taille de mot de 1944 et deux taux de codage. Nous pouvons observer que sur toutes les régions du SNR les vrai positifs

Table A.1: Résultats des décisions de la méthode proposée.

Calcul du syndrome *On-the-fly*	Calcul du syndrome typique	Résultat
Pass	Pass	Hit
Pass	Fail	False Alarm
Fail	Pass	Miss
Fail	Fail	Hit

dominent de plusieurs ordres de grandeur. Toutefois, nous proposons aussi une méthode pour valider les décisions erronées en effectuant un calcul du syndrome lors du décodage du mot suivant.

Le principal avantage de la méthode proposée est l'accélération ou la diminution de latence du décodage. Dans la région de faible SNR, il y a plus de possibilités pour la réduction de latence en raison des fluctuations des messages qui rendent inutile le calcul du syndrome. La Figure A.12 montre les résultats obtenus pour la norme IEEE 802.11n avec plusieurs tailles de mots et taux de codage. Une réduction de latence correspond à l'accélération du décodage si le facteur d'utilisation du décodeur est proche de 100%. Nos résultats montrent une accélération de l'ordre de 90%.

A.3.2 Architecture des mémoires

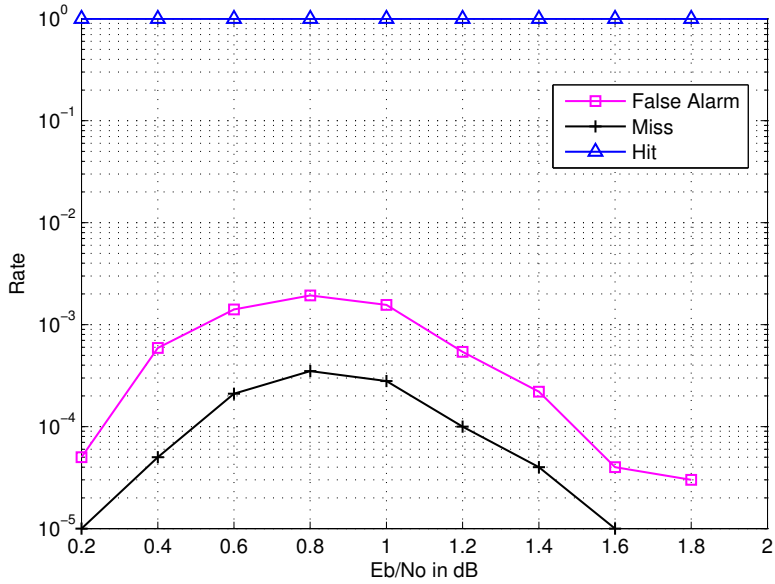
Le sous-système de mémoire du décodeur est un module essentiel en raison de la surface et de la consommation d'énergie. La Figure A.7 montre que les mémoires représentent au moins 80% de la surface du décodeur. Nous examinons l'organisation mémoire afin d'obtenir un compromis optimal surface/consommation. En outre, nous avons proposé une structure mémoire générique permettant des accès mémoires sans conflit pour tous les niveaux de parallélisme d'un décodeur QC-LDPC.

Mémoire des messages a posteriori

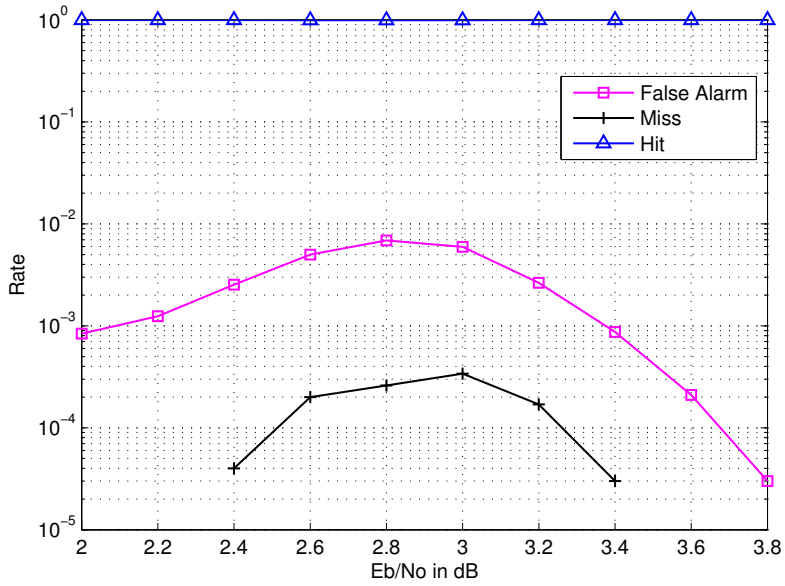
Ce module est initialisé avec les messages *intrinsèques* (messages du canal) et sa taille correspond à la taille du mot. Un vecteur de décision (résultat binaire) sur cette mémoire génère le message décodé. Nous introduisons deux niveaux d'organisation pour cette mémoire en raison de la structure du code (QC-LDPC).

1. Micro-organisation: Ce niveau d'organisation est assuré par la sous-matrice identité décalée dans $\boldsymbol{H}$. Les lignes consécutives peuvent être traitées en parallèle par la distribution de messages voisins (adjacents) entre les processeurs.

2. Macro-organisation: Chaque fois qu'un bloc de lignes est traité c_i blocs de colonnes sont utilisés, où c_i est le nombre des éléments non nuls dans une ligne. Ce niveau d'organisation est défini comme l'allocation de blocs de colonnes et des banques de mémoire permettant des accès aux c_i blocs sans conflit.

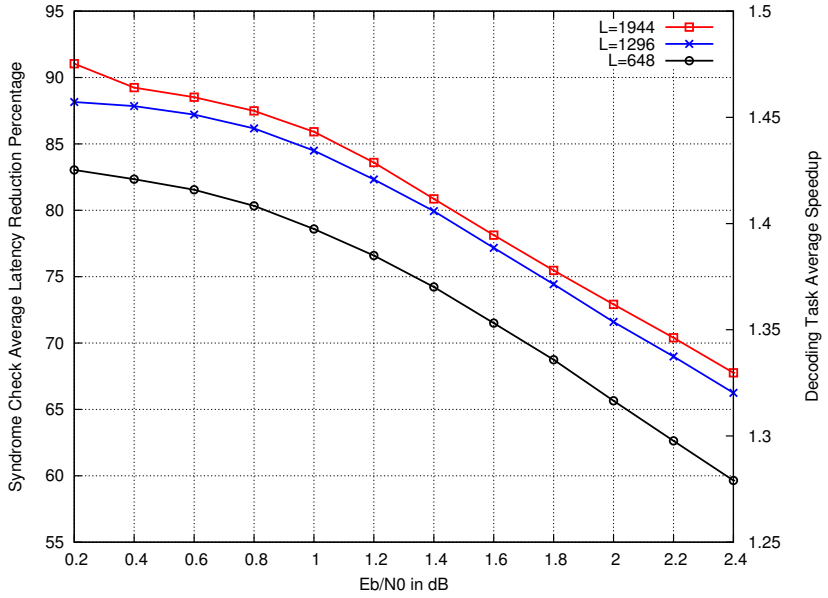


(a) Taux de codage 1/2

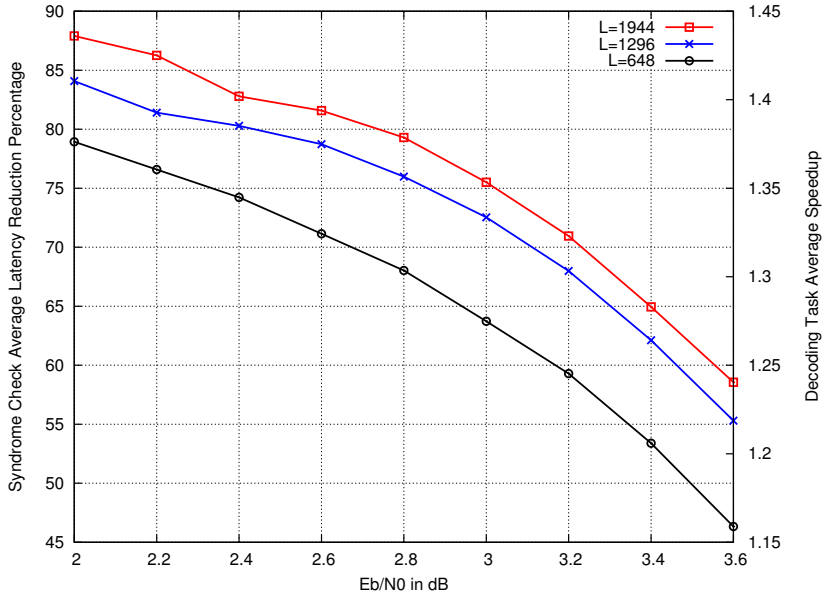


(b) Taux de codage 5/6

Figure A.11: Résultats des décisions de la méthode proposée.



(a) Taux de codage 1/2



(b) Taux de codage 5/6

Figure A.12: Diminution moyenne de latence et accélération correspondante du décodage.

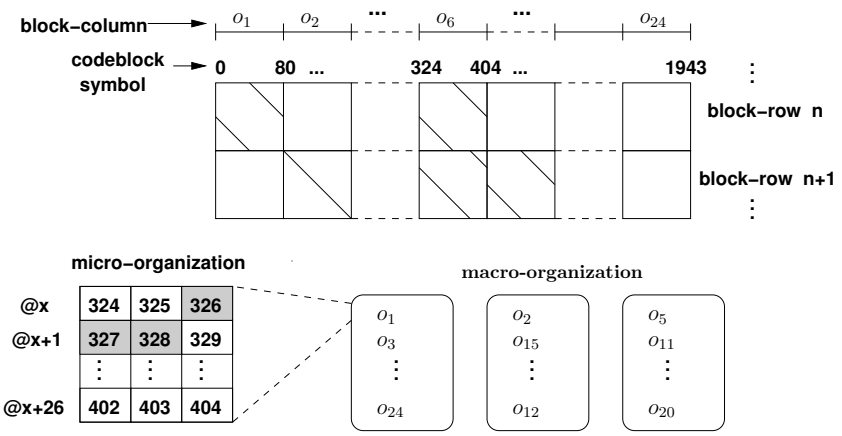


Figure A.13: Exemple des niveaux d'organisation pour la mémoire des messages a posteriori.

La Figure A.13 illustre les deux niveaux d'organisation pour un bloc de colonnes (symboles $\{324, 325, \ldots, 404\}$) avec $Z = 81$, une taille de mot de 1944 et un taux de codage de 1/2 dans la norme IEEE 802.11n. La macro-organisation correspond à l'allocation d'un ensemble d'objets $\mathcal{O} = \{o_1, o_2, \ldots, o_{24}\}$ (blocs de colonnes) des banques de mémoire.

Nous posons la macro-organisation de la mémoire comme un problème de coloration de graphe. Nous construisons des *graphes de conflit* à partir de la matrice de parité pour chaque bloc de lignes. Le nombre chromatique des graphes de conflit définit le nombre de banques de mémoire. La Figure A.14 illustre un exemple de macro-organisation résolu par les graphes de conflit.

Nous explorons l'espace de conception de cette mémoire en termes de taille des bancs mémoire et de division de la mémoire. La Figure A.15 montre l'exploration de l'espace de conception pour la mémoire des messages a posteriori dans une technologie CMOS $65nm$ (avec $V_{dd} = 1.32V$). La figure montre la surface de la mémoire et la consommation moyenne d'énergie par itération pour quelques cas d'utilisation (la taille de mot N et le taux de codage R) pour un décodeur multi-mode pour les normes IEEE 802.11n et 802.16e [6]. Nous explorons dans cette figure deux configurations pour le débit de 12 et 32 messages/cycle en utilisant des mémoires à accès aléatoire (RAM) double port avec une fréquence d'horloge de 648 et 243MHz respectivement. Cette exploration fournit un moyen de sélectionner une partition

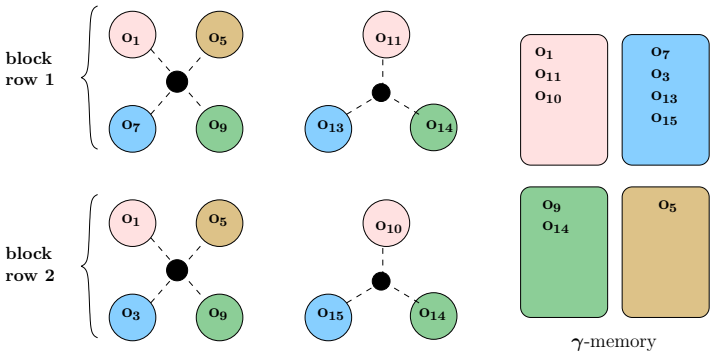


Figure A.14: La macro-organisation et les graphes de conflit.

optimale de la mémoire en considérant la flexibilité requise dans le décodeur.

Entrelacement de la mémoire

Nous identifions deux types de conflits mémoire: les conflits dus à la structure de la matrice de parité et la valeur de décalage par chaque sous-matrice identité, et les conflits dus au *saut* de ligne de chaque sous-matrice identité. Pour résoudre ces conflits nous proposons d'utiliser une mémoire entrelacée. Une mémoire entrelacée est divisée en plusieurs blocs, le nombre de blocs représente le degré d'entrelacement. Nous étudions la division verticale et horizontale de la mémoire dans le but de réduire les conflits. La Figure A.16 montre un exemple d'une mémoire entrelacée avec un degré d'entrelacement horizontal $m = 5$ et vertical $s = 2$. Chaque module enregistre $k = 3$ messages par adresse.

Après une analyse algébrique de l'entrelacement nous examinons les avantages de cette technique. Une réduction du nombre de conflits permet une réduction de la fréquence ou une augmentation de débit. Dans la Figure A.17 nous montrons les fréquences obtenues pour chaque cas d'utilisation (avec un taux de codage R et un facteur d'expansion Z) à partir d'une mémoire non-entrelacée pour un décodeur avec 30 processeurs pour la norme IEEE 802.11n dans une technologie CMOS $65nm$. L'entrelacement avec une configuration $m = 3$ et $s = 3$ permet une réduction de la fréquence entre 25%-50% en fonction du cas d'utilisation. Alternativement, le débit peut être augmenté dans la même proportion si la fréquence non-entrelacée est conservée.

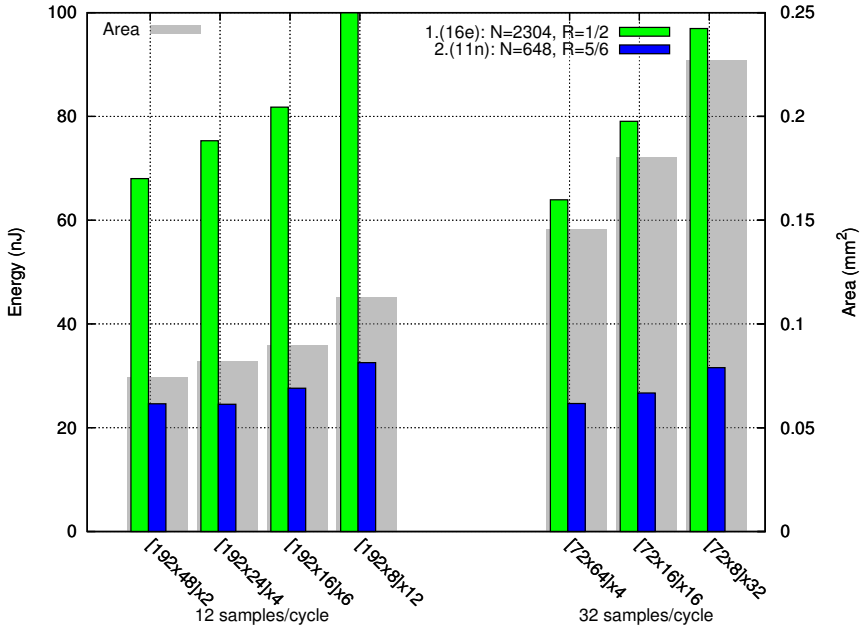


Figure A.15: L'exploration de la mémoire des messages a posteriori.

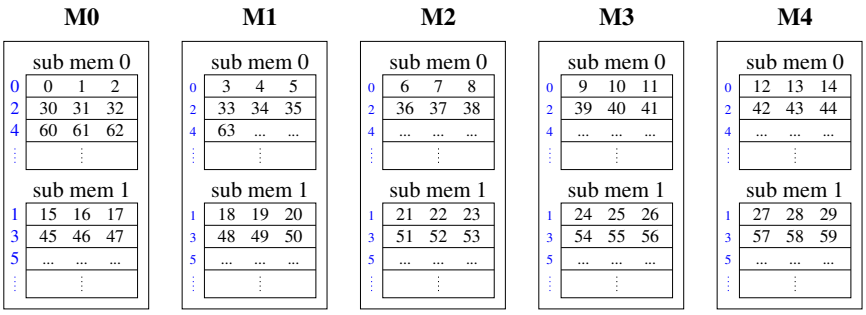


Figure A.16: Exemple d'une mémoire entrelacée avec m=5, s=2 and k=3.

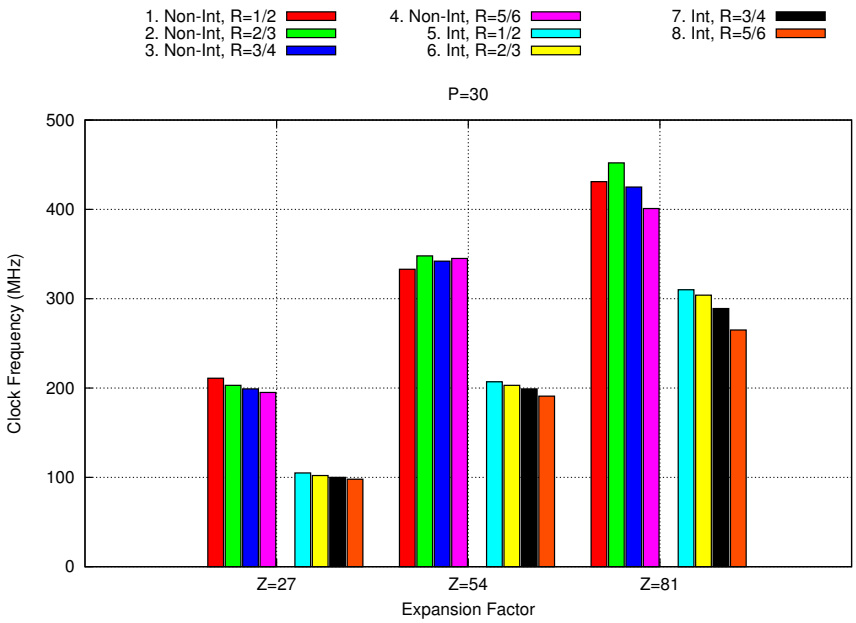


Figure A.17: Fréquence d'horloge des mémoires entrelacées/non-entrelacées.

Mémoire des messages extrinsèques

Cette mémoire enregistre les messages générés après chaque traitement de ligne dans $\boldsymbol{H}$. La taille de cette mémoire correspond au nombre d'arêtes dans le graphe du code. L'utilisation d'un algorithme basé sur le *min-sum* permet une réduction de la taille de cette mémoire. Au lieu de mémoriser les messages de toutes les colonnes de chaque contrainte, seulement la valeur de deux messages et les signes des messages sont enregistrées. Pour les codes de la norme IEEE 802.11n nous avons obtenu une réduction de 65% avec une quantification de 8-bits.

Une exploration de l'espace de conception permet la sélection d'une partition optimale de la mémoire en fonction des cas d'utilisation, la surface et la consommation d'énergie. La Figure A.18 montre l'exploration de l'espace de conception pour la mémoire de messages extrinsèques dans une technologie CMOS $65nm$ (avec $V_{dd} = 1.32V$ et $F = 648MHz$). La figure montre la surface de la mémoire et la consommation moyenne d'énergie par itération pour quelques cas d'utilisation (la taille de mot N et le taux de codage R) pour un décodeur multi-mode en utilisant des RAM double port.

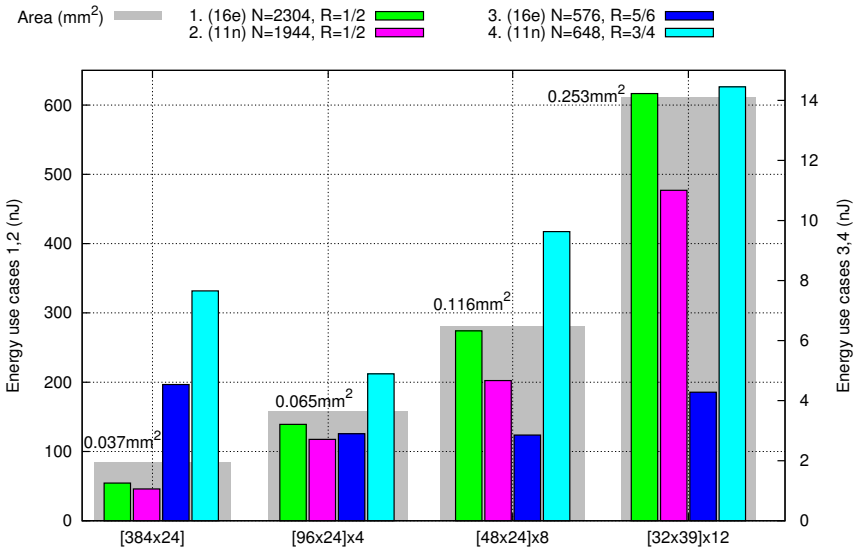


Figure A.18: L'exploration de la mémoire des messages extrinsèques.

A.4 Implémentation d'un décodeur multi-mode

Dans cette thèse nous présentons une implémentation d'un décodeur multi-mode efficace en termes d'énergie consommée pour les normes IEEE 802.11n et 802.16e. Après une exploration de l'espace de conception pour les mémoires nous avons proposé une organisation des données qui réduit la complexité des unités d'entrelacement (les arêtes du graphe). Notre implémentation dans une technologie CMOS $65nm$ a atteint une surface de $0.95mm^2$ et une efficacité énergétique de $47pJ/bit/iter$. Le décodeur est basé sur l'algorithme SCMS.

A.4.1 Conception

Il est nécessaire d'analyser les besoins du décodeur en termes de contraintes temps réel et de taille des mémoires. Il y a aussi des exigences de flexibilité en raison de plusieurs cas d'utilisation. Les cas d'utilisation sont différenciés par la taille de mot, le taux de codage, le facteur d'expansion et le mode d'operation (11n/16e).

La Table A.2 montre les caractéristiques des cas d'utilisation les plus et les moins exigeants.

Table A.2: Caractéristiques des codes LDPC standardisés.

Cas d'utilisation	Le plus exigeant		Le moins exigeant	
Standard 802.xx	11n	16e	11n	16e
Taille de mot	1944	2304	648	576
Taux de codage	1/2		5/6	
Dimensions de $\boldsymbol{H}$ ($m_b \times n_b$)	12x24	12x24	4x24	4x24
Z	81	96	27	24
Latence	8μs	0.25ms	8μs	0.25ms
Degré ligne c_i	7,8	6,7	22	20
Lignes dans $\boldsymbol{H}$	972	1152	108	96
Arêtes de graphe	6966	7296	2376	1920

A.4.2 Panorama de l'architecture et résultats

Nous avons conçu un décodeur avec 15 processeurs série de type SCMS avec une quantification sur 6-bits. Le décodeur exécute un maximum de 8 itérations. Le sous-système mémoire comprend l'enregistrement de la matrice de parité dans une mémoire ROM (mémoire morte accessible uniquement en lecture), une mémoire des messages a posteriori divisée en 5 blocs et une mémoire des messages extrinsèques répartie entre les processeurs. L'architecture générale du décodeur est illustrée dans la Figure A.8.

La répartition des données que nous avons proposé dans la mémoire des messages a posteriori permet l'utilisation d'un réseau de permutation de faible complexité. Cela résulte de l'alignement des données pour la majorité des cas d'utilisation. La Figure A.19 montre l'organisation de la mémoire au niveau de la répartition et du partitionnement des données.

Dans le Chapitre 6 nous présentons les détails d'implémentation de chaque module du décodeur: architecture des processeurs, architecture des mémoires et des exemples de l'utilisation du réseau de permutation. Les optimisations proposées dans cette thèse ont permis la réalisation d'un décodeur efficace en termes de surface et consommation d'énergie. Nous avons comparé le décodeur implémenté avec l'état de l'art afin de démontrer les avantages de notre approche. La Figure A.20 montre la distribution de la surface du décodeur multi-mode implémenté. Bien entendu le sous-système de mémoire occupe la plupart de la surface et il est donc l'élément essentiel. Effectivement l'optimisation du décodeur passe essentiellement par l'optimisation du sous-système de mémoire.

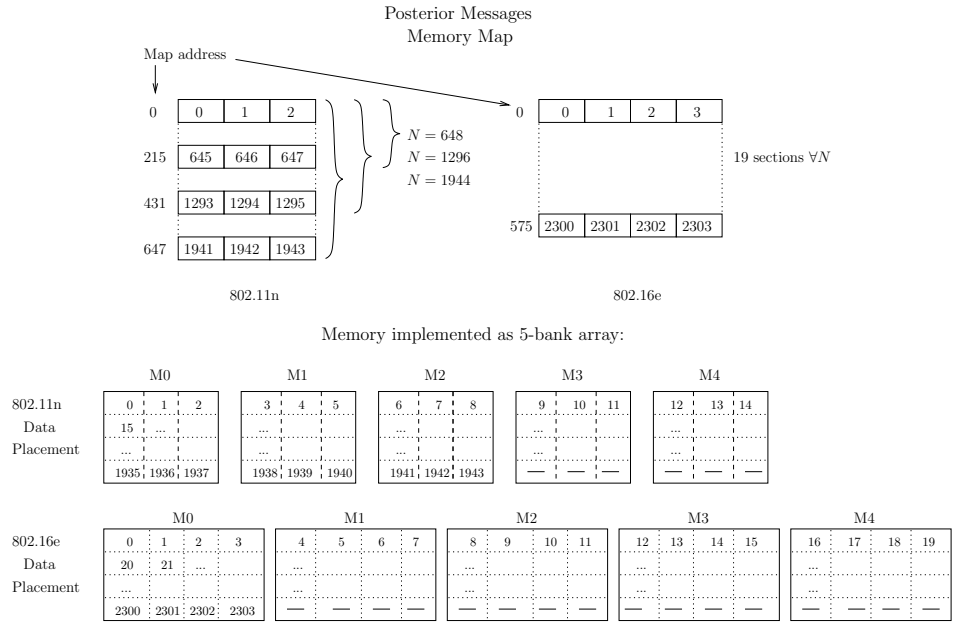


Figure A.19: L'organisation de la mémoire des messages a posteriori implementé.

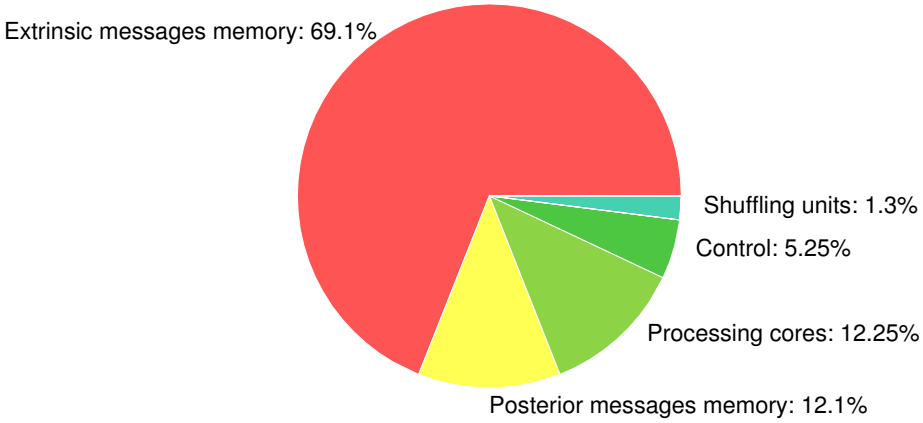


Figure A.20: Distribution de la surface du décodeur implémenté.

A.5 Gestion de la puissance

Les chapitres précédents ont examiné les optimisations de la conception des architectures des décodeurs LDPC. Dans le Chapitre 7 nous examinons la gestion de la puissance pour les décodeurs itératifs au niveau du système. Nous nous intéressons à l'optimisation de l'utilisation du décodeur pour obtenir un fonctionnement à faible puissance. Nous abordons ce sujet à deux niveau différents: le contrôle du nombre d'itérations par la détection de la convergence et l'adaptation dynamique de la puissance (la tension d'alimentation et la fréquence d'horloge) par la prédiction du nombre d'itérations nécessaires pour la convergence en satisfaisant des contraintes temps réel.

A.5.1 Contrôle de l'itération

Les algorithmes de décodage itératif sont intrinsèquement dynamiques puisque le nombre d'itérations dépend de plusieurs facteurs externes. Des politiques optimales pour le contrôle d'itérations (aussi connues comme *critères d'arrêt*) devraient identifier des trames décodable et indécodable afin d'économiser l'énergie des l'opérations inutiles du décodeur. La convergence est détectée par le calcul du syndrome (A.1) et normalement la divergence est détectée par un maximum d'itérations.

Après une comparaison des performances avec l'état de l'art nous proposons un critère *hybride* pour le contrôle d'itérations des décodeurs de type SCMS. Ce critère est basé sur l'observation du nombre de messages effacés et du nombre de contraintes de parité respectées. Ces deux valeurs sont des *mesures* pour la convergence du code et ils permettent de suivre la dynamique du processus de décodage. Basé sur les caractéristiques du comportement des mesures, nous avons proposé un critère d'arrêt qui permet de réduire le nombre moyen de décisions erronées (*de faux negatifs*). Un faux negatif dans ce contexte correspond à l'achèvement du nombre maximal d'itérations. De cette façon le taux de faux negatifs montre l'efficacité du critère d'arrêt. Dans la Figure A.21 nous montrons le nombre moyen d'itérations obtenu par plusieurs critères: [67] (Shin), [65] (Kienle), [69] (Chen) et le critère proposé. En outre, nous montrons les performances de deux cas limites:

1. Calcul du syndrome: le décodeur s'arrête dès que le syndrome est valide.

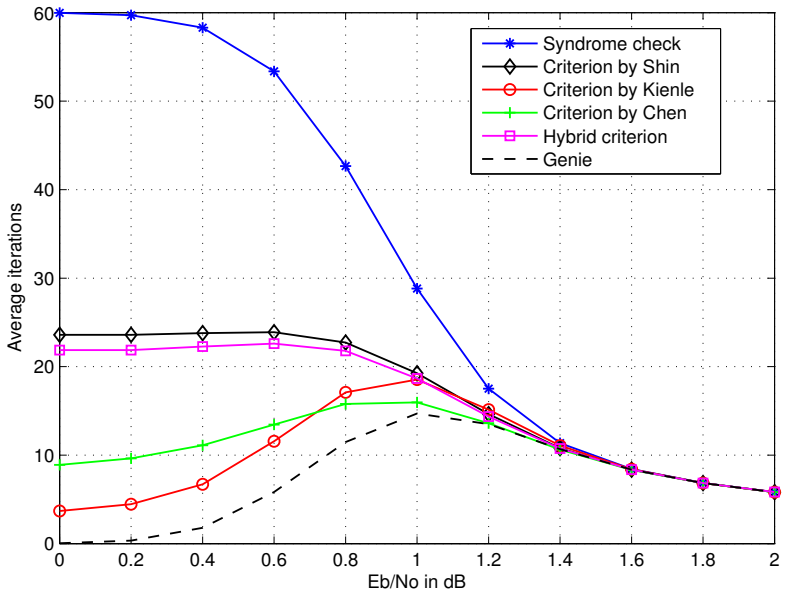


Figure A.21: Nombre moyen d'itérations des critères d'arrêt.

2. Un *génie*: le décodeur a une connaissance préalable de la trame transmise, dans ce cas le décodage ne serait pas démarré pour les trames indécodables.

La Figure A.21 montre que les critères d'arrêt sont importants seulement dans la zone de faible SNR, cependant nous sommes intéressés à la performance des critères. La Figure A.22 montre le taux de faux negatifs des critères d'arrêt utilisés. Le critère proposé montre une plus grande efficacité que l'état de l'art.

A.5.2 Prédiction du nombre d'itérations

Dans cette section, nous proposons d'aborder le problème de la réduction de la puissance des décodeurs itératifs d'une manière différente. Notre approche est basée sur les observations suivantes:

- Les décodeurs sont généralement dimensionnés pour exécuter un nombre maximal d'itérations en satisfaisant des contraintes temps réel.
- Il est bien connu que le décodage sans erreurs est réalisable avec quelques itérations dans de bonnes conditions de canal.

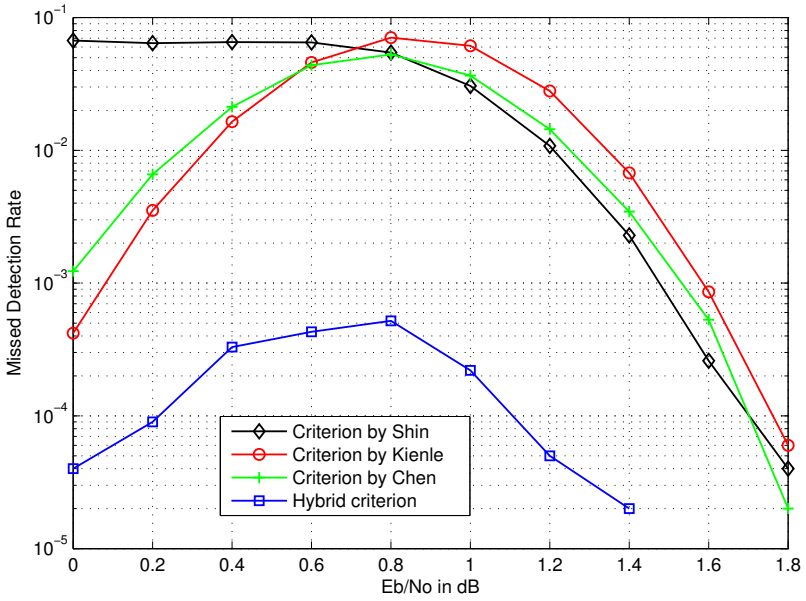


Figure A.22: Taux de faux negatifs des critères d'arrêt.

- Dans de mauvaises conditions de canal une trame ne pourrait pas être décodée avec le nombre maximum d'itérations.

La Figure A.23 illustre la situation typique pour une tâche de décodage réalisée à deux niveaux de puissance. Les deux instances garantissent le respect de la contrainte temporelle (*deadline*). Toutefois, la tâche à haute puissance peut réclamer le temps restant pour utiliser d'autres modes de faible puissance. La relation entre les différents modes disponibles peut garantir une réduction de consommation d'énergie du décodeur.

Nous soutenons qu'un décodeur avec une puissance gérable peut être contrôlé en suivant la dynamique du processus de décodage. Nous proposons un algorithme *en-ligne* qui suit le processus de décodage afin de trouver un mode de puissance approprié pour finir le décodage dans la contrainte temps réel. Un algorithme en-ligne signifie qu'il faut choisir une action lors de l'exécution de la tachê de décodage. Contrairement à un algorithme *hors-ligne* où toutes les situations possibles de la tâche sont connues avant l'exécution.

Nous formulons un problème de décision pour choisir pour chaque itération le mode de puissance du décodeur qui garantit le respect de la contrainte temporelle. Nous proposons une heuristique pour l'optimisation de la con-

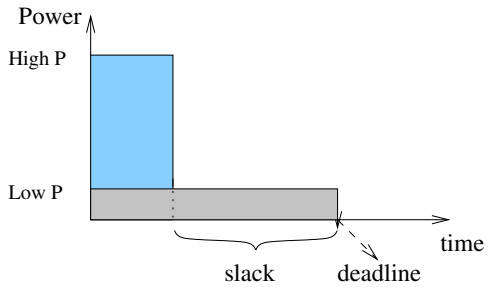


Figure A.23: Exemple d'une tâche de décodage à deux niveaux de puissance.

sommation d'énergie avec une contrainte temporelle. Le processus de décision est basé sur la prédiction du nombre d'itérations qui restent pour atteindre la convergence. En effet nous utilisons une mesure de la convergence pour suivre la dynamique du processus de décodage. Pour le cas des codes LDPC nous utilisons le nombre de contraintes de parité respectées pendant chaque itération, d'autre part, pour les codes Turbo, nous utilisons le nombre de changements de signe des messages a posteriori pendant chaque itération. Les deux mesures affichent un comportement caractéristique qui nous permet de dériver une politique de contrôle. La politique proposée s'applique aux décodeurs itératifs en général, mais nous avons montré le concept pour les décodeurs de type Turbo et LDPC.

Nous étudions l'implémentation de la politique de contrôle proposée dans une technologie CMOS $65nm$ et nous montrons des résultats intéressants pour la réduction d'énergie (jusqu'à 54%) avec une dégradation de la performance inférieure à 0,1dB.

A.6 Perspectives

Le travail de recherche effectué au cours de cette thèse peut être étendu dans le cadre des architectures versatiles pouvant traiter à la fois les codes Turbo et LDPC:

- L'étude des organisations du sous-système de mémoire qui permettent de maximiser la réutilisation des mémoires.
- L'enquête sur les avantages quantitatifs de l'utilisation de plusieurs algorithmes de décodage dans le même décodeur multi-mode.
- La conception des politiques de contrôle pour la gestion de la puissance spécifique pour les décodeurs unifiés Turbo/LDPC.

Bibliography

[1] C. Berrou, A Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-correcting coding and decoding: Turbo Codes," *IEEE International Conference on Communication*, pp. 1064–1070, May 1993.

[2] R. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inf. Theory*, vol. 7, pp. 21–28, January 1962.

[3] 3GPP UMTS, "General UMTS Architecture," *3GPP TS 23.101 version 7.0.0*, 2007.

[4] 3GPP LTE, "Evolved Universal Terrestrial Radio Access (EUTRA) and Evolved Universal Terrestrial Radio Access Network (EUTRAN)," *3GPP TS 36.300*, 2008.

[5] IEEE-802.11n, "Wireless LAN Medium Access Control and Physical Layer Specifications: Enhancements for Higher Throughput," *P802.11n-2009*, October 2009.

[6] IEEE-802.16e, "Air Interface for Fixed and Mobile Broadband Wireless Access Systems," *P802.16e-2005*, October 2005.

[7] IEEE-802.15.3c, "Amendment 2: Millimeter-wave-based Alternative Physical Layer Extension," *802.15.3c-2009*, 2009.

[8] ETSI DVB-S2, "Digital video broadcasting, second generation," *ETSI EN 302307*, vol. 1.1.1, 2005.

[9] IEEE-802m, "TGm System Requirements Documents (SRD)," *802.16m-10/0003r1*, 2010.

[10] 3GPP LTE-Advanced, "Requirements for Further Advancements for E-UTRA (LTE-Advanced)," *3GPP TSG RAN TR 36.913 v8.0.0*, 2009.

[11] Enrico Macii, "Lecture Notes: Introduction to Low-Power Design," *Politecnico di Torino*, 2007.

[12] International Technology Roadmap for Semiconductors, "System Drivers Report," 2010.

[13] M. Mansour and N. Shanbhag, "High-Throughput LDPC Decoders," *IEEE Trans. on VLSI Systems*, vol. 11, no. 6, pp. 976–996, December 2003.

[14] V. Savin, "Self-Corrected Min-Sum Decoding of LDPC Codes," in *Proc. of IEEE International Symposium on Information Theory*, 2008, pp. 146–150.

[15] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Tech. J.*, vol. 27, pp. 379–423 and 623–656, 1948.

[16] D. J. C. MacKay and R. M. Neal, "Good Codes Based on Very Sparse Matrices," in *Cryptography and Coding. 5th IMA Conf., Lecture Notes in Computer Science*, October 1995, vol. 1025, pp. 100–111.

[17] G.D. Forney and D.J. Costello, "Channel Coding: The Road to Channel Capacity," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1150 –1177, 2007.

[18] T. Richardson and R. Urbanke, *Modern Coding Theory*, Cambridge University Press, New York, NY, USA, 2008.

[19] A. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *Information Theory, IEEE Transactions on*, vol. 13, no. 2, pp. 260 – 269, Apr. 1967.

[20] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *Information Theory, IEEE Transactions on*, vol. 20, no. 2, pp. 284–287, 1974.

[21] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. on Inf. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[22] N. Wiberg, *Codes and Decoding on General Graphs*, Ph.D. thesis, Linköping Univ., Linköping, Sweden, 1996.

[23] R. Tanner, "A Recursive Approach to Low Complexity Codes," *Information Theory, IEEE Transactions on*, vol. 27, no. 5, pp. 533 – 547, Sept. 1981.

[24] F.R. Kschischang, B. Frey, and H.A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, pp. 498–519, Febraury 2001.

[25] H. Wymeersch, *Iterative Receiver Design*, Cambridge University Press, New York, NY, USA, 2007.

[26] M. Fossorier, "Quasi-Cyclic Low-Density Parity-Check Codes from Circulant Permutation Matrices," in *IEEE Trans. Information Theory*, 2004, vol. 50, pp. 1788–1793.

[27] M. Mansour, "A Turbo-Decoding Message-Passing Algorithm for Sparse Parity-Check Matrix Codes," *IEEE Trans. on Signal Processing*, vol. 54, no. 11, pp. 4376–4392, November 2006.

[28] J. Chen and M. Fossorier, "Near Optimum Universal Belief Propagation based Decoding of LDPC Codes," in *IEEE Trans. on Comm.*, 2002, pp. 406–414.

[29] A. Blanksby and C. Howland, "A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Code Decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 404–412, March 2002.

[30] F. Guilloud, *Generic Architecture for LDPC Codes Decoding*, Ph.D. thesis, ENST Paris, Paris, France, 2004.

[31] F. Quaglio, *Implementation of Architectures for High Performance and Flexible Decoders*, Ph.D. thesis, Politecnico di Torino, Torino, Italy, 2006.

[32] M. Alles, T. Vogt, and N. Wehn, "FlexiChap: A Reconfigurable ASIP for Convolutional, Turbo and LDPC Code Decoding," in *5th International Symposium on Turbo Codes and Related Topics* , 2008, pp. 84–89.

[33] F. Naessens, B. Bougard, S. Bressinck, L. Hollevoet, P. Raghavan, L. Van der Perre, and F. Catthoor, "A Unified Instruction Set Programmable Architecture for Multi-Standard Advanced Forward Error Correction," in *IEEE Workshop on Signal Processing Systems*, 2008, pp. 31–36.

[34] C.-H. Liu, C.-C. Lin, S.-W. Yen, C.-L. Chen, H.-C.-Chang, C.-Y. Lee, Y.-S. Hsu, and S.-J. Jou, "Design of a Multimode QC-LDPC Decoder Based on Shift-Routing Network," in *IEEE Trans. on Circuits and Systems II: Express Briefs*, 2009, vol. 59, pp. 734–738.

[35] Y. Sun and J.R. Cavallaro, "A Low-Power 1-Gbps Reconfigurable LDPC Decoder Design for Multiple 4G Wireless Standards," in *Proc. of IEEE International System-on-Chip Conference*, 2008, pp. 367–370.

[36] Y.-L. Wang, Y.-L. Ueng, C.-L. Peng, and C.-J. Yang, "Processing-Task Arrangement for a Low-Complexity Full-Mode WiMAX LDPC Codec," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 2010.

[37] Jie Jin and Chi-Ying Tsui, "An Energy Efficient Layered Decoding Architecture for LDPC Decoder," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 8, pp. 1185 –1195, 2010.

[38] Bo Xiang, Rui Shen, An Pan, Dan Bao, and Xiaoyang Zeng, "An Area-Efficient and Low-Power Multirate Decoder for Quasi-Cyclic Low-Density Parity-Check Codes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 10, pp. 1447 –1460, 2010.

[39] W.J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

[40] E. Zimmermann, P. Pattisapu, K. Bora, and G. Fettweis, "Reduced Complexity LDPC Decoding using Forced Convergence," *Proc. of the 7th International Symposium on Wireless Personal Multimedia Communications*, pp. 12–15, September 2004.

[41] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, "Precomputation-based Sequential Logic Optimization for Low Power ," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 426 –436, Dec. 1994.

[42] M. Rovini, G. Gentile, and L. Fanucci, "Multi-Size Circular Shifting Networks for Decoders of Structured LDPC Codes," *IEE Electronics Letters*, pp. 938–940, August 2007.

[43] C. Struder, N. Preyss, C. Roth, and A. Burg, "Configurable High-Throughput Decoder Architecture for Quasi-Cyclic LDPC Codes," in

Proceedings of the 42th Asilomar Conference on Signals, Systems and Computers, October 2008.

[44] C. Struder, *Iterative MIMO Decoding: Algorithms and VLSI Implementation Aspects*, Ph.D. thesis, ETH Zürich, Zürich, Switzerland, 2009.

[45] Yeong-Luh Ueng, Yu-Lun Wang, Chi-Yu Lin, Jen-Yuan Hsu, and Pangan Ting, "Modified Layered Message Passing Decoding with Dynamic Scheduling and Early Termination for QC-LDPC Codes," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, May 2009, pp. 121 –124.

[46] A. Heim and U. Sorger, "Turbo Decoding: Why Stopping-Criteria Do Work ," in *5th International Symposium on Turbo Codes and Related Topics*, 2008, pp. 255–259.

[47] G. Lechner and J. Sayir, "On the Convergence of Log-Likelihood Values in Iterative Decoding," in *Mini-Workshop on Topics in Information Theory, Essen, Germany*, September 2002.

[48] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, third edition, 2003.

[49] J. Dielissen and A. Hekstra, "Non-fractional Parallelism in LDPC Decoder Implementations," in *Proc. of 2007 Design, Automation and Test in Europe*, Nice, France, 2007.

[50] Z. Cui, *Low-Complexity High-Speed VLSI Design of Low-Density Parity-Check Decoders*, Ph.D. thesis, Oregon State University, Oregon, USA, 2007.

[51] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory System Optimization for FPGA-Based Implementation of Quasi-Cyclic LDPC Codes Decoders," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 2010.

[52] C. Marchand, J.-B. Dore, L. Conde-Canencia, and E. Boutillon, "Conflict Resolution for Pipelined Layered LDPC Decoders," in *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, 2009, pp. 220 –225.

[53] P. Keyngnaert, B. Demoen, B. De Sutter, B. De Sus, and K. De Bosschere, "Conflict Graph Based Allocation of Static Objects to Memory Banks," *Informal proceedings of the First workshop on Semantic, Program Analysis, and Computing Environments*, pp. 131–142, September 2001.

[54] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X. Hu, "Reduced-Complexity Decoding of LDPC Codes," *IEEE Trans. Comms.*, vol. 53, pp. 1288–1299, August 2005.

[55] David J. C. MacKay and M. J. Postol, "Weaknesses of Margulis and Ramanujan–Margulis low-density parity-check codes," in *Proceedings of MFCSIT2002, Galway*. 2003, vol. 74 of *Electronic Notes in Theoretical Computer Science*, Elsevier.

[56] R. Koetter and P. Vontobel, "Graph-Covers and Iterative Decoding of Finite Length Codes," in *International Symposium on Turbo Codes and Related Topics*, 2005.

[57] S. Landner and O. Milenkovic, "Algorithmic and Combinatorial Analysis of Trapping Sets in Structured LDPC Codes," in *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, 2005, vol. 1, pp. 630 –635 vol.1.

[58] Changyan Di, D. Proietti, I.E. Telatar, T.J. Richardson, and R.L. Urbanke, "Finite-Length Analysis of Low-Density Parity-Check Codes on the Binary Erasure Channel," *Information Theory, IEEE Transactions on*, vol. 48, no. 6, pp. 1570 –1579, June 2002.

[59] L. Dolecek, Zhengya Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "Analysis of Absorbing Sets for Array-Based LDPC Codes," in *Communications, 2007. ICC '07. IEEE International Conference on*, 2007, pp. 6261 –6268.

[60] C.-H. Liu, C.-C. Lin, H.-C. Chang, C.-Y. Lee, and Y. Hsu, "Multimode Message Passing Switch Networks Applied for QC-LDPC Decoder," in *IEEE International Symposium on Circuits and Systems*, 2008, pp. 752–755.

[61] Daesun Oh and K.K. Parhi, "Low-Complexity Switch Network for Reconfigurable LDPC Decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 1, pp. 85 –94, 2010.

[62] Jun Lin, Zhongfeng Wang, Li Li, Jin Sha, and Minglun Gao, "Efficient Shuffle Network Architecture and Application for WiMAX LDPC Decoders," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 56, no. 3, pp. 215 –219, 2009.

[63] S. Borkar, "Design Challenges of Technology Scaling," *Micro, IEEE*, vol. 19, no. 4, pp. 23 –29, 1999.

[64] G. Glikiotis and V. Paliouras, "A Low-Power Termination Criterion for Iterative LDPC Code Decoders," *IEEE Workshop on Signal Processing Systems Design and Implementation*, pp. 122–127, November 2005.

[65] F. Kienle and N. Wehn, "Low Complexity Stopping Criterion for LDPC Code Decoders," in *Proc. of IEEE VTC 2005-Spring*, 2005, pp. 606–609.

[66] Jin Li, Xiao hu You, and Jing Li, "Early Stopping for LDPC Decoding: Convergence of Mean Magnitude (CMM)," in *IEEE Communications Letters*, 2006, number 9.

[67] D. Shin, K. Heo, S. Oh, and J. Ha, "A Stopping Criterion for Low-Density Parity-Check Codes," in *Proc. of IEEE VTC 2007-Spring*, 2007, pp. 1529–1533.

[68] Z. Cui, L. Chen, and Z. Wang, "An Efficient Early Stopping Scheme for LDPC Decoding," in *13th NASA Symposium on VLSI Design*, 2007.

[69] Y.H Chen, Y.J. Chen, X.Y Shih, and A.Y Wu, "A Channel-Adaptive Early Termination Strategy for LDPC Decoders," in *IEEE Workshop on Signal Processing Systems*, 2009, pp. 226–231.

[70] L. Benini, R. Bogliolo, and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. on VLSI Systems*, vol. 8, pp. 299–316, 2000.

[71] R.Y. Shao, S. Lin, and M. Fossorier, "Two Simple Stopping Criteria for Turbo Decoding," *IEEE Trans. in Commun.*, vol. 47, pp. 1117–1120, Aug. 1999.

[72] A. Matache, S. Dolinar, and F. Pollara, "Stopping Rules for Turbo Decoders," *TMO Progree Report*, vol. 42, pp. 42–142, Aug. 2000.

[73] W. Wang and G. Choi, "Speculative Energy Scheduling for LDPC Decoding," in *8th International Symposium on Quality Electronic Design*, 2007, pp. 79–84.

[74] W. Wang, G. Choi, and K. Gunnam, "Low-Power VLSI Design of LDPC Decoder Using DVFS for AWGN Channels," in *Proc. of the 22nd International Conference on VLSI Design*, 2009, pp. 51–56.

[75] S. E. Krafft and D. Richard, "Power Savings Technique for Iterative Decoding," *US Patent Application Publication*, vol. US 2007/0113149 A1, May 2007.

[76] Ying Tan, P. Malani, Qinru Qiu, and QingWu, "Workload Prediction and Dynamic Voltage Scaling for MPEG Decoding," in *Design Automation, 2006. Asia and South Pacific Conference on*, 2006, p. 6 pp.

[77] Hwisung Jung and M. Pedram, "Continuous Frequency Adjustment Technique Based on Dynamic Workload Prediction," in *VLSI Design, 2008. VLSID 2008. 21st International Conference on*, 2008, pp. 249 –254.

[78] S. Irani, G. Singh, S.K. Shukla, and R.K. Gupta, "An Overview of the Competitive and Adversarial Approaches to Designing Dynamic Power Management Strategies," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 12, pp. 1349 – 1361, 2005.

[79] P. Macken, M. Degrauwe, M.V. Paemel, and H. Orguey, "A Voltage Reduction Technique for Digital Systems," in *IEEE International Solid State Circuits Conference*, 1990, pp. 238–239.

[80] Yanbin Liu and A.K. Mok, "An Integrated Approach for Applying Dynamic Voltage Scaling to Hard Real-Time Systems," in *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, May 2003, pp. 116 – 123.

[81] S. Irani, S. Shukla, and R. Gupta, "Algorithms for Power Savings," in *Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms*, 2003, pp. 37–46.

[82] E. Amador, R. Knopp, V. Rezard, and R. Pacalet, "Dynamic Power Management on LDPC Decoders," in *Proc. of IEEE Computer Society Annual Symposium on VLSI*, July 2010, pp. 416–421.

[83] S. Irani, S. Shukla, and R. Gupta, "Competitive Analysis of Dynamic Power Management Strategies for Systems with Multiple Power Savings States," in *Proc. of the 2002 Design, Automation and Test in Europe*, 2002, pp. 117 – 123.

[84] W. Kim, M.S. Gupta, G.Y. Wei, and D. Brooks, "System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators," in *IEEE 14th International Symposium on High Performance Computer Architecture*, 2008, pp. 123–134.

[85] F. Su, W.H. Ki, and C.Y. Tsui, "Ultra Fast Fixed-Frequency Hysteretic Buck Converter with Maximum Charging Current Control and Adaptive Delay Compensation for DVS Applications," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 815–822, April 2008.

[86] C.W. Chang and Y.E. Chen, "A CMOS True Single-Phase-Clock Divider with Differential Outputs," in *IEEE Microwave and Wireless Components Letters*, 2009, vol. 19, pp. 813 – 815.

[87] J. Kaza and C. Chakrabarti, "Energy-Efficient Turbo Decoder," in *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing*, 2002, pp. 3093 – 3096.

[88] S. Kunze, E. Matuš, G. Fettweis, and T. Kobori, "A Multi-user Approach Towards a Channel Decoder for Convolutional, Turbo and LDPC Codes," in *Signal Processing (SiPS), 2010 Workshop on*, 2010, pp. 386–391.

[89] Yang Sun and Joseph Cavallaro, "A Flexible LDPC/Turbo Decoder Architecture," *Journal of Signal Processing Systems*, pp. 1–16, 2010, 10.1007/s11265-010-0477-6.

[90] T.S.V. Gautham, A. Thangaraj, and D. Jalihal, "Common Architecture for Decoding Turbo and LDPC Codes," Jan. 2010, pp. 1 –5.

[91] A. Trofimenko, A. Efimov, A.V. Belogolovy, and V.A. Chernyshev, "Unified Decoder for Convolutional, Turbo and LDPC Codes," *US Patent Application Publication*, vol. US 2010/0146360 A1, Jun. 2010.

[92] M. Scarpellino, A. Singh, E. Boutillon, and G. Masera, "Reconfigurable Architecture for LDPC and Turbo Decoding: a NoC Case Study," in *Spread Spectrum Techniques and Applications, 2008. IEEE 10th INternational Symposium on*, 2008, pp. 671–676.

[93] F. Jiang, E. Psota, and L.C. Perez, "The Generator and Parity-Check Matrices of Turbo Codes," in *Information Sciences and Systems, 2006 40th Annual Conference*, 2006, pp. 1451–1454.

[94] A.J. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," in *IEEE J. Sel. Areas Commun.*, 1998, vol. 16, pp. 260–264.

[95] Yang Sun, Yuming Zhu, M. Goel, and J.R. Cavallaro, "Configurable and Scalable High Throughput Turbo Decoder Architecture for Multiple 4G Wireless Standards," jul. 2008, pp. 209 –214.

[96] Ji-Hoon Kim and In-Cheol Park, "A Unified Parallel Radix-4 Turbo Decoder for Mobile WiMAX and 3GPP-LTE," sep. 2009, pp. 487 –490.

[97] C. Schurgers, F. Cathoor, and M. Engels, "Memory Optimization of MAP Turbo Decoder Algorithms," in *IEEE Trans. on Very Large Scale Integration Systems*, 2001, vol. 9, pp. 305–312.

[98] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, "Architectural Strategies for Low-Power VLSI Turbo Decoders," in *IEEE Trans. on Very Large Scale Integration Systems*, 2002, vol. 10, pp. 279–285.

[99] E. Amador, R. Pacalet, and V. Rezard, "Optimum LDPC Decoder: A Memory Architecture Problem," in *Proc. of 46th Annual ACM/IEEE Design Automation Conference*, July 2009, pp. 891–896.

[100] O.Y. Takeshita, "On Maximum Contention-Free Interleavers and Permutation Polynomials over Integer Rings," *Information Theory, IEEE Transactions on*, vol. 52, no. 3, pp. 1249 –1253, mar. 2006.