



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

---

# Architectures for soft-decision decoding of non-binary codes

Esta tesis doctoral está sometida a procesos de protección o transferencia de tecnología o de conocimiento, por lo que los siguientes contenidos están inhibidos en la publicación en los repositorios institucionales. Autorizado por la Comisión de Doctorado de la Universitat Politècnica de València con fecha 10 de septiembre de 2013.

---

First version: 31st of January 2013  
Reviewed version: 20th of August 2013  
Francisco Miguel García Herrero



# Relación de contenidos inhibidos

La siguiente relación de contenidos debe protegerse para asegurar el éxito del proceso de patente de parte de este trabajo:

- La sección “2.3 Multiple-vote algorithm and architecture”, que incluye las subsecciones: “2.3.1 Multiple-Vote Symbol Flipping Decoding Algorithm”, “2.3.2 Parameters of MV-SFA and FER performance comparisons”, “2.3.3 Partial parallel architecture for MV-SFA” y “2.3.4 Conclusions of the multiple-vote symbol flipping algorithm”.
- La sección “2.5 Conclusions” de la Parte 2 de la tesis.
- Las referencias directas a resultados o procedimientos relacionados con los algoritmos GMV-SFA o MV-SFA en la Parte 3 “Conclusion and future works”.
- Las siguientes figuras: “2.20 FER performance of ES-GBFDA and MV-SFA for a (1536,1344) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel”, “2.21 FER performance of Min-Max algorithm, ES-GBFDA and MV-SFA for a (837,723) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel”, “2.22 Complete partial parallel architecture for MV-SFA”, “2.23 CNU units for MV-SFA” y “2.24 VNU unit for MV-SFA”.
- El algoritmo: Multiple-Vote Symbol Flipping Decoding Algorithm.

---

# Abstract

This thesis studies the design of non-binary error-correction decoders for high-speed modern communication systems. The objective is to propose low complexity decoding algorithms for non-binary low-density parity-check (NB-LDPC) and Reed-Solomon codes, to implement efficient hardware architectures.

In the first part of the thesis, we analyze the bottlenecks of the existing algorithms and architectures for NB-LDPC decoders and we propose low-complexity high-speed solutions based on symbol-flipping algorithms. First, flooding schedule solutions are designed with the aim of obtaining the highest-throughput possible without considering coding gain. Two different decoders based on clipping and blocking techniques are proposed; however, the maximum frequency is limited due to an excessive wiring. For this reason, we explore some methods to reduce the routing problems in NB-LDPC. A half-broadcasting architecture for symbol-flipping algorithms that mitigates the routing congestion is proposed. As the flooding schedule solutions with higher throughput are sub-optimal from a frame error rate performance point of view, we decide to design solutions for the serial schedule, with the objective of reaching higher throughput but keeping the coding gain of the original symbol flipping algorithms. Two serial schedule algorithms and architectures are introduced, reducing the required area resources of the decoders and increasing the maximum speed achievable. Finally, we generalize symbol-flipping algorithms and we show that some particular cases can achieve a coding gain close to Extended Min-sum and Min-max algorithms with lower complexity. An efficient architecture is proposed, showing that area resources are reduced to half compared to a direct mapping solution.

In the second part of the thesis, soft-decision decoding Reed-Solomon algorithms are compared, concluding that low complexity Chase (LCC) algorithm is the most efficient solution if high-speed is the main objective. However, LCC schemes are based on interpolation, which introduces some hardware limitations due to its complexity. In order to reduce complexity without modifying performance, we propose a soft-decision LCC scheme based on hard-decision algorithms. An efficient architecture is designed for this new scheme.

---

# Resumen

En esta tesis se estudia el diseño de decodificadores no-binarios para la corrección de errores en sistemas de comunicación modernos de alta velocidad. El objetivo es proponer soluciones de baja complejidad para los algoritmos de decodificación basados en los códigos de comprobación de paridad de baja densidad no-binarios (NB-LDPC) y en los códigos Reed-Solomon, con la finalidad de implementar arquitecturas hardware eficientes.

En la primera parte de la tesis se analizan los cuellos de botella existentes en los algoritmos y en las arquitecturas de decodificadores NB-LDPC y se proponen soluciones de baja complejidad y de alta velocidad basadas en el volteo de símbolos. En primer lugar, se estudian las soluciones basadas en actualización por inundación con el objetivo de obtener la mayor velocidad posible sin tener en cuenta la ganancia de codificación. Se proponen dos decodificadores diferentes basados en *clipping* y técnicas de bloqueo, sin embargo, la frecuencia máxima está limitada debido a un exceso de cableado. Por este motivo, se exploran algunos métodos para reducir los problemas de rutado en códigos NB-LDPC. Como solución se propone una arquitectura basada en difusión parcial para algoritmos de volteo de símbolos que mitiga la congestión por rutado. Como las soluciones de actualización por inundación de mayor velocidad son sub-óptimas desde el punto de vista de capacidad de corrección, decidimos diseñar soluciones para la actualización serie, con el objetivo de alcanzar una mayor velocidad manteniendo la ganancia de codificación de los algoritmos originales de volteo de símbolo. Se presentan dos algoritmos y arquitecturas de actualización serie, reduciendo el área y aumentando de la velocidad máxima alcanzable. Por último, se generalizan los algoritmos de volteo de símbolo y se muestra como algunos casos particulares puede lograr una ganancia de codificación cercana a los algoritmos Min-sum y Min-max con una menor complejidad. También se propone una arquitectura eficiente, que muestra que el área se reduce a la mitad en comparación con una solución de mapeo directo.

En la segunda parte de la tesis, se comparan algoritmos de decodificación Reed-Solomon basados en decisión blanda, concluyendo que el algoritmo de baja complejidad Chase (LCC) es la solución más eficiente si la alta velocidad es el objetivo

---

principal. Sin embargo, los esquemas LCC se basan en la interpolación, que introduce algunas limitaciones hardware debido a su complejidad. Con el fin de reducir la complejidad sin modificar la capacidad de corrección, se propone un esquema de decisión blanda para LCC basado en algoritmos de decisión dura. Por último se diseña una arquitectura eficiente para este nuevo esquema.



# Resum

En aquesta tesi s'estudia el disseny de descodificadors no-binàris per a la correcció d'errors en sistemes de comunicació moderns d'alta velocitat. L'objectiu és proposar solucions de baixa complexitat per als algorismes de decodificació basats en els codis de comprovació de paritat de baixa densitat no-binàris (NB-LDPC) i en els codis Reed-Solomon, amb la finalitat d'implementar arquitectures hardware eficients.

A la primera part de la tesi s'analitzen els colls d'ampolla existents en els algorismes i en les arquitectures de descodificadors NB-LDPC i es proposen solucions de baixa complexitat i d'alta velocitat basades en el volteig de símbols. En primer lloc, s'estudien les solucions basades en actualització per inundació amb l'objectiu d'obtenir la major velocitat possible sense tenir en compte el guany de codificació. Es proposen dos decodificadors diferents basats en *clipping* i tècniques de bloqueig, però, la freqüència màxima és limitada a causa d'un excés de cablejat. Per aquest motiu, s'exploren alguns mètodes per reduir els problemes de rutat als codis NB-LDPC. Com a solució es proposa una arquitectura basada en difusió parcial per algorismes de volteig de símbols que mitiga la congestió per rutat. Com les solucions de actualització per inundació de major velocitat són sub-òptimes des del punt de vista de capacitat de correcció, dissenyarem solucions per a la actualització sèrie, amb l'objectiu d'assolir una major velocitat mantenint el guany de codificació dels algorismes originals de volteig de símbol. Es presenten dos algorismes i arquitectures d'actualització sèrie, reduint l'àrea i augmentar la velocitat màxima assolible. Per últim, es generalitzen els algorismes de volteig de símbol i es mostra com en alguns casos particulars pot aconseguir un guany de codificació proper als algorismes Min-sum i Min-max amb una menor complexitat. També es proposa una arquitectura eficient, que mostra que l'àrea es redueix a la meitat en comparació amb una solució de mapeig directe.

En la segona part de la tesi, es comparen algorismes de decodificació Reed-Solomon basats en decisió tova, conclouent que l'algoritme de baixa complexitat Chase (LCC) és la solució més eficient si l'alta velocitat és l'objectiu principal. No obstant això, els esquemes LCC es basen en interpolació, que introdueix algunes limitacions

---

hardware a causa de la seua complexitat. Per tal de reduir la complexitat sense modificar la capacitat de correcció, es proposa un esquema de decisió tova per LCC basat en algoritmes de decisió dura. Per últim es va dissenyar una arquitectura eficient per aquest nou esquema.

# Acknowledgments

First, I would like to thank my supervisor Prof. Javier Valls for the guidance and support provided not just during the development of the Ph.D. degree, but also during the Master's thesis and the Bachelor degree project.

Besides, I am also grateful to the faculty members, students and staff of GISED and Department of Electronic Engineering in EPSG, specially with Prof. María José Canet.

For their technical advice, suggestions and collaboration in different projects, I would like to thank Prof. Pramod Kumar Meher, Prof. Mark Flanagan and Dr. Kiran Gunnam.

Furthermore, I would like to thank Prof. David Declercq for making possible my visit at ENSEA and giving me the chance of working in some of his projects and starting new ones. I would also like to express my gratitude to Prof. Declercq and Dr. Erbao Li for the valuable time spent on many fruitful discussions.

This research was supported by VALi+d grant (Grant No. ACIF/2011/023), sponsored by Generalitat Valenciana (Conselleria d'Educació), and a FPU grant (Grant No. AP2010-5178) sponsored by the Spanish Government. I would like to thank the sponsors.

For the feedback offered during the review of this manuscript, I would like to express my sincere gratitude to Professor Antonio García, Professor Alexandre Graell and Professor Camille Leroux. With their comments the final version provides a wider outlook of the work.

Finally, my deepest gratitude goes to my parents for their love, infinite patience and continuous encouragement. I don't have enough words to thank them for all the efforts that they have done to help me throughout my life. Also, I would like to show my greatest appreciation to Empar, for her understanding and her good sense of humor, cheering me up and supporting me in difficult times. Without them, it would have been impossible to develop this work.



# Contents

Relación de contenidos inhibidos	I
Abstract	I
Resumen	III
Resum	VI
Acknowledgments	VIII
Contents	XI
List of Acronyms	XXI
Preface	1
I Non-binary low-density parity-check decoding	7
1 State of the art of non-binary low-density parity-check decoding algorithms	9
1.1 Background concepts of non-binary low-density parity-check codes . . . . .	9
1.2 Basics of parity check matrix construction . . . . .	12
1.3 Qary-Sum-Product derived algorithms: Check node update function complexity and routing problems . . . . .	15

1.4 Forward-backward and T-EMS algorithms . . . . .	19
1.4.1 Forward-backward algorithm . . . . .	19
1.4.2 T-EMS algorithm . . . . .	25
1.5 Symbol-flipping algorithms . . . . .	33
1.6 Advantages of non-binary low-density parity-check decoding . . . . .	39
1.7 Conclusions . . . . .	41
2 Contributions to non-binary low-density parity-check decoding	43
2.1 Parallel schedule algorithms and architectures . . . . .	43
2.1.1 Modified GBFDA . . . . .	44
2.1.2 Finite precision analysis for M-GBFDA: control of the data growth . . . . .	45
2.1.3 Blocking technique . . . . .	48
2.1.4 Frame error rate performance comparison between NB-LDPC decoders. . . . .	50
2.1.5 Partial parallel architectures for the flooding schedule . . . . .	51
2.1.6 Conclusions of the architectures for the parallel schedule of M-GBFDA. . . . .	70
2.2 Serial schedule algorithms and architectures . . . . .	72
2.2.1 Enhanced Serial GBFDA . . . . .	72
2.2.2 Alternative voting process for serial schedule of GBFDA . . . . .	74
2.2.3 Alternative initialization for serial schedule of GBFDA . . . . .	76
2.2.4 Simplified ES-GBFDA . . . . .	78
2.2.5 Frame error rate performance comparison between NB-LDPC decoders. . . . .	81
2.2.6 Partial parallel architectures for the serial schedule . . . . .	81
2.2.7 Conclusions of the architectures for the serial schedule of GBFDA . . . . .	88
2.3 Multiple-vote algorithm and architecture . . . . .	90
2.3.1 Multiple-Vote Symbol Flipping Decoding Algorithm . . . . .	91
2.3.2 Parameters of MV-SFA and FER performance comparisons . . . . .	95
2.3.3 Partial parallel architecture for MV-SFA . . . . .	97
2.3.4 Conclusions of the multiple-vote symbol flipping algorithm . . . . .	111
2.4 Comparisons with other works . . . . .	112
2.5 Conclusions . . . . .	116

II	Algebraic soft-decision Reed-Solomon decoding	119
3	State of the art of algebraic soft-decision Reed-Solomon decoding algorithms	121
3.1	Background concepts of Reed-Solomon codes . . . . .	121
3.2	Complexity of soft-decision algorithms for Reed-Solomon codes . . . . .	123
3.3	Low complexity Chase algorithm for Reed-Solomon soft-decision . . . . .	125
3.4	Conclusions . . . . .	128
4	Contributions to algebraic soft-decision decoding algorithms	129
4.1	Low complexity Chase decoding algorithm based on HDD . . . . .	129
4.1.1	Multiplicity assignment . . . . .	130
4.1.2	Syndrome computation . . . . .	131
4.1.3	Key equation solver . . . . .	133
4.1.4	Chien search, Forney and test-vector selection . . . . .	133
4.1.5	Frame error rate performance for LCC based on HDD . . . . .	134
4.2	Architecture for LCC decoder based on HDD . . . . .	136
4.2.1	Multiplicity-assignment unit . . . . .	136
4.2.2	Syndromes computation unit . . . . .	137
4.2.3	Key equation solver unit . . . . .	145
4.2.4	Chien search, Forney and test-vector selection units . . . . .	146
4.2.5	Area and timing complexity analysis . . . . .	148
4.2.6	FPGA and ASIC results . . . . .	149
4.3	Comparisons with other works . . . . .	150
4.4	Conclusions . . . . .	152
III	Conclusion and future works	153
5	Conclusion and future works	155
5.1	Conclusions . . . . .	155
5.2	Future research lines . . . . .	158
Bibliography		161





# List of Figures

1.1	Different representations for NB-LDPC codes: a) Tanner graph, b) Parity matrix and c) Parity check equations . . . . .	10
1.2	Tanner graph representation for one iteration with flooding or parallel scheduling . . . . .	11
1.3	Tanner graph representation for one iteration with serial layered scheduling . . . . .	11
1.4	FER performance of log-QSPA algorithm, T-EMS algorithm, Min-max algorithm and GBFDA for a (837,723) NB-LDPC code over GF(32) with a binary phase-shift keying (BPSK) modulation in an additive white Gaussian noise (AWGN) channel. . . . .	34
2.1	Effect of the modified tentative decoding with the number of iterations	46
2.2	FER and BER performance of Min-max algorithm, GBFDA, M-GBFDA and MD algorithm for a (837,723) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel. . . . .	52
2.3	CNU unit for M-GBFDA message passing architecture. . . . .	53
2.4	VNU basic cell for M-GBFDA message passing architecture with clipping. . . . .	55
2.5	VNU unit for M-GBFDA message passing architecture with clipping.	55
2.6	VNU basic cell for M-GBFDA message passing architecture with blocking. . . . .	57
2.7	VNU unit for M-GBFDA message passing architecture with blocking.	57

2.8	Extra hardware to compute blocking technique with $r = Q_b$ for M-GBFDA. . . . .	58
2.9	Complete message passing architecture for M-GBFDA. . . . .	61
2.10	Partial CNU unit for M-GBFDA half-broadcasting architecture. . .	68
2.11	CNU-VNU unit for M-GBFDA half-broadcasting architecture. . .	69
2.12	Complete half-broadcasting architecture for M-GBFDA. . . . .	69
2.13	FER performance of Min-max, symbol flipping and SSRB algorithms, for a (403,255) NB-LDPC code over GF(32) and a (255,175) NB-LDPC code over GF(256) with a BPSK modulation in an AWGN channel. . . . .	75
2.14	FER performance of Min-max algorithm, GBFDA, ES-GBFDA and SES-GBFDA for a (837,723) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel. . . . .	82
2.15	Complete partial parallel architecture for ES-GBFDA and SES-GBFDA. . . . .	83
2.16	CNU basic cell for ES-GBFDA and SES-GBFDA. . . . .	84
2.17	VNU unit for ES-GBFDA and SES-GBFDA. VNU unit for SES-GBFDA does not include the interconnections marked with *. . . .	85
2.18	Basic cell for ES-GBFDA ( $d_v = 4$ ). . . . .	85
2.19	Basic cell for SES-GBFDA ( $d_v = 4$ ). . . . .	87
2.20	FER performance of ES-GBFDA and MV-SFA for a (1536,1344) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel. . . . .	98
2.21	FER performance of Min-Max algorithm, ES-GBFDA and MV-SFA for a (837,723) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel. . . . .	99
2.22	Complete partial parallel architecture for MV-SFA. . . . .	100
2.23	CNU units for MV-SFA: a) CNU unit for hard decision symbols. b) CNU unit for a test vector $i$ . . . . .	102
2.24	VNU unit for MV-SFA. . . . .	103
2.25	General architecture for a 4-minimum finder in a 27-element list. .	105

2.26 Architecture for the first stage ( $l = 0$ ) of a 4-minimum finder in a 27-element list. . . . .	106
2.27 Architecture for stages which are different from the first one ( $l > 0$ ) of a 4-minimum finder in a 27-element list. The index 0 of the inputs is for the 4th minimums and the index 3 is for the 1st minimums. . . . .	108
3.1 Block diagram of a generic Hard-Decision RS decoder . . . . .	122
3.2 Test vectors mapped in a hypercube for a decoder with $\eta = 3$ . . . . .	126
3.3 Test vectors mapped in a binary tree for a decoder with $\eta = 3$ . . . . .	127
4.1 Block diagram of the LCC soft-decoder based on decoding failure . . . . .	130
4.2 FER performance for a (255,239) RS code applying HDD and an LCC decoder based on HDD for $\eta = 3$ and $\eta = 4$ over an AWGN channel with a BPSK modulation . . . . .	135
4.3 Architecture for the partial syndrome computation with the common symbols . . . . .	137
4.4 Architecture for the contribution to the syndrome of the uncommon symbols . . . . .	139
4.5 Storage of the $2t$ syndromes of the $2^n$ test-vectors . . . . .	144
4.6 KES unit for computing two test-vectors . . . . .	146
4.7 (a) Basic $c_j$ block. (b) Chien search block diagram . . . . .	147
4.8 Forney's algorithm classical architecture . . . . .	147

*List of Figures*

---

# List of Tables

2.1	Operations and complexity of the techniques for control of data growth in one iteration of M-GBFDA (worst case) . . . . .	48
2.2	Theoretical estimation of area-complexity for M-GBFDA message passing architecture with clipping and $Q_b = 5$ bits, $r = 5$ bits (639246 <i>XOR</i> gates) . . . . .	65
2.3	Theoretical estimation of area-complexity for M-GBFDA message passing architecture with blocking and $Q_b = 5$ bits, $r = 5$ bits (633162 <i>XOR</i> gates) . . . . .	65
2.4	Theoretical estimation of area-complexity for M-GBFDA message passing architecture with blocking and $Q_b = 5$ bits, $r = 4$ bits (699618 <i>XOR</i> gates) . . . . .	66
2.5	Theoretical estimation of area-complexity for M-GBFDA message passing architecture with saturation and $Q_b = 6$ bits, $r = 5$ bits (803722 <i>XOR</i> gates) . . . . .	66
2.6	Theoretical estimation of area-complexity for M-GBFDA message passing architecture with saturation and $Q_b = 7$ bits, $r = 5$ bits (920618 <i>XOR</i> gates) . . . . .	67
2.7	Implementation results for Virtex VI FPGA device and 90nm process ASIC of the decoder architecture for M-GBFDA with clipping and blocking . . . . .	67
2.8	Implementation results for Virtex VI FPGA device and 90nm process ASIC of the decoder half-broadcasting architecture for M-GBFDA with blocking . . . . .	71
2.9	Complexity of simplified Min-max, SSRB and two symbol-flipping algorithms . . . . .	76

2.10	Percentage of symbols that received $\beta$ votes in ES-GBFDA for $E_b/N_0 = 4.6dB$ . . . . .	79
2.11	Percentage of symbols that received $\beta$ votes in ES-GBFDA for $E_b/N_0 = 5dB$ . . . . .	79
2.12	Percentage of symbols that were voted at the previous iteration and are also voted at the current one in ES-GBFDA . . . . .	80
2.13	ASIC results for a 90nm CMOS process of ES-GBFDA, SES-GBFDA and M-GBFDA partial parallel architectures . . . . .	89
2.14	Theoretical estimation of area-complexity for MV-SFA message passing architecture with clipping and $Q_b = 7$ bits and $Q'_b = 5$ bits, (1527228 XOR gates) . . . . .	110
2.15	Theoretical estimation of area-complexity for $R$ minimum finder in a $d_c$ list (radix $R$ , $R = L = 4$ ) and $Q_b = 7$ bits . . . . .	110
2.16	Estimated results of ES-GBFDA and MV-SFA partial parallel architectures . . . . .	111
2.17	Efficient architectures for high-rate NB-LDPC codes . . . . .	114
2.18	Upper and lower bounds for binary and non-binary LDPC architectures . . . . .	116
4.1	Timing diagram of the syndrome processing for eight test-vectors . . . . .	145
4.2	Theoretical estimation of the area-complexity of the proposed LCC decoder . . . . .	150
4.3	Equivalence in XOR gates of hardware resources . . . . .	151
4.4	XOR-gate complexity of LCC decoders for (255,239) RS code . . . . .	152
4.5	XOR-gate complexity of a hard-decision decoder for (255,239) RS code . . . . .	152

# List of Algorithms

1	Qary-Sum-Product algorithm . . . . .	16
2	Extended Min-Sum algorithm . . . . .	16
3	Min-max algorithm . . . . .	17
4	Forward-backward algorithm . . . . .	20
5	Improved Min-max algorithm . . . . .	24
6	Trellis EMS algorithm . . . . .	25
7	Generalized Bit-Flipping algorithm . . . . .	35
8	Modified tentative decoding algorithm . . . . .	44
9	M-GBFDA with blocking technique . . . . .	49
10	Enhanced Serial GBFDA . . . . .	73
11	Simplified ES-GBFDA . . . . .	80
12	Multiple-Vote Symbol Flipping Decoding algorithm . . . . .	96
13	Syndrome computation for LCC decoding . . . . .	132
14	Inversionless Berlekamp-Massey algorithm . . . . .	134





# List of Acronyms

**ASD** Algebraic Soft-Decision

**ASIC** Application-Specific Integrated Circuit

**AWGN** Additive White Gaussian Noise

**BER** Bit Error Rate

**BGMD** Bit-level Generalized Minimum Distance

**BM** Berlekamp-Massey

**BPSK** Binary Phase-Shift Keying

**CMOS** Complementary Metal-Oxide-Semiconductor

**CNU** Check Node Update

**DF** Decoding Failure

**EMS** Extended Min-Sum

**ES-GBFDA** Enhanced Serial Generalized Bit Flipping Decoding Algorithm

**FEC** Forward Error Correction

**FER** Frame Error Rate

**FPGA** Field Programmable Gate Array

**GBFDA** Generalized Bit Flipping Decoding Algorithm

**GF** Galois Field

**GMV-SFA** Generalized Multiple-Vote Symbol Flipping Algorithm

**HDD** Hard-Decision Decoder

**HDL** Hardware Description Language

<b>iBM</b>	Inversionless Berlekamp-Massey
<b>KES</b>	Key-Equation Solver
<b>KV</b>	Koetter-Vardy
<b>LCC</b>	Low Complexity Chase
<b>LDPC</b>	Low-Density Parity-Check
<b>LLR</b>	Log-Likelihood Ratio
<b>LUT</b>	Lookup Table
<b>MD</b>	Majority-logic Decodable
<b>M-GBFDA</b>	Modified-Generalized Bit Flipping Decoding Algorithm
<b>MV-SFA</b>	Multiple-Vote Symbol Flipping Decoding Algorithm
<b>NCG</b>	Net Coding Gain
<b>NB-LDPC</b>	Non-Binary Low-Density Parity-Check
<b>OIF</b>	Optical Internetworking Forum
<b>PDM-QPSK</b>	Polarization-Division-Multiplexed Quadrature Phase-Shift Keying
<b>QAM</b>	Quadrature Amplitude Modulation
<b>QSPA</b>	Qary-Sum Product Algorithm
<b>RAM</b>	Random-Access Memory
<b>ROM</b>	Read-Only Memory
<b>RS</b>	Reed-Solomon
<b>SES-GBFDA</b>	Simplified Enhanced Serial Generalized Bit Flipping Decoding Algorithm
<b>SNR</b>	Signal-to-Noise Ratio
<b>SSRB</b>	Serial Symbol-Reliability Based
<b>T-EMS</b>	Trellis Extended Min-Sum
<b>TV</b>	Test Vector
<b>UiBM</b>	Ultra-Ultra Folded inversionless Berlekamp-Massey
<b>VLSI</b>	Very Large Scale Integration
<b>VNU</b>	Variable Node Update

# Preface

Since analog-to-digital converters allow high speed, hardware implementation of error correction decoders based on soft information for 100Gbps communication systems are closer to become possible. Before high-speed analog-to-digital converters, only error correction decoders based on hard-decision were able to reach high throughput, because adding channel information required long processing time. This fact limited the maximum coding gain achievable by the decoders as they did not take into account the degree of reliability of each received symbol considering all of them as equally reliable. Nowadays, this problem is solved and the processing of the log-likelihood ratio of the received symbols can be performed at high speed. This fact has initiated an interest in soft-decision decoding for communication systems such as optical ones. Because of that, Optical Inter-networking Forum (OIF) proposed a white paper about forward error correction (FEC) solutions to be applied to a 100Gbps optical transport system. The main constraint of this document is that the net coding gain (NCG) performance has to be greater than 10dB. As the solution for this requirement is not standardized, coding theory and very large scale integration (VLSI) researchers have focused all their efforts on finding the most suitable solution. To accomplish both coding gain and speed constraints, non-binary low-density parity-check (NB-LDPC) codes are the strongest FEC candidates, because they have some advantages compared to their binary counterparts. However, hardware implementation of these codes are far from accomplish the constraints efficiently. With the actual state of the art 1515 NB-LDPC decoders running in parallel are required to get a throughput of 100Gbps, in other case, the decoder will be a bottle neck. More realistic schemes talk about using 4 parallel NB-LDPC decoders of 25Gbps, unfortunately, the existing proposals are far from this speed. On the other hand, NB-LDPC suffer a degradation called error floor which reduces coding gain at high frame error rates. To suppress the unwanted error floor very long codewords are required; however, the larger the codeword is, the lower the throughput is. To avoid long codewords and error floor, the concatenation of a hard-decision Reed-Solomon (RS) decoder with the NB-LDPC is the most efficient solution. In this way, RS cleans up the residual errors after the NB-LDPC decoder. In addition, OIF's white paper limits the codes to high-rate ones, with a rate bigger than 79.5% for NB-LDPC codes.

With regard to RS, the (255,239)RS code has been adopted as it was the commonest in past optical communication standards.

In this manuscript, several proposals for a low-complexity high-speed NB-LDPC decoder are included. Even if they do not reach 25Gbps, they increase considerable throughput of these kind of decoders at the cost of some performance loss. Moreover, a low-complexity high-speed soft-decision decoder for RS codes is proposed. Although only hard-decision RS decoders have been considered in optical communication schemes, a soft-decision RS decoder with similar complexity can be useful. It can be released that no final solution for optical communications has been reached, however, some interesting conclusions have been obtained from this work that will be profitable in future works.

## Objectives

The main objective of this thesis is to propose NB-LDPC and RS decoding algorithms with low complexity and the minimum performance loss degradation to reach high-throughput architectures. To reach this main target it is necessary to accomplish other intermediate objectives:

1. Make a deep review of the state of the art in NB-LDPC and RS decoding.
2. Study the different existing algorithms and conclude what makes them complex.
3. Study the architectures proposed by other authors and detect the bottlenecks that produce inefficient solutions.
4. Propose new low complexity algorithms or schemes for decoding NB-LDPC and RS codes.
5. Design new architectures for the existing algorithms and for the new proposals.
6. Implement the designed decoders and make the necessary modifications to reach the highest possible throughput.

## Methodology

Next, we describe the methodology followed to achieve the previously described objectives.

First, a study of the contributions of the coding theory community is done evaluating the possible interest of different decoding algorithms from a hardware perspective. For the algorithmic solutions classified as *hardware friendly*, software models

are developed. These models describe a complete communication system which includes encoder, modulator channel, demodulator and decoder. The decision of the programming language (Matlab, Mex, C or C++) is taken depending on the complexity and time required by simulations. Monte Carlo simulations are applied to the software models to compute the Frame Error Rate (FER) performance and the Bit Error Rate (BER) performance. These results are compared with the ones already published to validate them. To compute the Frame Error Rate and the Bit Error Rate for each value of  $E_b/N_0$  a random message is transmitted each time and 50 error packages are detected. After the reference is validated, a finite precision analysis is performed.

Before starting the design of a new architecture, we analyze the ones already existing and we study their bottlenecks. After this, we design new architectures trying to solve these problems. To compare its efficiency in a fair way we apply analytic methods which do not depend on technology. These analytic methods consist in the computation of memory, control and arithmetic hardware resources of an architecture in terms of *XOR* gates as well as their critical path.

To validate the results of this analysis we make an HDL model of the proposed architectures and extract synthesis and post place and route reports for FPGA and ASIC technologies. Xilinx and Cadence tools were used for FPGA and ASIC implementations respectively; Modelsim is the software used to verify the hardware models.

Finally, we compare the implementation results of the proposed solution with the ones found in literature. This methodology is iterative, so after one design is finished we try to modify or propose a new algorithm with lower complexity and we start all the process again from the software model to the HDL description. In the same way, different versions of the same architectures are evaluated as well as reviews of the state of the art of the algorithms proposed by the coding theory community are constantly performed.

## Contributions

In this thesis several algorithms, methods and hardware architectures for implementing NB-LDPC and soft-decision RS decoders have been proposed and improved. Below the main contributions of this thesis are enumerated.

1. A modified tentative decoding algorithm for decoders based on Generalized Bit Flipping Algorithm that reduces both arithmetic and memory resources and its derived partial parallel architecture based on the message passing scheme.

2. A simplified data growth control method for Generalized Bit Flipping Algorithm that reduces the number of clock cycles required by the Variable Node Update processor and allows higher throughput and its derived partial parallel architecture based on half-broadcasting.
3. An enhanced serial description of Generalized Bit Flipping Algorithm that improves the Frame Error Rate performance and reduces the storage resources compared to the previous contributions. Also an efficient partial parallel architecture is proposed.
4. A simplified version of the enhanced serial description of Generalized Bit Flipping Algorithm that allows the quantization of intrinsic information with one bit with negligible performance loss.
5. An alternative initialization that avoids clipping techniques to control the data growth and hence reduces area and critical path.
6. An efficient partial parallel architecture combining both alternative initialization and simplified version of the enhanced serial description for Generalized Bit Flipping Algorithm.
7. A generalization of symbol-flipping algorithms based on a multiple voting procedure that reaches a similar performance to the algorithms derived from Qary-Sum Product but with less complexity.
8. An efficient architecture for a particular case of the generalized symbol-flipping algorithm that reduces to half hardware resources compared with a direct mapping and outperforms in terms of area over throughput the solutions found in literature.
9. A Low Complexity Chase decoder for Reed-Solomon codes that avoids the interpolation and reduces complexity by applying hard-decision techniques. An architecture is also proposed.

Some contributions of this thesis have been already published in the next journal and conference papers:

- International Journals

1. F. Garcia-Herrero, J. Valls, P.K. Meher, "High speed RS(255,239) decoder based on LCC decoding," *Circuits, Systems, and Signal Processing*, Vol.30, No.6, pp.1643-1669, May.2011
2. F. Garcia-Herrero, M.J. Canet, J.Valls, M.Flanagan, "Serial Symbol-Reliability Based Algorithm For Decoding Non-Binary LDPC Codes," *IEEE Communications Letters*, Vol.16 No.6, pp.909-912, Jun.2012

3. F. Garcia-Herrero, M.J. Canet, J. Valls, "Architecture of Generalized Bit-Flipping Decoding for High-Rate Non-binary LDPC Codes," *Circuits, Systems, and Signal Processing*, Vol.32 No.2, pp.727-741, April 2013
4. F. Garcia-Herrero, M.J. Canet, J. Valls, "Non-Binary LDPC Decoder Based on Simplified Enhanced Generalized Bit Flipping Algorithm," *IEEE Transactions on Very Large Scale Integration*, to be published 2013

- International Conferences

1. F. Garcia-Herrero, M.J. Canet, J. Valls, "Decoder for an Enhanced Serial Generalized Bit Flipping Algorithm," *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Sevilla, Spain, Dec. 2012
2. F. Garcia-Herrero, M.J. Canet, J. Valls, "High-Speed NB-LDPC Decoder For Wireless Applications," *International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, Okinawa, Japan, Nov. 2013

Other contributions which have been submitted for publication are:

- International Journals

1. F. Garcia-Herrero, E. Li, D. Declercq and J. Valls, "Multiple-Vote Symbol Flipping Decoder for Non-Binary LDPC Codes," *IEEE Transactions on Very Large Scale Integration*, submitted on May 2013

Also a patent has been written: F. Garcia-Herrero, D. Declercq, E. Li, and J. Valls, "Generalized Symbol Flipping Decoder for Non-Binary LDPC Codes".

A previous work related with Reed-Solomon soft-decision was developed during the Master's thesis and it was published in an international journal. The conclusions of this paper are useful for understanding some of the choices that were done in the Reed-Solomon soft-decision topic during the development of this thesis. Part of the Master's thesis is included in:

- International Journal

1. F. Garcia-Herrero, M.J. Canet, J. Valls, P.K. Meher, "High-Throughput Interpolator Architecture for Low-Complexity Chase Decoding of RS Codes," *IEEE Transactions on Very Large Scale Integration*, Vol.20, No.3, pp.568-573, Mar.2012

Finally, estimation of hardware complexity of Trellis Extended Min Sum algorithm over high order Galois Field was performed and included in an international conference. This work helped to see the benefits and drawbacks of working with a Min Sum derived algorithms with high degree of parallelism. Part of the conclusions obtained from this work laid the foundation of the multiple voting algorithm for symbol flipping developed in this thesis.

- International Conference

1. E. Li, D. Declercq, K. Gunnam, F. Garcia-Herrero, J. Omar and J. Valls, “Low Latency T-EMS decoder for NB-LDPC codes,” Asilomar Conference on Signals, Systems and Computers 2013

## **Thesis structure**

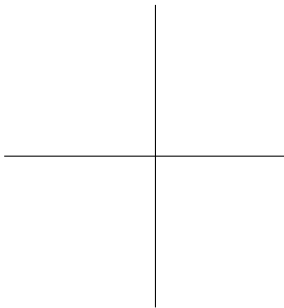
The rest of the manuscript has three blocks. The first one deals with NB-LDPC decoders. The second block is focused on low-complexity algebraic soft-decision RS decoding. Each block is divided in two chapters. The first makes a brief summarize of the basic concepts of the codes under study, and reviews the state of the art concluding which are the bottlenecks to improve and the requirements and more interesting topics of research from the author’s perspective. The second chapter of each block includes the most relevant contributions of the thesis, first explaining the algorithmic novelties and its impact in coding gain and then detailing the derived architectures. Each chapter includes its own conclusions and at the end of each block comparisons with the most efficient results found in literature are performed. The last block summarizes the most important contributions and conclusions and also a describes future research topics with some definite possible ideas.





Part I

**Non-binary low-density  
parity-check decoding**





# Chapter 1

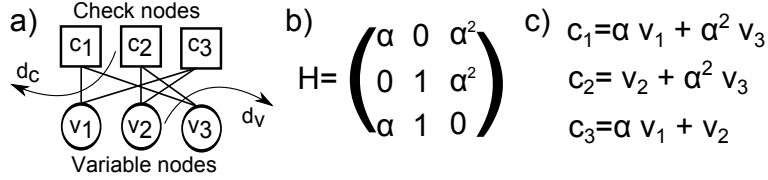
## State of the art of non-binary low-density parity-check decoding algorithms

In this chapter the reader will find: i) a brief summary of the basic concepts of non-binary low-density parity-check codes; ii) a detailed description of the main algorithms derived from Qary Sum-Product; iii) a discussion, under a VLSI perspective, of the state of the art of efficient algorithms to solve the check node update of extended Min-Sum and Min-max algorithms; iv) a review of symbol-flipping algorithms for high-rate codes; and finally, v) a comparison between binary low-density parity-check codes and the non-binary ones, highlighting the reasons that make implementation of non-binary low-density parity-check decoders an interesting topic of research.

### 1.1 Background concepts of non-binary low-density parity-check codes

Let us consider a  $(N, K)$  NB-LDPC code over  $\text{GF}(q)$  ( $q = 2^p$ ) with codeword length  $N$ , information length  $K$  and code rate  $R = K/N$ , lengths are expressed in terms of  $\text{GF}(q)$  symbols. This NB-LDPC code is defined by a very sparse parity check matrix  $\mathbf{H}$ , which has  $N$  columns and  $M$  rows. The relation between the matrix and the code parameters is  $K = N - \text{rank}(\mathbf{H})$ . The non-zero coefficients of  $\mathbf{H}$  are  $h_{m,n}$ , where  $m$  is the row index and  $n$  the column index. Each row represents a parity check equation, named as check node. The number of non-zero elements in a row is known as check node degree and is denoted as  $d_c$ . The

columns represent the variable nodes. The number of non-zero elements of the column is known as variable node degree and is denoted as  $d_v$ .  $\mathcal{N}(m)$  defines the set of variable nodes connected to the  $m$ -th check node, and  $\mathcal{M}(n)$  the set of check nodes connected to the  $n$ -th variable node. This matrix can be represented by a Tanner graph, Fig. 1.1. The nodes of this graph can be separated into variable and check nodes, where the edges may only connect two nodes not residing in the same class. The squares represent the  $M$  parity check nodes of the code and the circles are the  $N$  variable nodes. Each check node  $m$  has  $d_c$  incoming connections with the variable nodes that belong to  $\mathcal{N}(m)$ . In the same way, each variable node  $n$  has  $d_v$  incoming connections with the check nodes that belong to  $\mathcal{M}(n)$ . This representation will be useful in the following sections to easily analyze the complexity of the different decoding algorithms for these codes.

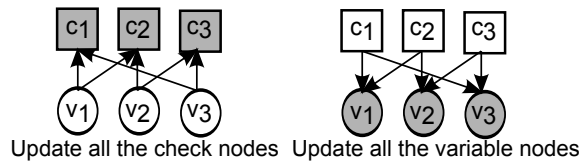


**Figure 1.1:** Different representations for NB-LDPC codes: a) Tanner graph, b) Parity matrix and c) Parity check equations

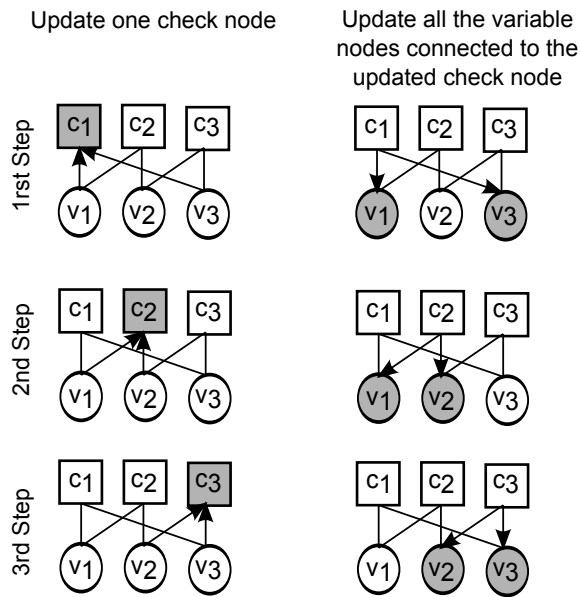
Considering a  $K$ -length vector message to be encoded,  $\mathbf{m}$ , the codeword generated by the NB-LDPC encoder is  $\mathbf{c} = \mathbf{m}\mathbf{G}$ . Where  $\mathbf{G}$  is a  $K \times n$  generator matrix, which should accomplish that  $\mathbf{c}\mathbf{H}^T = 0$ . The generator matrix can be computed applying Gaussian elimination over  $\text{GF}(q)$ . There are different efficient methods to perform and implement the encoder step, however, this is not the aim of this manuscript.

The transmitted codeword symbols are  $\mathbf{c} = (c_0, c_1, \dots, c_{N-1})$  and the received symbol sequence is defined as  $\mathbf{y} = (y_0, y_1, \dots, y_{N-1})$ , where  $\mathbf{y} = \mathbf{c} + \mathbf{e}$  and  $\mathbf{e}$  is the error vector introduced by the channel. The log-likelihood ratio (LLR) reliability is computed as  $L_n[x] = \log [P(c_n = \alpha^x | \mathbf{y}_n) / P(c_n = x | \mathbf{y}_n)]$ , for each  $x \in \text{GF}(q)$ . The LLR vector of the  $n$ -th symbol is  $\mathbf{L}_n = (L_n[0], L_n[1], \dots, L_n[q-1])$ . The hard-decision based on  $\mathbf{L}_n$  is called  $z_n \in \text{GF}(q)$ , which is the symbol with the most reliable value in  $\mathbf{L}_n$ . All the decoding NB-LDPC algorithms work using the channel information (LLRs) as initialization and then applying iteratively the check node update (CNU) and variable node update (VNU) functions following one of the two main schedules: parallel (flooding) schedule Fig. 1.2 or serial schedule Fig. 1.3. In each iteration, a decoder with parallel schedule first updates all the CNUs and after that using the new CNU information updates all the VNUs. The serial schedule can be classified as layered or shuffled. One iteration of a decoder based on a layered scheduling is completed updating one CNU and after that updating

all the VNUs connected to this check node, repeating this process until all the CNUs are updated. A decoder based on shuffled schedule updates one VNU and then uses this new information to update all the CNUs connected to this variable node. To complete one iteration all the VNUs and CNUs are updated following the same process. The serial schedule, compared to flooding one, reduces the number of iterations required by the decoding algorithm to get a certain performance.



**Figure 1.2:** Tanner graph representation for one iteration with flooding or parallel scheduling



**Figure 1.3:** Tanner graph representation for one iteration with serial layered scheduling

## 1.2 Basics of parity check matrix construction

There is a wide variety of methods to construct parity check matrices for NB-LDPC codes. However, from a hardware point of view, the most convenient are the ones that leads to structured parity check matrices. This means that the matrix can be divided in different parts which can be associated to independent processors. Inside the structured matrices we find quasi-cyclic codes, which have special properties that can be taken into account to reduce the complexity of the derived architectures. Several classes and methods to obtain quasi-cyclic codes are described at [1]. In this section we do not detail any particular class of non-binary quasi-cyclic LDPC, but we just include some of the main concepts about its construction.<sup>1</sup>

First, the length of the codeword in terms of symbols,  $N$ , has to be chosen. According to this we can select a field that accomplishes  $(q-1) \times (q-1) \geq N$ . After this, a  $(q-1) \times (q-1)$  matrix composed of different  $\text{GF}(q)$  symbols is built. The selection of the symbols of each row depends on the class of the code which is selected. Usually there is some kind of relation between two consecutive rows, for example the second row is equal to the first row after being multiplied by a symbol and subtracted one (multiplicative group class) or the second row is equal to the first plus a random symbol (additive subgroups class). These methods leads to a matrix,  $\mathbf{U}$ , which usually has zero elements in the main diagonal.

$$\mathbf{U} = \begin{bmatrix} 0 & \alpha^x & \alpha^y & \dots & \alpha^z \\ \alpha^z & 0 & \alpha^x & \alpha^y & \dots \\ \dots & \alpha^y & \alpha^z & 0 & \alpha^x \\ \alpha^x & \alpha^y & \dots & \alpha^z & 0 \end{bmatrix}$$

Each nonzero element of  $\mathbf{U}$  is associated to a  $(q-1) \times (q-1)$  quasi-cyclic matrix. The matrix that represents  $\alpha^0 = 1$  has the symbol  $\alpha^0 = 1$  in the first location of the first row,  $\alpha^2$  in the location element of the second row,  $\alpha^3$  in the third location of the third row and so on. If the quasi-cyclic matrix is associated to the symbol  $\alpha^x$  the first row has  $\alpha^x$  in the location  $x$  of the first row,  $\alpha^{(x+1)}$  in the location  $x+1$  of the second row, etc. When  $\alpha^{(q-2)}$  is reached, the next row starts with  $\alpha^0$  again, that is the reason why is called quasi-cyclic. The matrix associated to the zero element is a  $(q-1) \times (q-1)$  all-zero matrix. Next, the quasi-cyclic matrices for the zero element,  $\alpha^0$  and  $\alpha^2$  over  $\text{GF}(4)$  are included as an example:

<sup>1</sup>Random-like LDPC codes have also been deeply analyzed in the literature due to its good performance [2], [3], [4]. However, the derived architectures based on binary random-like LDPC codes leads to high power consumption, dense wiring and low area-speed ratios [5], [6], [7]. All these disadvantages are even more highlighted in the non-binary case where the size of the Galois field is  $q > 2$ , making its implementation impractical.

$$\mathbf{Q}_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{Q}_{\alpha^0} = \begin{bmatrix} \alpha^0 & 0 & 0 \\ 0 & \alpha^1 & 0 \\ 0 & 0 & \alpha^2 \end{bmatrix}$$

$$\mathbf{Q}_{\alpha^2} = \begin{bmatrix} 0 & 0 & \alpha^2 \\ \alpha^0 & 0 & 0 \\ 0 & \alpha^1 & 0 \end{bmatrix}$$

Once we know  $\mathbf{Q}$  for each element of the field, we replace each element in  $\mathbf{U}$  by its associated  $\mathbf{Q}$ . This procedure is called matrix expansion as it is done as follows:

$$\mathbf{U}' = \begin{bmatrix} \mathbf{Q}_0 & \mathbf{Q}_{\alpha^x} & \mathbf{Q}_{\alpha^y} & \dots & \mathbf{Q}_{\alpha^z} \\ \mathbf{Q}_{\alpha^z} & \mathbf{Q}_0 & \mathbf{Q}_{\alpha^x} & \mathbf{Q}_{\alpha^y} & \dots \\ \dots & \mathbf{Q}_{\alpha^y} & \mathbf{Q}_{\alpha^z} & \mathbf{Q}_0 & \mathbf{Q}_{\alpha^x} \\ \mathbf{Q}_{\alpha^x} & \mathbf{Q}_{\alpha^y} & \dots & \mathbf{Q}_{\alpha^z} & \mathbf{Q}_0 \end{bmatrix}$$

Depending on the length of the codeword,  $d_c$  consecutive columns of  $\mathbf{U}$  are selected, accomplishing that  $N = d_c \times (q - 1)^2$ . To select the number of rows of the parity check, two different methods can be applied depending on the constraints. If the constraint is  $d_v$  we just select  $d_v$  consecutive rows. However, if the rate of the code is the constraint, we have to look for the  $N - K$  linearly independent rows that accomplish the rate constraint. In this case,  $d_v$  depends on the rate requirements. The resulting parity check matrix is  $\mathbf{H}$ . Next we include an example for a GF(4) matrix.

**Example 1.2.1** *Let us assume that the class of the matrix that we applied gives as result:*

$$\mathbf{U} = \begin{bmatrix} 0 & \alpha^2 & \alpha^1 \\ \alpha^1 & 0 & \alpha^2 \\ \alpha^2 & \alpha^1 & 0 \end{bmatrix}$$

*The required codeword length is  $N = 9$  in terms of GF(4) symbols, so  $d_c = N/(q - 1) = 3$ .  $d_v = 2$  due to a design constraint. However, as we will consider the main zero diagonal,  $d_c = 2$ .*

<sup>2</sup>Note that if the main zero diagonal is included, the codeword length is equal to  $N = (d_c + 1) \times (q - 1)$ .

After computing  $\mathbf{U}$ , we replace the nonzero elements by its associated  $\mathbf{Q}$  matrices (we expand the matrix).

$$\mathbf{U}' = \begin{bmatrix} \mathbf{Q}_0 & \mathbf{Q}_{\alpha^2} & \mathbf{Q}_{\alpha^1} \\ \mathbf{Q}_{\alpha^1} & \mathbf{Q}_0 & \mathbf{Q}_{\alpha^2} \\ \mathbf{Q}_{\alpha^2} & \mathbf{Q}_{\alpha^1} & \mathbf{Q}_0 \end{bmatrix}$$

Then,  $d_c + 1 = 2$  columns and  $d_v = 2$  rows are selected. We select the first and the third row and all the columns because in this case  $d_c = (q - 1)$ . So,  $\mathbf{H}$  as a function of the quasi-cyclic matrices is:

$$\mathbf{H} = \begin{bmatrix} \mathbf{Q}_0 & \mathbf{Q}_{\alpha^2} & \mathbf{Q}_{\alpha^1} \\ \mathbf{Q}_{\alpha^2} & \mathbf{Q}_{\alpha^1} & \mathbf{Q}_0 \end{bmatrix}$$

Including all the coefficients  $\mathbf{H}$  is :

$$\mathbf{H} = \left[ \begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & \alpha^2 & 0 & \alpha^1 & 0 \\ 0 & 0 & 0 & \alpha^0 & 0 & 0 & 0 & 0 & \alpha^2 \\ 0 & 0 & 0 & 0 & \alpha^1 & 0 & \alpha^0 & 0 & 0 \\ \hline 0 & 0 & \alpha^2 & 0 & \alpha^1 & 0 & 0 & 0 & 0 \\ \alpha^0 & 0 & 0 & 0 & 0 & \alpha^2 & 0 & 0 & 0 \\ 0 & \alpha^1 & 0 & \alpha^0 & 0 & 0 & 0 & 0 & 0 \end{array} \right]$$

As it is shown, each sub-matrix has at most one nonzero element per column and row, due to the use of the quasi-cyclic expansion. This fact allows hardware designers to increase the degree of parallelism compared to other code structures. As there is no dependency between the nonzero elements of each sub-matrix, an entire sub-matrix can be computed in parallel each time. In addition, all the sub-matrices involved in the same row can also be computed in parallel, as each one is working with a different symbol from the codeword and it does not cause any memory conflict.

The previous principles can be combined with other techniques to have a higher degree of freedom in the code design. For example, the masking technique proposed also in [1] reduces the number of nonzero elements of  $\mathbf{H}$  by introducing more zeros in some locations according to the definition of a mask. Other example is the random distribution of elements of a different field in the locations defined by the original  $\mathbf{H}$ . This method replaces the nonzero elements of  $\mathbf{H}$  by a random nonzero element of another field, in order to accomplish with constraints of the Galois Field or the codeword length in terms of bits. However, even applying these transformations, the parity check matrix  $\mathbf{H}$  keeps its structure and hence the properties that makes this methods interesting from a hardware point of view.



To sum up, the quasi-cyclic matrices proposed at [1] have special properties that allow hardware designers to implement more efficient decoders. For this reason they are the most extended in the VLSI community. During this manuscript codes based on these matrices are applied unless stated other wise.

### 1.3 Qary-Sum-Product derived algorithms: Check node update function complexity and routing problems

The complexity of the first proposed NB-LDPC decoding algorithm, Qary Sum-Product Algorithm (QSPA) [8], was too high to be applied in hardware implementations, so new versions of the algorithm in the frequency domain [9], the logarithmic domain [10] and mixed domain [11] were proposed, improving the efficiency of the CNU and VNU processing and reducing the number of required bits by the quantize version of the algorithm without introducing a degradation in the frame error rate (FER) coding gain. However, the low throughput and large amount of area required by the derived architectures [11] forced researchers to propose simplified algorithms derived from QSPA. The most important contributions are Extended Min-Sum (EMS) algorithm [12] and Min-max algorithm [13]. These algorithms reduce complexity of the decoding process introducing a small performance loss for some codes compared to QSPA, however, they were far from allowing high-speed (tens of Gbps) hardware implementations. So the effort of most of the researchers in this area has been the reduction of the CNU function which involves the real complexity of the decoding process. Next, the parallel schedules of QSPA (Algorithm 1), EMS (Algorithm 2) and Min-max (Algorithm 3) algorithms are detailed.

All QSPA, EMS and Min-max have three main steps: initialization, CNU/VNU process and tentative decoding.  $L_n(a)$ ,  $R_{m,n}(a)$  and  $W_{m,n}(a)$  are defined as the channel information, the messages exchanged from check node to variable node and the messages exchanged from variable node to check node of an element  $a \in GF(q)$ , respectively.

The difference in initialization between QSPA/EMS and Min-max is that in the first ones the LLRs are normalized to a random symbol (for example the symbol  $0 \in GF(q)$ ) and in the second one the LLRs are normalized to the most reliable symbol (hard-decision  $z_n$ ). This different initialization introduces changes in the iterative process, in particular at step A2 of the check node, which can be simplified in Min-max algorithm with a negligible performance loss compared to QSPA/EMS algorithm (less than 0.1dB of difference between EMS and Min-max).

In the CNU/VNU process, step A1 calculates symbols  $a_{n'}$  that satisfy the parity check equation when  $a_n = a$ ,  $a \in GF(q)$ . In [12] and [14], a reduction of the configuration sets (combinations of symbols) that are included in  $A_{m,n}(a)$  at step

---

**Algorithm 1** Qary-Sum-Product algorithm

---

**Input:**  $L_n(a) = \log [P(c_n = 0 | \mathbf{y}_n)/P(c_n = a | \mathbf{y}_n)]$ ,  $W_{m,n}(a) = L_n(a)$   
 Iterative process  
**for**  $j = 1 \rightarrow It_{max}$  **do**  
     Check node processing  
     A1 :  $A_{m,n}(a) := \{a_{n'} \in \text{conf}(n_m, n_c) \mid h_{m,n}a_n + \sum_{n' \in \mathcal{N}(m) \setminus n} h_{m,n'}a_{n'} = 0\}$   
     A2 :  $R_{m,n}(a) = \log(\sum_{a_{n'} \in A_{m,n}(a), n' \in \mathcal{N}(m) \setminus n} e^{\sum_{n' \in \mathcal{N}(m) \setminus n} W_{m,n'}(a_{n'})})$   
     Variable node processing  
     A3 :  $W'_{m,n}(a) = L_n(a) + \sum_{m' \in \mathcal{M}(n) \setminus m} R_{m',n}(a)$   
     A4 :  $W'_{m,n}(a) = W_{m',n}(0)$   
     A5 :  $W_{m,n}(a) = W'_{m,n}(a) - W'_{m,n}$   
     Tentatively decoding  
     A6 :  $W_n(a) = L_n(a) + \sum_{m' \in \mathcal{M}(n)} R_{m',n}(a)$   
     A7 :  $\tilde{c}_n = GFmin(W_n(a))$   
     **if**  $\tilde{\mathbf{c}} \times \mathbf{H}^T = 0$  **then**  
         SKIP  
     **end if**  
**end for**  
**Output:**  $\tilde{\mathbf{c}}$

---



---

**Algorithm 2** Extended Min-Sum algorithm

---

**Input:**  $L_n(a) = \log [P(c_n = 0 | \mathbf{y}_n)/P(c_n = a | \mathbf{y}_n)]$ ,  $W_{m,n}(a) = L_n(a)$   
 Iterative process  
**for**  $j = 1 \rightarrow It_{max}$  **do**  
     Check node processing  
     A1 :  $A_{m,n}(a) := \{a_{n'} \in \text{conf}(n_m, n_c) \mid h_{m,n}a_n + \sum_{n' \in \mathcal{N}(m) \setminus n} h_{m,n'}a_{n'} = 0\}$   
     A2 :  $R_{m,n}(a) = \min_{a_{n'} \in A_{m,n}(a), n' \in \mathcal{N}(m) \setminus n} (\sum_{n' \in \mathcal{N}(m) \setminus n} (W_{m,n'}(a_{n'})))$   
     Variable node processing  
     A3 :  $W'_{m,n}(a) = L_n(a) + \sum_{m' \in \mathcal{M}(n) \setminus m} R_{m',n}(a)$   
     A4 :  $W'_{m,n}(a) = W_{m',n}(0)$   
     A5 :  $W_{m,n}(a) = W'_{m,n}(a) - W'_{m,n}$   
     Tentatively decoding  
     A6 :  $W_n(a) = L_n(a) + \sum_{m' \in \mathcal{M}(n)} R_{m',n}(a)$   
     A7 :  $\tilde{c}_n = GFmin(W_n(a))$   
     **if**  $\tilde{\mathbf{c}} \times \mathbf{H}^T = 0$  **then**  
         SKIP  
     **end if**  
**end for**  
**Output:**  $\tilde{\mathbf{c}}$

---

---

**Algorithm 3** Min-max algorithm

---

**Input:**  $L_n(a) = \log [P(c_n = z_n | \mathbf{y}_n) / P(c_n = a | \mathbf{y}_n)]$ ,  $W_{m,n}(a) = L_n(a)$

Iterative process

**for**  $j = 1 \rightarrow It_{max}$  **do**

  Check node processing

  A1 :  $A_{m,n}(a) := \{a_{n'} | h_{m,n}a_n + \sum_{n' \in \mathcal{N}(m) \setminus n} h_{m,n'}a_{n'} = 0\}$

  A2 :  $R_{m,n}(a) = \min_{a_{n'} \in A_{m,n}(a), n' \in \mathcal{N}(m) \setminus n} (\max_{n' \in \mathcal{N}(m) \setminus n} (W_{m,n'}(a_{n'})))$

  Variable node processing

  A3 :  $W'_{m,n}(a) = L_n(a) + \sum_{m' \in \mathcal{M}(n) \setminus m} R_{m',n}(a)$

  A4 :  $W'_{m,n} = \min_{a \in GF(q)} W'_{m,n}(a)$

  A5 :  $W_{m,n}(a) = W'_{m,n}(a) - W'_{m,n}$

  Tentatively decoding

  A6 :  $W_n(a) = L_n(a) + \sum_{m' \in \mathcal{M}(n)} R_{m',n}(a)$

  A7 :  $\tilde{c}_n = GFmin(W_n(a))$

**if**  $\tilde{\mathbf{c}} \times \mathbf{H}^T = 0$  **then**

    SKIP

**end if**

**end for**

**Output:**  $\tilde{\mathbf{c}}$

---

A1 is performed. Just the  $n_m < q$  more reliable symbols are taken into account and only combinations (configurations) with  $n_c$  symbols different from the hard decision  $z_n$  are analyzed. The alternative to the method based on reduction of the number of configurations is described at step A2 of Min-max algorithm. This requires calculating all the possible combinations of  $GF(q)$  symbols at a CNU of degree  $d_c$ .

Regardless of the method applied at step A1, step A2 of EMS algorithm computes the sum of  $W_{m,n'}(a_{n'})$ , approximated by the maximum at Min-max algorithm. This sum can be understood as the computation of the reliability of the selected symbols  $a_{n'}$ , when  $a_n = a$ . Approximating the sum of the  $d_c$  reliability values by the maximum of the  $d_c$  values, leads to a small performance loss but avoids the data growing in the CNU. From all the possible paths (or configurations) that satisfy the parity check equations, the minimum (more reliable path),  $R_{m,n}(a)$ , is selected for each value of  $a$ . Hence, if EMS method is applied  $\mathbf{R}_{m,n}$  is a  $n_m$ -length vector else, if the full number of comparisons are performed it is a  $q$ -length vector. It is important to remark that step A2 of EMS algorithm is in turn a simplification derived from the logarithmic formulation of QSPA (Algorithm 1), where the logarithm of a sum of exponentials is replaced by the sum of the logarithm of each exponential (the sum of exponents). This simplification introduces differences in the values of the exchanged messages that must be corrected applying offset or scaling. Even after this correction, EMS introduces some performance loss compared to QSPA. However, complexity of QSPA check node does not justify the difference

of coding gain. Comparing QSPA and EMS, both require the computation of the additions involved in  $\sum_{n' \in \mathcal{N}(m) \setminus n} W_{m,n'}(a_{n'})$ , but the first one needs to process: i) the exponential operation, which is memory hungry (and hence area and power consuming); ii) the addition of the exponential values, which increases the number of bits required by the finite precision analysis (increasing storage resources and routing), unlike the minimum operation performed in EMS; and iii) the logarithm of the sum, which has similar problems to the exponential operation, due to the use of look up tables for implementation. The previous differences make QSPA implementations very inefficient. Compared to EMS or Min-Sum architectures, the best QSPA design is about seven times less efficient, with an improvement of less than 0.2dB in coding gain [15]. Although the rest of the steps are similar for both QSPA and EMS algorithms, QSPA processing is 40% costlier because messages exchanged between check node and variable node and vice versa, require 40% more bits in the finite precision analysis. Even if some of the simplification and optimization methods that will be described in the following pages can be applied also to QSPA, we will just focus on EMS and Min-max because the check node is too complex for efficient and high-throughput implementations.

Step A3 in QSPA, EMS and Min-max algorithms adds to the channel information ( $L_n(a)$ ) the extrinsic information computed with the updated check nodes connected to this variable node ( $\sum_{m' \in \mathcal{M}(n) \setminus m} R_{m',n}(a)$ ). Step A4 and A5 normalizes  $W_{m,n}(a)$  to the symbol 0 or  $z_n$ , depending on the initialization of the algorithm, to ensure the computational stability. As it has been discussed before, in step A5 of EMS and Min-max algorithm, offset or scaling can be applied to improve the performance and compensate the differences of processing with QSPA. The offset or scaling values can be obtained applying density evolution or via simulation [12]. Note that  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  is also a  $n_m$ -element or  $q$ -element vector (depending on the number of configuration sets), for each pair of  $m$  and  $n$ .

The iterative process ends when reaches the maximum number of iterations or when all the parity check equations are satisfied, if an early stopping criterion is set. Step A6 of the tentative decoding performs a similar equation to step A3 but it takes into account all the check nodes  $\mathcal{M}(n)$  connected to the variable node  $n$ . Step A7 looks for the GF symbol associated to the minimum value of  $W_n(a)$  (most reliable symbol).

From the description of QSPA, EMS and Min-max algorithms two main bottlenecks can be observed: i) the computational complexity of the check node (steps A1 and A2); and ii) the exchange of vectors (of  $n_m$  or  $q$  elements) between CNU and VNU ( $\mathbf{R}_{\mathbf{m},\mathbf{n}}$  and  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$ ).

To avoid an inefficient way of solving steps A1 and A2 several algorithms were proposed, such as the forward-backward algorithm (included at [13]), the bubble check algorithm [16] and some trellis based algorithms: [17] (which introduces an early error floor degradation according to [18]), [19] and [20]. These solutions

have made possible to apply NB-LDPC decoders to a wide range of scenarios such as wireless communications [21], which was not possible with the complexity of the original QSPA. In the following section, we will explain with detail two of these proposals to solve the CNU: forward-backward from [13] and trellis-EMS (T-EMS) [19]. However, we can advance that even with the reduction of complexity of these techniques, the derived latency and throughput are far from high-speed constraints of some communication systems.

On the other hand, the simplification proposed at [12] and [14] selecting just  $n_m$  most reliable symbols is efficient for high order fields  $\text{GF}(64)$  and above. For high order fields,  $n_m$  can be very small compared to  $q$  ( $n_m \ll q$ ) without performance degradation and with a real saving of area. However, for fields under  $\text{GF}(64)$ , if  $n_m \ll q$ , there is a performance degradation; and if  $n_m < q$  is chosen there is no area saving due to the need of storing  $\mathbf{R}_{\mathbf{m},\mathbf{n}}$  and  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  with their associated GF symbols. Even in cases with area saving, the exchange of vectors between CNU and VNU has associated a routing congestion problem. This problem was already reported in binary LDPC [22], where the number of connections required to share information between the CNU and the VNU units was higher enough to reduce the maximum frequency achievable of the derived architectures, due to the density of the required wiring. In the binary case the output of CNU and VNU is a scalar, so we can expect that the problem in NB-LDPC decoding will be multiplied at least by  $n_m$ . For this reason different algorithms need to be introduced, even if they include some performance degradation. These algorithms will be explained at Section 1.5 of the manuscript.

## 1.4 Forward-backward and T-EMS algorithms

### 1.4.1 Forward-backward algorithm

Forward-backward is a recursive algorithm that computes in an efficient way the CNU function when the length of the incoming messages from the VNU is of  $q$ -elements (*i.e.* step A2 of Algorithm 3). As can be seen at Algorithm 4, two matrices of size  $q \times (d_c - 1)$ ,  $\mathbf{F}$  and  $\mathbf{B}$ , are needed to store the forward and backward metric results. The first element of column of  $\mathbf{F}$ ,  $F_0(a)$  (with  $a \in \text{GF}(q)$ ) is initialized with the information associated to the first VNU connected to the processed CNU  $W_{m,n_0}(a)$ . For a matrix  $\mathbf{H}$  with non-zero coefficients different from one, the values of  $W_{m,n_0}(a)$  stored in  $F_0(a)$  are shifted by  $h_{m,n_0}^{-1}$  ( $F_0(a) = W_{m,n_0}(h_{m,n_0}^{-1}a)$ ) to perform the product of the parity check equation. A similar reasoning is followed for the initialization of the vector  $\mathbf{B}$  with the only difference that in this case the last element,  $B_{d_c-1}(a)$ , is the one initialized. The recursive process for the forward metrics of a node  $i$  consists in comparing the column computed in the previous iteration,  $F_{i-1}(a')$ , with the incoming message from the

**Algorithm 4** Forward-backward algorithm**Input:**  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  of check node  $m$ 

Forward-backward metrics

$$F_0(a) = W_{m,n_0}(h_{m,n_0}^{-1}a)$$

$$B_{d_c-1}(a) = W_{m,n_{d_c-1}}(h_{m,n_{d_c-1}}^{-1}a)$$

**for**  $i = 1 \rightarrow d_c - 1$  **do**

$$F_i(a) = \min_{a'+h_{m,n_i}a''=a}(\max(F_{i-1}(a'), W_{m,n_i}(a'')), a', a'' \in \text{GF}(q))$$

$$B_{d_c-1-i}(a) = \min_{a'+h_{m,n_i}a''=a}(\max(B_{d_c-1-i+1}(a'), W_{m,n_{d_c-1-i}}(a'')), a', a'' \in \text{GF}(q))$$

**end for**

Merge metrics

$$M_0(a) = B_1(a)$$

$$M_{d_c-1}(a) = F_{d_c-2}(a)$$

**for**  $d = 1 \rightarrow d_c - 2$  **do**

$$M_d(a) = \min_{a'+a''=-h_{m,n_d}a}(\max(F_{d-1}(a'), B_{d+1}(a'')), a', a'' \in \text{GF}(q))$$

**end for****Output:**  $\mathbf{R}_{\mathbf{m},\mathbf{n}} = \mathbf{M}$ 

$i$  VNU,  $W_{m,n_i}(a'')$ ). The relation between  $a'$  and  $a''$  has to be  $a' + h_{m,n_i}a'' = a$  to ensure that the parity check equation will be satisfied. To process all the possible combinations of  $a'$  and  $a''$  that accomplish  $a' + h_{m,n_i}a'' = a$ ,  $q^2$  comparisons are required. Note that  $a'$  is not multiplied by  $h_{m,n_{i-1}}$  because the product was performed in the previous iteration. It is very important to remark that we cannot process efficiently the  $i$  iterations in parallel, because it is a recursive algorithm, we need to compute first  $F_{i-1}(a)$  to get  $F_i(a)$ , so at least  $d_c - 1$  iterations are required to compute the forward step. The same schedule, but backwards, is followed with  $\mathbf{B}$ . As  $\mathbf{F}$  and  $\mathbf{B}$  have independent outputs, they can be calculated in parallel. The combination of both  $\mathbf{F}$  and  $\mathbf{B}$  results with the merge metrics are stored in a matrix of size  $q \times d_c$ ,  $\mathbf{M}$ . The merge metrics are the same as the ones followed by the forward-backward metrics, but having as input the matrices  $\mathbf{F}$  and  $\mathbf{B}$  instead of the information from the VNU.  $\mathbf{M}_d$  is processed by means of combining  $\mathbf{F}_{d-1}$  and  $\mathbf{B}_{d+1}$ .  $\mathbf{F}_d$  and  $\mathbf{B}_d$  are not involved in the computation of  $\mathbf{M}_d$  because only the extrinsic information of the node  $d$  is required. To compute the merge metrics  $d_c - 2$  iterations are required. Hence, to compute a CNU of degree  $d_c$ ,  $2d_c - 3$  iterations are required, which is equivalent to a latency of  $2d_c - 3$  clock cycles if forward and backward metrics are performed in parallel. If merge metrics are not calculated after the full  $\mathbf{F}$  and  $\mathbf{B}$  are available, but starts when the first pair of  $F_{i-1}(a')$  and  $B_{i+1}(a'')$  to compute  $M_d(a)$  are available, the number of iterations can be reduced to  $d_c - 1$ . So, to set latency to  $d_c - 1$  clock cycles four parallel processors are required one for each forward and backward metrics and two processors for the merge metrics. It is not possible to solve a CNU with less number of iterations than this.

Next we include two examples that illustrate forward-backward algorithm, the first one shows the schedule of the forward, backward and merge metrics and the second one details the processing of one forward iteration.

**Example 1.4.1** *Let us assume a check node  $m$  with  $d_c = 7$ . Next the computation per iteration is included, considering the higher degree of parallelism: one forward processor  $F_x$ , one backward processor  $B_x$  and two merge processors  $M_x$  and  $M'_x$ .*

**Initialization:**

$$F_0(a) = W_{m,n_0}(h_{m,n_0}^{-1}a)$$

$$B_6(a) = W_{m,n_6}(h_{m,n_6}^{-1}a)$$

$$M_x = \text{Idle}$$

$$M'_x = \text{Idle}$$

**1st iteration:**

$$F_1(a) = \min_{a'+h_{m,n_1}a''=a}(\max(F_0(a'), W_{m,n_1}(a'')))$$

$$B_5(a) = \min_{a'+h_{m,n_5}a''=a}(\max(B_6(a'), W_{m,n_5}(a'')))$$

$$M_x = \text{Idle}$$

$$M'_x = \text{Idle}$$

**2nd iteration:**

$$F_2(a) = \min_{a'+h_{m,n_2}a''=a}(\max(F_1(a'), W_{m,n_2}(a'')))$$

$$B_4(a) = \min_{a'+h_{m,n_4}a''=a}(\max(B_5(a'), W_{m,n_4}(a'')))$$

$$M_x = \text{Idle}$$

$$M'_x = \text{Idle}$$

**3rd iteration:**

$$F_3(a) = \min_{a'+h_{m,n_3}a''=a}(\max(F_2(a'), W_{m,n_3}(a'')))$$

$$B_3(a) = \min_{a'+h_{m,n_3}a''=a}(\max(B_4(a'), W_{m,n_3}(a'')))$$

$$M_3(a) = \min_{a'+a''=-h_{m,n_3}a}(\max(F_2(a'), B_4(a'')))$$

$$M'_x = \text{Idle}$$

**4th iteration:**

$$F_4(a) = \min_{a'+h_{m,n_4}a''=a}(\max(F_3(a'), W_{m,n_4}(a'')))$$

$$B_2(a) = \min_{a'+h_{m,n_2}a''=a}(\max(B_3(a'), W_{m,n_2}(a'')))$$

$$M_2(a) = \min_{a'+a''=-h_{m,n_2}a}(\max(F_3(a'), B_3(a'')))$$

$$M'_4(a) = \min_{a'+a''=-h_{m,n_4}a}(\max(F_3(a'), B_5(a'')))$$

**5th iteration:**

$$F_5(a) = \min_{a'+h_{m,n_5}a''=a}(\max(F_4(a'), W_{m,n_5}(a'')))$$

$$B_1(a) = \min_{a'+h_{m,n_1}a''=a}(\max(B_2(a'), W_{m,n_1}(a'')))$$

$$M_1(a) = \min_{a'+a''=-h_{m,n_1}a}(\max(F_4(a'), B_2(a'')))$$

$$M'_5(a) = \min_{a'+a''=-h_{m,n_5}a}(\max(F_4(a'), B_6(a'')))$$

**6th iteration:**

$$F_x(a) = \text{Idle}$$

$$B_x(a) = \text{Idle}$$

$$M_0(a) = B_1(a)$$

$$M'_6(a) = F_5(a)$$

This example shows us that due to the dependency of the data between the forward-backward metrics and the merge it is not possible to solve the CNU with less than  $d_c - 1$  iterations (clock cycles). Although the merge processors are unused during the first  $\lceil d_c/2 \rceil$  iterations, these two processors are required to reach the lowest latency.

**Example 1.4.2** For simplicity, let us assume a parity check matrix  $\mathbf{H}$  with non-zero coefficients equal to  $1 \in GF(4)$ . The decoding algorithm is Min-max and  $\mathbf{F}_{i-1} = [18 \ 15 \ 0 \ 6]$  and  $\mathbf{W}_{\mathbf{m}, \mathbf{n}_i} = [0 \ 32 \ 13 \ 19]$ . The elements of vectors  $\mathbf{F}_{i-1}$  and  $\mathbf{W}_{\mathbf{m}, \mathbf{n}_i}$  are associated to the symbols  $0, 1, \alpha$  and  $\alpha^2$ , respectively, so the hard-decision symbol of  $\mathbf{F}_{i-1}$  is  $\alpha$  and the hard decision symbol of  $\mathbf{W}_{\mathbf{m}, \mathbf{n}_i}$  is  $0$  (as the minimum value correspond to the most reliable symbol). The operation addition over  $GF(4)$  is defined as:

$\oplus$	0	1	$\alpha$	$\alpha^2$
0	0	1	$\alpha$	$\alpha^2$
1	1	0	$\alpha^2$	$\alpha$
$\alpha$	$\alpha$	$\alpha^2$	0	1
$\alpha^2$	$\alpha^2$	$\alpha$	1	0

To compute  $F_i(a) = \min_{a'+h_{m,n_i}a''=a}(\max(F_{i-1}(a'), W_{m,n_i}(a'')))$ , all the  $\mathbf{F}_{i-1}$  and  $\mathbf{W}_{\mathbf{m}, \mathbf{n}_i}$  combinations of  $a'+h_{m,n_i}a''=a$  for  $a = \{0, 1, \alpha, \alpha^2\}$  should be calculated. As we assume that  $h_{m,n_i} = 1$ , the condition is simplified to  $a' + a'' = a$ . The maximums are computed as follows:

For  $a = 0$ :

- $\max(F_{i-1}(a' = 0), W_{m,n_i}(a'' = 0)) = \max(18, 0) = 18$
- $\max(F_{i-1}(a' = 1), W_{m,n_i}(a'' = 1)) = \max(15, 32) = 32$
- $\max(F_{i-1}(a' = \alpha), W_{m,n_i}(a'' = \alpha)) = \max(0, 13) = 13$
- $\max(F_{i-1}(a' = \alpha^2), W_{m,n_i}(a'' = \alpha^2)) = \max(6, 19) = 19$

For  $a = 1$ :

- $\max(F_{i-1}(a' = 0), W_{m,n_i}(a'' = 1)) = \max(18, 32) = 32$
- $\max(F_{i-1}(a' = 1), W_{m,n_i}(a'' = 0)) = \max(15, 0) = 15$
- $\max(F_{i-1}(a' = \alpha), W_{m,n_i}(a'' = \alpha^2)) = \max(0, 19) = 19$
- $\max(F_{i-1}(a' = \alpha^2), W_{m,n_i}(a'' = \alpha)) = \max(6, 13) = 13$

For  $a = \alpha$ :



- $\max(F_{i-1}(a' = 0), W_{m,n_i}(a'' = \alpha)) = \max(18, 13) = 18$
- $\max(F_{i-1}(a' = 1), W_{m,n_i}(a'' = \alpha^2)) = \max(15, 19) = 19$
- $\max(F_{i-1}(a' = \alpha), W_{m,n_i}(a'' = 0)) = \max(0, 0) = 0$
- $\max(F_{i-1}(a' = \alpha^2), W_{m,n_i}(a'' = 1)) = \max(6, 32) = 32$

For  $a = \alpha^2$ :

- $\max(F_{i-1}(a' = 0), W_{m,n_i}(a'' = \alpha^2)) = \max(18, 19) = 19$
- $\max(F_{i-1}(a' = 1), W_{m,n_i}(a'' = \alpha)) = \max(15, 13) = 15$
- $\max(F_{i-1}(a' = \alpha), W_{m,n_i}(a'' = 1)) = \max(0, 32) = 32$
- $\max(F_{i-1}(a' = \alpha^2), W_{m,n_i}(a'' = 0)) = \max(6, 0) = 6$

To perform the maximum of all the combinations,  $q^2$  comparisons are required. So, if we want to perform each forward (or backward/merge) step in one clock cycle,  $q^2$  comparators per processor are necessary to get the maximums. Finally, the minimum for each  $a \in GF(4)$  is calculated:

$$\text{For } a = 0: F_i(a = 0) = \min(18, 32, 13, 19) = 13$$

$$\text{For } a = 1: F_i(a = 1) = \min(32, 15, 19, 13) = 13$$

$$\text{For } a = \alpha: F_i(a = \alpha) = \min(18, 19, 0, 32) = 0$$

$$\text{For } a = \alpha^2: F_i(a = \alpha^2) = \min(19, 15, 32, 6) = 6$$

The result is coherent, because the addition of the hard decision of both inputs,  $\alpha + 0 = \alpha$ , and the most reliable symbol at the output of the forward step,  $F_i(a = \alpha) = 0$ , leads to the same symbol value.

To calculate the minimums  $q \times (q - 1)$  comparisons are needed. Hence, if we want to achieve the higher degree of parallelism  $q^3 \times (q - 1)$  comparisons are required to perform one complete iteration of the forward (or backward/merge) metrics in one clock cycle. According to Example 1.4.1, four processors are necessary to complete all the algorithm in  $d_c - 1$  clock cycles, so a complete CNU unit will need  $4 \times q^3 \times (q - 1)$  comparators to get the lowest latency (and the highest speed).

As we can see from the previous examples, latency grows with  $d_c$  and the number of comparators depends on the size of the field,  $q$ . Systems such as optical communication ones require high-rate codes, which have a large  $d_c$  value. In consequence of this, latency will grow and speed will be reduced for this kind of codes, which

is totally the opposite to what it is looked for in this high-speed systems. In addition, area resources and routing (wiring between comparators) will grow with the field if a fully parallel implementation is done. In case that a fully parallel implementation is avoided, serial implementations will make latency also dependent on  $q$ , reducing speed even more. In conclusion, for applications such as optical communications with high order fields and high-rate, forward-backward algorithm is not efficient enough to get a throughput as high as is required.

To reduce the complexity of the forward-backward algorithm, a simplified version where all the coefficients of the matrix  $\mathbf{H}$  are equal to  $1 \in \text{GF}(q)$  (as in Example 1.4.2) can be applied if the improved version of the Min-max algorithm in [23] is used. The improved Min-max algorithm performs the products by the parity check matrix coefficients outside of the forward-backward algorithm in order to avoid GF products inside the CNU. This modification of the algorithm does not degrade its performance, but has a real saving of area in the derived NB-LDPC decoder architectures. With this proposal GF multipliers are not required because the products can be implemented changing the writing address of the memories. Unfortunately, this improvement does not change the previously mentioned bottleneck in latency and speed of NB-LDPC codes. The improved version of the algorithm is included next.

---

**Algorithm 5** Improved Min-max algorithm

---

**Input:**  $L_n(a) = \log [P(c_n = z_n | \mathbf{y}_n) / P(c_n = a | \mathbf{y}_n)]$ ,  $W_{m,n}(a) = L_n(h_{m,n}a)$

Iterative process

**for**  $j = 1 \rightarrow It_{max}$  **do**

  Check node processing

  A1 :  $A_{m,n}(a) := \{a_{n'} \mid a_n + \sum_{n' \in \mathcal{N}(m) \setminus n} a_{n'} = 0\}$

  A2 :  $R_{m,n}(a) = \min_{a_{n'} \in A_{m,n}(a), n' \in \mathcal{N}(m) \setminus n} (\max_{n' \in \mathcal{N}(m) \setminus n} (W_{m,n'}(a_{n'})))$

  Variable node processing

  A3 :  $W'_{m,n}(h_{m,n}a) = L_n(a) + \sum_{m' \in \mathcal{M}(n) \setminus m} R_{m',n}(h_{m,n}a)$

  A4 :  $W'_{m,n} = \min_{a \in \text{GF}(q)} W'_{m,n}(a)$

  A5 :  $W_{m,n}(a) = W'_{m,n}(a) - W'_{m,n}$

  Tentatively decoding

  A6 :  $W_n(a) = L_n(a) + \sum_{m' \in \mathcal{M}(n)} R_{m',n}(h_{m,n}a)$

  A7 :  $\tilde{c}_n = \text{GFmin}(W_n(a))$

**if**  $\tilde{\mathbf{c}} \times \mathbf{H}^T = \mathbf{0}$  **then**

    SKIP

**end if**

**end for**

**Output:**  $\tilde{\mathbf{c}}$

---

Min-max and forward-backward algorithms can be also applied when the input vectors have  $n_m$  elements, but some trellis based algorithms ([17], [20]) became more efficient in these cases. However, these algorithms still have a latency bottle-

neck of  $d_c - 1$  clock cycles, even working with  $n_m$ -length vectors. For this reason we do not analyze them with detail in this manuscript. To the best knowledge of the author, the only algorithm that can achieve a fully parallel computation for the CNU update of EMS algorithm <sup>3</sup> is the one proposed at [19] and [25]. A detailed explanation of this algorithm is included in the following subsection.

### 1.4.2 T-EMS algorithm

In this subsection a trellis based algorithm to solve the CNU for EMS, T-EMS ([19], [25]), is described. The algorithm takes advantage of the GF product elimination at the CNU introduced at [23] and explained in the previous subsection. Unlike other trellis proposals like [20], this algorithm does not work with the  $n_m$  most reliable symbols in  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$ , but with full  $q$ -element vectors. The fact of working with all the elements in  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  does not increase complexity because T-EMS can reduce the number of configuration sets without searching the  $n_m$  most reliable symbols. In addition, this algorithm allows us a higher degree of parallelism than other proposals due to its delta-domain based trellis structure which avoids recursive computations.

---

#### Algorithm 6 Trellis EMS algorithm

---

**Input:**  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  and  $z_n$  of check node  $m$

Delta-domain

**for**  $j = 1 \rightarrow d_c$  **do**

A1:  $\Delta W'_{m,n_j}(\eta_j = a + z_{n_j}) = W_{m,n_j}(z_{n_j}) - W_{m,n_j}(a)$ ,  $a \in \text{GF}(q)$

**end for**

A2:  $\beta = \sum_{j=1}^{d_c} z_{n_j} \in \text{GF}(q)$

Syndrome reliability values from configuration sets

A3:  $\Delta W(a) = \min_{\eta'_j(a) \in \text{conf}(n_r, n_c)} \sum_{j=1}^{d_c} \Delta W'_{m,n_j}(\eta'_j(a))$ ,  $a \in \text{GF}(q)$

Extrinsic output messages and normal domain

**for**  $j = 1 \rightarrow d_c$  **do**

A4:  $\Delta R_{m,n_j}(a + \eta'_j(a)) = \Delta W(a) - \Delta W'_{m,n_j}(\eta'_j(a))$ ,  $a \in \text{GF}(q)$

A5:  $R_{m,n_j}(a + \beta + z_{n_j}) = -\Delta R_{m,n_j}(a)$ ,  $a \in \text{GF}(q)$

**end for**

**Output:**  $\mathbf{R}_{\mathbf{m},\mathbf{n}}$

---

Algorithm 6 transforms the VNU information into a delta-domain. In step A1, the difference between the reliability of the hard decision symbol  $W_{m,n_j}(z_{n_j})$  and the reliability of each symbol  $a \in \text{GF}(q)$ ,  $W_{m,n_j}(a)$ , is computed. This difference is associated to the symbol  $\eta_j = a + z_{n_j}$ , in other words, all the reliability values inside  $\Delta W'_{m,n_j}$  are shifted by the hard-decision  $z_{n_j}$ . Note that  $\eta_j = 0$  when  $a = z_{n_j}$ . Moreover, the syndrome equation of all the hard-decision symbols are calculated and stored at  $\beta$ , step A2. Steps A1 and A2 conform the delta-domain

<sup>3</sup>A similar method was proposed for Min-max algorithm, named relaxed Min-max [24].

transformation. As can be seen there is not dependency between the  $d_c$  values of  $j$  at step A1, so they can be computed in parallel. In addition, steps A1 and A2 are also independent.

Once  $\Delta W'_{m,n_j}(\eta_j)$  is calculated for all the  $j$  and  $\eta_j$  values, the configuration sets can be created. For a same value of  $\eta_j$  the  $n_r$  minimum values of  $\Delta W'_{m,n_j}(\eta_j)$  between the  $d_c$  possible ones are added to the configuration set. The minimum values are selected because they are the ones with less difference in terms of reliability with the hard-decision (remember step A1 of the delta-domain). The configuration sets include  $n_r \times d_c$  symbols. Next all the possible paths (combination of symbols) with up to  $n_c$  deviations from  $\eta = 0$  path (hard-decision path) are calculated, taken into account that for the same path there cannot be more than one deviation for the same value of  $j$ . These paths form the configuration sets denoted by  $conf(n_r, n_c)$ . The deviated paths represent other possible paths with high reliability, as its reliability only changes at  $n_c$  locations compared to the hard-decision path ( $\eta = 0$ ). The addition of all the  $\eta'_j(a) \in conf(n_r, n_c)$  symbols has to accomplish the condition of being equal to  $a$ . This means that for the syndrome  $a$ , in the delta domain, the most reliable paths will be formed by  $\eta'_j(a) \in conf(n_r, n_c)$  symbols. The search of the  $n_r$  minimum values are performed row-wise as they are comparing  $d_c$  different values of  $j$  with the same value of  $\eta_j$ . The independence between different values of  $\eta_j$  allow us to calculate the  $n_r \times d_c$  symbols in parallel.

Step A3 calculates the reliability of all the paths at  $conf(n_r, n_c)$  by adding the values  $\Delta W'_{m,n_j}(\eta'_j(a))$  involved in the path ( $\eta'_j(a) \in conf(n_r, n_c)$ ). From all the possible paths, selects the most reliable for each syndrome  $a$  (minimum operation). The result of step A3 is a column of syndrome reliability values,  $\Delta W(a)$ . As there is no dependence between results of the different paths, all this step can be computed in parallel.

Step A4 obtains the extrinsic information of each element that conforms the path, subtracting to the syndrome reliability value  $\Delta W(a)$  of the path the intrinsic reliability information  $\Delta W'_{m,n_j}(\eta'_j)$ . In a similar way, the associated symbol is computed, subtracting (which is the same as adding in GF) to the syndrome symbol, the symbol that conforms the path,  $a + \eta'_j(a)$ . When one value is not filled, the first minimum associated to the symbol is taken to fill it. If the first minimum belongs to one of the selected paths, second maximum of the row is taken. The last step of the algorithm, A5, makes the conversion to the normal domain, correcting the shift by  $z_n$  performed at step A1 and adding the value of hard-decision syndrome  $\beta$ , because the path of the hard-decision symbols ( $\eta = 0$ ) was the one taken as reference. Steps A4 and A5 can be updated in parallel because non-recursive functions are applied.

With the next example we will try to illustrate how this algorithm works:

**Example 1.4.3** Let us assume a check node  $m$  with  $d_c = 4$ ,  $n_r = 2$  and  $n_c = 2$ . The input information is the following:  $\mathbf{W}_{m,n_1} = [0 \ 45 \ -25 \ 55]$ ,  $\mathbf{W}_{m,n_2} = [0 \ -10 \ 30 \ -45]$ ,  $\mathbf{W}_{m,n_3} = [0 \ -25 \ 40 \ -45]$ ,  $\mathbf{W}_{m,n_4} = [0 \ -60 \ -5 \ -85]$ . Elements in the vector  $\mathbf{W}_{m,n_j}$  are sorted having as associated symbols  $0$ ,  $1$ ,  $\alpha$  and  $\alpha^2$ . The associated hard-decision symbols are the ones with the highest value of  $\mathbf{W}_{m,n_j}$ , so  $z_{n_1} = \alpha^2$ ,  $z_{n_2} = \alpha$ ,  $z_{n_3} = \alpha$  and  $z_{n_4} = 0$ .

The first step is the delta-domain:

- $\Delta W'_{m,n_1}(a + z_{n_1} = 0 + \alpha^2 = \alpha^2) = W_{m,n_1}(\alpha^2) - W_{m,n_1}(0) = 55 - 0 = 55$
- $\Delta W'_{m,n_1}(1 + \alpha^2 = \alpha) = W_{m,n_1}(\alpha^2) - W_{m,n_1}(1) = 55 - 45 = 10$
- $\Delta W'_{m,n_1}(\alpha + \alpha^2 = 1) = W_{m,n_1}(\alpha^2) - W_{m,n_1}(\alpha) = 55 - (-25) = 80$
- $\Delta W'_{m,n_1}(\alpha^2 + \alpha^2 = 0) = W_{m,n_1}(\alpha^2) - W_{m,n_1}(\alpha^2) = 55 - 55 = 0$

The sorted vector according to the position of the associated  $(0, 1, \alpha$  and  $\alpha^2)$  is:  $\Delta W'_{m,n_1} = [0 \ 80 \ 10 \ 55]$

- $\Delta W'_{m,n_2}(a + z_{n_2} = 0 + \alpha = \alpha) = W_{m,n_2}(\alpha) - W_{m,n_2}(0) = 30 - 0 = 30$
- $\Delta W'_{m,n_2}(1 + \alpha = \alpha^2) = W_{m,n_2}(\alpha) - W_{m,n_2}(1) = 30 - (-10) = 40$
- $\Delta W'_{m,n_2}(\alpha + \alpha = 0) = W_{m,n_2}(\alpha) - W_{m,n_2}(\alpha) = 30 - 30 = 0$
- $\Delta W'_{m,n_2}(\alpha^2 + \alpha = 1) = W_{m,n_2}(\alpha) - W_{m,n_2}(\alpha^2) = 30 - (-45) = 75$

Sorted vector:  $\Delta W'_{m,n_2} = [0 \ 75 \ 30 \ 40]$

- $\Delta W'_{m,n_3}(a + z_{n_3} = 0 + \alpha = \alpha) = W_{m,n_3}(\alpha) - W_{m,n_3}(0) = 40 - 0 = 40$
- $\Delta W'_{m,n_3}(1 + \alpha = \alpha^2) = W_{m,n_3}(\alpha) - W_{m,n_3}(1) = 40 - (-25) = 65$
- $\Delta W'_{m,n_3}(\alpha + \alpha = 0) = W_{m,n_3}(\alpha) - W_{m,n_3}(\alpha) = 40 - 40 = 0$
- $\Delta W'_{m,n_3}(\alpha^2 + \alpha = 1) = W_{m,n_3}(\alpha) - W_{m,n_3}(\alpha^2) = 40 - (-45) = 85$

Sorted vector:  $\Delta W'_{m,n_3} = [0 \ 85 \ 40 \ 65]$

- $\Delta W'_{m,n_4}(a + z_{n_4} = 0 + 0 = 0) = W_{m,n_4}(0) - W_{m,n_4}(0) = 0 - 0 = 0$
- $\Delta W'_{m,n_4}(1 + 0 = 1) = W_{m,n_4}(0) - W_{m,n_4}(1) = 0 - (-60) = 60$
- $\Delta W'_{m,n_4}(\alpha + 0 = \alpha) = W_{m,n_4}(0) - W_{m,n_4}(\alpha) = 0 - (-5) = 5$
- $\Delta W'_{m,n_4}(\alpha^2 + 0 = \alpha^2) = W_{m,n_4}(0) - W_{m,n_4}(\alpha^2) = 0 - (-85) = 85$

Sorted vector:  $\Delta W'_{m,n_4} = [0 \ 60 \ 5 \ 85]$

Writing  $\Delta \mathbf{W}'_{m,n}$  in its trellis representation:

$\eta/j$	1	2	3	4
0	0	0	0	0
1	80	75	85	60
$\alpha$	10	30	40	5
$\alpha^2$	55	40	65	85

Applying step A2 the syndrome of the hard-decision symbols is calculated as:

$$\beta = \sum_{j=1}^4 z_{n_j} = \alpha^2 + \alpha + \alpha + 0 = \alpha^2.$$

Next the configuration sets are built. The first step is searching  $n_r = 2$  minimums for the same  $\eta_j \neq 0$  value. Minimums are indicated in bold type.

$\eta/j$	1	2	3	4
0	0	0	0	0
1	80	<b>75</b>	85	<b>60</b>
$\alpha$	<b>10</b>	30	40	<b>5</b>
$\alpha^2$	55	<b>40</b>	<b>65</b>	85

The following step is computing all the possible paths with  $n_c = 2$  deviations, without taking more than one value in the same column (same  $j$  value).  $\text{conf}(n_r, n_c)$  for this example is as follows:

Paths for  $a = 1$  with one deviation:

- $\Delta W'_{m,n_2}(1) = 75$
- $\Delta W'_{m,n_4}(1) = 60$

Paths for  $a = 1$  with two deviations:

- $\Delta W'_{m,n_1}(\alpha) + \Delta W'_{m,n_2}(\alpha^2) = 10 + 40 = 50$
- $\Delta W'_{m,n_1}(\alpha) + \Delta W'_{m,n_3}(\alpha^2) = 10 + 65 = 75$
- $\Delta W'_{m,n_4}(\alpha) + \Delta W'_{m,n_2}(\alpha^2) = 5 + 40 = 45$
- $\Delta W'_{m,n_4}(\alpha) + \Delta W'_{m,n_3}(\alpha^2) = 5 + 65 = 70$

Paths for  $a = \alpha$  with one deviation:

- $\Delta W'_{m,n_1}(\alpha) = 10$

- $\Delta W'_{m,n_4}(\alpha) = 5$

Paths for  $a = \alpha$  with two deviations:

- $\Delta W'_{m,n_2}(1) + \Delta W'_{m,n_3}(\alpha^2) = 75 + 65 = 140$
- $\Delta W'_{m,n_4}(1) + \Delta W'_{m,n_2}(\alpha^2) = 60 + 40 = 100$
- $\Delta W'_{m,n_4}(1) + \Delta W'_{m,n_3}(\alpha^2) = 60 + 65 = 125$

Paths for  $a = \alpha^2$  with one deviation:

- $\Delta W'_{m,n_2}(\alpha^2) = 40$
- $\Delta W'_{m,n_3}(\alpha^2) = 65$

Paths for  $a = \alpha^2$  with two deviations:

- $\Delta W'_{m,n_2}(1) + \Delta W'_{m,n_1}(\alpha) = 75 + 10 = 85$
- $\Delta W'_{m,n_2}(1) + \Delta W'_{m,n_4}(\alpha) = 75 + 5 = 80$
- $\Delta W'_{m,n_4}(1) + \Delta W'_{m,n_1}(\alpha) = 60 + 10 = 70$

From the conf( $n_r, n_c$ ) the paths with the minimum reliability value, the ones with a closer reliability to the hard-decision, are chosen as indicates step A3. For  $a = 1$ :  $\Delta W'_{m,n_4}(\alpha) + \Delta W'_{m,n_2}(\alpha^2) = 5 + 40 = 45$ . For  $a = \alpha$ :  $\Delta W'_{m,n_4}(\alpha) = 5$ . For  $a = \alpha^2$ :  $\Delta W'_{m,n_2}(\alpha^2) = 40$ . Hence, the value of  $\Delta \mathbf{W}$  is:

$a$	$\Delta W$
0	0
1	45
$\alpha$	5
$\alpha^2$	40

And the associated information for the paths,  $\eta'_j(a)$ , is:

$a/j$	1	2	3	4
0	0	0	0	0
1	0	$\alpha^2$	0	$\alpha$
$\alpha$	0	0	0	$\alpha$
$\alpha^2$	0	$\alpha^2$	0	0

To perform step A4, the information of the selected paths is required.

For  $a = 1$

- $\Delta R_{m,n_1}(a + \eta'_1(a) = 1 + 0) = \Delta W(1) - \Delta W'_{m,n_1}(0) = 45 - 0 = 45$
- $\Delta R_{m,n_2}(a + \eta'_2(a) = 1 + \alpha^2 = \alpha) = \Delta W(1) - \Delta W'_{m,n_2}(\alpha^2) = 45 - 40 = 5$
- $\Delta R_{m,n_3}(a + \eta'_3(a) = 1 + 0) = \Delta W(1) - \Delta W'_{m,n_3}(0) = 45 - 0 = 45$
- $\Delta R_{m,n_4}(a + \eta'_4(a) = 1 + \alpha = \alpha^2) = \Delta W(1) - \Delta W'_{m,n_4}(\alpha) = 45 - 5 = 40$

For  $a = \alpha$

- $\Delta R_{m,n_1}(a + \eta'_1(a) = \alpha + 0) = \Delta W(\alpha) - \Delta W'_{m,n_1}(0) = 5 - 0 = 5$
- $\Delta R_{m,n_2}(a + \eta'_2(a) = \alpha + 0) = \Delta W(\alpha) - \Delta W'_{m,n_2}(0) = 5 - 0 = 5$
- $\Delta R_{m,n_3}(a + \eta'_3(a) = \alpha + 0) = \Delta W(\alpha) - \Delta W'_{m,n_3}(0) = 5 - 0 = 5$
- $\Delta R_{m,n_4}(a + \eta'_4(a) = \alpha + \alpha) = \Delta W(\alpha) - \Delta W'_{m,n_4}(\alpha) = 5 - 5 = 0$

For  $a = \alpha^2$

- $\Delta R_{m,n_1}(a + \eta'_1(a) = \alpha^2 + 0) = \Delta W(\alpha^2) - \Delta W'_{m,n_1}(0) = 40 - 0 = 40$
- $\Delta R_{m,n_2}(a + \eta'_2(a) = \alpha^2 + \alpha^2) = \Delta W(\alpha^2) - \Delta W'_{m,n_2}(\alpha^2) = 40 - 40 = 0$
- $\Delta R_{m,n_3}(a + \eta'_3(a) = \alpha^2 + 0) = \Delta W(\alpha^2) - \Delta W'_{m,n_3}(0) = 40 - 0 = 40$
- $\Delta R_{m,n_4}(a + \eta'_4(a) = \alpha^2 + 0) = \Delta W(\alpha^2) - \Delta W'_{m,n_4}(0) = 40 - 0 = 40$

So the value of  $\Delta \mathbf{R}_{\mathbf{m},\mathbf{n}}$  after applying A4 is:

$a + \eta'_j(a)/j$	1	2	3	4
0	0	0	0	0
1	45	$x$	45	$x$
$\alpha$	5	5	5	$x$
$\alpha^2$	40	$x$	40	40

The  $x$  indicates the unfilled values of  $\Delta \mathbf{R}_{\mathbf{m},\mathbf{n}}$ . To fill them, the first minimum of the corresponding row of  $\Delta \mathbf{W}'_{\mathbf{m},\mathbf{n}}$  is selected if it does not belong to one of the selected paths. Otherwise, the second minimum is applied. According to this the row of the symbol 1 is filled with the first minimum of  $W'_{m,n}(1)$  (60), because it does not belong to the selected paths. To fill the row of  $\alpha$  the first minimum cannot be applied because it belongs to some of the selected paths (path for  $a = 1$  and  $a = \alpha$ ), so the second minimum of  $W'_{m,n}(\alpha)$  (10) is applied. To fill the row of  $\alpha^2$  the second minimum of  $W'_{m,n}(\alpha^2)$  (65) is selected, because the first one is applied to compute the path for  $a = \alpha^2$ . The resulting  $\Delta \mathbf{R}_{\mathbf{m},\mathbf{n}}$  is:



$a + \eta'_j(a)/j$	1	2	3	4
0	0	0	0	0
1	45	60	45	60
$\alpha$	5	5	5	10
$\alpha^2$	40	65	40	40

The last step of the algorithm shifts the values of  $\Delta R_{\mathbf{m},\mathbf{n}}$  to compute the output of the normal domain.

For  $a = 0$

- $R_{m,n_1}(a + \beta + z_{n_1} = 0 + \alpha^2 + \alpha^2 = 0) = -\Delta R_{m,n_1}(0) = 0$
- $R_{m,n_2}(a + \beta + z_{n_2} = 0 + \alpha^2 + \alpha = 1) = -\Delta R_{m,n_2}(0) = 0$
- $R_{m,n_3}(a + \beta + z_{n_3} = 0 + \alpha^2 + \alpha = 1) = -\Delta R_{m,n_3}(0) = 0$
- $R_{m,n_4}(a + \beta + z_{n_4} = 0 + \alpha^2 + 0 = \alpha^2) = -\Delta R_{m,n_4}(0) = 0$

For  $a = 1$

- $R_{m,n_1}(a + \beta + z_{n_1} = 1 + \alpha^2 + \alpha^2 = 1) = -\Delta R_{m,n_1}(1) = -45$
- $R_{m,n_2}(a + \beta + z_{n_2} = 1 + \alpha^2 + \alpha = 0) = -\Delta R_{m,n_2}(1) = -60$
- $R_{m,n_3}(a + \beta + z_{n_3} = 1 + \alpha^2 + \alpha = 0) = -\Delta R_{m,n_3}(1) = -45$
- $R_{m,n_4}(a + \beta + z_{n_4} = 1 + \alpha^2 + 0 = \alpha) = -\Delta R_{m,n_4}(1) = -60$

For  $a = \alpha$

- $R_{m,n_1}(a + \beta + z_{n_1} = \alpha + \alpha^2 + \alpha^2 = \alpha) = -\Delta R_{m,n_1}(\alpha) = -5$
- $R_{m,n_2}(a + \beta + z_{n_2} = \alpha + \alpha^2 + \alpha = \alpha^2) = -\Delta R_{m,n_2}(\alpha) = -5$
- $R_{m,n_3}(a + \beta + z_{n_3} = \alpha + \alpha^2 + \alpha = \alpha^2) = -\Delta R_{m,n_3}(\alpha) = -5$
- $R_{m,n_4}(a + \beta + z_{n_4} = \alpha + \alpha^2 + 0 = 1) = -\Delta R_{m,n_4}(\alpha) = -10$

For  $a = \alpha^2$

- $R_{m,n_1}(a + \beta + z_{n_1} = \alpha^2 + \alpha^2 + \alpha^2 = \alpha^2) = -\Delta R_{m,n_1}(\alpha^2) = -40$
- $R_{m,n_2}(a + \beta + z_{n_2} = \alpha^2 + \alpha^2 + \alpha = \alpha) = -\Delta R_{m,n_2}(\alpha^2) = -65$
- $R_{m,n_3}(a + \beta + z_{n_3} = \alpha^2 + \alpha^2 + \alpha = \alpha) = -\Delta R_{m,n_3}(\alpha^2) = -40$

$$\bullet R_{m,n_4}(a + \beta + z_{n_4} = \alpha^2 + \alpha^2 + 0 = 0) = -\Delta R_{m,n_4}(\alpha^2) = -40$$

The output  $\mathbf{R}_{\mathbf{m},\mathbf{n}}$  is :

$(a + \beta + z_{n_j})/j$	1	2	3	4
0	0	-60	-60	-40
1	-45	0	0	-10
$\alpha$	-5	-65	-40	-60
$\alpha^2$	-40	-5	-5	0

As it is shown in Example 1.4.3 all the steps from T-EMS can be calculated in parallel. Delta-domain steps are computed column-wise, so  $d_c$  independent processors are required for each column. Each processor needs to compute  $(q-1)$  subtractions and  $(q-1)$  GF additions in parallel to implement A1 in one clock cycle. On the other hand, the computation of the configuration sets, performed row wise, requires  $(q-1)$  processors. Each processor needs more than  $(d_c-1)$  comparators to look for the  $n_r$  minimums of the  $d_c$  elements in a row when  $n_r > 1$ . In addition,  $\binom{q-1}{n_c} \times n_r^{n_c}$  sums are necessary in the worst case to compute all the possible paths [25] and comparators are required to select the  $(q-1)$  minimum ones. The extrinsic output messages can be calculated also in a column-wise manner with  $d_c$  processors with  $q$  subtractors and  $2q$  GF adders each processor. The previously mentioned resources allow us to reduce latency to three clock cycles (one for the delta-domain, one for the syndrome and configurations sets and another for the extrinsic information) if variables are stored between step A1 and A3 and between A3 and A5. This latency can be reduced to two clock cycles if the information is only stored between step A3 and A4 as indicated at [25], however for some cases this full parallelism introduces some drawbacks. As  $q$  and  $d_c$  increases, the complexity in terms of number of parallel processors and resources inside each processor grow. This fact makes not only consuming more area but also increasing considerably the wiring inside the CNU, up to a point that routing problems could be found inside the CNU unit for GF(16) and above for high-rate codes (with large  $d_c$ ). In addition, storage resources are registers, not memories because the fully parallel processing requires that all the information must be available during all the processing time. There cannot be idle cycles accessing to memory. The use of registers instead of memories enlarges area (storing one bit of RAM is three times smaller than storing one bit in a register). Moreover, the fact that the T-EMS algorithm exchanges  $q$ -length vectors between the CNU and the VNU makes that routing problems discussed at Section 1.3 appear.

In conclusion, T-EMS can reach a latency similar to binary LDPC, but above GF(8) it will suffer from routing problems inside the CNU unit and between CNU and VNU units. This routing problems will prevent T-EMS architectures from reaching a throughput of Gbps with high-rate codes over fields above GF(8). In addition, the derived architecture for this codes will consume a large amount of

area resources. This algorithm is optimum for GF(4) where higher speed than the ones derived from forward-backward [13] and bubble check [16] algorithms can be reached. If higher speed is desired for high-rates codes, some coding gain has to be sacrificed in order to reduce the complexity of the decoder and increase its efficiency. To do it other family of algorithms will be studied in the next section: symbol-flipping algorithms.

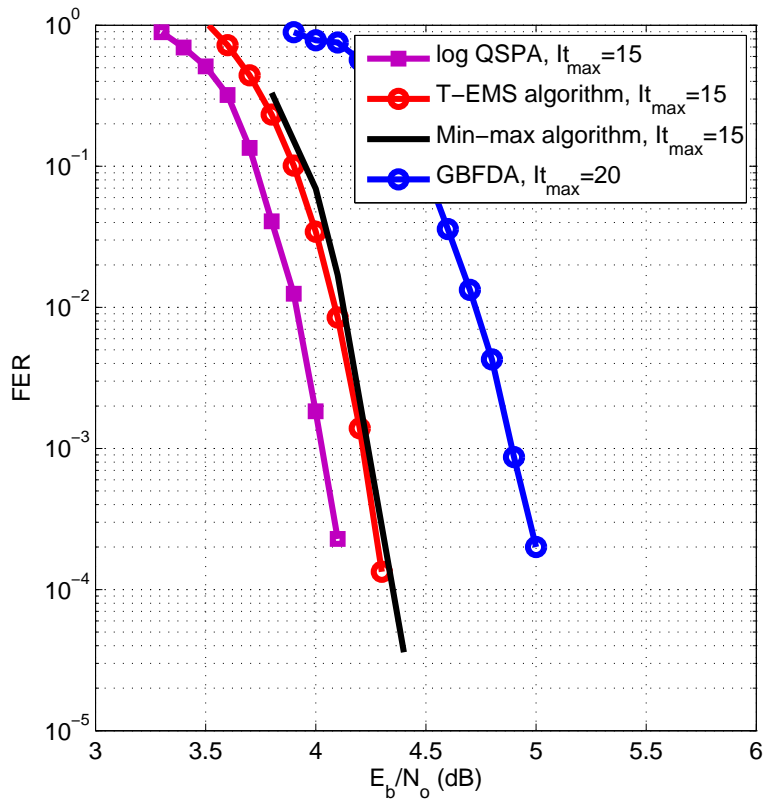
## 1.5 Symbol-flipping algorithms

At Section 1.3 and 1.4 algorithms derived from QSPA are analyzed concluding that EMS and Min-max involve high complexity in their check node update rules. The CNU algorithms of the different EMS and Min-max versions require the search of symbols that satisfy the parity check equations, working with vectors of  $q$  or  $n_m$  length which make use of soft-information from the channel. Techniques like forward-backward limit latency of the derived architectures due to the recursive process. Other solutions such as T-EMS, which allow us higher degree of parallelism, are limited by routing and area when the field order grows. So, if higher throughput and lower area are required, other alternatives must be studied.

Mainly, there are two solutions: reduce the number of iterations of the previously mentioned algorithms or reduce the complexity of the CNU. Both solutions lead to a performance loss. In order to reduce complexity of the CNU, hard-decision solutions can be applied. This means that instead of working with reliability vectors of  $q$  (or  $n_m$ ) length at the check node, only the hard-decision symbols are considered. This approach has been followed by symbol-flipping based algorithms such as majority-logic decodable (MD) algorithm [26] and generalized bit-flipping decoding algorithm (GBFDA) [27]. MD algorithm works efficiently under parity check codes based on Euclidean geometry [28] (with  $d_c = d_v$ ) and low rate; and GBFDA works better with high-rate codes and wider range of quasi-cyclic matrices [1] (such as the ones based on the multiplicative group of a finite field). The simplification of these algorithms at the CNU introduces some performance loss (between 0.7 and 1dB depending on the algorithm and the code, see Fig. 1.4), because of the lack of soft-information at the check node. This gap between the coding gain of EMS (or Min-max)<sup>4</sup> and symbol-flipping algorithms cannot be reduced by increasing the number of iterations of symbol-flipping algorithms. However, as we will discuss in next chapters, under the same performance, symbol-flipping architectures are more efficient than EMS or Min-max ones. Hence, if we can assume some performance loss, it is better in terms of hardware to apply symbol-flipping algorithms than to reduce the number of iterations of EMS or Min-max algorithms. For this reason, we include a detailed explanation of GBFDA, which is, to the best of the author's

<sup>4</sup>Note that comparisons with QSPA have been deliberately omitted because there is more than one order of magnitude in terms of complexity compared to EMS or Min-max and it just has a coding gain smaller than 0.16dB.

knowledge, the most efficient symbol-flipping algorithm for high-rate codes. Other symbol-flipping algorithms based on energy computation [29], [30] have not been taken into account due to the high performance loss introduced, compared to MD and GBFDA.



**Figure 1.4:** FER performance of log-QSPA algorithm, T-EMS algorithm, Min-max algorithm and GBFDA for a (837,723) NB-LDPC code over GF(32) with a binary phase-shift keying (BPSK) modulation in an additive white Gaussian noise (AWGN) channel.

In Algorithm 7 the GBFDA from [27] is described. This algorithm is based on a voting process.

The CNU is composed of steps A1 and A2, and computes all the operations over GF( $q$ ). Step A1 calculates the syndrome,  $s_m$ , for each check node  $m$ . If the syndrome is equal to zero, it means that the symbols  $Q_{m,n}$  involved in the computation, satisfy the parity check node equation  $m$ , so there is no need of flipping

**Algorithm 7** Generalized Bit-Flipping algorithm**Input:**  $Q_{m,n}^{(0)} = z_n$ ,  $\mathbf{W}_n^{(0)} = \mathbf{W}_{m,n}^{(0)} = \mathcal{F}(\mathbf{L}_n)$ 

Iterative process

**for**  $j = 1 \rightarrow It_{max}$  **do**

Check node processing

A1 :  $s_m = \sum_{n \in N(m)} h_{m,n} Q_{m,n}^{(j-1)}$

A2 :  $R_{m,n}^{(j)} = h_{m,n}^{-1} s_m - Q_{m,n}^{(j-1)}$

Variable node processing

A3 :  $\mathbf{W}_{m,n}^{(j)} = \mathbf{W}_{m,n}^{(j-1)} + \sum_{m' \in M(n) \setminus m} \delta(R_{m',n}^{(j)}, 1)$

A4 :  $Q_{m,n}^{(j)} = GFmax(\mathbf{W}_{m,n}^{(j)})$

A5 :  $\mathbf{W}_n^{(j)} = \mathbf{W}_n^{(j-1)} + \sum_{m' \in M(n)} \delta(R_{m',n}^{(j)}, v)$

Tentatively decoding

A6 :  $\tilde{c}_n^{(j)} = GFmax(\mathbf{W}_n^{(j)})$

**if**  $\tilde{\mathbf{c}} \times \mathbf{H}^T = 0$  **then**

SKIP

**end if****end for****Output:**  $\tilde{\mathbf{c}}$ 

the symbol values  $Q_{m,n}$ . However, if the result of step A1,  $s_m$ , is different from zero, new ‘candidates’ are proposed to flip  $Q_{m,n}$  values. Step A2 calculates the values of the new ‘candidates’ ( $R_{m,n}$ ) according to the syndrome information. In addition, step A2 ensures that the proposed ‘candidates’ ( $R_{m,n}$ ) satisfy the check node equations. This step calculates  $d_c$  ‘candidates’ for each check node  $m$ .

For the VNU a ‘voting process’ is carried out. Step A3 modifies  $\mathbf{W}_{m,n}$ , increasing in one vote the LLR information of the ‘candidates’ ( $R_{m,n}$ ). This is performed with the operation  $\delta(x, v)$  of A3 in which we ‘vote’ to the symbol  $x$  with an amplitude  $v$ . After the voting process of A3,  $\mathbf{W}_{m,n}$  values are sorted in step A4. The symbols with higher reliability (higher LLR and number of votes<sup>5</sup>) will be taken as the new hard-decision,  $Q_{m,n}$ . The iterative process will be repeated until the check nodes equations of all the rows are satisfied or the maximum number of iterations,  $It_{max}$ , is reached.

Steps A5 and A6 form the tentative decoding process. Step A5 accumulates in  $\mathbf{W}_n$  all the ‘votes’ for each symbol of the codeword. In step A6, the most reliable symbols of  $\mathbf{W}_n$  will be taken as the decoded codeword  $c_n$ .

Note that as the LLR and the voting domains are mixed at GBFDA, two parameters are required to transform the information: the function  $\mathcal{F}$  and the amplitude

<sup>5</sup>For GBFDA  $\mathbf{L}_n$  is defined as  $L_n[x] = \log[P(c_n = x | \mathbf{y}_n)/P(c_n = 0 | \mathbf{y}_n)]$  so the maximum value of  $L_n[x]$  is the most reliable.

value  $v$ . Function  $\mathcal{F}$  adapts the LLR values  $\mathbf{L}_n$  to the voting process, usually scaling the vector and performing a quantification. The amplitude  $v$  can be tuned to depending on  $\mathbf{W}_{m,n}$ . The common value for  $v$  is one for simplicity and  $\mathcal{F}$  is optimized via simulation to get the best performance.

Next, an example of how GBFDA works is included.

**Example 1.5.1** *In this example we will detail one iteration of GBFDA with parallel flooding schedule. Let us assume a parity matrix:  $\mathbf{H} = \begin{pmatrix} \alpha & 0 & \alpha^2 \\ 0 & 1 & \alpha^2 \\ \alpha & 1 & 0 \end{pmatrix}$*

*over  $GF(4)$ . The received LLR sequence is:  $L_1 = [0 \quad -4.7 \quad -23.2 \quad -3]$ ,  $L_2 = [0 \quad 3.2 \quad -7 \quad 24]$  and  $L_3 = [0 \quad 50 \quad 14 \quad -5]$ .  $\mathcal{F}$  multiplies  $\mathbf{L}_n$  by 0.5 and performs the round operation. The amplitude of the votes is  $v = 1$ . So  $\mathbf{W}_{m,n}$  is initialized to:  $\mathbf{W}_{1,1} = \mathbf{W}_{3,1} = \mathbf{W}_1 = [0 \quad -2 \quad -12 \quad -2]$ ,  $\mathbf{W}_{2,2} = \mathbf{W}_{3,2} = \mathbf{W}_2 = [0 \quad 2 \quad -4 \quad 12]$  and  $\mathbf{W}_{1,3} = \mathbf{W}_{2,3} = \mathbf{W}_3 = [0 \quad 25 \quad 7 \quad -3]$ .*

*The hard-decision symbols are:  $z_1 = 0$ ,  $z_2 = \alpha^2$  and  $z_3 = 1$ ; and  $Q_{1,1} = Q_{3,1} = z_1$ ,  $Q_{2,2} = Q_{3,2} = z_2$ ,  $Q_{1,3} = Q_{2,3} = z_3$ .*

*Remember that the product operation over  $GF(4)$  is defined as:*

$\otimes$	0	1	$\alpha$	$\alpha^2$
0	0	0	0	0
1	0	1	$\alpha$	$\alpha^2$
$\alpha$	0	$\alpha$	$\alpha^2$	1
$\alpha^2$	0	$\alpha^2$	1	$\alpha$

*First syndrome equations are computed:*

- $s_1 = h_{1,1}Q_{1,1} + h_{1,3}Q_{1,3} = \alpha \cdot 0 + \alpha^2 \cdot 1 = \alpha^2$
- $s_2 = h_{2,2}Q_{2,2} + h_{2,3}Q_{2,3} = 1 \cdot \alpha^2 + \alpha^2 \cdot 1 = 0$
- $s_3 = h_{3,1}Q_{3,1} + h_{3,2}Q_{3,2} = \alpha \cdot 0 + 1 \cdot \alpha^2 = \alpha^2$

*With this results we can know before computing step A2 that  $R_{2,2}$  and  $R_{2,3}$  candidates will be the same as  $Q_{2,2}$  and  $Q_{2,3}$ , because the second parity check equation is already satisfied. Using the syndrome information the ‘candidates’ for the voting process are calculated as follows:*

*For  $m = 1$ :*

- $R_{1,1} = h_{1,1}^{-1} \cdot s_1 - Q_{1,1} = \alpha^2 \cdot \alpha^2 - 0 = \alpha$
- $R_{1,3} = h_{1,3}^{-1} \cdot s_1 - Q_{1,3} = \alpha \cdot \alpha^2 - 1 = 0$

For  $m = 2$ :

- $R_{2,2} = h_{2,2}^{-1} \cdot s_2 - Q_{2,2} = \alpha^2$
- $R_{2,3} = h_{2,3}^{-1} \cdot s_2 - Q_{2,3} = 1$

For  $m = 3$ :

- $R_{3,1} = h_{3,1}^{-1} \cdot s_3 - Q_{3,1} = \alpha^2 \cdot \alpha^2 - 0 = \alpha$
- $R_{3,2} = h_{3,2}^{-1} \cdot s_3 - Q_{3,2} = 1 \cdot \alpha^2 - \alpha^2 = 0$

At step A3 the voting process with the extrinsic information is performed. As the matrix in the example has  $d_v = 2$  it receives just one vote, because  $d_v - 1 = 1$ . In bold are indicated the elements of  $W_{m,n}$  that receive votes.

For  $n = 1$

- $\mathbf{W}_{1,1} = \mathbf{W}_{1,1} + \delta(R_{3,1} = \alpha, v = 1) = [0 \quad -2 \quad \mathbf{-12+1} \quad -2]$
- $\mathbf{W}_{3,1} = \mathbf{W}_{3,1} + \delta(R_{1,1} = \alpha, v = 1) = [0 \quad -2 \quad \mathbf{-12+1} \quad -2]$

For  $n = 2$

- $\mathbf{W}_{2,2} = \mathbf{W}_{2,2} + \delta(R_{3,2} = 0, v = 1) = [\mathbf{0+1} \quad 2 \quad -4 \quad 12]$
- $\mathbf{W}_{3,2} = \mathbf{W}_{3,2} + \delta(R_{2,2} = \alpha^2, v = 1) = [0 \quad 2 \quad -4 \quad \mathbf{12+1}]$

For  $n = 3$

- $\mathbf{W}_{1,3} = \mathbf{W}_{1,3} + \delta(R_{2,3} = 1, v = 1) = [0 \quad \mathbf{25+1} \quad 7 \quad -3]$
- $\mathbf{W}_{2,3} = \mathbf{W}_{2,3} + \delta(R_{1,3} = 0, v = 1) = [\mathbf{0+1} \quad 25 \quad 7 \quad -3]$

After the voting process step A4 looks for the most reliable symbol with the updated information:

For  $n = 1$

- $Q_{1,1} = GFmax(\mathbf{W}_{1,1}) = GFmax([0 \quad -2 \quad -11 \quad -2]) = 0$
- $Q_{3,1} = GFmax(\mathbf{W}_{3,1}) = GFmax([0 \quad -2 \quad -11 \quad -2]) = 0$

For  $n = 2$

- $Q_{2,2} = GFmax(\mathbf{W}_{2,2}) = GFmax([1 \ 2 \ -4 \ 12]) = \alpha^2$
- $Q_{3,2} = GFmax(\mathbf{W}_{3,2}) = GFmax([0 \ 2 \ -4 \ 13]) = \alpha^2$

For  $n = 3$

- $Q_{1,3} = GFmax(\mathbf{W}_{1,3}) = GFmax([0 \ 26 \ 7 \ -3]) = 1$
- $Q_{2,3} = GFmax(\mathbf{W}_{2,3}) = GFmax([1 \ 25 \ 7 \ -3]) = 1$

As we can see, there is no flipping in the values ( $Q_{m,n}$  symbols at the beginning of the iteration are the same as  $Q_{m,n}$  symbols at the end), because the new ‘candidates’ have not receive enough votes to be ‘elected’, so more iterations will be required to produce a flipping. That is the reason why GBFDA and in general all the algorithms based on symbol flipping work better (converge faster) when codes with  $d_v \geq 3$  are used. As large  $d_v$  is, greater is the amount of information available at the VNU to chose the right candidate and faster the flipping of the symbols are produced.

Finally, step A5 accumulates all the votes performed at the location  $n$  of the code-word in the matrix  $\mathbf{W}_n$ :

$$\text{For } n = 1 : \mathbf{W}_1 = \mathbf{W}_1 + \delta(R_{3,1} = \alpha, v = 1) + \delta(R_{1,1} = \alpha, v = 1) = \\ [0 \ -2 \ -12+2 \ -2]$$

$$\text{For } n = 2 : \mathbf{W}_2 = \mathbf{W}_2 + \delta(R_{3,2} = 0, v = 1) + \delta(R_{2,2} = \alpha^2, v = 1) = \\ [0+1 \ 2 \ -4 \ 12+1]$$

$$\text{For } n = 3 : \mathbf{W}_3 = \mathbf{W}_{1,3} + \delta(R_{2,3} = 1, v = 1) + \delta(R_{1,3} = 0, v = 1) = \\ [0+1 \ 25+1 \ 7 \ -3]$$

Step A6 performs the tentative decoding:

$$\text{For } n = 1 : \tilde{c}_1 = GF(\mathbf{W}_1) = GFmax([0 \ -2 \ 10 \ -2]) = 0$$

$$\text{For } n = 2 : \tilde{c}_2 = GF(\mathbf{W}_2) = GFmax([1 \ 2 \ -4 \ 13]) = \alpha^2$$

$$\text{For } n = 3 : \tilde{c}_3 = GF(\mathbf{W}_{1,3}) = GFmax([1 \ 26 \ 7 \ -3]) = 1$$

As it can be seen in Example 1.5.1, GBFDA allows us the highest degree of parallelism with only one CNU processor, due to the use of just the hard-decision symbols. With this algorithm the complexity is moved from CNU to VNU, being one of the main problems the way of controlling the number of bits required for



$\mathbf{W}_{m,n}$  and  $\mathbf{W}_n$  (see next chapters). Moreover, the information exchanged between CNU and VNU,  $Q_{m,n}$  and  $R_{m,n}$  are not vectors as in EMS or Min-max, but scalar values. This fact reduces the complexity of the wiring between CNU and VNU units getting a similar routing congestion to the one reported at [22] for the binary case. Hence, for high-speed communication systems, algorithms derived from GBFDA will allow us to implement hardware architectures with a higher efficiency. The main drawback is that a performance in terms of error correction as high as EMS and Min-max cannot be reached with the actual state of the art, so one of the main points in the research of this field has to be focused on how to improve the coding gain of symbol-flipping algorithms without increasing the complexity.

## 1.6 Advantages of non-binary low-density parity-check decoding

After making a detailed revision of the existing NB-LDPC decoding algorithms and see that their main disadvantages for hardware implementation come from the fact of using Galois fields with  $q > 2$ , some logic questions arise: Do NB-LDPC codes have any advantage compared to binary LDPC? For the same complexity, can NB-LDPC outperform binary codes? Is the search of more efficient NB-LDPC decoder implementations justified?

As it has been reported several times in literature, a completely fair comparison between binary and non-binary LDPC codes is not realistic [31], [32]. The number of parameters that has to be taken into account is too large and usually the degree distribution, the girth and other variables from the code construction have different effects on binary and non-binary codes. Despite this fact, both coding theory and VLSI communities have attempted to compare these codes. Coding theory papers usually have tried to obtain codes with the same length in terms of bits, the same rate and the same degree distribution [31], [32], [8]. Simulations show that NB-LDPC codes with short and medium lengths obtain better performance, in terms of coding gain, than the binary ones. These simulations, performed over AWGN channel with BPSK modulation, show a difference of 0.2dB-0.5dB between binary and non-binary codes with GF(16) depending on different parameters. VLSI community comparisons of area, throughput and coding gain were usually done implementing codes of the same length in terms of bits but with different degree distributions. Differences in terms of coding gain over AWGN channel and BPSK modulation show that non-binary codes have a coding gain of 0.2dB-0.3dB with GF(32) compared to binary codes. Differences in terms of area and throughput show that NB-LDPC decoders are about  $q$  times less efficient than binary ones [24], [33]. This means that even reducing the number of iterations of the NB-LDPC decoders to get the same performance of binary ones, NB-LDPC decoders are less efficient than binary ones. However, comparisons of

these codes just take into account length and rate of the code, but not degree distribution or girth were considered. It is reasonable to think that maybe with a more fair comparison non-binary codes with GF(32) should obtain larger coding gain, as suggested in [32]. According to the previous mentioned conclusions we could summarize that NB-LDPC codes obtain some extra coding gain, compared to binary LDPC codes, but when the decoder is implemented, complexity is too high. Generally speaking, this last statement can be considered true when AWGN channel is considered and BPSK is the applied.

Recently some interesting studies have been done in the fields of optical communications and magnetic storage systems. These studies demonstrate that in certain scenarios, NB-LDPC are not just an efficient solution but maybe the only useful alternative to reach certain goals of next generation communication systems. These works analyze the effect of combining different modulations with NB-LDPC codes, such as: polarization-division-multiplexed quadrature phaseshift keying (PDM-QPSK) modulation [34], 8-PSK [35], QAM [36], 8-QAM, differential QPSK [37], etc. These researches show that NB-LDPC codes provide higher coding gain than the NCG<sup>6</sup> desired for next-generation optical communication systems, and some of them also prove that as higher the order the modulation is, higher the coding gain of NB-LDPC codes, compared to their binary counterparts. One very illustrative example is included at [38], where authors report coding gain between 0.29dB and 2.17dB for order of modulations between 16 and 64 compared to binary LDPC codes. Some of the previously reported experiments were performed over AWGN channel, but other use real optical channel. The ones based on real optical channels convert information of the optical fiber, after the demodulation, to the digital domain and then decode offline. On the other hand, the research in magnetic recording channels show that binary LDPC codes are not as efficient as NB-LDPC when burst errors are analyzed. In [39] an improvement of NB-LDPC compared to binary LDPC concatenated with Reed-Solomon codes is reported. This work concludes that NB-LDPC over GF(16) will be integrated in future hard disk drive systems, replacing binary LDPC and Reed-Solomon codes, as NB-LDPC have a good performance with both AWGN and burst errors.

For the previously mentioned reasons, we can conclude that the reduction of the complexity and the design of efficient implementations for NB-LDPC codes are justified even if they cannot be equal to the binary ones. Non negligible improvements in coding gain of more than 1dB with high order modulations and good behavior against burst errors make NB-LDPC codes great candidates for some scenarios in future optical communications and magnetic storage systems.

At this point, it can be advanced that BPSK modulation and AWGN channel are applied in the simulations included in this manuscript, in order to make fair com-

---

<sup>6</sup>The difference of  $E_b/N_0$  (or SNR) between the input of the system bit error rate and the output of the decoder is the defined as the NCG.

parisons with the existing hardware implementations. Simulations with magnetic recording channels, optical fiber and different modulations schemes will constitute future work. In either case, hardware improvements included in this document are independent of the selected channel and modulation.

## 1.7 Conclusions

In this chapter we have studied the main NB-LDPC algorithms, looking from a hardware point of view the ones which allow us to derive architectures with high speed for high-rate codes. The obtained conclusions are the following:

1. Algorithms derived from QSPA as EMS and Min-max require high complexity processing at the CNU due to the use of soft-decision information.
2. GF products can be avoided with methods such as the one proposed in the improved Min-max algorithm.
3. Forward-backward, bubble check or trellis based algorithms apply recursive methods that increase latency of the architectures, which is useful for wireless communication but not for other high-speed systems such as optical communication ones.
4. T-EMS is an special trellis algorithm that can support a higher degree of parallelism, however, its complexity is still high for codes over  $GF(16)$  and above if high throughput is the target.
5. Symbol flipping algorithms have a low complexity CNU based on computing just hard-decision information from the VNU.
6. The use of soft-decision allows EMS and Min-max to reach higher coding gain than symbol-flipping algorithms which lose channel information because they only compute hard-decision information at the CNU.
7. Symbol flipping algorithms reduce the wiring problems because exchange scalar messages between CNU and VNU, not vectors of length  $q$  or  $n_m$  as EMS and Min-max algorithms.



## Chapter 2

# Contributions to non-binary low-density parity-check decoding

In this chapter the reader will find the algorithmic and architectural contributions of the thesis to the field of non-binary low-density parity-check decoding. There are five main sections. The first one studies flooding schedule architectures for modified versions of GBFDA as well as the effects of routing and quantification for this schedule and proposes two architectures. The second section is focused on serial schedule in particular the layered one. This section includes two different high-speed architectures based on two enhanced versions of the serial GBFDA schedule. The third section takes advantage of the conclusions derived from the study of the state of the art and previous sections to propose a new multiple-vote algorithm which outperforms the coding gain of symbol-flipping algorithms, obtaining a performance close to EMS and Min-max with lower complexity. A derived architecture for the multiple-vote algorithm is proposed in the same section. The fourth section includes comparisons with the existing architectures found in literature and the fifth section outlines the conclusions of the chapter.

### 2.1 Parallel schedule algorithms and architectures

In this section several algorithmic simplifications, studies of quantification and finite precision techniques are proposed and analyzed with the aim of designing an efficient partial-parallel architecture for a NB-LDPC decoder with flooding (or parallel) schedule. Two are the main objectives: i) the reduction of storage resources; and ii) the reduction of latency and routing congestion.

### 2.1.1 Modified GBFDA

In GBFDA (Algorithm 7) tentative decoding requires storing, computing and sorting the values of  $\mathbf{W}_n$ , which is a matrix of size  $q \times N$ . The implementation of these tasks has a negative influence in the hardware architecture. In order to simplify hardware implementations of the algorithm we propose a new tentative decoding, maintaining the same CNU and VNU equations.

---

#### Algorithm 8 Modified tentative decoding algorithm

---

**Input:**  $Q_{m,n}^{(j)}$   
**for**  $n = 1 \rightarrow N$  **do**  
  **for**  $m = 1 \rightarrow M$  **do**  
    **if**  $h_{m,n} \neq 0$  **then**  
       $\tilde{c}_n^{(j)} = Q_{m,n}^{(j)}$   
      **SKIP**  
    **end if**  
  **end for**  
**end for**  
**Output:**  $\tilde{c}$

---

In Algorithm 8 the new tentative decoding is defined. It consists in the selection of one hard decision  $Q_{m,n}$  as decoded codeword symbol  $\tilde{c}_n$ , instead of using  $\mathbf{W}_n$  information. This allows us to eliminate  $\mathbf{W}_n$  and in consequence step A5 from GBFDA, as it does not participate at the check node or variable node processing. With this change in the tentative decoding, we approximate the hard-decision of the extrinsic information,  $\mathbf{W}_{m,n}$ , by the hard-decision of  $\mathbf{W}_n$ . Not considering the intrinsic information at  $\mathbf{W}_{m,n}$  for the tentative decoding introduces some differences in the performance for low number of iterations. These differences among  $\mathbf{W}_n$  and  $\mathbf{W}_{m,n}$  are reduced with the number of iterations, due to the fact that  $\mathbf{W}_{m,n}$  converges to  $\mathbf{W}_n$ . It is important to note that when the algorithm converges, symbols calculated as  $GFmax(\mathbf{W}_{m,n})$  satisfy the parity check equations and give a zero syndrome, which makes  $Q_{m,n}$  not to flip. So, if  $Q_{m,n}$  for all  $m$  and  $n$  values satisfy the parity check equations it means that  $\tilde{c}_n$  also satisfies the equations and  $Q_{m,n} = \tilde{c}_n$  for any  $m \in \mathcal{M}(n)$ . We call GBFDA with the modified tentative decoding Modified-GBFDA (M-GBFDA).

Fig. 2.1 shows FER performance of GBFDA and M-GBFDA with 10, 15, 20, 25 and 30 iterations, for a (837,723) NB-LDPC code <sup>1</sup> based on the quasi-cyclic

---

<sup>1</sup>Most of the examples, analysis and demonstrations of this manuscript will be performed with this code, because it is the most extended in the VLSI community and hence it is necessary to provide information according to it for fair comparisons with other authors. (837,723) and (837,726) NB-LDPC codes are chosen for comparisons in hardware implementation because they have large  $d_c$  ( $d_c = 26$  and  $d_c = 27$  respectively), large  $d_v$  ( $d_v = 4$ ) and a high Galois field order (GF(32)). Taking into account that it is not desirable to use codes with  $d_v$  bigger than 4 and  $d_c$

construction method proposed at [1]. This code has a parity check matrix  $\mathbf{H}$  with  $M = 124$ ,  $N = 837$ ,  $d_c = 26$ ,  $d_v = 4$  and  $h_{m,n} \in \text{GF}(32)$ . We can see in the figure that M-GBFDA requires more iterations to achieve the same performance as GBFDA when the algorithms have not reached the convergence<sup>2</sup>. However, for more than 30 iterations the FER performance is approximately the same for both algorithms, because they have already converged. We can see how the difference in coding gain is reduced from 0.07dB with 10 iterations to 0.007dB at 30 iterations, which illustrates our previous comments about the influence of the modified tentative decoding in the number of iterations.

To sum up, we can affirm that simplified tentative decoding increases the average number of iterations if an early stopping criterion is included in the decoder. However, it does not affect to the convergence, so in the worst case, the maximum number of iterations for both GBFDA and M-GBFDA will be the same. As the throughput of the decoder is limited by the maximum number of iterations in the worst case, not the average number, this technique is interesting because: i) saves storage resources of  $\mathbf{W}_n$ ; ii) reduces arithmetic resources required to compute step A5; and iii) eliminates clock cycles for sorting  $\mathbf{W}_n$  at step A6 of GBFDA. The only drawback is that if an stopping criterion is implemented the average number of iterations is increased. For implementations in next subsections, 20 iterations without early stopping are considered as the performance degradation is negligible, lower than 0.05dB.

### 2.1.2 Finite precision analysis for M-GBFDA: control of the data growth

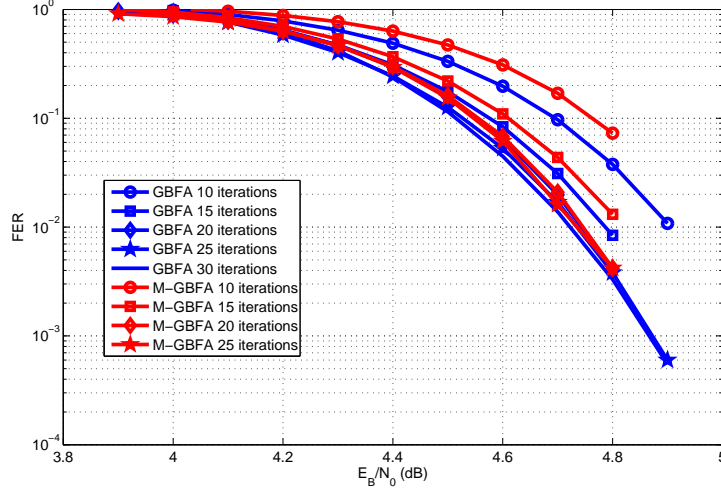
M-GBFDA and GBFDA require a method to avoid the data growth, because storage resources involve between 73 and 81% of the total area in the derived architectures as we will see in hardware sections of this manuscript. In addition, if the number of bits increases, the critical path and the routing problems will increase too. So, a method to avoid a significant increase of the number of bits, without losing coding gain, is needed to make an efficient implementation. Next, we include the data growth analysis for GBFDA, which was not included in the original paper [27] and the one for the M-GBFDA.

CNU (steps A1 and A2 of Algorithm 7) involve GF operations, so there are no problems of finite precision and data growth. However, the VNU (steps A3, A4 and A5) involve regular operations, and require a finite precision analysis. We

---

larger than 32, we can take the results for these codes as an upper bound. In other words, if one architecture is efficient for these codes, it will be efficient for other high-rate codes with lower  $d_c$  and  $d_v$ . In addition, it is not common to work with  $q > 64$ , due to the increase of channel information required.

<sup>2</sup>We can consider that GBFDA has converged with 20 iterations for this code, because increasing the number of iterations does not provide an important coding gain



**Figure 2.1:** Effect of the modified tentative decoding with the number of iterations

define two values for the quantification model:  $r$ , which is the number of bits to quantize  $\mathcal{F}(\mathbf{L}_n)$  in the initialization; and  $Q_b$ , which is the number of bits needed by  $\mathbf{W}_n$  and  $\mathbf{W}_{m,n}$  to grow in the execution of the iterative process. Note that the quantification of  $\mathbf{W}_n$  (steps A5 and A6) is not required for the M-GBFDA as  $\mathbf{W}_n$  is not involved.

First of all, we obtained by simulation the number of bits of  $\mathcal{F}(\mathbf{L}_n)$ ,  $r$ , for the initialization. For example, the chosen value for a (837,723) NB-LDPC code was  $r = 5$ , because with  $r$  higher than 5 there is no increase in coding gain and with  $r = 4$  there is a performance loss of 0.2dB.  $\mathcal{F}(\mathbf{L}_n)$  was initialized between a maximum of  $2^{r-1} - 1$  and a minimum  $-2^{r-1}$ , but we checked that all the negative values can be fixed to zero without performance loss, being the negative values the less reliable ones and the maximum positive value the hard-decision reliability. This reduction of information is similar to the selection of the  $n_m$  most reliable symbols at EMS algorithm [12]. In this way, the number of bits required for the quantification is reduced in one, because the bit of sign is not needed.

Once  $r$  was chosen, we fixed the number of bits  $Q_b$  for  $\mathbf{W}_n$  and  $\mathbf{W}_{m,n}$ . Unless stated otherwise, we consider the amplitude of the vote  $v = 1$  for the rest of the chapter. In that case, one value can be increased in each iteration at most in  $d_v - 1$  for  $\mathbf{W}_{m,n}$  and in  $d_v$  for  $\mathbf{W}_n$ , as we explained at Example 1.5.1. So, for  $It_{max}$  iterations, the maximum value that we can find in  $\mathbf{W}_n$  is  $2^r - 1 + d_v \times It_{max}$  and  $2^r - 1 + (d_v - 1) \times It_{max}$  for  $\mathbf{W}_{m,n}$ ; thus the value  $Q_b$  must be chosen consequently.



Following with the previous example for the (837,723) code, with  $It_{max} = 20$ ,  $d_v = 4$  and  $r = 5$ , the maximum possible value is 111 for  $\mathbf{W}_n$  and 91 for  $\mathbf{W}_{m,n}$ , so  $Q_b = 7$  bits are required to quantize  $\mathbf{W}_n$  and  $\mathbf{W}_{m,n}$  in the worst case if we allow data to grow. With the objective of reducing  $Q_b$  without introducing a performance degradation, a technique to control the data growth must be applied. The most common techniques are detailed below.

- Saturation:** Saturation is a common solution to avoid the growth of the data in signal processing. When saturation is applied to the GBFDA or the M-GBFDA, the initialization is done with  $r$  bits, and we only let data grow up to  $2^{Q_b} - 1$ , being  $r < Q_b < r + \log_2(d_r \times It_{max})$ . If  $Q_b \leq r$  the distances between the reliability values of  $\mathbf{W}_{m,n}$  are modified at the first iteration because no data growth is allowed. This causes a performance degradation. In fact, what is really happening when no data growth is allowed is that the most reliable value, which is saturated, is approaching to the rest of the voted values and, in some cases, unreliable symbols can even replace it and make the algorithm diverge. For this reason, saturation with  $Q_b = r$  causes an early floor degradation, which makes impractical the algorithm. On the other hand, the growth of one bit ( $Q_b = r + 1$ ) has a great impact in the architecture: it increases the area in 15% and reduces the throughput at least 14%, as explained in next sections.
- Clipping technique:** A method called clipping was proposed in [40] to control the data growth. With this technique,  $r$  bits are used for the initialization and  $Q_b = r$  bits are applied in the iterative process. The clipping technique is applied as follows to GBFDA (or M-GBFDA): when the biggest reliability of  $\mathbf{W}_{m,n}$  or  $\mathbf{W}_n$  is equal to  $2^{Q_b} - 1$  and receives a vote, the voted value is kept as  $2^{Q_b} - 1$  and the rest of the reliability values in  $\mathbf{W}_{m,n}$  or  $\mathbf{W}_n$  are subtracted by a vote. After the subtraction, negative values of  $\mathbf{W}_{m,n}$  or  $\mathbf{W}_n$  are fixed to zero. With this method, distances between  $\mathbf{W}_{m,n}$  or  $\mathbf{W}_n$  values of the most reliable symbols are kept in the same way as we let data grow, ensuring the convergence of the algorithm. Applying this technique  $\mathbf{W}_{m,n}$  and  $\mathbf{W}_n$  have a minimum value of zero and a maximum of  $2^{Q_b} - 1$ . The main disadvantage of this method is that extra complexity is required. Supposing that only one overflow is detected in  $\mathbf{W}_{m,n}$ ,  $q - 1$  subtractions to apply the clipping technique and  $q - 1$  comparisons with zero to fix the negative values to zero must be applied. These extra operations increase complexity and have a great impact on the hardware designs because depending on the degree of parallelism more area or clock cycles are needed to implement clipping. Table 2.1 shows the operations needed to apply clipping to M-GBFDA in the worst case. The worst case is considered as the one in which all the nonzero elements of the matrix overflow their  $\mathbf{W}_{m,n}$  values and, hence, the clipping technique must be applied for all of them.

**Table 2.1:** Operations and complexity of the techniques for control of data growth in one iteration of M-GBFDA (worst case)

Method	Comparisons with the local maximum	Comparisons with $2^{Q_b}$	Comparisons with zero	Subtractions
<b>Clipping</b>	0	$d_c \times M$	$(q-1) \times d_c \times M$	$(q-1) \times d_c \times M$
<b>Blocking</b> ( $r = Q_b$ )	0	$d_c \times M$	0	0
<b>Blocking</b> ( $r = Q_b - 1$ )	$d_c \times M$	$d_c \times M$	0	0

In the next subsection, a sub-optimal solution to data growth problem is proposed, trying to reduce the complexity of the control techniques to derive more efficient architectures. This technique is called blocking.

### 2.1.3 Blocking technique

Blocking technique applied to M-GBFDA is included at Algorithm 9. This algorithm initializes  $\mathcal{F}(\mathbf{L}_n)$  with  $r$  bits, setting its maximum value to  $2^r - 1$  and its minimum value to 0.  $\mathbf{W}_{m,n}$  is quantized with  $Q_b$  bits, so we let the data grow from  $2^r - 1$  to  $2^{Q_b} - 1$ . The set  $\mathcal{B}$  contains  $n$  and  $m$  indexes. In the initialization, the set is empty and, after that, it saves the indexes of the values in  $\mathbf{W}_{m,n}$  that are blocked at the current iteration and also the indexes of the ones blocked at the previous iterations. To perform or not step A3 of Algorithm 9, the blocking technique makes comparisons with the elements at  $\mathbf{W}_{m,n}$  that receive a vote. An overflow is detected in  $\mathbf{W}_{m,n}$  if the result of adding the votes to a reliability value of  $\mathbf{W}_{m,n}$  is bigger than  $2^{Q_b} - 1$ . In this case, the proposed technique blocks all the  $q$  reliability values in  $\mathbf{W}_{m,n}$ . Blocking  $\mathbf{W}_{m,n}$  means that until the iterative process finishes no more updates are allowed for this node. Note that distances in  $\mathbf{W}_{m,n}$  are kept when an overflow is detected because the corresponding  $q$  reliability values are blocked. The rest of nodes different from  $\mathbf{W}_{m,n}$  follow the same steps as usual if no overflows are detected.

Next we explain how the complete M-GBFDA with blocking works. Steps A1 and A2 are exactly the same as the ones in Algorithm 7. Step A3 detects the overflows produced by adding the votes to the reliability values  $\mathbf{W}_{m,n}$ . Comparisons with  $2^{Q_b}$  are made to decide if the blocking is applied. When an overflow is detected for a node with indexes  $n$  and  $m$ ,  $\mathbf{W}_{m,n}$  is not updated anymore, it is blocked, and  $Q_{m,n}$  is updated to  $R_{m,n}$  (step A4), as its maximum reliability value in  $\mathbf{W}_{m,n}$  is the one that caused the overflow. For the iteration  $j$ , the indexes  $n$  and  $m$  of each  $\mathbf{W}_{m,n}$  blocked during this iteration are included in  $\mathcal{B}^{(j)}$ , and also the indexes of the nodes blocked at the previous iterations, named as  $\mathcal{B}^{(j-1)}$  (step A3). If no overflow is detected at the  $j$ th iteration or at the previous iterations, for a given

**Algorithm 9** M-GBFDA with blocking technique**Input:**  $Q_{m,n}^{(0)} = z_n$ ,  $\mathbf{W}_{\mathbf{m},\mathbf{n}}^{(0)} = \mathcal{F}(\mathbf{L}_n)$ ,  $\mathcal{B}^{(0)} = \emptyset$ 

Iterative process

**for**  $j = 1 \rightarrow It_{max}$  **do**

Check node processing

  A1 :  $s_m = \sum_{n \in N(m)} h_{m,n} Q_{m,n}^{(j-1)}$

  A2 :  $R_{m,n}^{(j)} = h_{m,n}^{-1} s_m - Q_{m,n}^{(j-1)}$

Variable node processing

**if**  $\mathbf{W}_{\mathbf{m},\mathbf{n}}^{(j-1)} + \sum_{m' \in M(n) \setminus m} \delta(R_{m',n}^{(j)}, v) = 2^{Q_b}$  **then**

    A3 :  $\mathcal{B}^{(j)} \in \{\mathcal{B}^{(j-1)}, (m, n)\}$

    A4 :  $Q_{m,n}^{(j)} = R_{m,n}^{(j)}$

**end if**

  A5 :  $\mathbf{W}_{\mathbf{m},\mathbf{n}}^{(j)} = \mathbf{W}_{\mathbf{m},\mathbf{n}}^{(j-1)} + \sum_{m' \in M(n) \setminus m} \delta(R_{m',n}^{(j)}, v)$ ,  $m, n \notin \mathcal{B}^{(j)}$

  A6 :  $Q_{m,n}^{(j)} = GFmax(\mathbf{W}_{\mathbf{m},\mathbf{n}}^{(j)})$ ,  $m, n \notin \mathcal{B}^{(j)}$

Tentatively decoding

A7 : Algorithm 8

**if**  $\tilde{\mathbf{c}} \times \mathbf{H}^T = 0$  **then**

SKIP

**end if****end for****Output:**  $\tilde{\mathbf{c}}$

$n$  and a given  $m$ ,  $\mathbf{W}_{\mathbf{m},n}$  is updated applying step A5. As  $\mathbf{W}_{\mathbf{m},n}$  can grow from  $2^r - 1$  to  $2^{Q_b} - 1$ ,  $Q_{m,n}$  is also updated for the non-blocked indexes in step A6. Note that step A6 is not necessary when  $r = Q_b$  because the data growth is not allowed in  $\mathbf{W}_{\mathbf{m},n}$ . Step A7 implements the modified tentative decoding included in Algorithm 8.

In Table 2.1 the complexity of M-GBFDA with the blocking technique for two cases can be seen. With  $r = Q_b$  the blocking technique avoids  $(q - 1) \times d_c \times M$  comparisons with zero and  $(q - 1) \times d_c \times M$  subtractions however, the limitation of not letting most reliable information grow causes an early error floor that makes this case impractical for some applications. On the other hand, blocking technique with  $r = Q_b - 1$  also saves  $(q - 1) \times d_c \times M$  comparisons with zero and  $(q - 1) \times d_c \times M$  subtractions, but requires  $d_c \times M$  comparisons with the local maximum to perform step A6. This case has lower complexity than the clipping technique, however, in some cases the blocking technique prevents the updating of nodes that, even reaching the maximum value of quantization, have not converged yet. In other words, the blocking technique does not allow some symbols to flip if their reliability values have reached the maximum, this makes the algorithm be trapped due to erroneous decisions which cannot be corrected by flipping. This behavior is less common when  $r = Q_b - 1$  than with  $r = Q_b$ , because in the first case the algorithm has run more iterations unblocked, so blocking decisions are taken in most of the cases near the convergence region. This is why for  $r = Q_b - 1$  the blocking algorithm becomes trapped with less frequency than with  $r = Q_b$ , and as consequence error floor appears for the first case in a lower region. As the blocking technique introduces this degradation in the error correction, we consider it a sub-optimal solution to the data growth problem. Despite this problems, blocking technique can be applied in different scenarios and achieve high speed in hardware implementation architectures.

#### 2.1.4 Frame error rate performance comparison between NB-LDPC decoders

Fig. 2.2 compares the FER and BER performance of Min-max [13], MD [26], GBFDA with clipping [27] and M-GBFDA with clipping and blocking for the (837,723) NB-LDPC code. We used a BPSK modulation in an AWGN channel and encoded a different random message  $\mathbf{m}$  with each transmitted package. A complete finite precision analysis was made to fix the values of  $Q_b$  and  $r$ . Our simulations showed that  $Q_b$  must be set to 5 in GBFDA and M-GBFDA to avoid performance loss in the waterfall region with respect to the non-quantized algorithms. In addition, for  $r < 4$  there is an early error floor (above a FER of  $10^{-3}$ ) that makes the algorithm impractical for most communication systems. The special case of  $Q_b = r = 5$ , which reduces the complexity of the algorithm and the decoder architecture, generates an error floor at a FER of  $2 \times 10^{-4}$  (BER= $6 \times 10^{-7}$ ). To shift this error floor to a FER of  $2 \times 10^{-5}$  (BER= $7 \times 10^{-8}$ ),  $Q_b$  must be set

to 5 and  $r$  to 4. As we explained in the previous section, this error floor is due to the fact that we cannot modify the symbols in later iterations after being blocked, unlike the original GBFDA, which is able to replace  $Q_{m,n}$  symbols in every iteration. Anyway, there are many applications where our decoder can be used as an standalone FEC decoder, such as wireless communications, where the BER constraint is achieved. Additionally, the error floor of our proposal and the waterfall coding gain accomplish the constraints for future optical communications systems. In [41], authors limit the error floor introduced by the NB-LDPC decoder in a optical communication system to BER  $10^{-7}$ , because this degradation can be corrected by means of a hard decision FEC (such as a Reed-Solomon decoder), which is placed after the NB-LDPC decoder. Our proposal has a lower error floor than the constraint (error floor at BER= $7 \times 10^{-8}$ ). On the other hand, extrapolating the slope of the BER curve as in [41], our LDPC decoder meets the net coding gain (NCG) requirement of 10.8dB at BER  $10^{-15}$ .

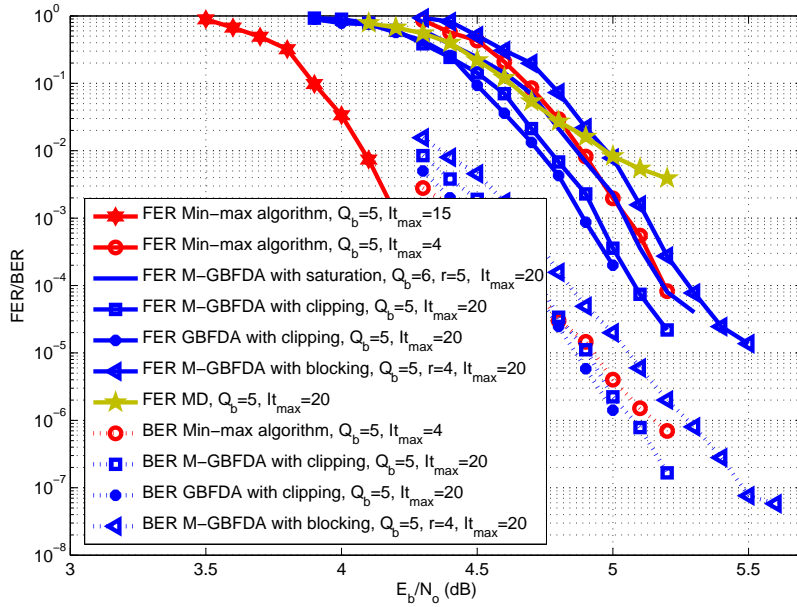
Other important issue to remark, which was mentioned in previous section, is that GBFDA cannot achieve the same performance as Min-max when both algorithms have converged (15 iterations for Min-max and 20 for GBFDA), because GBFDA works only with hard-decision information and Min-max uses soft-information (all the channel information available). Min-max achieves a coding gain of 0.6dB compared to GBFDA after the convergence. However, the complexity of GBFDA is much lower, so to do a fair comparison we consider Min-max with 4 iterations, which gets a coding gain similar to GBFDA. M-GBFDA with clipping and  $Q_b = 5$  obtains a coding gain of 0.15 dB compared to Min-max with 4 iterations and 5 bits. M-GBFDA with blocking technique and  $Q_b = 5$  and  $r = 4$  has only 0.07 dB of performance loss, compared to Min-max with 4 iterations and the previously mentioned error floor (BER  $7 \times 10^{-8}$ ).

The MD algorithm has lower complexity than M-GBFDA and provides good results for low-rate codes with high  $d_v$ , usually  $d_c = d_v$ . However, for high-rate codes, such as the (837,723) NB-LDPC code, it has an early error floor (FER of  $7 \times 10^{-2}$ ) as it is said in [42] and shown in Fig. 2.2. This turns the algorithm impractical for the applications that require such codes.

### 2.1.5 Partial parallel architectures for the flooding schedule

In this subsection two different kinds of partial parallel architectures <sup>3</sup> for M-GBFDA with flooding (or parallel) schedule are presented. First, message passing architectures are described. These designs separate hardware units into CNU and VNU processors following the equations of the algorithms. However, we will show that this division of hardware leads to routing congestion, limiting the maximum

<sup>3</sup>Fully parallel and serial architectures have not been considered because the first ones require an amount of area that is not affordable with the existing technology and the second ones have an extremely large latency.



**Figure 2.2:** FER and BER performance of Min-max algorithm, GBFDA, M-GBFDA and MD algorithm for a (837,723) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel.

frequency achievable because of the excess of wiring between processors (or hardware units). The second approach is a broadcasting architecture in which different processors separate the implemented equation not in CNU or VNU, but in an efficient way that reduces the amount of information exchanged between hardware units, reducing the routing problems.

### Message passing architecture for M-GBFDA with and without clipping

This architecture is divided into two main parts: the CNU unit and the VNU units. There are  $d_c \times d_v$  VNU units and one CNU unit. Three different architectures for the VNU units are exposed: one based on clipping technique, one to implement the blocking technique with  $r = Q_b$  and another to implement the blocking technique with  $r = Q_b - 1$ . With the exception of the VNU unit, the rest of the architecture is exactly the same for all data growth control techniques.

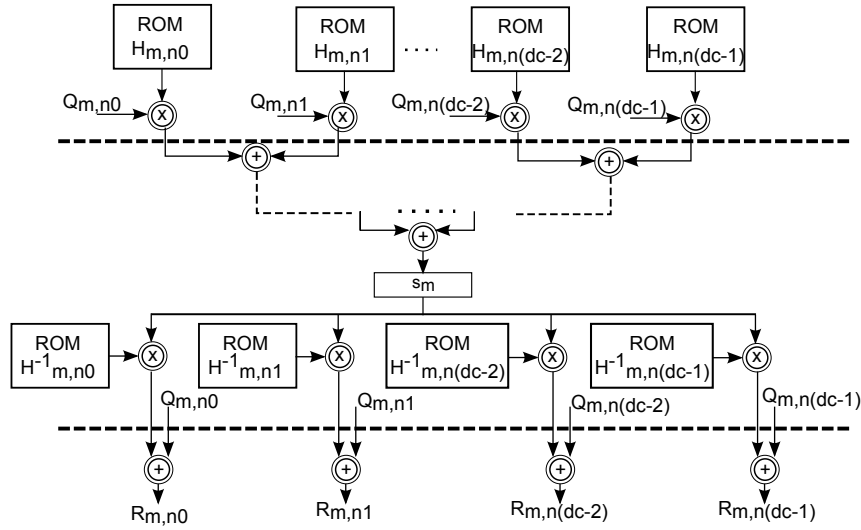


Figure 2.3: CNU unit for M-GBFDA message passing architecture.

1. **CNU unit:** As commented in previous sections CNU only requires operations over  $\text{GF}(q)$ , so all the operators included at the CNU unit, Fig. 2.3, are over  $\text{GF}$ . The proposed implementation for equation A1, which calculates the syndrome  $s_m$  for each check node, requires  $d_c$   $\text{GF}$  multipliers and a tree of  $d_c - 1$   $\text{GF}$  adders, so all the  $M$  syndromes can be computed in  $M$  clock cycles. The inputs of the  $d_c$   $\text{GF}$  multipliers are the  $d_c$  nonzero coefficients of the matrix  $\mathbf{H}$  for a check node  $m$ , and the  $Q_{m,n}$  hard decision values. So,  $d_c$  different ROMs of length  $M \times p$  are needed for storing the coefficients of

**H.** The results of the  $d_c$  multipliers are added with the tree of GF adders, giving as a result the syndrome  $s_m$ , which is stored in a register. For step A2,  $d_c$  GF multipliers make the product of  $s_m$  by the  $\mathbf{H}^{-1}$  coefficients. The outputs of these multipliers are added to the  $Q_{m,n}$  using  $d_c$  GF adders. As a result,  $d_c$   $R_{m,n}$  values for a check node  $m$  are obtained in parallel. These  $d_c$   $R_{m,n}$  values are connected to the inputs of the VNU units. In addition,  $2d_c$  pipeline registers of  $p$  bits are needed at the output of the GF multipliers to reduce the critical path of the architecture. These registers are drawn as dashed lines in Fig. 2.3.

## 2. VNU unit:

- **Common aspects for clipping and blocking:** The VNU unit does not involve GF( $q$ ) operations, it works with regular arithmetic. The architecture of this block is introduced in Fig.2.4 for clipping and in Fig.2.6 for blocking. There are  $d_c \times d_v$  VNU units, and each VNU unit computes  $Q_{m,n}$  for  $M/d_v$  different indexes  $(m,n)$ . For initializing the algorithm, the input  $\mathcal{F}(\mathbf{L}_n)$  is selected with the multiplexor M1 and, then, the RAM is filled with the corresponding reliability values. The RAM stores the quantized elements ( $Q_b$  bits) of  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$ <sup>4</sup> for  $M/d_v$  check nodes connected, that is,  $M/d_v \times q$  reliability values of  $Q_b$  bits. The  $M/d_v \times \lceil \log_2(M/d_v) \rceil$  ROM output indicates which check node we are working with (one of the  $M/d_v$  check nodes that are computed in a VNU cell). During the initialization the multiplexor M0 selects the input *Add-low*, which is connected to a  $p$ -bit counter, to initialize the elements of  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  stored in the RAM. So, the most significant bits of the address word select the check node which we work with and the less significant bits select the GF element whose reliability is processed.
- **Clipping technique** (Fig. 2.4 and 2.5): To implement step A3 of M-GBFDA the input  $R_{m,n}$  is selected with M0. The  $R_{m,n}$  signal, concatenated with the output of the ROM, is used as address of the RAM to select the reliability value in  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  that is going to be increased by '1' (remember that the amplitude of the vote is  $v = 1$ ). Once the addition is calculated (selecting '1' as an input of the adder/subtractor by using M2 and M3), the new  $W_{m,n}(R_{m,n})$  is stored again in the RAM. The AND gate compares the output of the adder with the maximum value,  $2^{Q_b} - 1$ , to generate a clipping signal, which indicates whether clipping has to be applied or not. This signal is stored in a  $(M/d_v)$ -bit register, where each bit corresponds to one of the  $M/d_v$  check nodes that are stored and computed in a single VNU cell. So, if one of the  $q$  reliability values of  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  exceeds the maximum value  $2^{Q_b} - 1$ , the bit corresponding to the  $m$  check node will be set to '1' in the  $(M/d_v)$ -

<sup>4</sup>Remember that  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  is a matrix of size  $M \times d_c$  which elements are vectors of length  $q$  ( $q$  different reliability values)



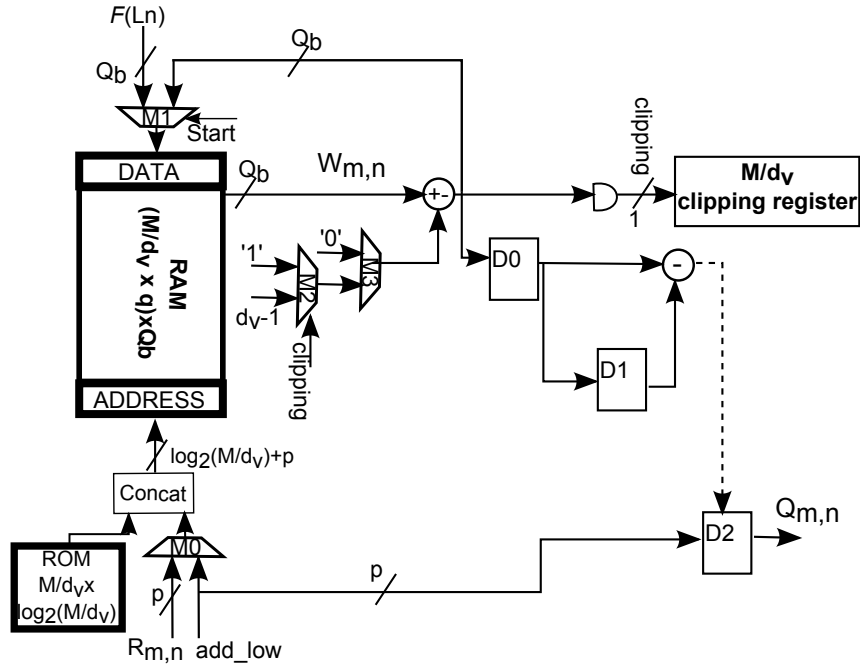


Figure 2.4: VNU basic cell for M-GBFDA message passing architecture with clipping.

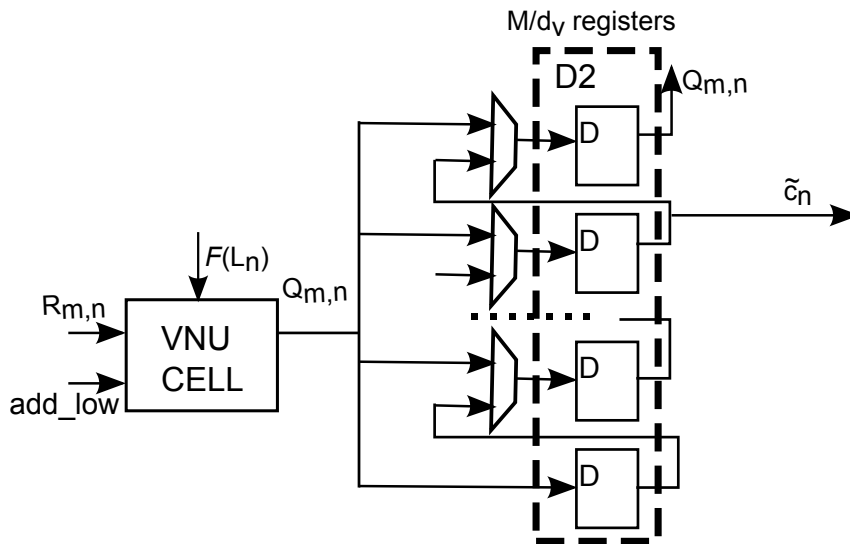


Figure 2.5: VNU unit for M-GBFDA message passing architecture with clipping.

bit register, indicating that clipping must be applied to the rest of the  $q - 1$  reliability values. It was shown by simulation that when clipping is applied the effect of subtracting the votes that has received the maximum or subtracting the constant value  $d_v - 1$ <sup>5</sup> leads to a similar performance, as the  $q - 1$  symbols different from the most reliable keep their distances. The subtraction is carried during the next step A4. In order to compute equation A4 and apply the clipping technique, a sorting process is followed. To process all the reliability values stored in the RAM, a counter is used as address (*Add-low* in Fig. 2.4). If the clipping signal of the corresponding check node is set to ‘1’ (clipping activated), the output of the RAM is subtracted by  $d_v - 1$  votes (the value  $d_v - 1$  is selected as input of the adder/subtractor using *M2* and *M3*). In addition, the value with or without clipping is stored in *D0* to be compared with the value of *D1*. The comparison is performed with a subtracter. If the value in *D0* is bigger than *D1*, the enable of *D1* is activated to store *D0* in *D1*, otherwise, the value of *D1* remains the same. At the same time, the value of the address (which is the hard-decision symbol of  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$ ) is stored in *D2*. So, in parallel,  $Q_{m,n}$  is found (step A4) and the clipping technique is applied.

The VNU cell computes  $M/d_v$  rows, so we need  $M/d_v$  registers of  $p$  bits to store the  $Q_{m,n}$  elements of each row. In Fig.2.5, we include the registers and the multiplexors required to store the GF symbols of step A4 for  $M/d_v$  check nodes. Note that with the registers of Fig.2.5 we do not need the register *D2* of Fig.2.4, so the enable signal and  $Q_{m,n}$  can directly pass to the registers of Fig.2.5. The outputs  $Q_{m,n}$  will be stored in the shift register following the order of check nodes from the matrix  $\mathbf{H}$ . Each cycle after completing the step A4, the shift register will shift and provide a different  $Q_{m,n}$  to the CNU unit to make the computation of A1 and A2. According to Algorithm 8, the output of some registers of the shift register *D2* can be taken as  $\tilde{c}_n$ . We only have to hard-wire  $N$  registers *D2* as outputs of the decoder, among the  $d_c \times d_v$  VNU units.

- **Blocking technique** (Fig.2.6, 2.7 and 2.8): To implement condition of step A3 and step A5 of Algorithm 9, the input  $R_{m,n}$  is selected with the multiplexor *M0*. The  $R_{m,n}$  signal, concatenated with the output of the ROM, is used as address of the RAM to select the reliability value of  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  which is going to be increased by ‘1’. Additionally, the *AND* gate checks if the selected  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  associated to the  $R_{m,n}$  is equal to the maximum,  $2^{Q_b} - 1$ . In this case, the output of the gate is ‘1’, the write enable of the RAM memory is disabled and the blocking register is updated. The blocking register is a  $M/d_v$ -length register, where each

<sup>5</sup>This value is chosen because it is the maximum number of votes that  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  can receive in one iteration. See Example 1.5.1

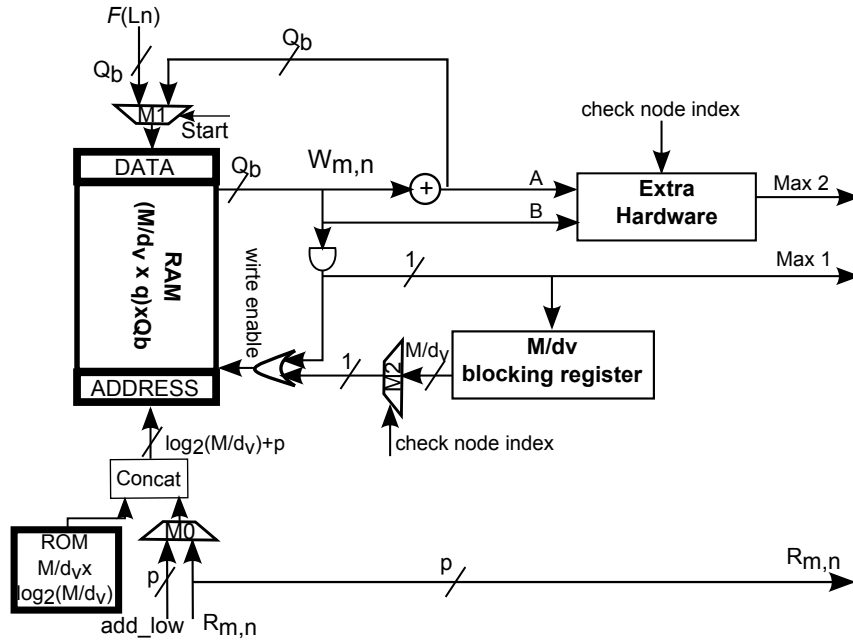


Figure 2.6: VNU basic cell for M-GBFDA message passing architecture with blocking.

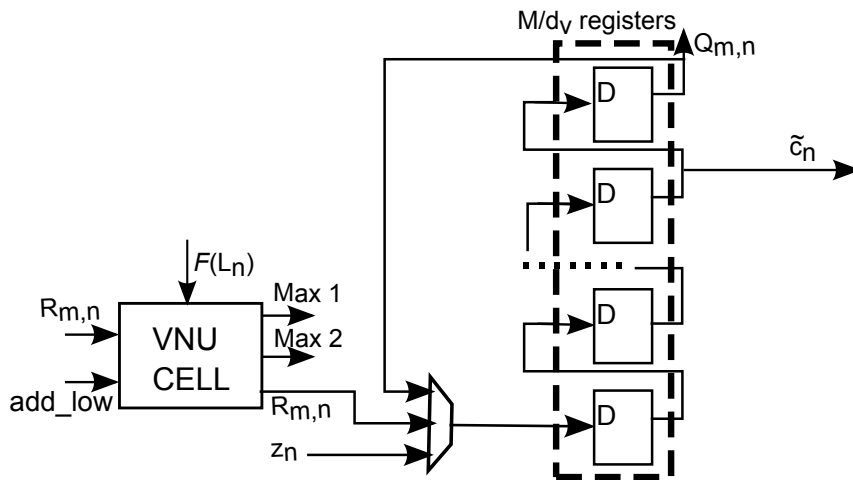
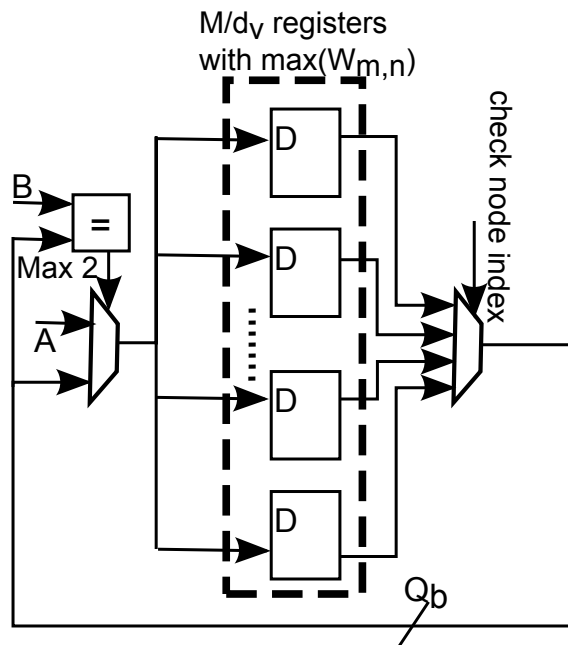


Figure 2.7: VNU unit for M-GBFDA message passing architecture with blocking.



**Figure 2.8:** Extra hardware to compute blocking technique with  $r = Q_b$  for M-GBFDA.

bit represents a different check node of the ones processed by this VNU cell. If the bit of a given  $m$  is activated, the  $q$  reliability values of the corresponding  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  must be blocked, so the write enable of the RAM memory for the addresses of these elements is disabled until the last iteration. This is done by means of the  $M/d_v$  to 1 multiplexer, which selects the bit of the blocking register corresponding to the check node we are working with. If this bit is equal to '1', it means that the current row was blocked in previous iterations, so the write enable of the RAM is disabled. As it has been mentioned above, the write enable is disabled when  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  is blocked in the current iteration, this is done with the *OR* gate. To ensure that  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  only contains the extrinsic information for a given  $m$  (condition of step A3 and step A5), the write enable is also disabled when we detect that the intrinsic information is at the output of the RAM. On the other hand, if the output of the multiplexer is equal to '0', and the output of the *AND* gate is '0', it means that the current row has not been blocked yet, so the voted value of  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  (increased by '1') is saved again in the RAM. In this way, step A5 is completed. Note that the implementation of the clipping technique needs at least  $M/d_v \times q$  more clock cycles per iteration, because all the elements in the RAM have to be subtracted, so at the end,  $M/d_v \times q \times It_{max}$  extra clock cycles will be needed compared to the blocking technique.

On the other hand, the difference between blocking with  $r = Q_b$  and  $r = Q_b - 1$  is how  $Q_{m,n}$  is updated. With  $r = Q_b$ ,  $Q_{m,n}$  is only updated with  $R_{m,n}$  if  $W_{m,n}(R_{m,n})$  read from the RAM is equal to the maximum  $2^{Q_b} - 1$ . When this happens, the output of the *AND* gate (*Max signal 1* in Fig.2.6) is equal to '1'. So, *Max signal 1* enables the update of the registers that store  $Q_{m,n}$  (Fig.2.7). With  $r = Q_b - 1$ ,  $Q_{m,n}$  is updated with  $R_{m,n}$  if  $W_{m,n}(R_{m,n})$  read from the RAM is equal to the maximum  $2^{Q_b} - 1$ , step A4 of Algorithm 9. Additionally, if the current  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  is not blocked,  $Q_{m,n}$  is updated with the GF symbol corresponding to the maximum value of  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$ . To implement this, some extra hardware is added: the box *Extra Hardware* in Fig.2.6, is detailed in Fig.2.8. There are  $M/d_v$  registers which store the maximum values of  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  obtained at the previous iterations. The register corresponding to the current check node is selected with the  $M/d_v$  to 1 multiplexer of  $Q_b$  bits. The output of the RAM (signal *B* in Fig.2.6 and 2.8) for the current check node is compared to the local maximum stored in the corresponding register. If the output of the RAM (signal *B*) is equal to the output of the multiplexer, the output of the adder (signal *A* in Fig.2.6 and 2.8) is stored as the new maximum in the corresponding register and a signal (*Max signal 2*) is activated to indicate that  $Q_{m,n}$  must be updated with  $R_{m,n}$ . So, *Max signal 1* and *Max signal 2* are used as control of the update of the registers that store  $Q_{m,n}$  (Fig.2.8).

Fig.2.7 shows the VNU unit, which includes the VNU cell of Fig.2.6 and the  $M/d_v$  registers of  $p$  bits required to store the  $Q_{m,n}$  elements corresponding to the  $M/d_v$  check nodes computed by a VNU unit. Additionally, a multiplexer is required. As said before, *Max signal 1* and *Max signal 2* control the update of  $Q_{m,n}$  for both blocking with  $r = Q_b$  and  $r = Q_b - 1$ . During the initialization process, the multiplexer of Fig.2.7 selects the  $z_n$  input. After the initialization, if the maximum condition (*Max signal 1* and *Max signal 2*) is detected,  $R_{m,n}$  is selected. In other cases, the shift register configuration is chosen with the multiplexer, ensuring that the values of the registers are not updated. As for clipping, the outputs  $Q_{m,n}$  will be stored in the shift register following the order of the matrix  $\mathbf{H}$  and  $N$  outputs will be hard wired as  $\tilde{c}_n$ .

3. **Complete architecture:** For both clipping and blocking techniques, the complete architecture is included in Fig.2.9. It requires one CNU unit and  $d_c \times d_v$  VNU units. With  $d_c$  multiplexers the correct  $Q_{m,n}$  from the VNU units are selected as inputs of the CNU unit.

For M-GBFDA with clipping technique the architecture first initializes the RAMs and computes  $Q_{m,n}$  values of the initialization applying the sorting process, so  $q \times (M/d_v) + 3$  clock cycles are spent in this task. After initialization, steps A1 and A2 of the CNU are computed, that takes  $M + 1$  clock cycles. During these clock cycles, step A3 can be also computed by the VNU units adding one extra clock cycle. After computing steps A1, A2 and A3, step A4 and the clipping are applied. As  $d_c \times d_v$  VNU units are working at the same time, only  $q \times (M/d_v) + 3$  clock cycles are needed. So, the total latency of the decoder is  $(M + 2 + q \times (M/d_v) + 3) \times It_{max} + q \times (M/d_v) + 3$  clock cycles.

For M-GBFDA with blocking technique the architecture first initializes the RAMs with the  $\mathbf{W}_{\mathbf{m},\mathbf{n}} = \mathcal{F}(\mathbf{L}_{\mathbf{n}})$  values and the registers with the  $Q_{m,n} = z_n$  values, so  $q \times (M/d_v)$  clock cycles are spent in this task. After initialization and 4 clock cycles of latency, all the VNU units and the CNU unit compute all the steps of Algorithm 9 simultaneously. Completing one iteration takes  $M$  clock cycles. So, the total latency of the decoder is  $(M \times It_{max} + q \times (M/d_v) + 4)$  clock cycles.

Architecture for M-GBFDA with blocking can also be used with saturation. Only the blocking register, the  $M/d_v$  to 1 multiplexer and the AND gate from Fig.2.6 must be eliminated. The extra hardware in Fig.2.8 is required to look for the maximum, which can vary from  $2^r - 1$  to  $2^{Q_b} - 1$ .

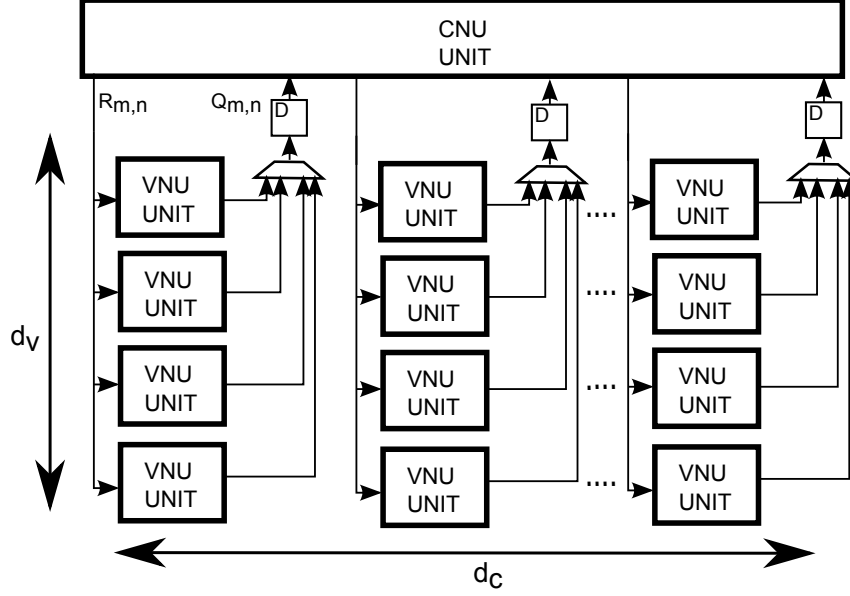


Figure 2.9: Complete message passing architecture for M-GBFDA.

#### Area and timing complexity for the (837,723) NB-LDPC code

Next, hardware results of the previous architectures for a (837,723) NB-LDPC code are detailed. First, analytic methods of area for M-GBFDA with saturation, clipping, blocking with  $r = Q_b = 5$  and  $r = Q_b - 1 = 4$  are included. Next, comparisons between synthesis and post place and route results are performed.

- **Analytic methods:** A theoretical analysis to compute area-complexity of the decoders in terms of *XOR* gates is applied. This method was proposed at [17] and gives a quite accurate approximation of the area required by the architectures. To apply the method first the number of storage, arithmetic and control resources must be estimated, next the amount of *XOR* gates per resource is required. This theoretical method emulates the behavior of ASIC synthesis tools. Applying this method we study: i) the advantage of using M-GBFDA instead of GBFDA, ii) the efficiency of the clipping and the blocking technique with a hardware perspective, and iii) a comparison between the blocking technique and saturation.

If GBFDA is implemented with our proposed architecture and no alternative tentative decoding, such as the one for M-GBFDA is used, the area requirements increase considerably. As was said in Section 2.1.1, we need to store  $\mathbf{W}_n$  if GBFDA is applied, so  $N \times q \times Q_b = 837 \times 31 \times 5 = 129735$  bits of

RAM are required, which is equivalent to an area of 129735 *XOR* gates. In addition, for step A5 and the sorting process of A6 at least  $d_c$  adders are required too, which is equivalent to  $26 \times 15 = 390$  *XOR* gates and  $5N = 4185$  registers are needed for storing the results of the sorting process  $\tilde{c}_n$ , so an increase of  $4185 \times 3 = 12555$  *XOR* gates is produced by this fact. To sum up, at least an increase of equivalent area of  $129735 + 390 + 12555 = 142680$  *XOR* gates, which is a  $142680/639246 \times 100 = 22\%$ <sup>6</sup> of the total area of the decoder, is avoided if M-GBFDA is implemented instead GBFDA. So, it can be affirmed that the modified tentative decoding proposed at Algorithm 9, with a cost of less than 0.05dB of performance loss and without penalization in throughput, is an efficient method to reduce area.

In Table 2.2 resources for the M-GBFDA decoder with clipping are included. Its total area in terms of *XOR* gates is 639246 *XOR* gates. In addition, latency for the decoder with clipping is  $(M + 2 + q \times (M/d_v) + 3) \times It_{max} + q \times (M/d_v) + 3 = (126 + 32 \times 31 + 3)20 + 32 \times 31 + 3 = 23415$  clock cycles. With regard to the architectures with blocking, for the decoder with  $r = Q_b$  (Table 2.3) an equivalent area of 633162 *XOR* gates is necessary; for the decoder with  $r = Q_b - 1$  (Table 2.4) the area is 699618 *XOR* gates. Latency in both blocking cases is  $(M \times It_{max} + q \times (M/d_v) + 4) = (124 \times 20 + 32 \times 31 + 4) = 3476$  clock cycles. If we compare the three architectures we can see that between the one based on clipping and the one with blocking and  $r = Q_b$  area is almost the same while the number of clock cycles is  $23415/3476 = 6.7$  times smaller for the blocking architectures. This fact, makes the blocking architecture 6.7 times faster and hence, 6.7 times more efficient. However, the reader should remember that M-GBFDA with  $r = Q_b$  introduces an early error floor, which is not useful for all kind of communication systems. On the other hand, if we compare the architecture with clipping with the one based on blocking and  $r = Q_b - 1$ , we can see that there is an increase of  $(699618 - 639246)/639246 \times 100 = 9\%$  in area. Nevertheless, in this case blocking is still more efficient because the reduction of latency is bigger than the increase of area. In this case the error floor is lower enough to consider the decoder with blocking and  $r = Q_b - 1$  as an interesting high-speed solution.

Previously, we explained that the architecture for M-GBFDA with the blocking technique can be applied for the saturation method if we just eliminate the resources needed to implement blocking. However, at least  $Q_b = 6$  and  $r = 5$  bits are needed for a sub-optimal implementation (with error floor), and  $Q_b = 7$  and  $r = 5$  for full precision. In such cases, although the latency remains the same,  $(803722 - 699618)/699618 \times 100 = 14.8\%$  and  $(920618 - 699618)/699618 \times 100 = 31.6\%$  more area than the one of the blocking architecture with  $r = Q_b$  decoder is required. The main advantage

<sup>6</sup>Compared to the clipping architecture. For details see Table 2.2.



of saturation is that has coding gain in the waterfall region of 0.1dB and 0.18dB, with  $Q_b = 6$  and  $Q_b = 8^7$ , respectively.

- **Synthesis and place and route results:** The decoders proposed in this manuscript were coded in VHDL. Synthesis and place and route of the cores were completed using the standard Cadence tools with a 90nm CMOS technology with nine layers and with a 40nm Virtex VI FPGA device. Table 2.7 shows the implementation results for M-GBFDA message passing architectures with clipping and blocking ( $r = Q_b$  and  $r = Q_b - 1$ ).

First, the implementation results for the 90nm CMOS technology are analyzed. Synthesis results show that the message passing architecture for the clipping technique has a minimum clock period of 2ns, so the throughput is  $(500\text{MHz} \times 837 \times 5\text{bits})/23415 = 89\text{Mbps}$  and a core area of  $12.7\text{mm}^2$ . However, after place and route the core area is  $13\text{mm}^2$  with a logic density of 79% and a frequency of 255MHz, providing a throughput of  $(255\text{MHz} \times 837 \times 5\text{bits})/3476 = 45\text{Mbps}$ , which is 51% less throughput than the one given by the synthesis process. The same happens to the message passing architecture for blocking with  $r = Q$  and  $r = Q_b - 1$ , the throughput obtained after synthesis (859Mbps and 633Mbps, respectively) doubles the one obtained after place and route (414Mbps and 316Mbps). So, it is important to remark that synthesis results are not valid for this kind of decoders in which around the 52% of the delay is due to the routing, not to the logic. On the other hand, according to the gate-account reports, the total amount of memory resources (507330 memory bits) and the ratio logic/memory (around 29%) are close to the analytic results, which validates this kind of approximations for area estimations.

Next, the implementation results for a 40nm Virtex VI FPGA device are analyzed. The proposed message passing architecture for M-GBFDA with blocking and  $r = Q_b$  requires 2904 slices and achieves a frequency of 227MHz, which is equivalent to a throughput of  $(227\text{MHz} \times 837 \times 5\text{bits}) / (3476) = 273\text{Mbps}$ . On the other hand, the proposed message passing architecture for M-GBFDA with blocking and  $r = Q_b - 1$  has a cost of 7694 slices and achieves a frequency of 200MHz, so a throughput of  $(200\text{MHz} \times 837 \times 5\text{bits}) / (3475) = 240\text{Mbps}$  is reached. The difference in terms of slices between the decoder with blocking and  $r = Q_b$  and  $r = Q_b - 1$  is about 62%. This could seem incoherent compared to the estimated and ASIC results in which the difference of area is 4%. However, if we look carefully at Table 2.7 (slices, RAM components, registers and shift registers) we can check that the results for FPGA satisfy the estimations. As said in Table 2.3 and 2.4, the memory resources for both decoders must be the same, and this is what happens in the FPGA implementations, where the required RAMB18E1 and

<sup>7</sup>Remember that saturation with full precision,  $Q_b = 8$ , has the same performance as clipping,  $Q_b = 5$ .

RAMB36E1 components are the same. In Table 2.3 and 2.4 the number of GF adders, multipliers and 4 to 1 multiplexors is the same, however, the number of adders is 3 times higher for the decoder with  $r = Q_b - 1$  and there are 16 times more 2 to 1 multiplexors, so the increase of 3.4 times more LUTs in the FPGA decoder implementation with blocking and  $r = Q_b - 1$  is justified. Moreover, if we compute the number of registers of the FPGA implementations taking into account the shift registers, with 31 registers each one, we obtain  $4758 + (520 \times 31) = 20858$  registers for the decoder based on blocking with  $r = Q_b$  and  $22121 + (520 \times 31) = 38241$  registers for the decoder with  $r = Q_b - 1$ , what satisfies the estimation of 19739 registers and 35859 registers, respectively. This reasoning can be extended to the clipping architecture. To sum up, the FPGA implementation results are coherent with the estimated results, because do not only the number of slices has to be taken into account but also the number of memories and registers, which is a high percentage of the total area. Finally, it is important to remark that the limited frequency in FPGA devices is not only due to the routing but mainly to the limitations of this kind of devices which have predefined wiring between cell and do not allow as much flexibility as ASIC.

We can conclude that throughput of message passing architectures for M-GBFDA with clipping or blocking is limited by routing congestion. This problem cannot be appreciate with synthesis tools for ASIC, as they do not include wiring models. On the other hand, blocking technique achieves at least 5.4 times higher speed on an FPGA device than the clipping one and at least 7 times on ASIC, with a similar area. The reasonably good performance of the blocking technique makes this solution more efficient than the clipping one, if high-speed architecture is looked for.

#### ***Half-broadcasting architecture for M-GBFDA with blocking***

A broadcasting technique was proposed in [22] for binary LDPC architectures to reduce routing problems. These problems are due to the excessive wiring between CNU units and VNU units, specially in fully-parallel and partial parallel architectures. As algorithms based on GBFDA just exchange scalar messages between CNU units and VNU units, such as in the binary case, we propose to apply the same methods to try to mitigate the problem in NB-LDPC architectures. As it was briefly discussed in the previous chapter, other algorithms such as EMS and Min-max can aggravate this routing problem due to the exchange of vectors with  $q$  (or  $n_m$ ) reliability values between CNU unit and VNU units.

Two types of broadcasting can be distinguished: full-broadcasting and half- broadcasting. On the first hand, full-broadcasting consists in sending only one message from the CNU units to the VNU units and also one message from the VNU units to the CNU units. This broadcasting method can be only applied to full parallel

**Table 2.2:** Theoretical estimation of area-complexity for M-GBFDA message passing architecture with clipping and  $Q_b = 5$  bits,  $r = 5$  bits (639246 XOR gates)

Processor	RAM (1bit)	Reg (1bit)	GF Mult	GF Adder	Add/Sub (5 bits)	ROM (1 bit)	Mux 2-1 (5 bits)
1 VNU cell Fig.2.4	$\frac{M}{d_v} \times q \times Q_b = 4960$	$2Q_b = 10$	0	0	2	$q(\frac{M}{d_v}) = 155$	3
1 VNU unit Fig.2.5	0	$(\frac{M}{d_v} \times p) = 155$	0	0	0	0	$\frac{M}{d_v} - 1 = 30$
VNU Total	$(d_c \times d_v)4960 = 515840$	$(d_c \times d_v)165 = 17160$	0	0	$(d_c \times d_v)2 = 208$	$(d_c \times d_v)155 = 16120$	$(d_c \times d_v)33 = 3432$
CNU Total	0	$(d_c + 1)p = 135$	$2d_c = 52$	$2d_c - 1 = 51$	0	$2d_c \times Q_b \times M = 32240$	0
Total	515840	17295	52	51	208	48360	3432
XOR/item	1	3	43	5	15	1	5
Total (XORs)	515840	51885	2236	255	3120	48360	17160+390

**Table 2.3:** Theoretical estimation of area-complexity for M-GBFDA message passing architecture with blocking and  $Q_b = 5$  bits,  $r = 5$  bits (633162 XOR gates)

Processor	RAM(1 bit)	Reg (1bit)	GF Mult.	GF Adder	Add +1 (5 bits)	ROM (1 bit)	Mux 2/4-1 (5 bits)
1 VNU cell (Fig.2.6)	$\frac{M}{d_v} \times p \times Q_b = 4960$	$\frac{M}{d_v} = 31$	0	0	1	$\lceil \log_2(\frac{M}{d_v}) \rceil \frac{M}{d_v} = 155$	2/0
1 VNU unit (Fig.2.7)	0	$\frac{M}{d_v} \times p = 155$	0	0	0	0	0/1
VNU total	$(d_c \times d_v)4960 = 515840$	$(d_c \times d_v)186 = 19344$	0	0	$(d_c \times d_v) = 104$	$(d_c \times d_v)155 = 16120$	$(d_c \times d_v)2 / (d_c + 1) \times d_v$
CNU total	0	$(3 \times d_c + 1)q = 395$	$2d_c = 52$	$2d_c - 1 = 51$	0	$2(d_c \times p \times M) = 32240$	0
Total	515840	19739	52	51	104	48360	208/130
XOR/item	1	3	43	5	11	1	5/15
Total (XORs)	515840	59217	2236	255	1144	48360	1040/1950

**Table 2.4:** Theoretical estimation of area-complexity for M-GBFDA message passing architecture with blocking and  $Q_b = 5$  bits,  $r = 4$  bits (699618 *XOR* gates)

Processor	RAM(1 bit)	Reg (1bit)	GF Mult.	GF Adder	Adder (5 bits)	ROM (1 bit)	Mux 2/4-1 (5 bits)
1 VNU cell (Fig.2.6 and 2.8)	$\frac{M}{d_v} \times q \times Q_b = 4960$	$\frac{M}{d_v} + \frac{M}{d_v} \times Q_b = 186$	0	0	2	$\lceil \log_2(\frac{M}{d_v}) \rceil \frac{M}{d_v} = 155$	33/0
1 VNU unit (Fig.2.7)	0	$\frac{M}{d_v} \times p = 155$	0	0	0	0	0/1
VNU total	$(d_c \times d_v)4960 = 515840$	$(d_c \times d_v)341 = 35464$	0	0	$2d_c + 2d_v = 208$	$(d_c \times d_v)155 = 16120$	$(d_c \times d_v)33 / (d_c + 1) \times d_v$
CNU total	0	$(3d_c + 1)p = 395$	$2d_c = 52$	$2d_c - 1 = 51$	0	$2(d_c \times p \times M) = 32240$	0
Total	515840	35859	52	51	208	48360	3432/130
<i>XOR</i> /item	1	3	43	5	15	1	5/15
Total ( <i>XORs</i> )	515840	107577	2236	255	3120	48360	17160/1950

**Table 2.5:** Theoretical estimation of area-complexity for M-GBFDA message passing architecture with saturation and  $Q_b = 6$  bits,  $r = 5$  bits (803722 *XOR* gates)

Processor	RAM(1 bit)	Reg (1bit)	GF Mult.	GF Adder	Adder (6 bits)	ROM (1 bit)	Mux 2/4-1 (5 bits)
1 VNU cell	$\frac{M}{d_v} \times q \times Q_b = 5952$	$\frac{M}{d_v} \times Q_b = 186$	0	0	2	$\lceil \log_2(\frac{M}{d_v}) \rceil \frac{M}{d_v} = 155$	33/0
1 VNU unit	0	$(\frac{M}{d_v} \times p) = 155$	0	0	0	0	0/1
VNU total	$(d_c \times d_v)5952 = 619008$	$(d_c \times d_v)341 = 35464$	0	0	$2d_c \times 2d_v = 208$	$(d_c \times d_v)155 = 16120$	$(d_c \times d_v)33 / (d_c \times d_v) + d_c$
CNU total	0	$(3d_c + 1)p = 395$	$2d_c = 52$	$2d_c - 1 = 51$	0	$2(d_c \times p \times M) = 32240$	0
Total	619008	35859	52	51	208	48360	3432/130
<i>XOR</i> /item	1	3	43	5	18	1	6/15
Total ( <i>XORs</i> )	619008	107577	2236	255	3744	48360	20592/1950

**Table 2.6:** Theoretical estimation of area-complexity for M-GBFDA message passing architecture with saturation and  $Q_b = 7$  bits,  $r = 5$  bits (920618 XOR gates)

Processor	RAM(1 bit)	Reg (1bit)	GF Mult.	GF Adder	Adder (7 bits)	ROM (1 bit)	Mux 2/4-1 (5 bits)
1 VNU cell	$\frac{M}{d_v} \times q \times Q_b =$ 6944	$\frac{M}{d_v} \times Q_b =$ 217	0	0	2	$\lceil \log_2(\frac{M}{d_v}) \rceil \frac{M}{d_v} =$ 155	33/0
1 VNU unit	0	$(\frac{M}{d_v} \times p) =$ 155	0	0	0	0	0/1
VNU total	$(d_c \times d_v)6944 =$ 722176	$(d_c \times d_v)372 =$ 38688	0	0	$2d_c \times 2d_v =$ 208	$(d_c \times d_v)155 =$ 16120	$(d_c \times d_v)33$ $(d_c \times d_v) + d_c$
CNU total	0	$(3d_c + 1)p =$ 395	$2d_c =$ 52	$2d_c - 1 =$ 51	0	$2(d_c \times p \times M) =$ 32240	0
Total	722176	39083	52	51	208	48360	3432/130
XOR/item	1	3	43	5	21	1	7/15
Total (XORs)	722176	117249	2236	255	4368	48360	24024/1950

**Table 2.7:** Implementation results for Virtex VI FPGA device and 90nm process ASIC of the decoder architecture for M-GBFDA with clipping and blocking

Algorithm	FPGA Area Synthesis	FPGA Thr./Freq. Synthesis	ASIC Area Synthesis	ASIC Thr./Freq. Synthesis	ASIC Area Place & Route	ASIC Throughput / Place & Route
Clipping	16884 reg. 11288 LUTs 105 RAMB18E1 7 RAMB36E1 4070 slices	44.6Mbps / 250MHz	12.7mm <sup>2</sup>	89Mbps / 500MHz	13mm <sup>2</sup>	45Mbps / 255MHz
Blocking ( $r = Q_b$ )	4758 reg. 7148 LUTs 520 shift reg. 105 RAMB18E1 7 RAMB36E1 2904 slices	273Mbps / 227MHz	12.5mm <sup>2</sup>	859Mbps / 714MHz	12.7mm <sup>2</sup>	414Mbps / 344MHz
Blocking ( $r = Q_b - 1$ )	22121 reg. 24673 LUTs 520 shift reg. 105 RAMB18E1 7 RAMB36E1 7694 slices	240Mbps / 200MHz	13mm <sup>2</sup>	633Mbps / 526MHz	13.28mm <sup>2</sup>	316Mbps / 263MHz

architectures, which is not our case. On the second hand, half-broadcasting consists in sending only one message from the CNU unit to the VNU units, but more messages are sent from the VNU units to the CNU unit. This method can be applied to partial parallel message passing architectures as the ones that we propose in the previous subsections, as shown in Fig. 2.10, Fig. 2.11 and Fig.2.12. However, with this half-broadcasting technique, the CNU and VNU equation are not implemented in separated processors, so now we cannot talk about CNU unit and VNU unit, but about Partial CNU unit and CNU-VNU unit. The message sent by the Partial CNU unit to the CNU-VNU units is the syndrome (Fig. 2.10), because this processor just computes step A1 from the check node. Each CNU-VNU unit includes a message passing VNU unit, which implements all the VNU steps, and the necessary hardware resources to compute  $R_{m,n}$  (step A2 from the CNU): a GF multiplier, a GF adder and a ROM with the  $h_{m,n}^{-1}$  coefficients (Fig. 2.11). In this way, the routing congestion is reduced because message passing architectures' wiring from the CNU unit to all the VNU units is saved. With this proposal it is not necessary to exchange  $d_c$  different  $R_{m,n}$  values of  $p$  bits to  $d_c \times d_v$  processors, just one syndrome value,  $s_m$ , needs to be broadcast. In addition, part of the wiring from  $Q_{m,n}$  signals is shorted because  $Q_{m,n}$  is used for obtaining  $R_{m,n}$  inside the CNU-VNU unit, where it is calculated. The disadvantage of this technique is that introduces an area overhead, as GF multipliers, GF adders and ROM memories are now in each CNU-VNU unit. These resources are increased  $d_v - 1$  times.

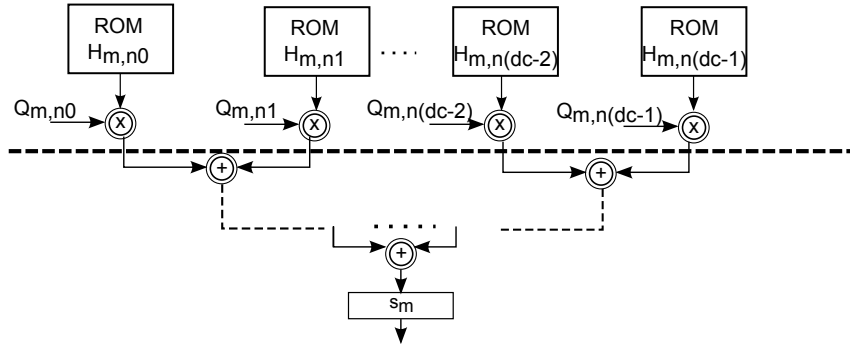


Figure 2.10: Partial CNU unit for M-GBFDA half-broadcasting architecture.

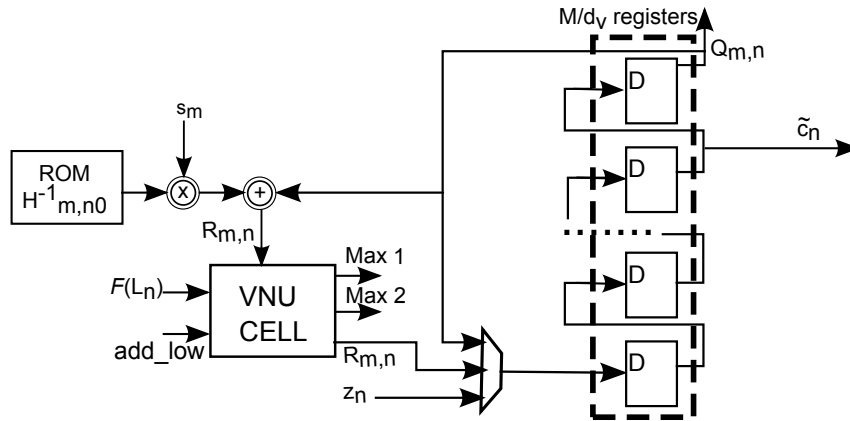


Figure 2.11: CNU-VNU unit for M-GBFDA half-broadcasting architecture.

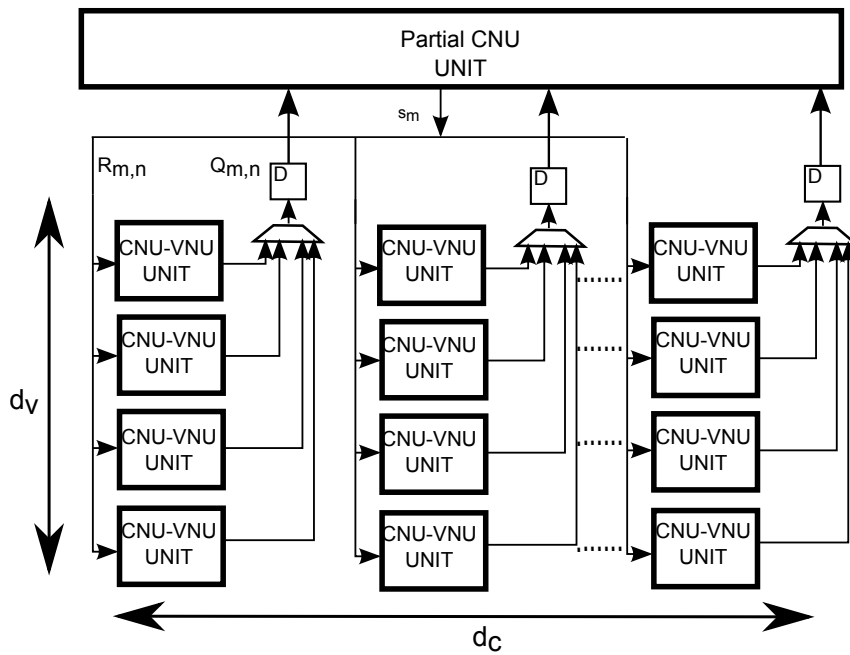


Figure 2.12: Complete half-broadcasting architecture for M-GBFDA.

### Area and timing complexity for the (837,723) NB-LDPC code

Next, hardware results of the half-broadcasting architectures based on the blocking technique for a (837,723) NB-LDPC code are included.

- Analytic methods:** Applying the analytic methods from [17], we can estimate an overhead in area of 52104 *XOR* gates for both half-broadcasting blocking architectures with  $r = Q_b$  and  $r = Q_b - 1$ . This is due to the increase of  $(d_v - 1) \times d_c$  GF multipliers,  $(d_v - 1) \times d_c$  GF adders and  $(d_v - 1) \times d_c M \times p$  ROM memories. Although the latency and the number of gates of the critical path is the same as the message passing architectures, frequency is higher because routing congestion is reduced, as it will be shown next with place and route results.
- Synthesis and place and route results:** Under the same FPGA and ASIC technologies applied to the message passing architectures previously reported, we obtain the results for half-broadcasting architecture included in Table 2.8. The architecture for blocking with  $r = Q_b$  reaches a frequency of 500 MHz with a core area of  $12.8mm^2$ . So, it provides a throughput of 601Mbps, which is 45% higher than the one achieved with the message passing architecture, at some extra area expense. The second one, blocking with  $r = Q_b - 1$ , reaches a frequency of 476 MHz with a core area of  $13.4mm^2$ . So, it provides a throughput of 573Mbps, which is 81% higher than the one achieved with the message passing architecture. We can affirm that, although there are some routing effects remaining, half-broadcasting technique mitigates considerably the effect for ASIC implementation increasing throughput. As an example the total wire length is reduced from 52163452  $\mu m$  for the conventional message passing architecture of the blocking algorithm to 44348801  $\mu m$  for the half-broadcasting one. Unfortunately, the effect of half-broadcasting on FPGA devices is reduced, throughput just increases 10% and 11% for blocking with  $r = Q_b$  and  $r = Q_b - 1$  respectively, at the cost of a similar increase of area.

#### 2.1.6 Conclusions of the architectures for the parallel schedule of M-GBFDA

In this subsection of the chapter we have proposed two different methods to reduce complexity of GBFDA with flooding schedule. In addition, some architectures have been proposed with the objective of reaching high speed with high-rate NB-LDPC codes. The conclusions of this part of the chapter are:

1. A simplified tentative decoding is proposed for GBFDA. The algorithm with this new tentative decoding is named M-GBFDA.



**Table 2.8:** Implementation results for Virtex VI FPGA device and 90nm process ASIC of the decoder half-broadcasting architecture for M-GBFDA with blocking

Algorithm	FPGA Area Synthesis	FPGA Thr./Freq. Synthesis	ASIC Area Synthesis	ASIC Thr./Freq. Synthesis	ASIC Area Place & Route	ASIC Throughput / Place & Route
Blocking ( $r = Q_b$ )	4888 reg. 7918 LUTs 520 shift reg. 105 RAMB18E1 7 RAMB36E1 3078 slices	300Mbps / 250MHz	$12.8mm^2$	859Mbps / 714MHz	$12.8mm^2$	601Mbps / 500MHz
Blocking ( $r = Q_b - 1$ )	21632 reg. 29965 LUTs 520 shift reg. 105 RAMB18E1 7 RAMB36E1 9592 slices	267Mbps / 222MHz	$13.2mm^2$	633Mbps / 526MHz	$13.4mm^2$	573Mbps / 476MHz

2. M-GBFDA has a negligible performance loss compared to GBFDA with 20 iterations. With less than 10 iterations, the difference in coding gain is considerable because the algorithm has not converged yet in most of the cases. Both GBFDA and M-GBFDA have less coding gain than EMS and Min-max algorithms.
3. Architectures derived from M-GBFDA have an important reduction in hardware resources compared to GBFDA, so this algorithm saves area.
4. To limit data growth, techniques such as saturation, clipping or the one proposed here, blocking, must be applied.
5. Saturation has an important increase of area if no early error floor is desired.
6. Clipping requires a large latency in partial parallel architectures for flooding schedule, that makes the decoder very slow. Clipping is the only technique to control data growth that does not introduce performance degradation.
7. Blocking allows decoders to have both small area and low latency, at a cost of an error floor. Error floor with  $r = Q_b - 1$  is lower enough to be suitable for several communication systems. Throughput of the architectures with blocking are at least 7 times faster than the ones based on clipping.
8. Partial parallel message passing architectures for M-GBFDA suffer the same routing problems as binary LDPC decoders.
9. Techniques to reduce routing congestion can increase speed up to 81% mitigating the wiring. The fastest partial parallel architecture for the (837,726)

NB-LDPC code, introduced in this subsection, with a FER performance applicable to most communications systems, is the one based on blocking with  $r = Q_b - 1$  and half-broadcasting, which reaches 575Mbps.

## 2.2 Serial schedule algorithms and architectures

Serial schedule reduces the number of iterations required by LDPC decoders to reach a certain performance, compared to flooding schedule. This makes serial schedule interesting if throughput is the priority. In addition, the fact of updating progressively check nodes and variable nodes, allow us to derive architectures with higher degree of efficiency, without including simplifications that introduce performance degradation, just taking advantage of some special conditions.

In this section, we propose two algorithms based on the serial schedule of GBFDA. The first one does a more efficient division of variable node information and adds some extra complexity that improves performance compared to the flooding schedule of GBFDA. The second proposal simplifies the first one making use of some statistical information. Additionally, a different initialization of the algorithm avoids using techniques to control data growth without any performance degradation. Both algorithms give as result two efficient partial parallel message passing architectures with high-throughput and lower area than the solutions of the previous section.

### 2.2.1 Enhanced Serial GBFDA

In Algorithm 10, the proposed Enhanced Serial GBFDA (ES-GBFDA) is described. In the same way as in the original GBFDA, we can distinguish two main steps in the iterative process: CNU, which computes steps A2 and A3, and VNU, which calculates steps A1, A4 and A5. CNU steps calculate the symbols with higher probability to satisfy the check node equations ( $R_n$ ) according to the deviation of the syndrome,  $s$ . VNU steps increase the reliability values (stored in  $\mathbf{W}_n$ ) associated to the candidates ( $R_n$ ) of the CNU step by means of a voting process.

As the description of the algorithm is serial, each check node is updated sequentially. So, after updating one check node,  $m$ , all the variable nodes connected to  $m$  are updated. For this reason a loop  $m$  is included. One iteration is ended when all check nodes have been updated (and hence all the variable nodes). This sequential update avoids storing syndromes for each one of the check nodes, because after the update of one check node the information is not necessary for the following one. Note that with this schedule we do not update first all the CNU and after that all the VNU, unlike flooding schedule algorithms such as M-GBFDA in the previous

section. Moreover, we do not need to store  $Q_{m,n}$  symbols as we can compute them performing the subtraction in step A1 and just storing  $Q_n$ .

Other advantages of the algorithm, which are not due to the serial schedule, are that votes are stored separately in  $\mathbf{W}_{m,n}$ , which will allow us to quantize the extrinsic information with lower number of bits. In addition, ES-GBFDA accumulates in step A4 the votes given to the symbols at previous iterations (from 1 to  $j - 1$ ), so,  $\mathbf{W}_{\mathbf{n}}^{(j-1)} - \mathbf{W}_{m,n}^{(j-1)}$  in A1 represents the extrinsic information obtained with the information from all the previous iterations. This last modification of the GBFDA not only reduces the number of iterations due to the serial description, but also improves the performance in the waterfall region because of using the intrinsic information from all the iterations to compute the extrinsic messages. Next, we give a more detailed explanation of the algorithm.

---

**Algorithm 10** Enhanced Serial GBFDA

---

**Input:**  $\mathbf{W}_{\mathbf{n}}^{(0)} = \mathcal{F}(\mathbf{L}_{\mathbf{n}})$ ,  $\mathbf{W}_{m,n}^{(0)} = 0$   
Iterative process  
**for**  $j = 1 \rightarrow It_{max}$  **do**  
  **for**  $m = 1 \rightarrow M$  **do**  
    Variable node processing:  
    A1 :  $Q_n^{(j)} = GFmax(\mathbf{W}_{\mathbf{n}}^{(j-1)} - \mathbf{W}_{m,n}^{(j-1)})$   
    Check node processing  
    A2 :  $s = \sum_{n \in N(m)} h_{m,n} Q_n^{(j)}$   
    A3 :  $R_n^{(j)} = h_{m,n}^{-1} s - Q_n^{(j)}$   
    Variable node processing  
    A4 :  $\mathbf{W}_{m,n}^{(j)}(R_n) = \mathbf{W}_{m,n}^{(j-1)}(R_n) + 1$   
    A5 :  $\mathbf{W}_{\mathbf{n}}^{(j)}(R_n) = \mathbf{W}_{\mathbf{n}}^{(j-1)}(R_n) + 1$   
    Tentatively decoding  
    A6 :  $\tilde{c}_n^{(j)} = GFmax(\mathbf{W}_{\mathbf{n}}^{(j)})$   
    **if**  $\tilde{\mathbf{c}} \times \mathbf{H}^T = 0$  **then**  
      SKIP  
    **end if**  
  **end for**  
**end for**  
**Output:**  $\tilde{\mathbf{c}}$

---

During the initialization,  $\mathbf{W}_{\mathbf{n}}$  is equal to the reliability values ( $\mathcal{F}(\mathbf{L}_{\mathbf{n}})$ ) and  $\mathbf{W}_{m,n}$  is an all-zero matrix. Step A1 sorts the extrinsic information to find the symbol with the maximum reliability, which will be taken as the new hard decision,  $Q_n$ . This extrinsic information is the result of subtracting to the  $\mathbf{W}_{\mathbf{n}}$  the intrinsic information of a check node  $m$ ,  $\mathbf{W}_{m,n}$ .  $\mathbf{W}_{\mathbf{n}}$  contains the channel information (reliability values of the initialization) plus the vote information (all received votes) of each symbol  $n$  in the codeword. Step A2 computes the syndrome  $s$ , which is

required in step A3 to calculate the candidates of the voting process,  $R_n$ . In step A4,  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  counts the votes (intrinsic information) that receive each symbol  $R_n$  during all the iterations. When the  $m$  loop ends,  $\mathbf{W}_{\mathbf{n}}$  is updated in step A5. The votes modify the values of  $\mathbf{W}_{\mathbf{n}}$  and  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$ , changing the result of the sorting process in step A1 and, hence, flipping  $Q_n$  symbols. Once the iterative process ends, step A6 performs the tentative decoding by finding the symbols associated to the maximum values of  $\mathbf{W}_{\mathbf{n}}$ . It is important to remark that ES-GBFDA does not include any modification in the tentative decoding, but requires a method to control the data growth of  $\mathbf{W}_{\mathbf{n}}$  and  $\mathbf{W}_{\mathbf{m},\mathbf{n}}$  or an alternative to avoid it. In the following subsection a discussion of an alternative initialization to avoid clipping is exposed.

### 2.2.2 Alternative voting process for serial schedule of GBFDA

With the aim of increasing the coding gain of ES-GBFDA, a version of the algorithm in which the amplitude of the vote depends on the channel information was also proposed. This algorithm, called serial symbol-reliability based (SSRB) algorithm, modifies steps A4 and A5 by  $\mathbf{W}_{\mathbf{m},\mathbf{n}}^{(j)}(R_n) = \mathbf{W}_{\mathbf{m},\mathbf{n}}^{(j-1)}(R_n) + \mathbf{L}_{\mathbf{n}}(R_n) \times \xi$  and  $\mathbf{W}_{\mathbf{n}}^{(j)}(R_n) = \mathbf{W}_{\mathbf{n}}^{(j-1)}(R_n) + \mathbf{L}_{\mathbf{n}}(R_n) \times \xi$ , respectively.

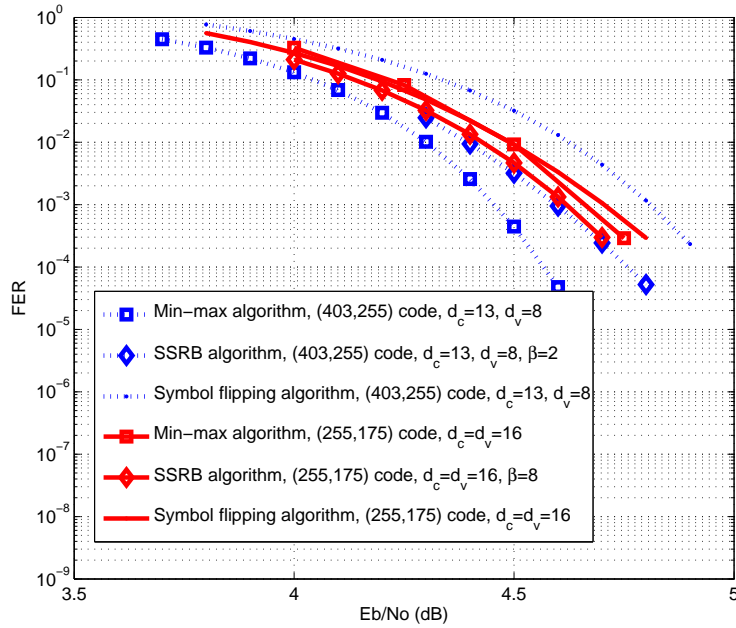
The constant  $\xi$  is computed via simulation. This constant speeds up the convergence of the algorithm and improves its error-correction performance because adjusts the importance of each vote. Its optimum value depends on  $d_v$  and  $q$ , and it is selected as a power of two to avoid products. A basic guideline for the choice of  $\xi$  is as follows: in general, for a given Galois field, a code with high  $d_v$  will require a lower value of  $\xi$  than a code with low  $d_v$ . If  $\xi$  is too large with a high  $d_v$ , an erroneous symbol flip becomes more probable, because information from the channel is ignored somehow, as we give a higher weight to the votes than to the LLRs. On the other hand, if  $d_v$  is low and  $\xi$  is low, more iterations are required to make a symbol flip because the votes do not have enough amplitude to modify the decisions of the channel information and this would slow down the convergence of the algorithm. For codes over different Galois field (different values of  $q$ ),  $\xi$  must be increased as we increase  $q$ . This is due to the fact that the average distance between LLRs gets larger as the field size grows. The factor  $\xi$  must correspondingly be increased in order to facilitate the flipping of the symbols.

As the vote amplitude in this case is the scaled value of the LLR, more soft information is processed and, hence, higher coding gain is obtained, with lower complexity than Min-max algorithm and the symbol-flipping from [26] and [40].

Table 2.9 compares the number of operations per iteration required by SSRB algorithm and the symbol flipping algorithms from [26] and [40]. It includes the number of GF additions, GF multiplications, regular comparisons and regular additions. In addition, we include the number of iterations required to achieve

similar FER performance (shown in Fig.2.13). The simplified version of Min-max algorithm in [23], which avoids using GF multiplications and performs the pre-processing of the GF additions, is also included in Table 2.9. As the parameter  $\xi$  is a power of 2, multiplications are not required. For comparison purposes, we measure the complexity as the total number of operations: (operations/iteration)  $\times$  (iterations).

As can be seen, for the (403,255) NB-LDPC code [1] the proposed algorithm is  $(428792 \times 20)/(125736 \times 10) = 6.8$  times less complex than the algorithm from [26], and  $(468999 \times 20)/(125736 \times 10) = 7.4$  times less complex than the one in [40]. Both algorithms from [26] and [40] have the same coding gain. Compared to Min-max algorithm, SSRB algorithm is  $(167608320 \times 7)/(125736 \times 10) = 93$  times less complex with a performance loss of 0.2dB. For the (255,175) NB-LDPC code, based on Euclidean geometry, there is a reduction of complexity of a factor of  $(2935280 \times 20)/(736400 \times 10) = 7.9$ ,  $(4389445 \times 20)/(736400 \times 10) = 11.9$  and  $(963379200 \times 10)/(736400 \times 10) = 1308$  times, compared to the algorithms from [26], [40] and [23], respectively, with almost the same coding gain.



**Figure 2.13:** FER performance of Min-max, symbol flipping and SSRB algorithms, for a (403,255) NB-LDPC code over GF(32) and a (255,175) NB-LDPC code over GF(256) with a BPSK modulation in an AWGN channel.

**Table 2.9:** Complexity of simplified Min-max, SSRB and two symbol-flipping algorithms

Algorithm	GF Add.	GF Mult.	Regular Addition	Regular comparisons	Operat./It.	It. Max.
SSRB	$2d_c \times M$	$3d_c \times M$	$2d_c \times M$	$q \times d_c \times M$		
(255,175)	5600	8400	5600	716800	736400	10
(403,225)	6448	9672	6448	103168	125736	10
[26]	$q \times d_c \times 4M$	0	$d_c \times M$	$q \times N$		
(255,175)	2867200	0	2800	65280	2935280	20
(403,225)	412672	0	3324	12896	428792	20
[40]	$(3d_c - 2) \times M$	$3d_c \times M$	$d_v \times N \times (2 + 2q) + (q - 1) \times N$	$2(q - 1)(d_v + 1) \times N$		
(255,175)	8050	8400	2162145	22108050	4389445	20
(403,225)	9176	9672	225277	224874	468999	20
Min-max [23]	Pre-processed	0	0	$(d_c - 2) \times (4q^2) \times M$		
(255,175)	-	0	0	963379200	963379200	10
(403,225)	-	0	0	16760832	16760832	7

As can be seen in Fig.2.13, for some codes SSRB algorithm has similar coding gain to Min-max. Unfortunately, this proposal has some drawbacks: i) it only works with  $d_v > 5$  codes and ii) a large number of bits is required to quantize the vote amplitude in order to get a performance similar to Min-max algorithm. For these reasons, this contribution is interesting for floating point digital signal processors (DSP) but not for ASIC or FPGA implementations, which is the perspective of this manuscript. For more results and extra details we refer to [43].

### 2.2.3 Alternative initialization for serial schedule of GBFDA

As we showed in the previous section, implementation of clipping technique requires a large number of clock cycles which reduces the throughput of the decoder, turning it useless for high speed applications. On the other hand, we can advance that implementations of clipping with higher degree of parallelism reduce the critical path and increase area. Moreover, other techniques as blocking, introduce error degradation in the performance, such as error floor, which requires from other concatenated decoder to correct it [41]. This last issue makes the solution sub-optimal and it can be a problem if we look for an standalone FEC decoder. In this subsection we propose to modify the way in which we represent channel information for the GBFDA serial schedule, instead of modifying the technique to control data growth.

First, let us define  $r$  as the number of bits that are required to compute the  $\mathcal{F}(\mathbf{L}_n)$  with  $L_n(x) = \log \left[ \frac{P(c_n=x | \mathbf{y}_n)}{P(c_n=0 | \mathbf{y}_n)} \right]$ . The value of  $\mathcal{F}(\mathbf{L}_n)$  for the most reliable symbol can vary between  $2^{r-1} - 1$  and 0 (when the symbol zero is the most reliable). An offset  $\gamma_n = 2^{r-1} - 1 - \max(\mathcal{F}(\mathbf{L}_n))$  is added in order to fix the value in  $\mathcal{F}(\mathbf{L}_n)$

for the most reliable symbol to the maximum amplitude  $2^{r-1} - 1$  for each  $n$ , as proposed for the MD algorithm at [42]. The absolute minimum of  $\mathcal{F}(\mathbf{L}_n)$  is  $-2^{r-1}$  (for the less reliable symbol).

On the other hand, [12] proposes to keep just  $n_m$  values in  $\mathbf{L}_n$  (with  $n_m < q$ ) to save memory in EMS. For GBFDA, a similar procedure is applied, as commented in Section 2.1.2, with a negligible performance loss: the values of  $\mathcal{F}(\mathbf{L}_n)$  between 0 and  $-2^{r-1}$ , which corresponds to the less reliable symbols, are fixed to 0. In this way,  $n_m$  is taken as the number of values between  $2^{r-1} - 1$  and 0 (so  $n_m$  is dynamical), and  $q - n_m$  unreliable values are forced to 0. So, the values in  $\mathcal{F}(\mathbf{L}_n)$  are positive (between  $2^{r-1} - 1$  and 0), and the number of bits required to quantize them is reduced by one (from  $r$  to  $r - 1$ ) due to the fact that the sign bit is not used.

Additionally, the data saved in  $\mathbf{W}_n$ , which is initialized to  $\mathcal{F}(\mathbf{L}_n)$ , increase with each vote (step A5). Being  $Q_b$  the number of bits used to quantize  $\mathbf{W}_n$ . Remember that  $Q_b$  would be bigger than  $r - 1$  if a technique to control the data growth was not applied as happened with the flooding schedule. In Section 2.1.5 was shown that a big percentage of hardware resources are memories whose size depends on  $Q_b$ , so the increase of  $Q_b$  has a great impact on the total area.

One possible solution to data growth problem was the clipping technique. This technique detects when a vote causes that  $\max(\mathbf{W}_n) > 2^{r-1} - 1$ . In this case, it subtracts one vote to all the reliability values saved in  $\mathbf{W}_n$ . After this subtraction: i) the maximum value in  $\mathbf{W}_n$  cannot be bigger than  $2^{r-1} - 1$ ; and ii) some values are -1 (those whose value was 0 before the subtraction). To avoid the need of a bit of sign, values lower than zero are fixed to zero. This fact makes that we lose information of some unreliable symbols each time clipping is applied, because we take as zero both negative and zero  $\mathbf{W}_n$  values. In this way, clipping is ignoring a certain number of unreliable symbols, without introducing any error degradation. Although this technique keeps the number of bits required for  $\mathbf{W}_n$  in  $Q_b = r - 1$ , it has several hardware drawbacks.

To avoid this kind of techniques and not increase the number of bits, we define  $\mathcal{F}(\mathbf{L}_n)$  normalized to the most reliable symbol,  $L_n(x) = \log \left[ \frac{P(c_n=x | \mathbf{y}_n)}{P(c_n=z_n | \mathbf{y}_n)} \right]$ . Now the reliability values of  $\mathcal{F}(\mathbf{L}_n)$  are between 0, for the most reliable symbol, and  $-2^r$ , for the less reliable symbol. Applying the idea of the existence of just  $n_m < q$  useful reliability values, we fix the values between  $-2^r$  and  $-2^{r-1}$  to  $-2^{r-1}$ , which is the same as fixing values between 0 and  $-2^{r-1}$  to 0 in the previous initialization. Moreover, as said before, clipping modifies (sets to 0) the information saved in  $\mathbf{W}_n$  for some unreliable symbols (those whose values are zero before the subtraction). Taking this into account, we studied the effect of modifying (setting to the unreliable value) the values saved in  $\mathcal{F}(\mathbf{L}_n)$  for some unreliable symbols. Our simulations showed that the  $\mathcal{F}(\mathbf{L}_n)$  values which are between  $-2^{r-1}$  and  $-2^{r-2}$

can be fixed to  $-2^{r-2}$  without any degradation in the performance with respect to applying clipping (see Subsection 2.2.5). In this way,  $\mathcal{F}(\mathbf{L}_n)$  values are between 0 for the most reliable symbol and  $-2^{r-2}$  for the less reliable symbol. Additionally,  $\mathbf{W}_n$  is quantized to  $Q_b = r - 1$  bits, so its minimum reliability value is  $-2^{r-2}$  and its maximum reliability value is  $2^{r-2} - 1$ . As  $\mathbf{W}_n$  is initialized to  $\mathcal{F}(\mathbf{L}_n)$ , the most reliable symbol in  $\mathbf{W}_n$  can grow from 0 to  $2^{r-2} - 1$ , that is, it can receive  $2^{r-2} - 1$  votes in step A5. If it receives more votes (the maximum value  $2^{r-2} - 1$  is reached), saturation is applied not performing step A5.

### 2.2.4 Simplified ES-GBFDA

In this subsection a modification of the ES-GBFDA is presented. Its purpose is to reduce the amount of information that we need to store. This is achieved by simplifying the information related with the number of votes. In ES-GBFDA, we accumulate the votes from all the previous iterations in  $\mathbf{W}_{m,n}$ . With the proposed simplification, we reduce the information of all the number of votes stored in ES-GBFDA to a constant value obtained by simulation to avoid a big increase of the maximum value of  $\mathbf{W}_{m,n}$  and  $\mathbf{W}_n$ .

The proposed modification is based on the analysis of the values of the votes through the iterations and the number of symbols that are voted in two consecutive iterations in Algorithm 10. Next, we include one example of analysis, particularized for the (837,723) NB-LDPC code. We used a BPSK modulation over an AWGN channel with 10000 test frames.

First, we studied  $\mathbf{W}_{m,n}$  in ES-GBFDA, for several  $E_b/N_0$ , to obtain the percentage of symbols that received from 2 to 8 votes during the different iterations up to the algorithm converges. Note that symbols which only received one vote were omitted because this unique vote can be considered as noise, they were voted erroneously if they are not voted again in the following iterations. Table 2.10 and 2.11 show the results of the analysis of  $\mathbf{W}_{m,n}$  for  $E_b/N_0=4.6\text{dB}$  and  $E_b/N_0=5\text{dB}$ , respectively. As can be seen in Table 2.10 and 2.11, a big percentage of voted symbols, which increases with  $E_b/N_0$ , received two votes (around 50% and 60% for  $E_b/N_0=4.6\text{dB}$  and  $E_b/N_0=5\text{dB}$ , respectively, for more than 4 iterations). So, if instead of accumulating the votes as in step A4 of ES-GBFDA we suppose that all the voted symbols always receive 2 votes during the previous iterations, we are approximating the behaviour of Algorithm 10. For example, using this approximation for 5 iterations and  $E_b/N_0=4.6\text{dB}$ , 52.2% of symbols will be unmodified and 27.18% will lose the information of just one vote when step A1 is computed (79.38% of the symbols suffer negligible changes). For higher  $E_b/N_0$ , the percentage of symbols that suffers negligible changes is even higher, 91.5% for  $E_b/N_0=5\text{dB}$  and 5 iterations. The same reasoning can be followed for other number of iterations, reaching similar results. Hence, we can conclude that there is a number of received votes,  $\beta$ , which is more common in  $\mathbf{W}_{m,n}$  for the symbols that are voted



more than once and we can approximate the accumulation of votes (step A4 in ES-GBFDA) by  $\beta$ .

Second, we studied the values of the candidates  $R_n$  during the voting process in ES-GBFDA, for several  $E_b/N_0$ . With this analysis we obtain the percentage of symbols that were voted at the iteration  $j - 1$  and are also voted at the current iteration  $j$ . Table 2.12 shows the obtained results for several  $E_b/N_0$ . The percentage of repeated votes is higher than 70% even for low  $E_b/N_0$ , being higher than 90% for  $E_b/N_0=5\text{dB}$  and more than 8 iterations. So, we can approximate the behavior of ES-GBFDA by taking into account only the voting information of iteration  $j - 1$ , especially for high  $E_b/N_0$ .

**Table 2.10:** Percentage of symbols that received  $\beta$  votes in ES-GBFDA for  $E_b/N_0 = 4.6\text{dB}$

$\beta$ / iter.	2	3	4	5	6	7	8	9	10
2	100%	65.4%	56%	52.2%	52.2%	50.2%	47%	47.9%	47%
3	0%	34.6%	27%	27.18%	27.18%	26.7%	23%	24.8%	24.9%
4	0%	0%	13%	12.8%	12.7%	13.3%	14%	13.7%	13.1%
5	0%	0%	4%	7.81%	6.98%	6.5%	9%	7.5%	7.6%
6	0%	0%	0%	0%	0.8%	3.2%	4%	3.4%	3.6%
7	0%	0%	0%	0%	0%	0%	2%	1.7%	1.8%
8	0%	0%	0%	0%	0%	0%	1%	1%	2%

**Table 2.11:** Percentage of symbols that received  $\beta$  votes in ES-GBFDA for  $E_b/N_0 = 5\text{dB}$

$\beta$ / iter.	2	3	4	5	6	7	8	9	10
2	100%	68.2%	62.8%	62.2%	62.2%	62.3%	62.09%	62.03%	62.1%
3	0%	31.78%	29.7%	29.3%	29.3%	29.13%	29.17%	29.11%	28.94%
4	0%	0%	7.37%	6.8%	6.5%	6.5%	6.69%	6.68%	6.6%
5	0%	0%	0%	1.5%	1.5%	1.6%	1.5%	1.5%	1.5%
6	0%	0%	0%	0%	0.4%	0.34%	0.36%	0.4%	0.4%
7	0%	0%	0%	0%	0%	0.024%	0.14%	0.16%	0.18%
8	0%	0%	0%	0%	0%	0%	0%	0%	0%

Based on these ideas, we propose a simplified version of the ES-GBFDA, SES-GBFDA, which is included in Algorithm 11. In this algorithm  $\mathbf{W}_{m,n}$  only indicates which symbols are voted at the current iteration  $j$  (step A4), supposing that they were the same at the previous ones. On the other hand, we introduce the parameter  $\beta$ , which is used in step A1 to indicate the number of votes that we suppose the symbols received at the previous iterations. So,  $\mathbf{W}_{m,n}$  is just a matrix of flags that indicates if  $\beta$  must be subtracted in step A1 or not. This parameter  $\beta$  can be obtained by analyzing the votes (Table 2.10 and 2.11) or with FER simulations, as shown in the next subsection. The complexity of SES-GBFDA is lower than

**Table 2.12:** Percentage of symbols that were voted at the previous iteration and are also voted at the current one in ES-GBFDA

$E_b/N_0$ / iter.	2	3	4	5	6	7	8	9	10
4.2 dB	73%	73%	73%	73%	73%	73%	73%	73%	73%
4.4 dB	73.9%	73.9%	73.9%	73.9%	73.9%	73.9%	73.9%	73.9%	73.9%
4.6 dB	74%	74.5%	74.7%	74.8%	74.7%	74.6%	74.5%	74.4%	74.3%
4.8 dB	72%	73%	75%	76%	77.4%	77.6%	77.6%	77.5%	77%
5 dB	74.2%	74.8%	77.2%	81.2%	86.9%	89.8%	90.9%	91.3%	91.4%

ES-GBFDA because additions are avoided in step A4. Additionally, the number of subtractions in A1 is also reduced: ES-GBFDA needs to compute a maximum of  $d_c \times M \times q$  subtractions by iteration (if all the possible symbols received at least one vote at the previous iterations), in contrast, GBFDA only computes  $d_c \times M$  subtractions because it only has into account the votes received at iteration  $j - 1$ .

---

**Algorithm 11** Simplified ES-GBFDA

---

**Input:**  $\mathbf{W}_n^{(0)} = \mathcal{F}(\mathbf{L}_n)$ ,  $\mathbf{W}_{m,n}^{(0)} = 0$

Iterative process

**for**  $j = 1 \rightarrow It_{max}$  **do**

**for**  $m = 1 \rightarrow M$  **do**

    Variable node processing:

$$A1 : Q_n^{(j)} = GFmax(\mathbf{W}_n^{(j-1)} - \mathbf{W}_{m,n}^{(j-1)} \times \beta)$$

    Check node processing

$$A2 : s = \sum_{n \in N(m)} h_{m,n} Q_n^{(j)}$$

$$A3 : R_n^{(j)} = h_{m,n}^{-1} s - Q_n^{(j)}$$

    Variable node processing

$$A4 : \mathbf{W}_{m,n}^{(j)}(R_n) = 1; \mathbf{W}_{m,n}^{(j)}(x \neq R_n) = 0$$

$$A5 : \mathbf{W}_n^{(j)}(R_n) = \mathbf{W}_n^{(j-1)}(R_n) + 1$$

    Tentatively decoding

$$A6 : \tilde{c}_n^{(j)} = GFmax(\mathbf{W}_n^{(j)})$$

**if**  $\tilde{c} \times \mathbf{H}^T = 0$  **then**

    SKIP

**end if**

**end for**

**end for**

**Output:**  $\tilde{c}$

---

### 2.2.5 Frame error rate performance comparison between NB-LDPC decoders

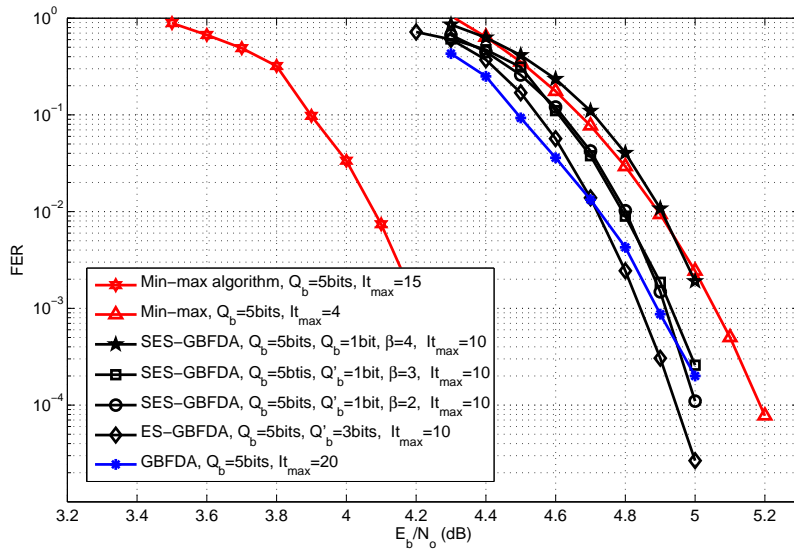
Fig. 2.14 compares the FER performance of the algorithms Min-max, GBFDA, ES-GBFDA and SES-GBFDA for the (837,723) NB-LDPC code. The initialization for ES-GBFDA and SES-GBFDA algorithms is the alternative one proposed on Section 2.2.3, so they do not implement clipping. GBFDA with parallel flooding includes the clipping technique.  $Q'_b$  is defined as the number of bits required to quantize  $\mathbf{W}_{m,n}$  in ES-GBFDA and SES-GBFDA. We used a BPSK modulation in an AWGN channel and encoded a different random message  $\mathbf{m}$  with each transmitted package.

First, a comparison among GBFDA algorithms is made. The serial descriptions of GBFDA, ES-GBFDA and SES-GBFDA, reduce by half the number of iterations required to converge, from 20 iterations for the flooding schedule to 10 iterations for the serial one. Moreover, ES-GBFDA has a coding gain of 0.05dB with respect to SES-GBFDA with the optimum  $\beta$  ( $\beta = 2$ ) and both improve the performance in the waterfall region compared to GBFDA with flooding schedule. In addition, note the importance of the selection of the optimum  $\beta$  for the SES-GBFDA. As Fig.2.14 shows, a wrong  $\beta$  can cause a non-negligible performance loss (there is a difference of 0.2dB between  $\beta = 2$  and  $\beta = 4$ ). These performance curves corroborate the statistical analysis showed in the previous subsection, where it was found that the most probable value for  $\beta$  is 2.

On the other hand, ES-GBFDA and SES-GBFDA ( $\beta = 2$ ) with 10 iterations have a performance loss of 0.65dB and 0.7dB with respect to Min-max with 15 iterations. However, for future hardware comparisons in Section 2.4, ES-GBFDA, SES-GBFDA and Min-max architectures must be compared under similar FER performance conditions to be fair. As reported in Section 2.1.4, the performance of Min-max with 4 iterations is similar to GBFDA with parallel flooding and 20 iterations and ES-GBFDA with 10 iterations, for the code under analysis.

### 2.2.6 Partial parallel architectures for the serial schedule

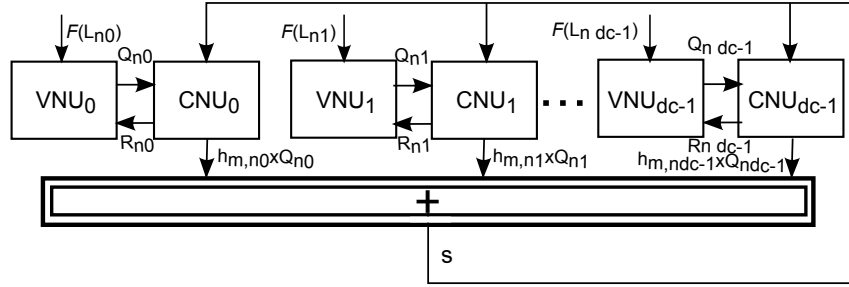
The following subsection is composed by two parts. In the first one, an architecture for ES-GBFDA and parallel implementation of the clipping technique is detailed. The second subsection shows how the ES-GBFDA architecture is modified to implement SES-GBFDA and include the new initialization method. Both architectures are based on message passing solutions, because unlike the implementation of flooding schedule where  $d_c \times d_v$  VNU units were required to reduce latency, the implementation of the serial schedule just needs  $d_c$  VNU units for high speed, so routing is already reduced without using broadcasting techniques.



**Figure 2.14:** FER performance of Min-max algorithm, GBFDA, ES-GBFDA and SES-GBFDA for a (837,723) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel.

### Partial parallel architecture for ES-GBFDA

The complete partial parallel architecture for the ES-GBFDA is included in Fig.2.15. The degree of parallelism of the architecture is  $d_c$ , so there are  $d_c$  CNU and VNU units working at the same time.



**Figure 2.15:** Complete partial parallel architecture for ES-GBFDA and SES-GBFDA.

1. **CNU unit:** The CNU unit shown in Fig.2.16 is implemented taking advantage of the quasi-cyclic structure of the codes proposed in [1]. The  $H_{m,n}$  block in Fig.2.16 generates the  $h_{m,n}$  coefficients required in step A2. The  $d_v \times p$  ROM stores the  $d_v$  exponents of the coefficients for the same column of the circulant permutation matrix before the dispersion or expansion (see the generation of the matrix in [1]). The  $(q-1) \times p$  ROM stores the powers of alpha ( $\alpha^0, \alpha^1, \dots, \alpha^{q-2}$ ). To compute the expansion, one of the exponents of the  $d_v \times p$  ROM is read and loaded in the cyclic counter. This counter, which begins in the read exponent and finishes after  $q-1$  cycles, is used as an address of the  $(q-1) \times p$  ROM, getting at the output the corresponding  $h_{m,n}$  of the expanded matrix. In the same way, the  $H_{m,n}^{-1}$  block generates the  $h_{m,n}^{-1}$  coefficients required in step A3. First, the  $h_{m,n}$  coefficient is multiplied by the output of the corresponding VNU unit using a GF multiplier, giving as a result  $h_{m,n} \times Q_n$ . As can be seen in Fig.2.16, the  $d_c$   $h_{m,n} \times Q_n$  values (one for each CNU unit) are added to obtain the syndrome,  $s$  (step A2). Then, in each VNU  $s$  is multiplied by the corresponding  $h_{m,n}^{-1}$  coefficient using a GF multiplier and the output is added to  $Q_n$ , getting  $R_n$  (step A3).
2. **VNU unit:** The VNU unit is included in Fig.2.17. Each RAM stores  $q-1$  values of  $R_n$ , which correspond to one of the  $d_c \times d_v$  sub-matrices of size  $(q-1) \times (q-1)$  obtained after the expansion. Data are saved in each RAM in a ping-pong manner to ensure a continuous processing (without idle times). Each VNU unit has  $q$  cells that compute:  $W_n(0) - W_{m,n}(0)$ ,  $W_n(\alpha^0) - W_{m,n}(\alpha^0)$ ,  $W_n(\alpha^1) - W_{m,n}(\alpha^1)$ , ...,  $W_n(\alpha^{q-2}) - W_{m,n}(\alpha^{q-2})$ . So, each cell processes the value of  $\mathbf{W}_n - \mathbf{W}_{m,n}$  associated to a different element of the field  $\text{GF}(q)$ . The decoder converts the  $p$  bits of  $R_n$  into a word of  $q$

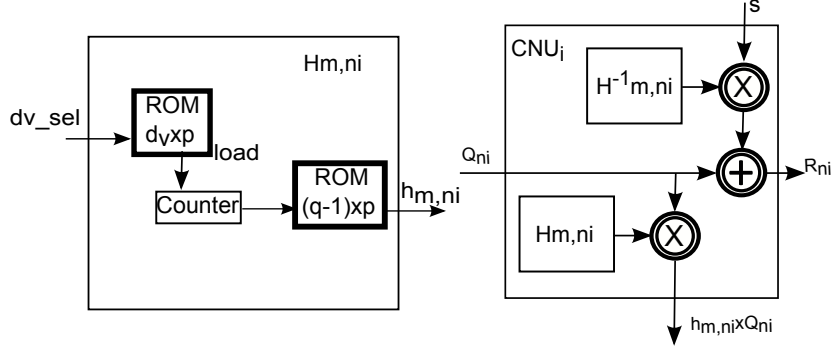
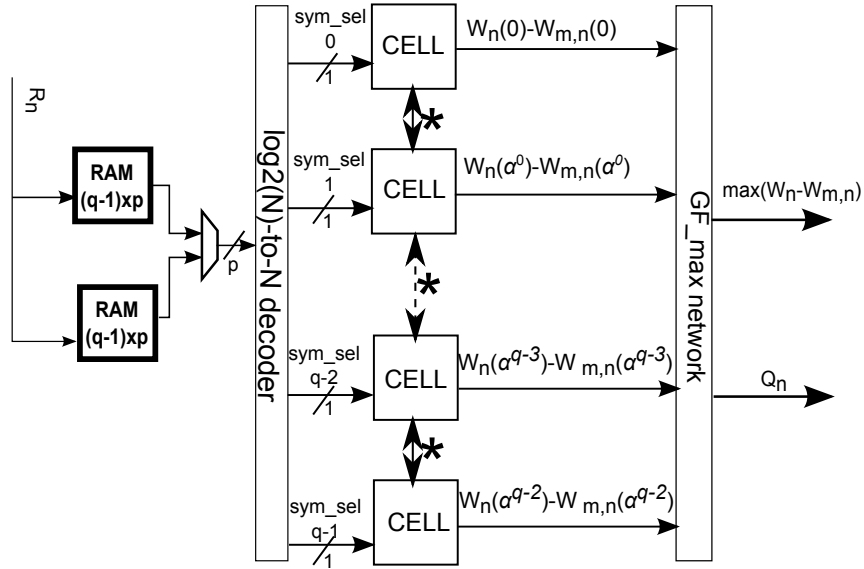


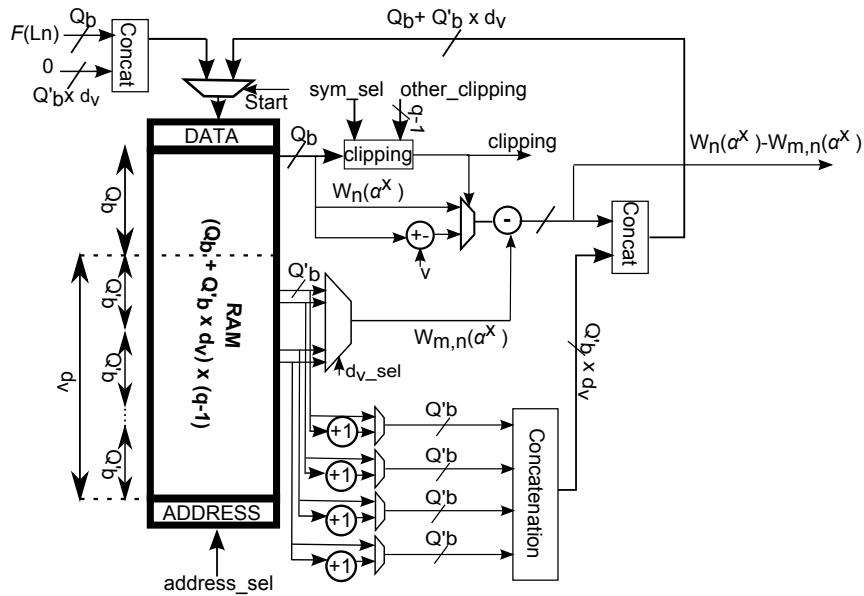
Figure 2.16: CNU basic cell for ES-GBFDA and SES-GBFDA.

bits ( $sym\_sel$ ). Only one bit of  $sym\_sel$  is set to one, activating the cell whose power of  $\alpha$  is equal to  $R_n$ . This indicates that the CNU votes that symbol. The output of all the cells are compared to look for the maximum value of  $\mathbf{W}_n - \mathbf{W}_{m,n}$  and its associated GF symbol ( $Q_n$  in step A1). These comparisons are carried out with a tree of comparators and multiplexors, named as *GF\_max network*.

The architecture of each cell processor is included in Fig.2.18. The RAM stores the  $\mathbf{W}_n$  and  $\mathbf{W}_{m,n}$  values for the  $\alpha$  element of the field that processes that cell. Each data word is split as follows: the  $Q_b$  most significant bits are the  $W_n(\alpha^x)$  value and the remaining  $d_v \times Q'_b$  bits store  $d_v$  different values of  $W_{m,n}(\alpha^x)$  quantized with  $Q'_b$  bits. There are  $q - 1$  different data words because of the size of the sub-matrix. The VNU processor works as follows. First, the RAM is filled with the reliability values concatenated with  $d_v \times Q'_b$  zeros during the initialization. After that, the RAM stores the updated information for  $W_n(\alpha^x)$  and the  $d_v$  different values of  $W_{m,n}(\alpha^x)$ . The update of  $W_n(\alpha^x)$  depends on: the value of the decoder output ( $sym\_sel$ ), the detection of clipping in other cells of the same VNU unit and the detection of clipping in the own cell. If the cell is selected and no clipping is detected in the own cell,  $W_n(\alpha^x) = W_n(\alpha^x) + 1$ . If the cell is not selected and clipping is detected in other cells,  $W_n(\alpha^x) = W_n(\alpha^x) - 1$ . In other cases,  $W_n(\alpha^x)$  remains the same. The signal  $dv\_sel$  selects the corresponding  $W_{m,n}(\alpha^x)$  from the  $d_v$  possible values.  $W_{m,n}(\alpha^x)$  represents the votes obtained by  $\alpha^x$  at the previous iterations and it is required for the computation of  $W_n(\alpha^x) - W_{m,n}(\alpha^x)$ . In addition,  $W_{m,n}(\alpha^x)$  is increased by one if  $sym\_sel=1$ , to take into account the vote made in the current iteration. Both  $W_n(\alpha^x)$  and the  $d_v$   $W_{m,n}(\alpha^x)$  updated values are concatenated and stored again in the RAM. Note that the clipping processing of one cell requires the clipping information of the other  $q - 1$  cells. This is done by broadcasting the clipping result of each



**Figure 2.17:** VNU unit for ES-GBFDA and SES-GBFDA. VNU unit for SES-GBFDA does not include the interconnections marked with \*.



**Figure 2.18:** Basic cell for ES-GBFDA ( $d_v = 4$ ).

cell to the others. In Fig.2.17 this bus is represented by bidirectional rows between cells. This interconnection between cells increases routing in the VNU unit, reducing the maximum operating frequency of the decoder.

Some pipeline registers included to reduce the critical path of the decoder has been omitted to simplify the figures. The latency of the decoder is  $(q - 1 + pipeline) \times d_v \times (It_{max} + 1) + (q - 1 + pipeline)$  clock cycles. Computing each sub-matrix takes  $(q - 1)$  clock cycles but the pipeline delay has to be added, so each sub-matrix needs  $(q - 1 + pipeline)$  clock cycles. As there are  $d_v$  sub-matrices, each iteration needs  $(q - 1 + pipeline) \times d_v$  clock cycles. One extra iteration is added for the initialization. Finally,  $(q - 1 + pipeline)$  clock cycles are needed to get the values of the decoded codeword after the iterative process.

### ***Partial parallel architecture for SES-GBFDA***

In this part of the subsection, the architecture proposed for the ES-GBFDA is modified to implement a SES-GBFDA architecture and the alternative initialization which does not require clipping. In this case, just the VNU unit changes, while the rest of the architecture remains the same. So, the new architecture has exactly the same latency than the one proposed for ES-GBFDA. However, the maximum frequency achievable is increased due to the elimination of the clipping, and this allows us to increment considerably the throughput.

Thanks to the new initialization, the different cells that compose the VNU unit are totally independent. As shown in Fig.2.17 the wires between cells (marked with \*) disappear in the architecture without clipping. This avoids  $q \times (q - 1)$  wires in each VNU unit reducing the routing congestion.

On the other hand, Fig.2.19 includes the basic cell of the VNU unit for SES-GBFDA. The new initialization makes the update of  $\mathbf{W}_n$  simpler than in ES-GBFDA: if the value of  $W_n(\alpha^x)$  read from the RAM is lower than the maximum value  $(2^{Q_b-1} - 1)$ , this value is incremented by one ( $W_n(\alpha^x) = W_n(\alpha^x) + 1$ ); if not, the maximum is reached and saturation is applied ( $W_n(\alpha^x) = W_n(\alpha^x)$ ), which means that step A5 is not performed. Additionally, there is no need of sharing information with other cells because there is no clipping. Moreover, in SES-GBFDA  $W_{m,n}(\alpha^x)$  is just a flag that indicates if the corresponding symbol is voted or not at the current iteration (step A4). The  $W_{m,n}(\alpha^x)$  values are stored in the RAM because this information is needed to compute step A1 in the next iteration. This is done by means of a  $\log_2(d_v)$ -to- $d_v$  decoder. This decoder generates a  $d_v$ -bit output that indicates which symbol is voted at the current iteration (only one bit is set to one if the cell is active,  $sym\_sel = 1$ , otherwise all the bits are set to zero because there is no vote). Finally, the parameter  $\beta$  of step A1 is introduced as follows: the flag  $W_{m,n}(\alpha^x)$  read from the RAM, which



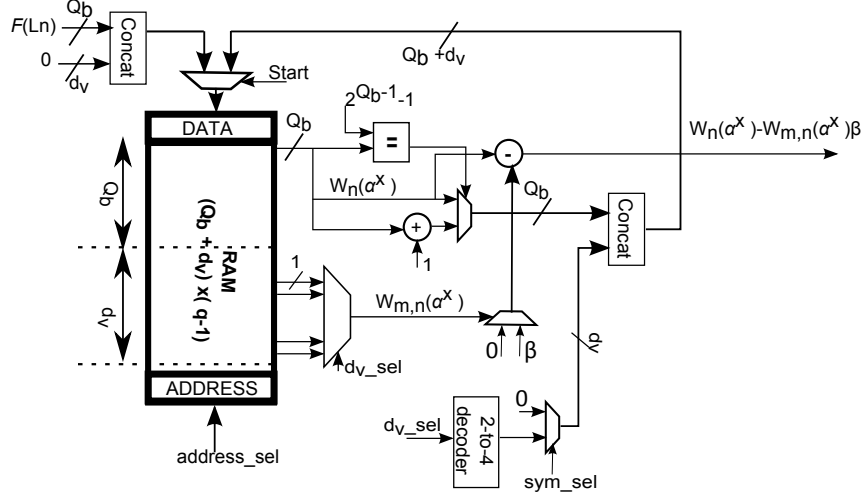


Figure 2.19: Basic cell for SES-GBFDA ( $d_v = 4$ ).

indicates if the symbol was voted in the previous iteration, controls a multiplexor to decide if  $\beta$  is subtracted or not from  $W_n(\alpha^x)$ .

In conclusion, RAM memory resources are reduced from  $(Q_b + d_v \times Q'_b) \times (q - 1)$  bits in ES-GBFDA to  $(Q_b + d_v) \times (q - 1)$  bits in SES-GBFDA. Also, the data path related to the votes is reduced from  $Q'_b$  bits to just one bit, and the  $d_v$   $Q'_b$ -adders needed to accumulate the votes in ES-GBFDA are eliminated.

#### Area and timing complexity for the (837,723) NB-LDPC code

The architectures described in the previous section for ES-GBFDA and SES-GBFDA were implemented using a 90nm CMOS standard cell library for a (837,723) NB-LDPC code.

Table 2.13 includes post place and route ASIC results of the proposed architecture for ES-GBFDA and for SES-GBFDA. In addition, we include again the ASIC results from the M-GBFDA architecture with the parallel schedule from Section 2.1.5 to make easier the comparisons.

First, ES-GBFDA and SES-GBFDA architectures are compared. On the one hand, the effect of not requiring clipping is analyzed by comparing the throughput of ES-GBFDA with the one of SES-GBFDA. The maximum frequency of the decoder is improved, increasing throughput by 16%. This increase of speed is achieved due to the reduction of the critical path and the independence between basic cells,

which implies a reduction of the wiring interconnection (from a total wire length of  $39105917\mu m$  to  $19686639\mu m$ ). On the other hand, the impact of simplifying ES-GBFDA is studied comparing area. With SES-GBFDA area is reduced by 44.6% compared to ES-GBFDA. This is mainly due to the reduction of memory resources achieved using one bit instead of  $Q'_b = 3$  bits to quantize the votes and the elimination of  $d_v$  adders in each basic cell. In terms of efficiency, SES-GBFDA decoder is 2.1 times more efficient than ES-GBFDA decoder.

Compared to half-broadcasting M-GBFDA architectures, ES-GBFDA and SES-GBFDA reach a lower maximum frequency, due to the length of the critical path. This bigger length is caused by a bigger degree of parallelism. Note that ES-GBFDA and SES-GBFDA increase the degree of parallelism in the VNU, processing  $q$  symbols at the same time, while M-GBFDA computes serially each one of the  $q$  symbols. This higher degree of parallelism was not possible for the flooding schedule of M-GBFDA because: i) the number of VNU units was higher,  $d_c \times d_v$ ; and ii)  $\mathbf{W}_{m,n}$  required more bits to be quantized. So trying to work with the same degree of parallelism with M-GBFDA would increase at least  $q$  times area, and the longer critical path and routing, even with half-broadcasting, would make speed decrease dramatically. However, even with lower frequency, ES-GBFDA and SES-GBFDA achieve a slightly higher throughput, thanks to the two times reduction of latency. It is also important to remark that ES-GBFDA has a similar area to half-broadcasting M-GBFDA architectures, because the extra area required by the parallel VNU processor is compensated by just requiring  $d_c$  VNU units and  $Q'_b = 3$  bits to quantize  $\mathbf{W}_{m,n}$ . Although, M-GBFDA with blocking architectures have similar efficiency in terms of hardware to ES-GBFDA architecture, the last one is more interesting because it does not introduce any performance degradation, such as error floor. SES-GBFDA has approximately half area compared to architectures based on blocking, so more than two times efficiency is achieved.

Compared to the architecture of M-GBFDA with clipping, both ES-GBFDA and SES-GBFDA reduce more than 14 times latency which makes them more efficient in terms of speed over area ratio. The performance is also slightly different, both ES-GBFDA and SES-GBFDA have a better slope at the waterfall region than the M-GBFDA with clipping.

### 2.2.7 Conclusions of the architectures for the serial schedule of GBFDA

In this subsection of the chapter we have proposed two algorithms to improve efficiency and reduce storage resources of GBFDA with the serial schedule. In addition, an alternative initialization is applied in order to reduce complexity. Moreover, some architectures have been derived with the objective of reaching high speed decoders for high-rate NB-LDPC codes. Conclusions of this part of the chapter are:

**Table 2.13:** ASIC results for a 90nm CMOS process of ES-GBFDA, SES-GBFDA and M-GBFDA partial parallel architectures

Algorithm	M-GBFDA	M-GBFDA	M-GBFDA	ES-GBFDA	SES-GBFDA
Data growth control technique	Clipping	Blocking $r = Q_b$	Blocking $r = Q_b - 1$	Clipping	None
Architecture	Message Passing	Half-broadcasting	Half-broadcasting	Message Passing	Message Passing
Code	(837,723)	(837,723)	(837,723)	(837,723)	(837,723)
Code Rate	0.86	0.86	0.86	0.86	0.86
GF	32	32	32	32	32
$(d_c, d_v)$	(26,4)	(26,4)	(26,4)	(26,4)	(26,4)
Process	90nm	90nm	90nm	90nm	90nm
Report	post-layout	post-layout	post-layout	post-layout	post-layout
$Q_b (Q'_b)$	5 bits	5 bits	5 bits	5 bits (3 bits)	5 bits (1 bit)
Area ( $mm^2$ )	13	12.8	13.4	12.2	6.6
Frequency (MHz)	255	500	476	238	277
Iterations	20	20	20	10	10
Latency (clock cycles)	23415	3476	3476	1620	1620
Throughput (Mbps)	45	601	573	615	717
Early Error Floor Degradation	NO	YES	YES	NO	NO
Efficiency (Mbps/ $mm^2$ )	3.46	46.9	42.7	50.4	108.6

1. A serial schedule algorithm which stores intrinsic information from all iterations is proposed. The algorithm is named ES-GBFDA. ES-GBFDA also makes an efficient storage of the information, separating the intrinsic vote information from the channel information. The algorithm does not introduce any performance degradation.
2. A simplified version of ES-GBFDA is proposed, SES-GBFDA. Based on statistical analysis and simulation a value  $\beta$  is obtained to approximate voting information from previous iterations. This simplification leads to a negligible performance loss and allows as to quantize the voting information with 1 bit.
3. An alternative initialization based on normalizing the channel information to the hard-decision reliability and eliminating some unreliable information is proposed. This initialization avoids using data growth control techniques such as saturation, clipping or blocking.
4. Partial parallel message passing architectures for ES-GBFDA and SES-GBFDA have been proposed. ES-GBFDA architecture implements a parallel version of clipping, showing that frequency is reduced compared to the alternative

initialization, implemented for the SES-GBFDA architecture. In addition, SES-GBFDA reduces to almost half the required area.

5. Both ES-GBFDA and SES-GBFDA have a similar throughput compared to M-GBFDA based on blocking architectures, however, their FER performance does not introduce any early error floor degradation. For the case of SES-GBFDA area is reduced to half. We conclude that ES-GBFDA and SES-GBFDA architectures are more efficient than the ones based on M-GBFDA.
6. All GBFDA derived algorithms, ES-GBFDA and SES-GBFDA, have less coding gain than EMS and Min-max algorithms, because only hard decision information is computed in their check nodes.

### 2.3 Multiple-vote algorithm and architecture

[REDACTED]

[REDACTED]



[REDACTED]

- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]

[REDACTED]

- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]

- [REDACTED]
- [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

- [REDACTED]
- [REDACTED]
- [REDACTED]

[REDACTED]

- [REDACTED]
- [REDACTED]
- [REDACTED]

[REDACTED]

[REDACTED]



[REDACTED]

[REDACTED]

### 2.3.2 Parameters of MV-SFA and FER performance comparisons

[REDACTED]

[REDACTED]

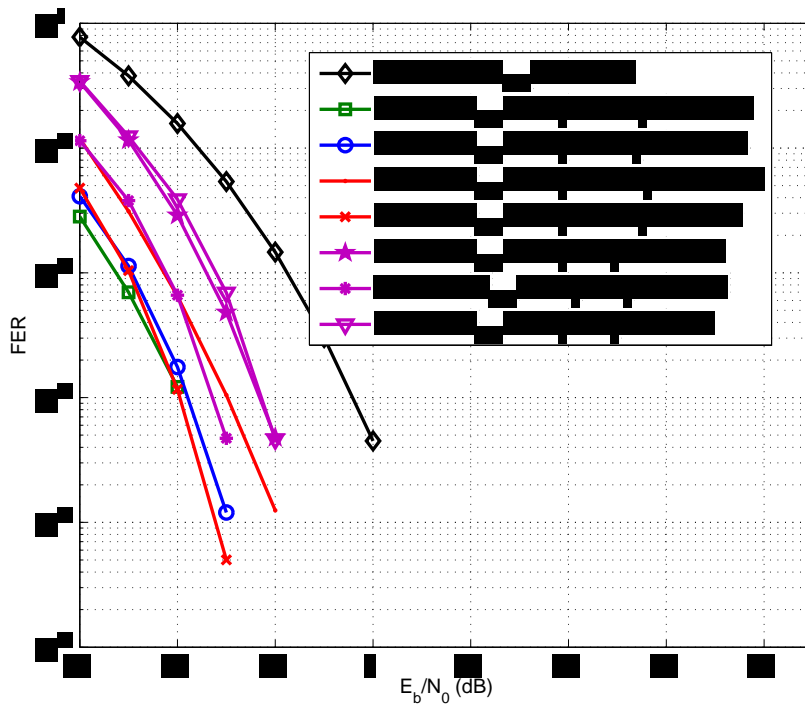


[REDACTED]

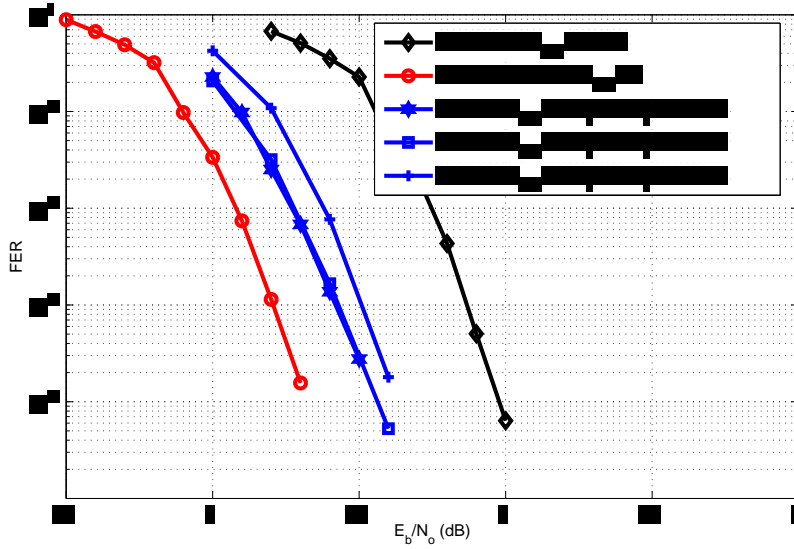
[REDACTED]

### 2.3.3 Partial parallel architecture for MV-SFA

[REDACTED]



**Figure 2.20:** FER performance of ES-GBFDA and MV-SFA for a (1536,1344) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel.



**Figure 2.21:** FER performance of Min-Max algorithm, ES-GBFDA and MV-SFA for a (837,723) NB-LDPC code over GF(32) with a BPSK modulation in an AWGN channel.

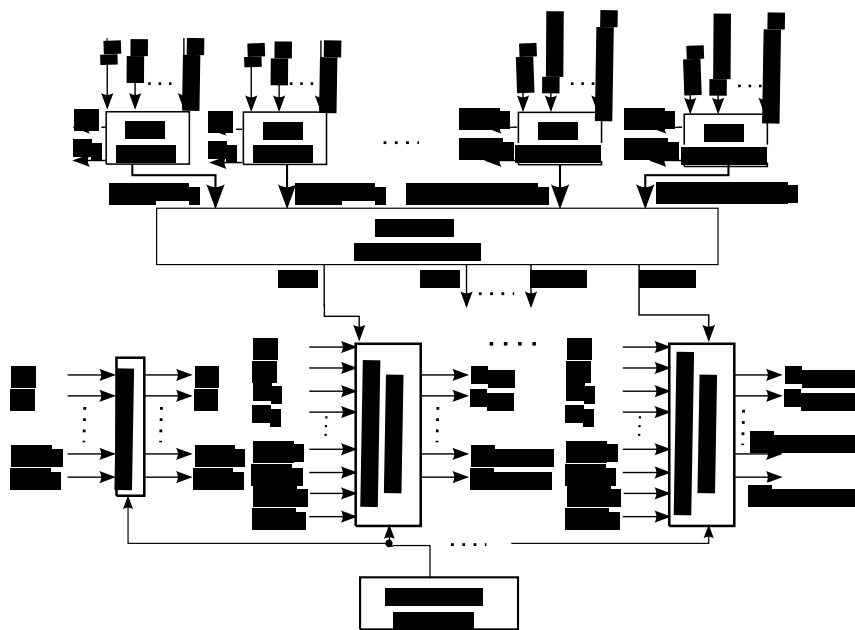
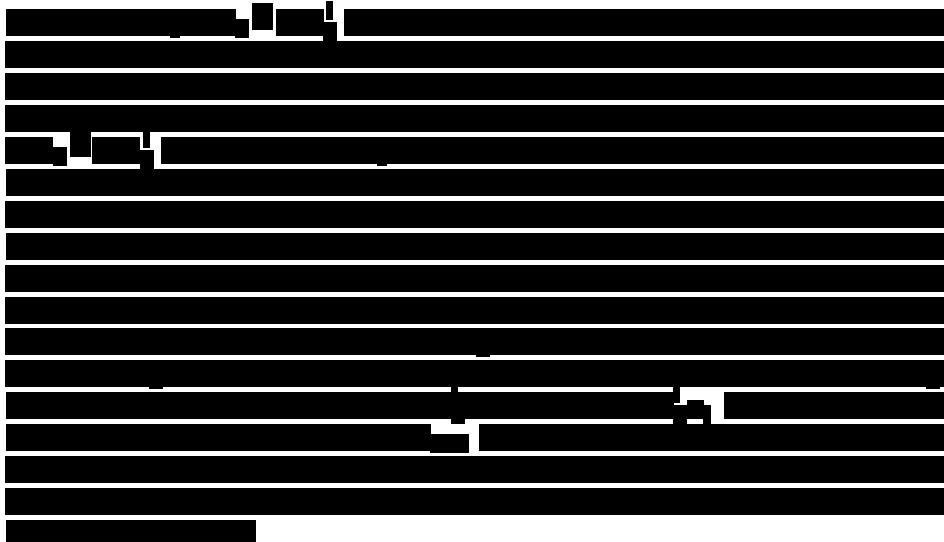


Figure 2.22: Complete partial parallel architecture for MV-SFA.

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

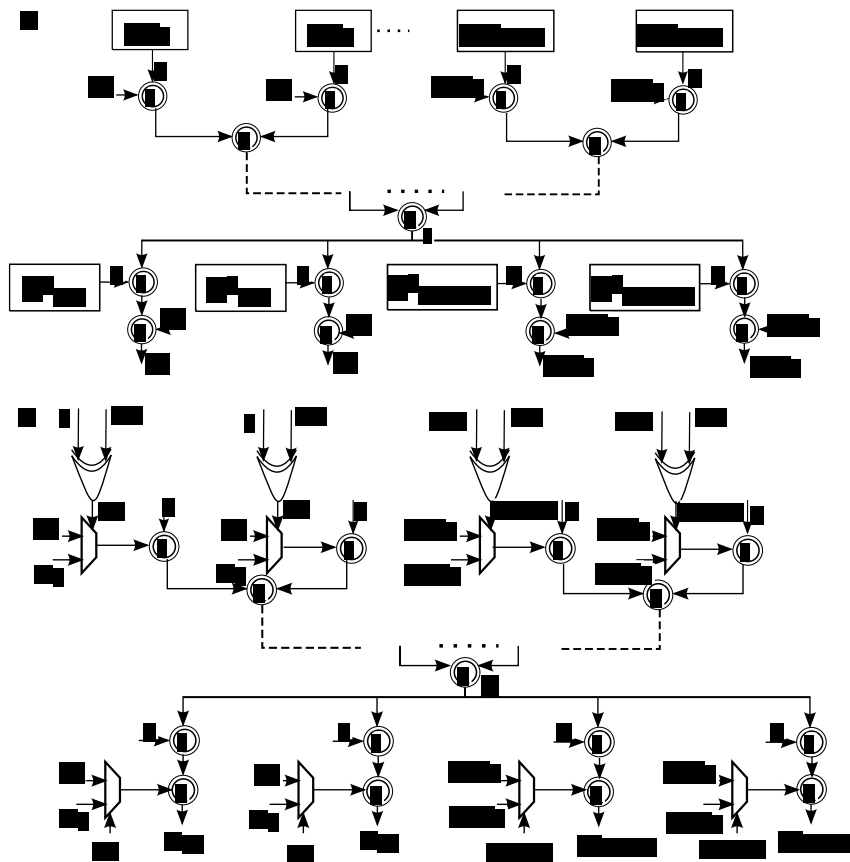


Figure 2.23: CNU units for MV-SFA: [REDACTED]



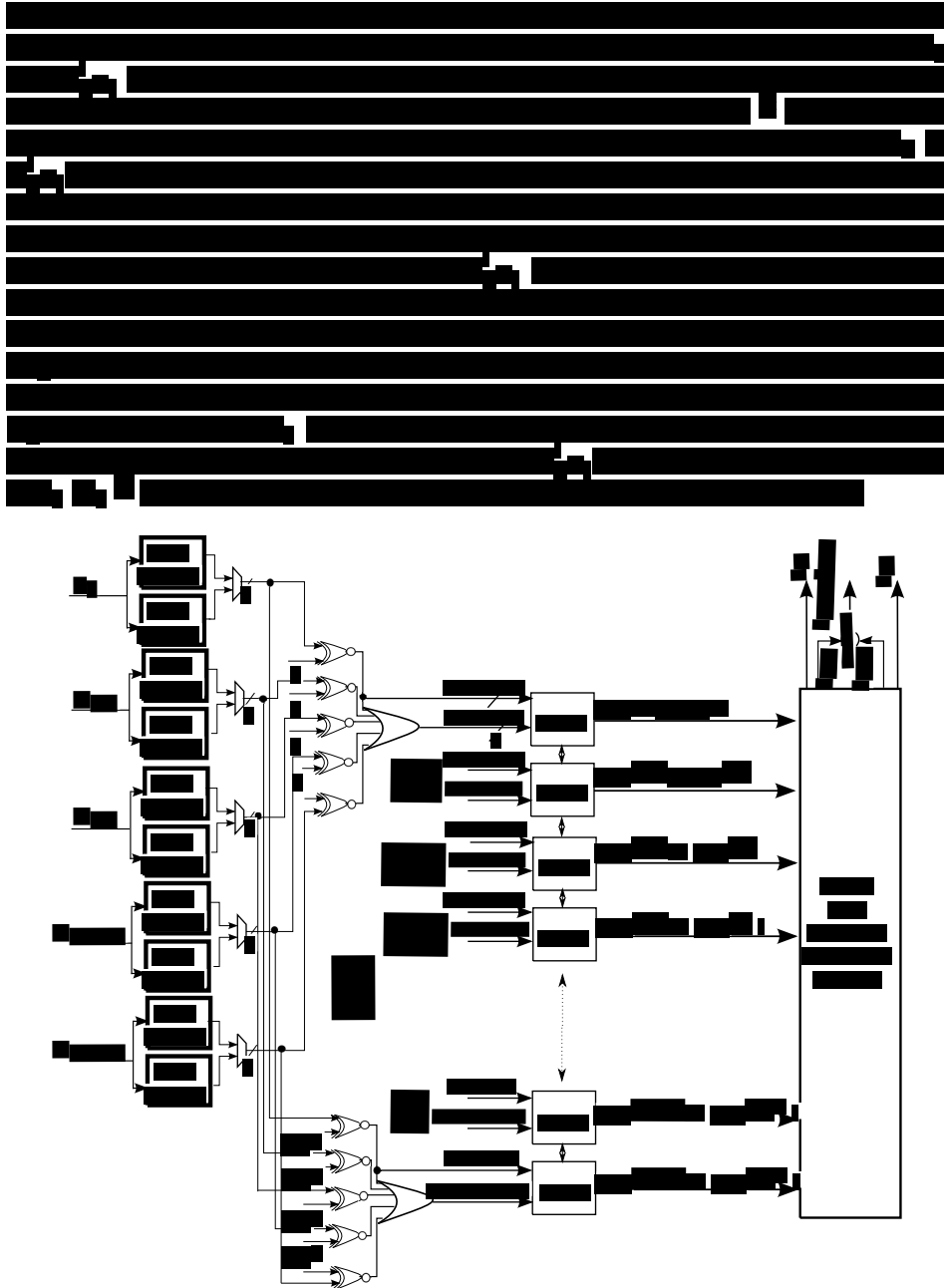


Figure 2.24: VNU unit for MV-SFA.

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

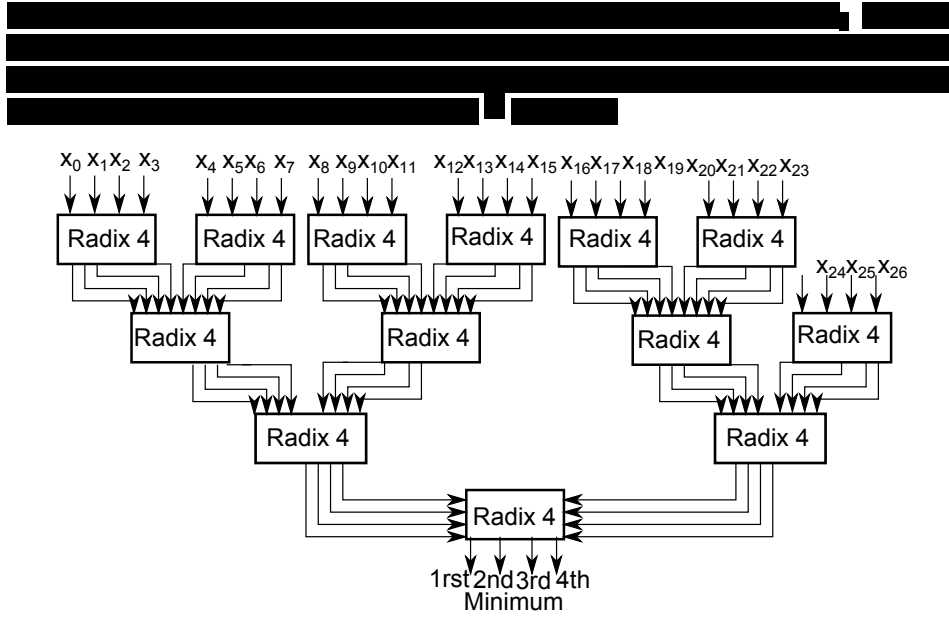
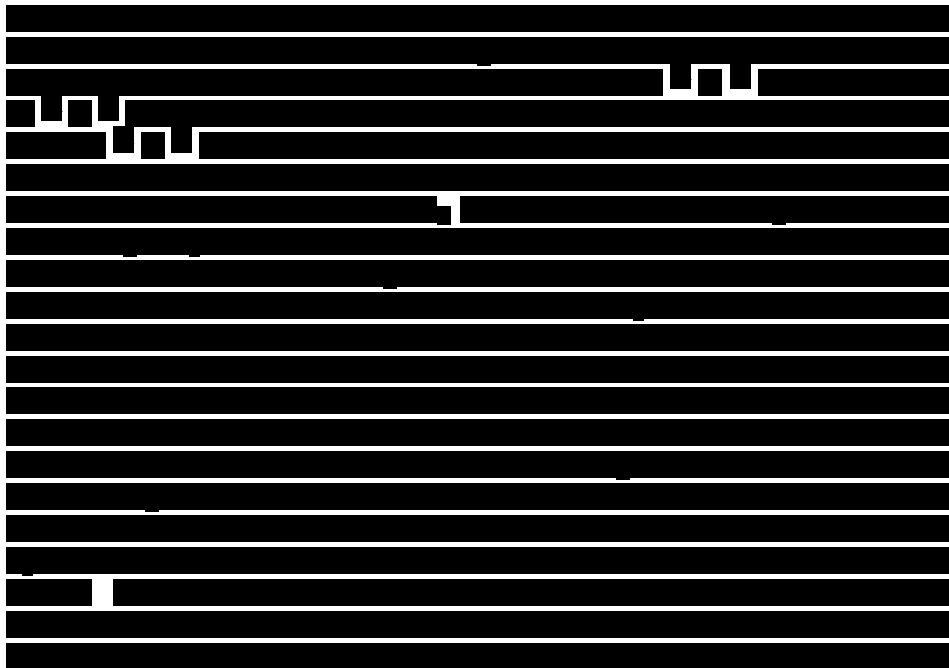
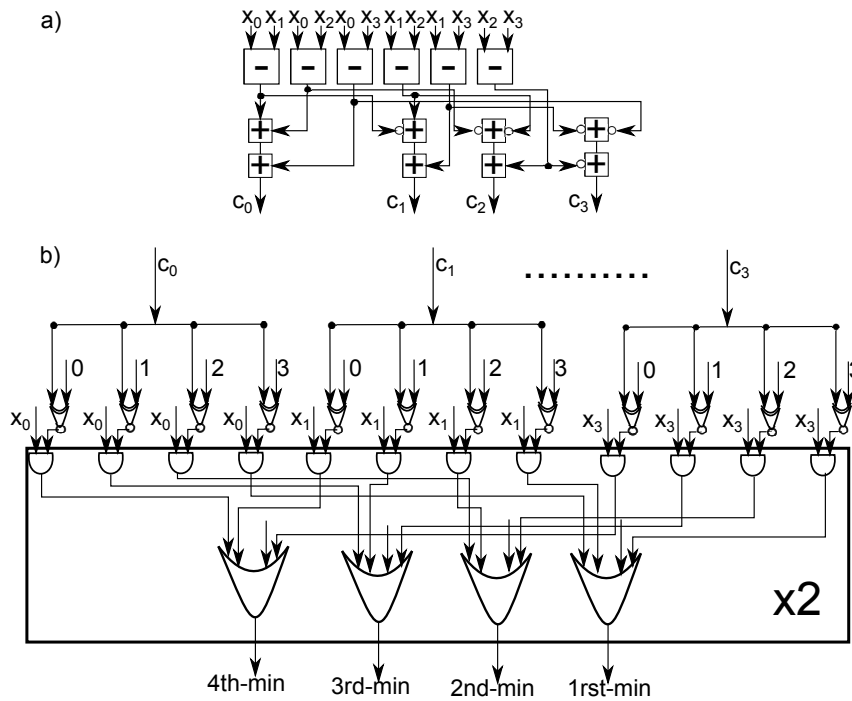


Figure 2.25: General architecture for a 4-minimum finder in a 27-element list.





**Figure 2.26:** Architecture for the first stage ( $l = 0$ ) of a 4-minimum finder in a 27-element list.

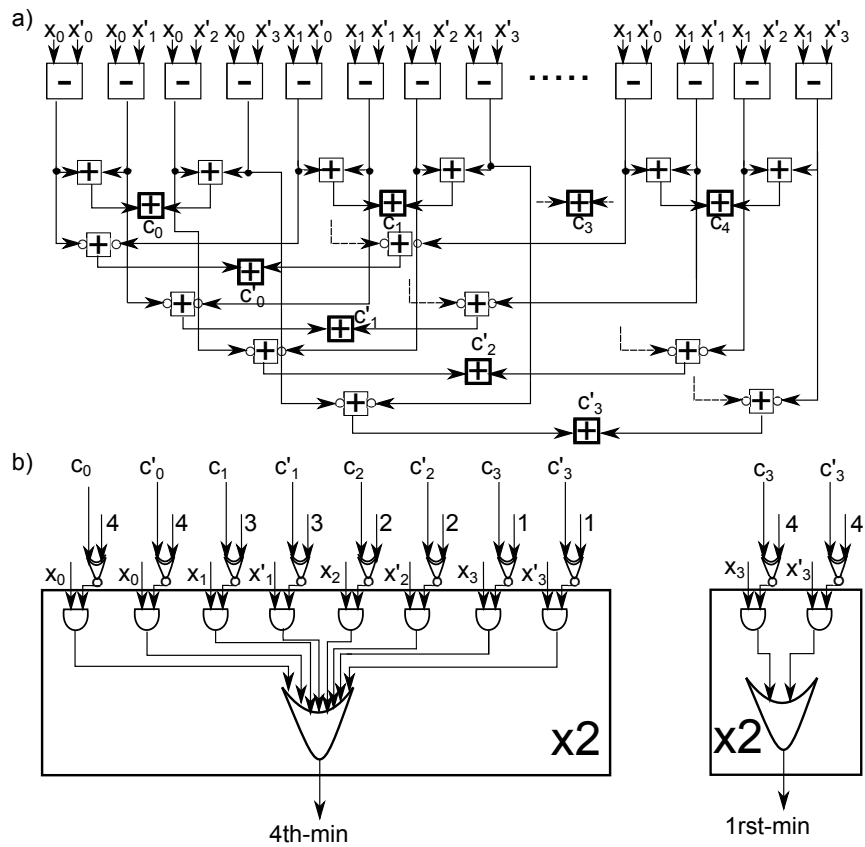
[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]



**Figure 2.27:** Architecture for stages which are different from the first one ( $l > 0$ ) of a 4-minimum finder in a 27-element list. The index 0 of the inputs is for the 4th minimums and the index 3 is for the 1rst minimums.

[REDACTED]





**Table 2.16:** Estimated results of ES-GBFDA and MV-SFA partial parallel architectures

Algorithm	ES-GBFDA	MV-SFA
Code	(837,723)	(837,726)
Code Rate	0.86	0.86
GF	32	32
$(d_c, d_v)$	(26,4)	(27,4)
Process	90 nm	analytic methods
$Q_b (Q'_b)$	5 bits (3 bits)	7 bits (5 bits)
Iterations	10	10
Frequency (MHz)	238	238
Latency (clock cycles)	1620	1845
Throughput (Mbps)	615	540
Gate count ( <i>XORs</i> )	847K	1.5M
Performance loss compared to EMS and Min-max ( <i>dB</i> )	0.7	0.26
Efficiency (Mbps/M- <i>XORs</i> )	726	360

### 2.3.4 Conclusions of the multiple-vote symbol flipping algorithm

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

## 2.4 Comparisons with other works

The first architecture for a NB-LDPC code was proposed in 2007, the code had half-rate and short codeword length and reached 44Mbps [45]. Since then, a large number of papers about this topic have been published. Most of them were more focused on providing simplifications and reductions of complexity of EMS and Min-max decoding algorithms, specially the check node processing, than in hardware architectures. This was due to the fact that there were some important bottlenecks in both storage resources and speed, that only could be solved from an algorithmic perspective.

First, most of the results related with ASIC technology were based on estimations or analytic methods, because the objective of these works was to have an approximate idea of the area of the derived architectures. These analytic methods allowed authors to quantize the hardware improvements derived from their algorithmic proposals as fast as new algorithmic improvements were proposed. There was no a real interest in implementation as decoders were to slow and big, and in a short period other contributions would come out. So, there was a high probability that during the implementation of the architecture a different algorithmic proposal was published and a more efficient architecture would be proposed. Changes in algorithms were faster than the time required to complete a full design, so there was no a real motivation to make HDL models. Most of these works were proposed during 2010 and provided a new method to evaluate in a short period of time the interest of the NB-LDPC algorithms [46], [47], [48], [49], [17] and [50]. However, the increase of area due to routing and speed cannot be estimated in an accurate way with these methods. Moreover, during this time solutions for wireless communications were proposed [21], accomplishing area and speed constraints for this kind of applications based on real implementation results. This first period was also interesting because researchers started to give results of decoders for high-rate codes with medium GF length, highlighting the difficulties of reaching high-speed and low area with architectures derived from EMS and Min-max algorithms.

Although some synthesis results were published before 2012, [11], [51], [52], they were all focused on half-rate codes. The first main contribution to high-rate codes decoding architectures based on synthesis results was published at the beginning of 2012 in [18]. The architecture was a configurable decoder which reached a throughput 16Mbps with and area of 1.37M gates for the (837,726) NB-LDPC code with a 180nm CMOS process. Thanks to the real implementation of this architecture, accurate results in terms of area and critical path were provided for the first time. During 2012, more architectures have been published including now synthesis results, being the most efficient ones [53], [54] and [55]. Unfortunately, even the fastest architecture, proposed in [55], has a limited throughput of 64Mbps with an area of 1.29M gates for the (837,726) NB-LDPC code with a 180nm CMOS process. This limitations in throughput are produced because VLSI designers base

their architectures on EMS and Min-max algorithms and they do not want to introduce any performance loss. However, it seems that if higher throughput wants to be achieved some coding gain needs to be sacrificed. Following this reasoning, a low complexity algorithm similar to T-EMS, named as relaxed Min-max [24], has been proposed. Despite its low complexity throughput is still low, 66Mbps. Other architectures derived from T-EMS algorithm with higher degree of parallelism are estimated to reach a throughput of 337Mbps [25]. These estimations have to be corroborated by place and route results, because it is easy that for codes over field above GF(16), routing congestion may reduce the maximum frequency achievable. Despite that, T-EMS algorithm is an interesting option for codes over low field order and, in addition, more simplifications, a part from the ones proposed in [24] must be developed if higher throughput wants to be reached with GF(16) and above. This simplifications will introduce also some performance degradation, but this will be future work.

Before making comparison with other works, it is important to remark two more ideas: i) to the best knowledge of the author, only the work published in [53] provides post place and route results; and ii) none previous work on architectures for symbol-flipping algorithm with high-rate codes has been proposed a part from the ones included in this manuscript, even when at [42] was shown that these algorithms can reach 193Mbps with half-rate codes. With regard to the first issue, we think that it is very important to provide in the future post place and route results, because wiring problems and routing congestion are very likely to be suffered by LDPC decoders [22] and specially in the non-binary ones (long discussions about this topic have been included along this manuscript). With synthesis results we can have an idea of complexity in terms of area and throughput, but only after place and route we can certainly know what is the real speed and size of the decoder. About symbol-flipping for high-rate codes, we think that they can be an interesting alternative to EMS and Min-max algorithms, as their CNU is less complex and allows us to derive more simple architectures with higher degree of parallelism. More work has to be done related with this topic, which has been abandoned mainly for the performance loss that introduces. However, as we will see next, decoders with similar performance to EMS and Min-max algorithms and higher hardware efficiency can be derived from GMV-SFA, in particular MV-SFA.

In Table 2.17, the most efficient architectures found in literature are included <sup>8</sup>. In addition, from all our proposals detailed in this chapter we have just selected the most efficient in terms of hardware derived from GBFDA, SES-GBFDA architecture, and the architecture for MV-SFA. As there is a different FER performance in both contributions, we have included results from other works for FER performance close to GBFDA and with the original FER performance detailed in each publication. To include results for a FER performance close to GBFDA we have

<sup>8</sup>Note that the fully parallel T-EMS architecture is not included because only estimation of the CNU unit is done for a (837,726) NB-LDPC code

**Table 2.17:** Efficient architectures for high-rate NB-LDPC codes

Algorithm	Min-Max [53]	Min-Max [54]	Simplify MS [55]	Relaxed Min-Max [24]	SES-GBFDA	MV-SFA
Code	(837,726)	(837,726)	(837,726)	(837,726)	(837,723)	(837,726)
Code Rate	0.86	0.86	0.86	0.86	0.86	0.86
GF	32	32	32	32	32	32
$(d_c, d_v)$	(27,4)	(27,4)	(27,4)	(27,4)	(26,4)	(27,4)
Process	90 nm	130 nm	180 nm	180nm	90 nm	analytic methods
Report	post-layout	synthesis	synthesis	synthesis	post-layout	analytic methods
$Q_b (Q'_b)$	7 bits	5 bits	5 bits	5 bits	5 bits (1 bit)	7 bits (5 bits)
Frequency (MHz)	260	500	200	200	277	238
Iterations	15/4	15/4	15/4	15/4	10	10
Latency (clock cycles)	N/A	28215 / 7524	12995 / 3454	12675 / 3380	1620	1845
Throughput (Mbps)	29/108.75	64/240	64/240	66/247	717	540
Throughput-90nm (Mbps)	29/108.75	107/400	149.3/560	154/576.3	717	540
Gate count ( $XORs$ )	1.32M	1.06M	806K	544K	468K	1.5M
Area ( $mm^2$ )	N/A	N/A	N/A	N/A	6.6	N/A
Performance loss compared to EMS ( $dB$ )	0.05/0.8	0.05/0.8	0/0.75	0.07/0.82	0.7	0.26
Efficiency-90nm (Mbps/M- $XORs$ )	21.96/82.4	101/377	185/695	283/1059	1532	360

calculated throughput and latency with four iterations, because we have shown in Section 2.1.4, 2.2.5 and 2.3.2 that Min-max algorithm with four iterations reaches a FER performance close to GBFDA for the (837,726) NB-LDPC code. So SES-GBFDA is compared to other architectures with four iterations and MV-SFA is compared with the original number of iterations, because FER performance cannot be included in the efficiency parameter and we want to perform comparisons as fair as possible. We want to remark once again that SES-GBFDA cannot reach higher coding gain with higher number of iterations because it is limited by the lack of soft information in the check node. Finally, for comparisons between the proposals of this manuscript, we refer the reader to the end of each subsection.

First, SES-GBFDA is compared to [53] , [54], [55] and [24] with 4 iterations. In terms of hardware resources, SES-GBFDA requires 2.26, 2.8, 2.97 and 1.2

times less area than [53], [54], [55] and [24], respectively. In terms of speed, the throughput achieved by SES-GBFDA is 6.5, 1.8, 1.3 and 1.2 times higher compared to [53], [54], [55] and [24] respectively. Under a similar FER performance, SES-GBFDA proposal is at least 30.8% more efficient than EMS and Min-max architectures.

With regard to MV-SFA<sup>9</sup>, the comparisons are performed with [55] and [24] with 15 iterations. MV-SFA has a performance loss of 0.19dB compared to the most efficient Min-max proposal. In terms of area requires 2.7 times more area than the architecture in [24]. However, MV-SFA reaches a throughput 3.5 times faster. Hence, MV-SFA is 27% more efficient than the decoder in [24] at a cost of 0.19dB of performance loss. Compared to the most efficient MS architecture, our decoder requires 1.86 times more area, but reaches a throughput 3.6 times higher. The decoder for MV-SFA is 1.95 times more efficient compared to the one in [55] based on EMS, but a difference of 0.26dB in coding gain should be taken into account.

To complete this section of comparisons we include upper and lower bounds of complexity and efficiency in terms of Mbps/M-XORs. As an upper bound (highest complexity), the most efficient architecture for log-QSPA is considered [15]. As lower bound, the binary decoder from [56] is selected, following the most recent non-binary VLSI works [24]. Note that Table 2.18, which includes these architectures, is separated from the rest of designs found in literature because comparisons in this case are just a reference of the order of complexity, not fair comparisons, as QSPA is too inefficient from a hardware point of view and the binary code only has the same length in terms of bits and the same rate but not the same degree distribution. In addition, the reader should also remember that NB-LDPC codes can gain advantage over binary codes when high order modulations are applied or burst errors should be corrected (see Section 1.6). For this reasons a fair and generalizable comparison between the binary and NB-LDPC is not possible, and the next results shown are just useful to understand the magnitude of the problem of designing architectures for NB-LDPC codes. As can be seen in Table 2.18 the QSPA decoder from [15] requires 3.5 times more area than the MV-SFA one. Moreover, QSPA latency is about 2.4 times higher than MV-SFA, providing a throughput 2.4 times lower than our proposal. For the same 90nm technology we can conclude that QSPA is 8.5 times less efficient than MV-SFA decoder. The binary decoder from [56] has a latency 3.7 times lower and a throughput 3.8 times higher than our proposal. The binary decoder with the same length and rate as the non-binary one requires 3 times less area than the MV-SFA architecture. In terms of efficiency measured as Mbps/M-XORs, the binary decoder is about 12 times more efficient than the MV-SFA decoder for the same code length and rate.

<sup>9</sup>Remember that area results are overestimated to compensate some possible deviations of the analytic method. Throughput results are computed using the post place and route frequency of ES-GBFDA, because both MV-SFA and ES-GBFDA have the same critical path. Latency is estimated duplicating the number of pipeline stages.

**Table 2.18:** Upper and lower bounds for binary and non-binary LDPC architectures

Algorithm	log-QSPA [15]	Binary Min-sum [56]
Code	(837,726)	(4191,3602)
Code Rate	0.86	0.86
GF	32	2
$(d_c, d_v)$	(27,4)	(36,4)
Process	90 nm	65 nm
$Q_b$	7 bits	5 bits
Frequency (MHz)	250	300
Iterations	5	15
Latency (clock cycles)	4460	495
Throughput (Mbps)	223	2100
Gate count ( <i>XORs</i> )	5.3M	495K
Efficiency-90nm (Mbps/M- <i>XORs</i> )	42	4242

## 2.5 Conclusions

[REDACTED]

[REDACTED]









**Part II**

**Algebraic soft-decision  
Reed-Solomon decoding**





## Chapter 3

# State of the art of algebraic soft-decision Reed-Solomon decoding algorithms

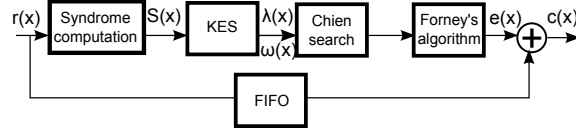
In this chapter the reader will find: i) a brief summary of the basic concepts of Reed-Solomon (RS) codes; ii) comparisons between algebraic soft-decision algorithms for high-rate codes with the aim of concluding which is the most interesting from a hardware perspective suitable for high speed; and iii) the principles of low complexity Chase decoding algorithm.

### 3.1 Background concepts of Reed-Solomon codes

Let us define a  $(N, K)$ RS code over  $\text{GF}(q)$  ( $q = 2^p$ ). The total number of symbols that constitutes the codeword  $c(x)$  is given by  $N = q - 1$ . If systematic encoding is performed,  $c(x)$  includes  $K$  information symbols and  $N - K = 2t$  parity symbols incorporated by the encoder. The received polynomial at the decoder is  $r(x) = c(x) + e(x)$ , where  $e(x)$  is the error of the channel, and  $c(x)$  is the transmitted codeword.

A hard-decision RS decoder is composed of four blocks (Fig.3.1). Those are (i) syndrome-computation, (ii) key-equation solver (KES), (iii) Chien search, and (iv) Forney's algorithm.

In the syndrome-computation step,  $2t$  syndromes of the message polynomial  $r(x)$  are computed as



**Figure 3.1:** Block diagram of a generic Hard-Decision RS decoder

$$S_i = \sum_{j=0}^{n-1} r_j \alpha^i, \text{ for } i=1 \text{ to } 2t \quad (3.1)$$

where,  $\alpha$  is an element of the field  $\text{GF}(q)$  and  $r_j$  is each one of the coefficients from  $r(x)$ .  $r(x)$  has erroneous symbols if at least one of the  $2t$  syndromes is different from zero. In the next step, an error locator polynomial  $\lambda(x)$ , and a magnitude polynomial  $\omega(x)$ , are computed by a KES algorithm, solving Equation 3.2.

$$\lambda(x) \cdot S(x) = \omega(x) \bmod x^{n-k} \quad (3.2)$$

The solution of Equation 3.2 is obtained by the KES, which can be implemented applying either Euclid's algorithm or Berlekamp-Massey algorithm.  $S(x)$  is a polynomial composed of  $S_i$  syndrome coefficients computed in Equation 3.1.

After the KES block, the Chien search is performed to extract the positions of the errors in the polynomial  $e(x)$  applying Equation 3.3.

$$\lambda(\alpha^{-i}) = 0, \text{ for } i = 0 \text{ to } n \quad (3.3)$$

During this step a decoding failure (DF) can be detected. The KES process provides an error locator polynomial  $\lambda(x)$ , with the same degree as the number of detected errors [57]. This happens only if there are no more than  $t$  error symbols on the message. Therefore, for a received message with  $h$  errors,  $\lambda(x)$  is a polynomial of degree  $h$ , where  $h < t + 1$ . In other words,  $t$  is the maximum degree of the locator polynomial  $\lambda(x)$ . The zeros of Equation 3.3 are the roots of  $\lambda(x)$  and indicate the error position. A decoding failure happens if a received message generates an error locator polynomial  $\lambda(x)$  of degree higher than the number of roots obtained during the Chien search. In that case the number of errors is higher than  $t$ , and therefore, the decoder cannot recover the message correctly.

The last step of the hard-decision decoder is the implementation of Forney's algorithm to compute the magnitude of the errors,  $e(x)$ , at the error locations extracted by the Chien Search. The magnitudes are calculated evaluating the power of  $\alpha$

that represents an error position, where  $\lambda'(x)$  is the formal derivative of  $\lambda(x)$ , Equation 3.4.

$$e_i = \frac{\omega(\alpha^i) \cdot \lambda(\alpha^i)}{\lambda'(\alpha^i)}, \text{ if } \lambda(\alpha^i) = 0 \text{ and } e_i = 0, \text{ if } \lambda(\alpha^i) \neq 0 \quad (3.4)$$

Once the error locations and their magnitudes are determined,  $e(x)$  is subtracted from  $r(x)$  to recover the transmitted message.

### 3.2 Complexity of soft-decision algorithms for Reed-Solomon codes

Hard decision decoders (HDD) involve low complexity, but they do not use the soft information from the channel. This fact turns their FER performance not acceptable in many future and even current communication and storage systems. To provide some coding gain, different soft-decision methods have been proposed: algebraic soft-decision (ASD) algorithms [58], maximum likelihood decoding [59], reliability based ordered statistics decoding [60] and belief propagation decoding [61]. Maximum likelihood and reliability based ordered statistics decoding for RS have a complexity that grows exponentially with the codeword length in terms of bits,  $N \times p$ , so it is not convenient for high-rate codes, usually with large code length. Belief propagation for RS is more complex than NB-LDPC decoding and although belief propagation can get 0.7dB of coding gain compared to other ASD algorithms for high-rate codes, its complexity is higher enough for not being considered by the VLSI community. The most efficient algorithms from a hardware point of view are ASD ones. Some architectures derived from these algorithms can reach a throughput of Gbps, however, their coding gain is limited, even with the highest complexity.

The first ASD method was Koetter-Vardy (KV) algorithm [58]. KV algorithm consists of three main steps: multiplicity assignment, interpolation and factorization. The multiplicity assignment step creates a matrix which is proportional to the reliability information from the channel, this matrix is named as multiplicity. Depending on the value of the multiplicity, a symbol is considered more or less reliable. The levels of multiplicity applied in the algorithm depends on the degree of precision that is required to classify the reliability of the different symbols. As it is reasonable to assume, with more levels of multiplicity, bigger coding gain can be obtained. During the interpolation process a bivariate polynomial is computed taking into account the multiplicities of the symbols. The larger is the maximum multiplicity, the larger is the complexity of the interpolation. Even with the minimum multiplicity value, interpolation is the stage that involves major part of the decoding complexity. To solve interpolation step, the most commonly used al-

gorithm is Nielsen's one [62]. In [63] an interpolation technique for reducing the computational complexity, named Lee-O'Sullivan interpolation, was proposed. Although this algorithm is more efficient than Nielsen's one, it cannot be described as a recursive method, which does not allow us to use information from previous processed points [64]. The bivariate polynomial obtained with the interpolation contains the locations and magnitudes of the error in their roots. To extract this information a factorization process is required. To compute factorization Roth-Ruckenstein algorithm is applied [65].

The main problems of KV are: i) its complexity grows fast with the maximum multiplicity; and ii) works with all the symbols from the codeword, which requires the processing of an extremely large number of points at each iteration of the interpolation. This makes difficult to achieve high throughput with a reasonable area [66].

A technique called re-encoder was proposed to reduce the number of interpolation points [67]. Using this method, based on the use of erasure decoders, the complexity of the interpolation step is reduced: for a  $(N, K)$ RS code, instead of  $N$  points, just  $N - K$  points need to be interpolated. This method improves significantly the throughput and reduces the area requirements. But, interpolation and factorization complexity are still high.

In addition, two decoding methods were proposed to reduce the maximum value of multiplicity. They are: low complexity Chase (LCC) decoding [68] and the bit-level generalized minimum distance (BGMD) decoding [69].

LCC decodes different versions of the received sequence, building a list of test-vectors in which only the unreliable elements are modified. LCC requires a maximum multiplicity of one, because it just has to distinguish to different levels of reliability, reliable symbols and unreliable symbols. For high rate codes, LCC decoding with a short number of test-vectors achieves the same FER performance as KV algorithm with  $m = 4$  [64]. Due to its advantage of a maximum multiplicity one, interpolation and factorization complexities are reduced, if these are combined with re-encoder. LCC proposals in literature are based on Nielsen's interpolation since it needs to change points during the interpolation process. So, Lee-O'Sullivan's interpolation cannot be used for LCC. Several algorithms and architectures have been proposed to implement LCC decoders [68], [64], [70] and [71]. The main schedules for solving LCC efficiently are the hypercube and the binary tree. Solutions based on hypercube apply a low complexity algorithm named backward interpolation [64]. Backward interpolation reduces the size of the storage of the intermediate interpolation results using Nielsen's algorithm special properties. For the binary tree based a modified version of Nielsen's is proposed in [71]. This algorithm reduces the number of steps required by the interpolation algorithm to process the test vectors. Hence, the derived architectures have a more reduced latency than the ones based on backward interpolation, but at a cost of an slightly

increase of area. Moreover, regardless of the interpolation algorithm, since the LCC works with a maximum multiplicity of one, an alternative method called free-factorization, which avoids the Roth-Ruckenstein factorization step, is introduced in [70] for reducing the complexity of LCC decoder. The most efficient LCC decoder to the knowledge of the author consists of a re-encoder, a Nielsen's interpolator, a method to extract the roots of the interpolator and one erasure decoder [70].

BGMD decoder [69] works with a maximum multiplicity two, and for high-rate codes achieves the same performance as KV algorithm with  $m = 4$ . The derived architectures are more efficient in terms of area and throughput than the ones of KV algorithm, and can incorporate the re-encoding technique. Architectures for BGMD decoder were proposed in [72] and [73]. BGMD decoder consists of a re-encoder along with a Nielsen's interpolator combined with backward interpolator [73] or the Lee-O'Sullivan interpolator [72], followed by factorization. Since its maximum multiplicity is two, no free-factorization methods can be applied to it. BGMD is found to be less efficient in terms of area and speed than LCC for the same FER performance, [70], [73], [74]. We detail LCC principles in the next section.

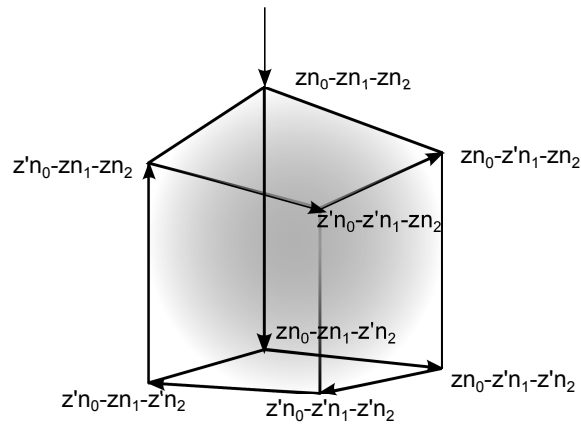
### 3.3 Low complexity Chase algorithm for Reed-Solomon soft-decision

The main steps for LCC decoding are: i) multiplicity assignments, ii) creation of the test-vectors, iii) decoding and iv) selection of the correct test-vector. To assign multiplicities a ratio between the probability of the hard-decision symbol,  $z_n$ , and the second best decision,  $z'_n$ , is established, Equation 3.5. This ratio indicates how reliable the hard-decision is.

$$\Pi_{1 \times n} = \left[ \frac{P(r_0|P(z_0))}{P(r_0|P(z'_0))} \dots \frac{P(r_{n-1}|P(z_{n-1}))}{P(r_{n-1}|P(z'_{n-1}))} \right] \quad (3.5)$$

For LCC decoding, the maximum multiplicity takes a value of one, and represents the most reliable symbols, which are the ones with higher ratio in Equation 3.5. The rest of the symbols have zero-multiplicity. With the multiplicity information LCC decoding creates  $2^\eta$  different test-vectors. Each test-vector is a different received sequence of symbols with a probability similar to the hard decision sequence. The parameter  $\eta$  represents the number of points with the worst probability ratio between the hard-decision and second best decision that will be considered. For these  $\eta$  positions, zero-multiplicity will be assigned. In order to keep the decoder complexity low to avoid impractical implementations, it must be accomplished that  $\eta < 2t$ , as indicated in [68]. To build the  $2^\eta$  test-vectors, both  $z_n$  and  $z'_n$  will

be considered in the zero-multiplicity locations, as they have similar probabilities to be correct. So, the decoder needs to process all the possible combinations of  $z_n$  and  $z'_n$  in the  $\eta$  locations, which leads to  $2^\eta$  different vectors to test. As it would be totally inefficient using  $2^\eta$  decoders or iterate over the same decoder  $2^\eta$  times, computation can be shared, mapping the common points in a hypercube or a binary tree, as we can see at Fig. 3.2 and 3.3. As it is very likely that different test vectors will lead to different decoded codewords, a method to select which test-vector has provided the correct solution, if it exists, is required.



**Figure 3.2:** Test vectors mapped in a hypercube for a decoder with  $\eta = 3$ .

LCC decoding obtains its coding gain not because of the algorithm applied to decode each test-vector, but due to the fact that there is a high probability that considering  $z'_n$  instead of  $z_n$  some errors are corrected before the decoding. Selecting the locations in which the hard decision is weak due to the noise, and changing it by the second best decision reduces the number of symbols with errors to  $t$  in some cases, and allows the decoder to recover the correct message. It is important to remark that the algorithm to be applied to decode the test-vectors is not specified in the original paper where Chase's method was published, [75]. Using interpolation it is not mandatory, but convenient. Applying interpolation methods to the binary tree [71] or the hypercube mapping [70] have shown some advantages as resource and computation sharing compared to KV algorithm. However, this method has also some drawbacks, like requiring two hard decision decoders based on erasures, one for the re-encoder and another to avoid factorization. Moreover, applying interpolation for decoding RS has not any advantage from a coding gain perspective when multiplicity is one. In [76] it is demonstrated and illustrated with an example that there is no difference between the coding gain of a decoder based on Nielsen's algorithm with multiplicity equal to one and a HDD decoder. However, the use of traditional HDD methods based on key-equation solvers (KES)



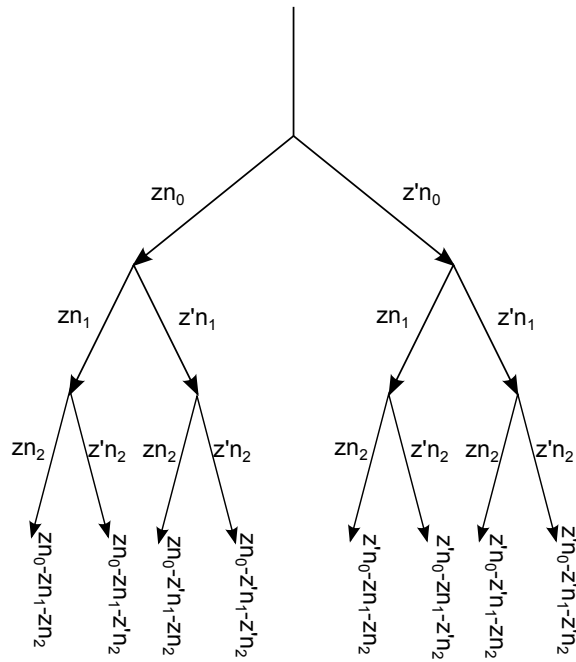


Figure 3.3: Test vectors mapped in a binary tree for a decoder with  $\eta = 3$ .

like Berlekamp-Massey or Euclidean algorithms [76], [77] have not been considered suitable to recover the information, because they involve more products than methods based on interpolation using the re-encoding technique. Such an affirmation is true if no simplifications are made and the decoding of the test-vectors is performed in brute force manner, applying one HDD to each test-vector.

In the next chapter, we will show that the test-vectors have some common points that allow us to reduce the complexity and increase the speed if HDD is applied to LCC decoding.

### 3.4 Conclusions

In this chapter we review the basics of Reed-Solomon hard-decision and soft-decision decoding. The conclusions are:

- Hard-decision Reed-Solomon algorithms have a limited error correction capacity.
- If higher coding gain wants to be achieved, soft-decision algorithms are necessary.
- Algebraic soft-decision algorithms are the soft-decision solutions that allow us a moderate increase of coding gain with reduce complexity.
- Algebraic soft-decision algorithms based on low complexity Chase decoding are more efficient than other algorithms such as Koetter-Vardy, because work with a lower multiplicity, which reduces the number of computations required.
- Low complexity Chase decoding schemes are based on interpolation, which requires, a part from the interpolator, a re-encoder, an erasure decoder, Chien search block and Forney's algorithm.
- It is not mandatory to use interpolation for low complexity Chase decoding, just a decoder that corrects  $t$  error by test vector is necessary.

## Chapter 4

# Contributions to algebraic soft-decision decoding algorithms

In this chapter the reader will find the algorithmic and architectural contributions of the thesis to the field of algebraic soft-decision decoding. There are two main sections. The first one proposes a new scheme to perform low complexity Chase decoding algorithm based on hard decision decoding algorithms. The second section includes the proposed architecture derived from the low complexity Chase algorithm. Comparisons with the existing architectures for this algorithm are also performed in this section. The last section summarizes the conclusions of this chapter.

### 4.1 Low complexity Chase decoding algorithm based on HDD

The method proposed in this section combines the hard-decision decoding techniques with LCC decoding to decode the RS codes using the channel information.

The problem we face up while we are using a procedure like hard-decision decoding for LCC arises mainly due to the fact that LCC decoder requires to decode  $2^n$  test-vectors. If the decoding of each test-vector is performed directly by an independent hard decision decoder, the computational complexity would increase prohibitively. Furthermore, a mechanism to choose the correct decoded vector is not proposed yet for HDD. Therefore, a selection process to choose the correct test-vector is needed and modifications are required in the traditional HDD steps to reduce its complexity.

The block diagram of the proposed decoder is shown in Fig.4.1. The steps of the decoder are: multiplicity assignment, syndrome computation, KES, Chien Search, decoding failure (DF) detection and Forney's algorithm. The main ideas are: i) to share the computation of the common symbols among the test-vectors in the syndrome computation stage to exclude the redundant processing; and ii) to select the correct test-vector by means of detecting a decoding failure.

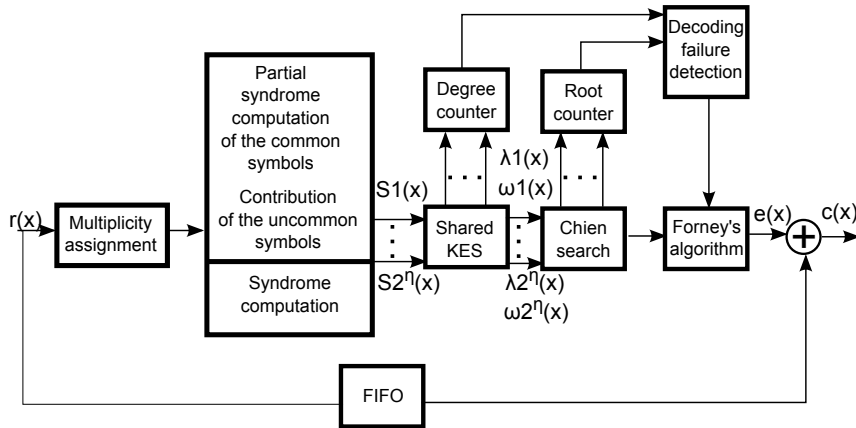


Figure 4.1: Block diagram of the LCC soft-decoder based on decoding failure

#### 4.1.1 Multiplicity assignment

The multiplicity assignment for the proposed LCC decoding algorithm is different from that used in works based on interpolation. With the HDD there is no need of re-encoding to reduce the number iterations. Hence, it is not necessary to force  $2t$  unreliable symbols, because it is not required to mark  $2t$  erasures in the re-encoding process<sup>1</sup>. With the LCC decoder based on HDD, zero-multiplicity is assigned just to  $\eta$  symbols to create the  $2^\eta$  test-vectors. The rest of symbols are considered as one-multiplicity.

To classify the multiplicity of the symbols, the probability ratios for the received message  $r(x)$  (Equation 3.5) are sorted. The  $\eta$  symbols with the minimum ratio values are denoted as zero-multiplicity. The locations of these  $\eta$  symbols will be the ones with different values for the test-vectors,  $z_n$  or  $z'_n$ .

<sup>1</sup>Note that the re-encoder is implemented by means of an erasure decoder, as we mention in the previous chapter

### 4.1.2 Syndrome computation

To apply HDD to LCC decoding a modification of the conventional syndrome processing is required. For doing this it is necessary to use multiplicity information of the LCC decoder. Algorithm 1 describes the proposed modified syndrome computation. First, the  $n - \eta$  symbols with one-multiplicity are used for the partial syndrome computation of the common symbols of all the test-vectors (step A1). Second, the contribution to the syndrome of the hard-decision and the second hard-decision values of each one of the  $\eta$  zero-multiplicity symbols are calculated (step A2). Finally, the syndrome computation is finalized by combining the results of the previous computation to form the  $2^\eta$  test-vectors (step A3).

In the description of Algorithm 13,  $z_n$  represents the hard-decision of the received symbols,  $z'_n$  is the second best decision, and  $\mathcal{M}_n$  denotes the multiplicities. The computation of partial values of the  $2t$  syndromes having the common symbols is represented as  $sc_g$ . The representation  $s_g^j$  is used for the contribution to the syndromes of the uncommon  $\eta$  locations with multiplicity zero. The subindex  $g$  indicates the syndrome which is computed. The superscript  $j$  represents the number of test-vector from the  $2^\eta$  possible ones. If the information is computed with hard-decision symbols, the syndrome for the uncommon symbol is represented as  $s_g^j$ ; if the second best decision is computed, the syndrome is denoted as  $s'_g{}^j$ . For the final syndrome representation of each test-vector, we use  $S_c^d$ , where  $c$  represents the syndrome that is stored, (from 1 to  $2t$ ), and  $d$  is the binary representation of the number of test-vectors. The binary representation of the test-vectors is used to indicate that '0' is equivalent to a contribution made with a  $z_n$  symbol and '1' is used to indicate that the computation is done by means of  $z'_n$ . For example, if LCC decoding works with  $\eta = 4$  and a syndrome  $S_2^{0101}$  is evaluated, it means that the syndrome corresponds to the  $\alpha^2$  evaluation, and the contributions for the  $\eta$  uncommon symbols used to compute the syndrome are a combination of  $z_{n3}$ ,  $z'_{n2}$ ,  $z_{n1}$  and  $r'_{n0}$ .

Note that such a syndrome computation based on two steps, i) the partial syndrome computation of the common symbols and ii) inclusion of contribution to the syndrome of the uncommon symbols, is less complex than the brute-force one. This is due to two facts: first, the products of the common symbols are not repeated for each test-vector; second, the  $\eta$  different symbols only require  $2\eta$  products per syndrome, (*i.e.*, two products per symbol per syndrome: one for  $z_n$  and other for  $z'_n$ ), instead of the  $2^\eta$  products per syndrome that a straight-forward method would have required. The output of the syndrome-computation is  $2t$  syndromes per test-vector.

---

**Algorithm 13** Syndrome computation for LCC decoding

---

**Input:**  $z_n, z'_n, \mathcal{M}_n$   
**for**  $i = 0 \rightarrow N$  **do**  
    **if**  $\mathcal{M}_i = 1$  **then**  
        **for**  $g = 1 \rightarrow 2t$  **do**  
            A1:  
             $sc_g = z_i \cdot \alpha^{g \cdot i} + sc_g$   
        **end for**  
    **else**  
        **if**  $j < (\eta + 1)$  **then**  
            **for**  $g = 1 \rightarrow 2t$  **do**  
                A2:  
                 $s_g^j = z_i \cdot \alpha^{g \cdot i}$   
                 $s'_g{}^j = z'_i \cdot \alpha^{g \cdot i}$   
                 $j = j + 1$   
            **end for**  
        **end if**  
    **end if**  
**end for**  
**for**  $b = 1 \rightarrow 2t$  **do**  
    A3:  
     $S_b^{00\dots 0} = sc_b + s_b^\eta + s_b^{(\eta-1)} + \dots + s_b^0$   
     $S_b^{00\dots 1} = sc_b + s_b^\eta + s_b^{(\eta-1)} + \dots + s'_b{}^0$   
    .....  
     $S_b^{11\dots 1} = sc_b + s'_b{}^\eta + s'_b{}^{(\eta-1)} + \dots + s'_b{}^0$   
**end for**  
**Output:**  $S_{b-00\dots 0}, S_{b-00\dots 1}, \dots, S_{b-11\dots 1}, b=1$  to  $2t$

---

### 4.1.3 Key equation solver

Once the syndromes of the  $2^n$  test-vectors are computed, KES is applied to each of the test-vectors. There are two algorithms for the realization of KES that have been widely studied from a hardware perspective. One of them is Berlekamp-Massey (BM) algorithm [78], [79], [80] and the other is Euclidean algorithm [81], [82], [83]. The original descriptions of both algorithms have been modified, and different versions have been proposed in literature. The recent-most versions are the Ultra-Ultra Folded inversionless Berlekamp-Massey (UiBM) algorithm [79] and the modified Euclidean (ME) [81] algorithm, which provide the best throughput and area performances.

The inputs (syndromes) and outputs ( $\lambda(x)$ ,  $\omega(x)$ ) of both algorithms, are identical and the techniques and the definitions presented in this manuscript can be applied to both of them. We have preferred to use the BM algorithm for the proposed LCC decoder, because of its potential for lower latency of implementation compared to Euclidean algorithm. However, we have not selected the UiBM version because it requires processing intermediate polynomials with higher order than previous versions of BM, and consequently requires more registers. Instead of UiBM, the inversionless Berlekamp-Massey (iBM) algorithm from [78] is used in the proposed decoder, Algorithm 14.

The outputs of KES process in LCC decoder are error locator polynomials,  $\lambda(x)$ , and a magnitude polynomial,  $\omega(x)$ , for each of the test-vectors. These polynomials have a maximum degree of  $t$ . The degree of the  $\lambda(x)$  of each test-vector is stored to perform the necessary comparisons to detect the occurrence of decoding failure.

### 4.1.4 Chien search, Forney and test-vector selection

The next step is to extract the roots of  $\lambda(x)$ . The most efficient method is the Chien search. One Chien search for each test-vector is performed, and the number of zeros is counted. After the completion of Chien Search, the number of zeros/roots of this step, and the degree of  $\lambda(x)$  are compared. The first test-vector that has the same degree as the number of roots is selected to run the Forney's algorithm. The fact of having the same degree and number of roots ensures that the test-vector is free from decoding failure. Therefore, if one of the  $2^n$  test-vectors has less than  $t + 1$  errors then decoder will be able to correct them in all cases. The comparison between roots and polynomial degree conforms the test-vector selection step.

**Algorithm 14** Inversionless Berlekamp-Massey algorithm

---

**Input:**  $2t$  syndromes,  $\lambda(x) = 1$ ,  $A(x) = 1$ ,  $k = 0$ ,  $\gamma = 1$ **for**  $r = 0 \rightarrow 2t$  **do**

A1:  $\delta = \sum_{n=0}^t \lambda_n \cdot S_{r+1-n}$

$T(x) = \gamma \cdot \lambda_i(x) - \delta \cdot A(x) \cdot x$

A2:

**if**  $\delta \neq 0$  and  $k \geq 0$  **then**

$A(x) = \lambda(x)$

$\gamma = \lambda$

$k = -k - 1$

**else**

$A(x) = A(x) \cdot x$

$\gamma = \gamma$

$k = k + 1$

**end if**

$\lambda(x) = T(x)$

**end for**

$\omega_i = \sum_{n=0}^i \lambda_n \cdot S_{i-n}$

**Output:**  $\lambda(x)$  and  $\omega(x)$ 

---

#### 4.1.5 Frame error rate performance for LCC based on HDD

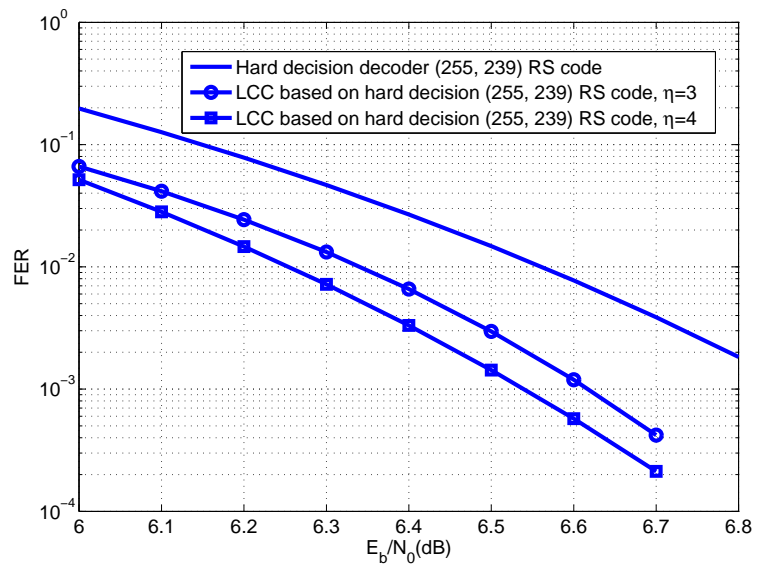
The proposed LCC decoding based on HDD has the same performance as the theoretic Chase decoder and the one reported in LCC works based on interpolation methods.

Fig.4.2 shows the FER performance for a  $(255, 239)^2$  RS code using the proposed LCC scheme with  $\eta = 3$  and  $\eta = 4$ . We have simulated the proposed decoder for BPSK modulated message with AWGN. It is shown that LCC gets around 0.3dB of coding gain at a FER of  $10^{-3}$  with moderated complexity and 10 bits to quantize the channel information required to compute the multiplicities. Unlike other codes such as LDPC, Reed-Solomon based on soft-decision does not introduce any degradation such as error-floor, so coding gain for lower FER values can be estimated extrapolating the slope from the figure.

---

<sup>2</sup>The  $(255, 239)$ RS code is considered in this manuscript because it is the commonest code in VLSI literature and optical communication standards.





**Figure 4.2:** FER performance for a (255,239) RS code applying HDD and an LCC decoder based on HDD for  $\eta = 3$  and  $\eta = 4$  over an AWGN channel with a BPSK modulation

## 4.2 Architecture for LCC decoder based on HDD

In this section, a high-throughput architecture for the different stages of the proposed LCC decoder is detailed.

### 4.2.1 Multiplicity-assignment unit

The first step of LCC decoding is the multiplicity-assignment, which implemented by the multiplicity-assignment unit. The proposed method makes use of the  $\eta$  least reliable symbols, which can be found in parallel by using  $\eta$  comparators,  $\eta$  registers of length  $Q_b$ <sup>3</sup> to store the channel information, and  $\eta$  registers of  $p$  bits each to store the position of the unreliable symbols. When the first ratio of probabilities between a hard-decision symbol and the second-best decision arrives to the multiplicity unit, the  $\eta$  registers are loaded with the channel information and the position of the symbol. The incoming ratios (corresponding to the newly received symbols) are compared with the ones stored in the  $\eta$  registers. The lowest reliability value at each moment is compared with the reliability of the new symbol, and if it found to be more reliable than the latter, then all the values in the  $\eta$  registers are shifted through one position, so that the most unreliable value takes the position of the second most unreliable one, and the new symbol takes the position of the most unreliable value. As a result, the least unreliable value is flushed out of the array of  $\eta$  registers when a more unreliable value arrives at. If the new symbol is less unreliable than the most unreliable symbol, then the second most-unreliable symbol at that moment is compared with a new symbol. If the second most-unreliable symbol is found to be more reliable than the new symbol, then all the values in the  $\eta - 1$  registers are shifted through one position so that the second most-unreliable value takes the position of the third most unreliable one, and the new symbol takes the position of the second most-unreliable value. The process is repeated for all the symbols in the registers, because these symbols are placed across the array of registers in the descending order of unreliability and the arrays of registers hold the  $\eta$  least reliable symbols. The location of the symbol from which the content of the registers needs to be shifted is identified by  $\eta$  comparators. After all the  $N$  symbols of the received message are passed through this process, the block of  $\eta$  registers of  $p$ -bit each contains the symbols with zero-multiplicity.

---

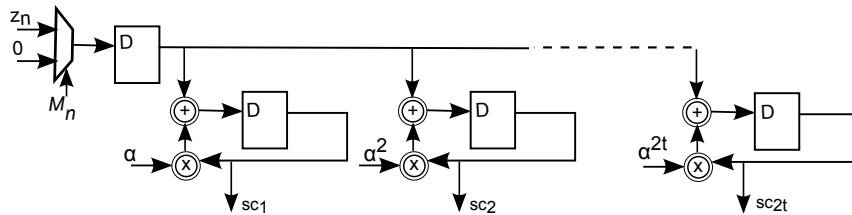
<sup>3</sup>Let us assume that  $Q_b$  is the number of bits required to quantize the ratio of probabilities from Equation 3.1 without introducing performance loss

### 4.2.2 Syndromes computation unit

The syndrome computation unit performs the syndrome computation by means of Algorithm 13. It has three blocks: i) the partial syndrome computation of the common symbols, ii) the computation of the contribution to the syndromes from the differing symbols, and iii) the final computation of all the syndromes from previous computations. In the next sub-sections we have illustrated the complete syndrome computations of a (7,5)RS code over GF(8) by three examples. Examples 4.2.1, 4.2.2 and 4.2.3, respectively explain the partial syndrome computation of the common symbols, evaluation of the differing symbols and total computation. Note that all these examples are included only to clarify the behavior of the proposed architectures. The order of Galois field and the RS code of these examples are different from the field and the RS code used in the implementation of the architectures.

#### *Computation of partial syndrome of the common symbols*

The computation of partial syndrome of the common symbols is shown in Fig.4.3. It is based on the recursive computation of the Horner's rule for polynomial evaluation. A multiplexor is used at the input for selecting the symbols with one-multiplicity for the computation of syndrome, as they are the common symbols for all the test-vectors. The  $\eta$  symbols with zero-multiplicity are replaced by a null symbol, for not affecting the result of the evaluation of the common symbols. The creation and evaluation of the new polynomial composed of common symbols of the test-vectors is illustrated in Example 4.2.1. The partial syndrome computation circuit involves  $2t$  constant multipliers,  $2t$  adders,  $2t + 1$  registers and a multiplexor as shown in Fig.4.3.



**Figure 4.3:** Architecture for the partial syndrome computation with the common symbols

**Example 4.2.1** Consider a (7,5) RS code over GF(8). In this code, two syndromes can be found. Assume that the received hard decision vector is :  $\alpha^3 \cdot x^6 + \alpha^0 \cdot x^5 + \alpha^5 \cdot x^4 + \alpha^4 \cdot x^3 + \alpha^5 \cdot x^2 + \alpha^4 \cdot x + \alpha^3$  and the multiplicity assignment is as follows :  $\{0,1,1,1,0,1,1\}$ , as we work with  $\eta = 2$ . For the two unreliable positions,

$x^6$  and  $x^2$ , the second best decision is  $\alpha^0$  and  $\alpha^3$ , respectively. To calculate the partial syndrome computation, we replace the symbols of zero-multiplicity by the null symbol, having as result  $\chi(x) = 0 \cdot x^6 + \alpha^0 \cdot x^5 + \alpha^5 \cdot x^4 + \alpha^4 \cdot x^3 + 0 \cdot x^2 + \alpha^4 \cdot x + \alpha^3 = \alpha^0 \cdot x^5 + \alpha^5 \cdot x^4 + \alpha^4 \cdot x^3 + \alpha^4 \cdot x + \alpha^3$ . Applying the Horner's rule, we evaluate the new polynomial with  $\alpha$  and  $\alpha^2$  to obtain the two syndromes of the (7,5) RS code:

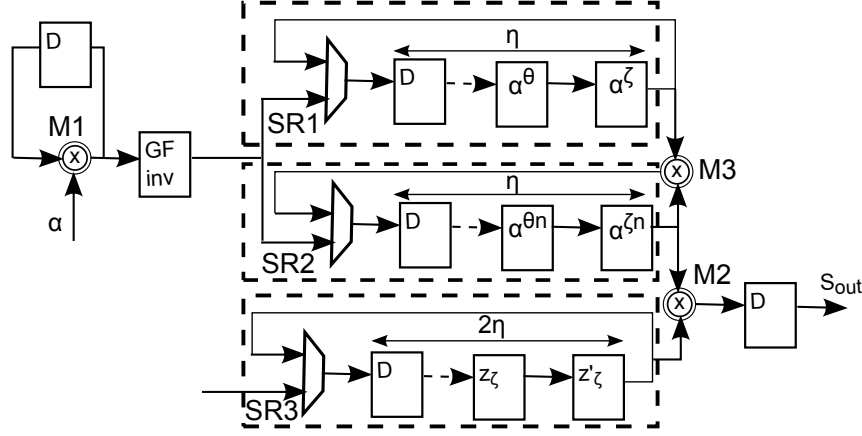
$$sc_1 = \chi(\alpha) = \alpha^4, sc_2 = \chi(\alpha^2) = \alpha^3$$

#### ***Evaluation of contribution of the differing symbols to the syndrome***

A circuit for computing the contribution to the syndrome of the differing symbols is depicted in Fig.4.4. For the  $\eta$  zero-multiplicity symbols, hard-decision and second most-reliable decision symbols are stored in a group of  $2\eta$  registers (SR3). This information is stored during the partial syndrome computation with the common symbols. In Example 4.2.2, we can see that we have to store  $\alpha^3$  and  $\alpha^0$  for the first differing symbol (as these are the values of the hard-decision and second most-reliable decision symbols of the first differing symbol) and the  $\alpha^5$  and  $\alpha^3$  for the second differing symbol, so four values are stored as per requirement for  $\eta = 2$ .

To know the contribution of the differing symbols to the final evaluation of syndromes, the position of the symbol in the received polynomial must be known. This position allows us to obtain the appropriate power of  $\alpha$  for the evaluation of the differing symbols. The power of  $x$  variable in the positions of the differing symbols must be known for evaluating each one of those differing symbols regardless of the rest of the symbols. In the Example 4.2.1, these powers are six and two, because the positions of the zero-multiplicity symbols correspond to the  $x^6$  and the  $x^2$  received symbols. As Example 4.2.2 shows, if we know the value of these powers we also know that for the first syndrome (evaluated in  $\alpha$ ) the evaluation of the variable  $x$  in the differing symbols are  $\alpha^6$  and  $\alpha^2$ . For the next syndrome (evaluated in  $\alpha^2$ ) the evaluation of the variable  $x$  in the special symbols is  $(\alpha^2)^6 = \alpha^{12} = \alpha^5$  and  $(\alpha^2)^2 = \alpha^4$ .

For this task, an exponentiation circuit consisting of a constant multiplier (M1) and a register is used, where the successive power of  $\alpha$ , starting from  $\alpha^0$  and ending at  $\alpha^{N-1}$  are evaluated by recursive multiplications. When a zero-multiplicity symbol is computed, the value of the register is passed through an inverter circuit and stored in two different registers (SR1-SR2). The knowledge of the power of  $\alpha$  allows us to count the location of a zero-multiplicity symbol, but the problem is that the received symbols do not start with the element with  $x^0$  of the polynomial; instead they start with  $x^{n-2}$ , as detailed in Example 4.2.2. Therefore, for the partial syndrome computation, the inverter transforms  $\alpha^y$  into  $\alpha^{n-y-1}$  using the properties of binary fields [84], explained in Example 4.2.2.



**Figure 4.4:** Architecture for the contribution to the syndrome of the uncommon symbols

The product of the stored hard-decision and the second most-reliable decision (stored in SR3) by the suitable power of  $\alpha$  (stored in SR2) is performed with the multiplier M2 to obtain their contribution for syndromes of each test-vector. However, only the contribution to the first syndrome (evaluated on  $x = \alpha$ ) can be computed as mentioned previously. That is the reason of using a double storage for the powers that represents the positions of the zero-multiplicity symbols. Using the third multiplier (M3) and the shift registers SR1-SR2, (shown in Fig. 4.4), the powers that represent the locations for the first syndrome can be transformed into the powers required for the rest of the syndromes. The shift-register, SR1 in the upper block of Fig.4.4 contains the values of the powers that have been stored for the first syndrome and the other shift-register (SR2) stores a new power every time it is shifted, which is calculated by applying Horner's recursions again. In Example 4.2.2, SR1 and SR2 store the first  $\alpha^6$  and  $\alpha^2$  (these values are only suitable for the first syndrome). With the multiplier M3 we perform the multiplication between  $\alpha^6$  of SR1 and SR2 to obtain the value for the second syndrome  $\alpha^6 \cdot \alpha^6 = (\alpha^2)^2 = \alpha^{12} = \alpha^5$ , which is stored in SR2. After a shift of SR1 and SR2, the new output value is  $\alpha^2$ , and following the same reasoning of  $\alpha^6$ , with the multiplier M3,  $\alpha^2 \cdot \alpha^2 = (\alpha^2)^2 = \alpha^4$  is computed and stored in SR2.

As a generic example, when the symbol of the  $\zeta$  location of the polynomial is evaluated, the products of  $z_\zeta$  and  $\alpha^\zeta$  and of  $z'_\zeta$  and  $\alpha^\zeta$  are computed by M2. After that, the two shift registers (SR1-SR2) are shifted and SR2 stores a new value,  $\alpha^{2\zeta}$ , which is the result of the multiplication of the outputs of SR1 and SR2 (by M3). This new value is used for the computation of the contribution to the second syndrome (evaluated on  $x = \alpha^2$ ). This example can be extended to the rest of

the syndromes, knowing that shift-register SR1 keeps  $\alpha^{\zeta}$  and the SR-2 stores  $\alpha^{\zeta n}$ , with  $n$  from 0 to  $2t$ .

So the proposed architecture allows us to compute the evaluation of the  $\eta$  special symbols independently, to have the values of the different hard-decision and second best-decision symbol contributions to the syndromes of each test-vector.

**Example 4.2.2** *Assuming the conditions of Example 4.2.1, as we have two different symbols for each one of the two unreliable positions of the received polynomial:  $\alpha^3$  and  $\alpha^0$  for  $x^6$ , and  $\alpha^5$  and  $\alpha^3$  for  $x^2$ . The received symbols are ordered from  $x^6$  to  $x^0$ . This allows us to apply the Horner's rule during the partial syndrome computation, but it makes difficult to compute the powers of the unreliable symbols for the contribution of the differing symbols. To make the evaluation of the differing symbols,  $\alpha^3 \cdot x^6$  or  $\alpha^0 \cdot x^6$  and  $\alpha^5 \cdot x^2$  or  $\alpha^3 \cdot x^2$ , we proceed with the following steps:*

- Store the differing symbols, hard decision and second hard decision, in SR3. The symbols are ordered in SR3 from input to output as follows:  $\alpha^3$ ,  $\alpha^5$ ,  $\alpha^0$  and  $\alpha^3$ .
- Perform the evaluation of  $x^6$  and  $x^2$ , and store the result in SR1 and SR2.

For the evaluation of  $x^6$  and  $x^2$ , we use a Galois field property. This property uses the inversion definition in which the inverse of a given power of  $\alpha$  power is another power of  $\alpha$ , such that multiplication of both those powers of  $\alpha$  gives the result  $\alpha^0 = 1$ . If we extract all the inverses of a field  $GF(q)$ , where  $N = q$ , we can prove that the inverse of a power  $\alpha^i$  is always  $\alpha^{N-1-i}$ , for all the elements in the field including  $\alpha^0$ , whose inverse is  $\alpha^7$  that is the same as  $\alpha^0$  ( $\alpha^7 = \alpha^0$ ).

Different powers of  $\alpha$  (from  $\alpha$  to  $\alpha^6$ ) are calculated in successive clock cycles by Multiplier M1, where the output of the multiplier is returned back as input through a register. Since the received symbols are ordered from  $x^6$  to  $x^0$ , for the evaluation of the differing symbols we need reverse order of computation of the powers of  $\alpha$  from  $\alpha^6$  to  $\alpha$  instead of  $\alpha$  to  $\alpha^6$ . So we add to the multiplier M1 an inverter of the field, that makes this transformation:  $\{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}^{-1} \implies \{\alpha^6, \alpha^5, \alpha^4, \alpha^3, \alpha^2, \alpha\}$ . The inversion operator can be implemented in different ways: one simple option is to use a memory with all the inverses of a given field, other is to implement by a combinational operator. The outputs of the inverter are stored in SR1 and SR2 while the symbol of zero-multiplicity is stored in SR3. The followed steps are executed thereafter.

- Multiply the contents of SR2 and SR3:  $\alpha^3 \cdot \alpha^6$ , by the multiplier M2. This product gives the evaluation of  $\alpha^3 \cdot x^6$  in  $\alpha$ , for the contribution of the first differing symbol  $z_n$  ( $\alpha^3$ ) to the first syndrome (evaluated on  $x = \alpha$ ).

- Perform a shift on SR3 by one position (right) to change the value of the output register from the  $z_n$  of the first symbol ( $\alpha^3$ ) to the  $z'_n$  of the first symbol ( $\alpha^0$ ).
- Multiply the values of the outputs of SR2 and SR3:  $\alpha^0 \cdot \alpha^6$ , (using the multiplier M2) to obtain  $\alpha^0 \cdot x^6$ , for the contribution of the first differing symbol  $z'_n$  ( $\alpha^0$ ) to the first syndrome (evaluated on  $x = \alpha$ ).
- Shift the contents of SR1 and SR2 one position (right) to change the value of the output from  $\alpha^6$  to  $\alpha^2$ . This changes the evaluation of  $x$  from the first differing symbol position ( $x^6$ ), to the second differing symbol position ( $x^2$ ). Multiply the outputs of SR1 and SR2 with M3, before the shift, to compute:  $\alpha^6 \cdot \alpha^6$ , and store the result in SR2. The stored value provides the second syndrome (evaluated on  $x = \alpha^2$ ) of the first differing symbol, at the position of  $x^6$ . Shift the content of SR3 one position (right) to change the value of the output register from the  $z'_n$  of the first symbol ( $\alpha^0$ ) to the  $z_n$  of the second symbol ( $\alpha^5$ ).
- Multiply (using M2) the values of the output registers SR2 and SR3:  $\alpha^5 \cdot \alpha^2$ , to find  $\alpha^5 \cdot x^2$ , for the contribution of the second differing symbol  $z_n$  ( $\alpha^5$ ) to the first syndrome (evaluated on  $x = \alpha$ ).
- Shift the content of SR3 by one position (right) to change the value of the output register from the  $z_n$  of the second symbol ( $\alpha^5$ ) to the  $z'_n$  of the second symbol ( $\alpha^3$ ).
- Multiply with the values of the output registers SR2 and SR3: ( $\alpha^3 \cdot \alpha^2$ , (using M2) for the evaluation of  $\alpha^3 \cdot x^2$ , to estimate the contribution of the second differing symbol  $z'_n$  ( $\alpha^3$ ) to the first syndrome (evaluated on  $x = \alpha$ ).
- Shift the contents of SR1 and SR2 one position (right) to change the value of the output from  $\alpha^2$  to  $\alpha^6$  in SR1 and from  $\alpha^2$  to  $\alpha^{12}$  in SR2. The change of SR2 allows us to make the evaluation of  $x$  from the second differing symbol position ( $x^2$ ), to the first differing symbol position ( $x^6$ ), but this time on the second syndrome. Multiply the outputs of SR1 and SR2 (using M3), before the shift, to compute:  $\alpha^2 \cdot \alpha^2$  and store it in SR2. The stored value provides the second syndrome (evaluated on  $x = \alpha^2$ ) of the second differing symbol, at the position of  $x^2$ . Perform a shift on SR3 by one position (right) to change the value of the output register from the  $z'_n$  of the second symbol ( $\alpha^3$ ) to the  $z_n$  of the first symbol ( $\alpha^3$ ).
- Multiply the values of the output registers SR2 and SR3:  $\alpha^3 \cdot \alpha^{12}$  (using M2), to find  $\alpha^3 \cdot x^6$ , which gives the contribution of the first differing symbol  $z_n$  ( $\alpha^3$ ) to the second syndrome (evaluated on  $x = \alpha^2$ ).
- Shift the content of SR3 one position (right) to change the value of the output register from the  $z_n$  of the first symbol ( $\alpha^3$ ) to the  $z'_n$  of the first symbol ( $\alpha^0$ ).

- Multiply the output of the registers SR2 and SR3:  $\alpha^0 \cdot \alpha^{12}$ , to find  $\alpha^0 \cdot x^6$ , the contribution of the first differing symbol  $z'_n$  ( $\alpha^0$ ) to the second syndrome (evaluated on  $x = \alpha^2$ ).
- Shift the content of SR2 by one position (right) to change the value of the output from  $\alpha^{12}$  to  $\alpha^4$ . This changes the evaluation of  $x$  from the first differing symbol position ( $x^6$ ), to the second differing symbol position ( $x^2$ ), this time in the second syndrome (evaluated on  $x = \alpha^2$ ). Shift the content of SR3 too by one position (right) to change the value of the output register from the  $z'_n$  of the first symbol ( $\alpha^0$ ) to the  $z_n$  of the second symbol ( $\alpha^5$ ).
- Multiply the outputs of SR2 and SR3:  $\alpha^5 \cdot \alpha^4$  (using M2) to find  $\alpha^5 \cdot x^2$ , the contribution of the second differing symbol  $z_n$  ( $\alpha^5$ ) to the second syndrome (evaluated on  $x = \alpha^2$ ).
- Make the last shift of SR3 by one position (right) to change the value of the output register from the  $z_n$  of the second symbol ( $\alpha^5$ ) to the  $z'_n$  of the second symbol ( $\alpha^3$ ).
- Multiply the contents of SR2 and SR3:  $\alpha^3 \cdot \alpha^4$ , to find  $\alpha^3 \cdot x^2$ , the contribution of the second differing symbol  $z'_n$  ( $\alpha^3$ ) to the second syndrome (evaluated on  $x = \alpha^2$ ).

As a result the contribution of all the differing symbols to the pair of syndromes is obtained for the hard decision and the second best decision.

### Total computation

To combine the results of the partial computation of syndromes obtained for the common symbols and the contribution of the differing symbols a  $2t \cdot 2^\eta$  matrix of registers is required for  $2t$  syndromes per test-vector of LCC. At the end of the computations pertaining to the common symbols, the values stored at the registers of Fig.4.3 are loaded into the registers of Fig.4.5 (the wires  $sc_1$ ,  $sc_2$  and  $sc_{2t}$  of Fig.4.3 are connected to the wires with the same name in Fig.4.5). In addition, the wire marked  $S_{out}$  (right side register of Fig.4.4) is connected to the left input with the same name as Fig.4.5. The function of this block is described next.

After the result of the common syndrome-computation is loaded, the contribution of the hard-decision or the second hard-decision of the  $\eta$  different symbols is added to the partial syndrome of common symbols stored in the matrix of registers, as explained in Example 4.2.3.

As a generic example, if we consider the case of  $\eta = 3$ , there are six different symbols: three corresponding to the hard-decision ( $z_\zeta$ ,  $z_\theta$  and  $z_\xi$ ) and three corresponding to the second best decision ( $z'_\zeta$ ,  $z'_\theta$  and  $z'_\xi$ ). As shown in Table 4.1,



during the first clock cycle, after the result of common computation is loaded (at clock cycle 0), the contribution of the first differing symbol hard-decision,  $z_\zeta$  (input  $S_{out}$ ), is added to the values of registers  $S_1^0, S_1^2, S_1^4$  and  $S_1^6$ , and the content of the rest of the registers remain the same since they are disabled. During the next cycle, (cycle 1) the second best-decision contribution ( $z'_\zeta$ ) is added to  $S_1^1, S_1^3, S_1^5$  and  $S_1^7$  registers. The same procedure is applied to the symbols  $z_\theta, z_\xi, z'_\theta$  and  $z'_\xi$  according to the schedule of Table 4.1, until the combinations shown on the summary column are achieved.

These combinations are equivalent to the values of the test-vectors and allow us to compute the syndrome  $S_1$  for all the vectors without computing twice the value of the common symbols. As each row of the matrix of registers of Fig.4.5 behaves as a shift-register, the rest of syndromes ( $S_2 \dots S_{2t}$ ) are computed by shifting the values from one register to another. This reduces the requirement of the number of adders.

**Example 4.2.3** In Example 4.2.1 we have shown the computation of partial syndromes  $sc_1 = \alpha^4$  and  $sc_2 = \alpha^3$ . In Example 4.2.2 we have shown the computation of the contribution of the differing symbols: for the first syndrome ( $\alpha^3 \cdot \alpha^6, \alpha^0 \cdot \alpha^6, \alpha^5 \cdot \alpha^2, \alpha^3 \cdot \alpha^2$ ) and for the second syndrome ( $\alpha^3 \cdot \alpha^{12}, \alpha^0 \cdot \alpha^{12}, \alpha^5 \cdot \alpha^4, \alpha^3 \cdot \alpha^4$ ). Finally, we combine these results obtained in Examples 4.2.1 and 4.2.2 as follows to obtain the four test vectors syndromes: First syndrome for the test vector  $z_{n0}, z_{n1}$ :

- $S_1^0 = sc_1 + \alpha^3 \cdot \alpha^6 + \alpha^5 \cdot \alpha^2$

First syndrome for the test vector  $z_{n0}, z'_{n1}$ :

- $S_1^1 = sc_1 + \alpha^3 \cdot \alpha^6 + \alpha^3 \cdot \alpha^2$

First syndrome for the test vector  $z'_{n0}, z_{n1}$ :

- $S_1^2 = sc_1 + \alpha^0 \cdot \alpha^6 + \alpha^5 \cdot \alpha^2$

First syndrome for the test vector  $z'_{n0}, z'_{n1}$ :

- $S_1^3 = sc_1 + \alpha^0 \cdot \alpha^6 + \alpha^3 \cdot \alpha^2$

Second syndrome for the test vector  $z_{n0}, z_{n1}$ :

- $S_2^0 = sc_2 + \alpha^3 \cdot \alpha^6 + \alpha^5 \cdot \alpha^4$

Second syndrome for the test vector  $z_{n0}, z'_{n1}$ :

- $S_2^1 = sc_2 + \alpha^3 \cdot \alpha^6 + \alpha^3 \cdot \alpha^4$

Second syndrome for the test vector  $z'_{n0}, z_{n1}$ :

- $S_2^2 = sc_2 + \alpha^0 \cdot \alpha^6 + \alpha^5 \cdot \alpha^4$

Second syndrome for the test vector  $z'_{n0}, z'_{n1}$ :

- $S_2^3 = sc_2 + \alpha^0 \cdot \alpha^6 + \alpha^3 \cdot \alpha^4$

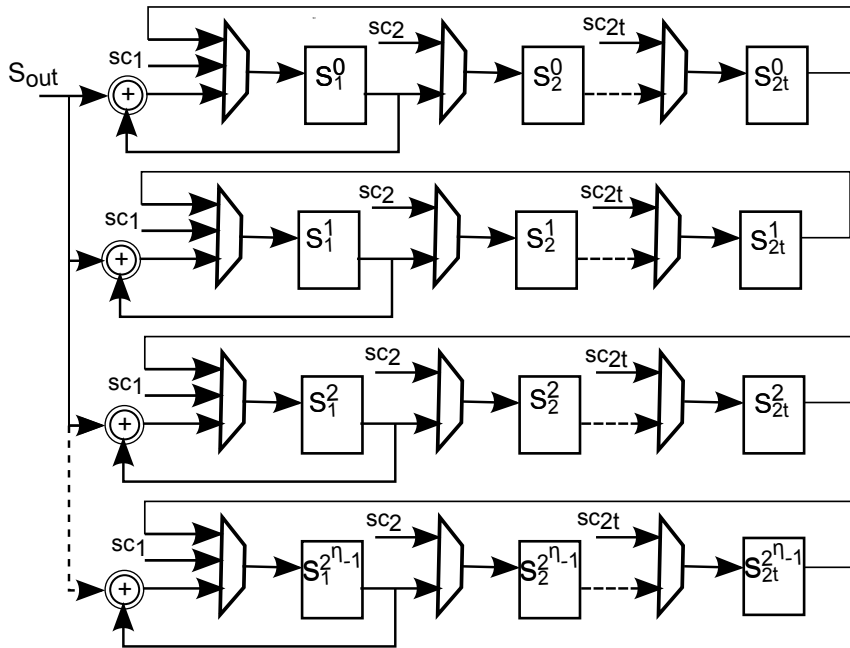


Figure 4.5: Storage of the  $2t$  syndromes of the  $2^n$  test-vectors

**Table 4.1:** Timing diagram of the syndrome processing for eight test-vectors

Clock cycle	0	1	2	3	4	5	Summary
$S_1^0$	$z_\zeta$	Disable	$z_\theta$	Disable	$z_\xi$	Disable	HD-HD-HD
$S_1^1$	Disable	$z'_\zeta$	$z_\theta$	Disable	$z_\xi$	Disable	HD-HD-2HD
$S_1^2$	$z_\zeta$	Disable	Disable	$z'_\theta$	$z_\xi$	Disable	HD-2HD-HD
$S_1^3$	Disable	$z'_\zeta$	Disable	$z_\theta$	$z_\xi$	Disable	HD-2HD-2HD
$S_1^4$	$z_\zeta$	Disable	$z_\theta$	Disable	Disable	$z'_\zeta$	2HD-HD-HD
$S_1^5$	Disable	$z'_\zeta$	$z_\theta$	Disable	Disable	$z'_\zeta$	2HD-HD-2HD
$S_1^6$	$z_\zeta$	Disable	Disable	$z'_\theta$	Disable	$z'_\zeta$	2HD-2HD-HD
$S_1^7$	Disable	$z'_\zeta$	Disable	$z_\theta$	Disable	$z'_\zeta$	2HD-2HD-2HD

### 4.2.3 Key equation solver unit

The next unit of the proposed decoder is the KES. As shown in Algorithm 14, there are two parts in the iBM algorithm: the discrepancy computation (Step A1) and the updating of the polynomials (Step A2). Due to the data dependency, it is not possible to compute A1 and A2 at the same time; therefore, A2 must be computed after A1. As the number of test-vectors is always an even number for any  $\eta$ , two test-vectors can use the same KES unit. Then, the discrepancy of odd test-vectors can be computed while the polynomials of the even vectors are updated and vice versa. This shared architecture allows us to reduce the number of multipliers and adders in the KES unit.

To improve the throughput of the decoder, the matrix of syndrome registers (Fig.4.5) is duplicated to save the syndrome results from a message in two identical storage units. This will help to compute the syndromes of the next message concurrently with KES computation of the previous message. The proposed architecture is based on the parallel form of the iBM algorithm [78], which we have modified to share the arithmetic resources for computing two KES at a time. As shown in Fig.4.6, two test-vectors are computed at the same time. Multiplexors make a selection of the even or the odd test-vectors. When the discrepancy  $\delta$  of an even test-vector is calculated, the updating of the odd  $\lambda(x)$  is computed simultaneously, and vice versa. The polynomials  $A(x)$  and  $\lambda(x)$  require  $t + 1$  registers for each test-vector, the  $\omega(x)$  coefficients require  $t$  registers for each test-vector and the discrepancy  $\delta$  need one more register per test-vector. The number of multipliers and adders is three in each KES unit, and  $2^{\eta-1}$  KES units are used to compute the iBM algorithm for the  $2^\eta$  test-vectors.

On the other hand, the reason to not use the UiBM [79] is that although requires one multiplier less than our proposed architecture, the number of registers is  $2t$  higher. Since, the proposed architecture is based on multiplexing of KES computation of the test-vectors, the number of arithmetic elements (as multipliers) is reduced to half, unlike the storage elements (registers), which remain the same. Since we have  $2^\eta$  test-vectors,  $2^{\eta-1}$  multipliers are saved with the UiBM, but  $2t \cdot 2^\eta$

more registers are required because it works with polynomials of bigger order. For example, for a (255,239)RS code with  $\eta = 3$  applying the proposed scheme based on iBM instead of the UiBM, involves seven times less *XOR* gates, without increase in latency.

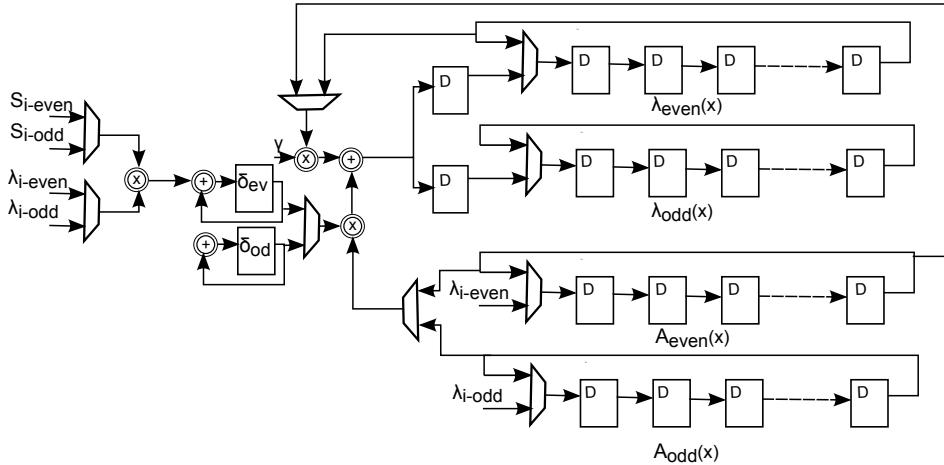


Figure 4.6: KES unit for computing two test-vectors

#### 4.2.4 Chien search, Forney and test-vector selection units

After  $\lambda(x)$  and  $\omega(x)$  are computed  $2^n$  Chien searches are performed in parallel. The conventional architecture based on  $t$  constant multipliers and  $t$  adders are replicated  $2^n$  times to extract the locations of the errors and the number of roots that are used in DF detection. For each test-vector  $t + 1$  registers are used to store  $\lambda(x)$ ,  $t$  registers are used to store  $\omega(x)$ ,  $t$  registers are used for the evaluation of  $\lambda'(x)$  on the error symbols,  $t$  registers are used to store the locations of the errors and  $t$  multiplexors are used to select between the desired coefficients of the polynomials to evaluate  $\lambda_j$  and the power of  $\alpha$  used to perform the computation of the Horner's recursions (Fig.4.7). Due to the properties of the formal derivative of a polynomial whose coefficients are GF elements,  $\lambda'(x)$  evaluation does not need extra arithmetic resources, as this value can be calculated with the evaluation of the even coefficients of  $\lambda(x)$  [78]. However an extra counter is needed for each test-vector, placed after Chien search. Each counter increases the count with every root of  $\lambda(x)$ . The maximum number of the count is  $t$ . A comparator checks if the degree of  $\lambda(x)$  is the same as the number of roots of  $\lambda(x)$  and generates the DF signals.

Only one Forney block is needed by the proposed architecture, Fig.4.8. In the Forney block, the locations of the test-vector free of DF are loaded in  $t$  registers. The evaluation of  $\lambda'(x)$  in these locations are stored in  $t$  additional registers and one Chien search for the selected  $\omega(x)$  is performed. This Chien search needs  $t$  constant multipliers,  $t$  adders and  $t$  registers. To match the corresponding evaluation of  $\omega(x)$  with the stored evaluation of  $\lambda'(x)$ , the error locations are compared with a counter that counts all the possible locations in the polynomial (at the same time the Chien search for the selected  $\omega(x)$  is performed). A control signal is activated when the counter and one of the stored locations are the same, showing that the associated value of evaluation of  $\lambda'(x)$  to this location must be read to obtain a correct match with the evaluation of  $\omega(x)$ . A multiplier, which makes the products between the correct  $\lambda'(x)$  evaluations and the  $\omega(x)$  ones, is needed to compute the magnitudes of the errors. Finally, the addition of these magnitudes to the received message is made with another adder.

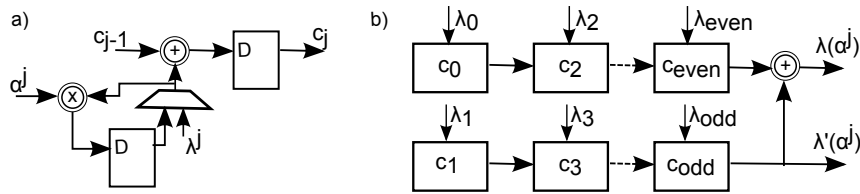


Figure 4.7: (a) Basic  $c_j$  block. (b) Chien search block diagram

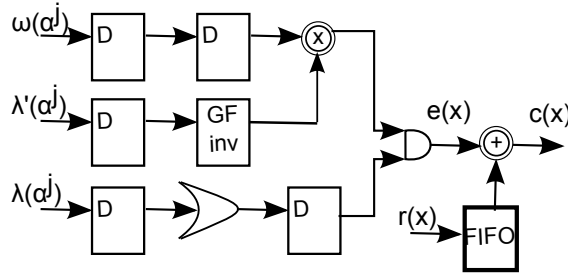


Figure 4.8: Forney's algorithm classical architecture

### 4.2.5 Area and timing complexity analysis

The syndrome computation latency is equal to the sum of the latency of the partial syndrome computation of one of the common symbols and the latency of the contribution to the syndromes of the differing symbols. The computation of the syndromes for the common symbols takes  $N + 1$  clock cycles, because all the  $N$  symbols of the received message must be processed and there is a delay of one cycle caused by the input register. It is important to remark that the storage of the  $\eta$  unreliable symbols is done during the common computation. As it was explained in previous subsections, every time that a zero multiplicity is detected in the input, the hard decision and the second best decision of this symbol is stored in parallel on the SR3 (Fig.4.4), avoiding the increase of latency during this step. In addition, the multiplier M1 (Fig.4.4) starts to compute locations during the common computation, and stores the locations of the differing symbols in SR1 and SR2 (Fig.4.4) each time a zero multiplicity is detected. Hence, the generation of the locations for the differing symbols is done in parallel with the common computation, so it does not increase latency on the common symbols or the differing symbols computations. The computation pertaining to the differing symbols involves  $2\eta$  contributions per syndrome for  $2t$  syndromes. It introduces a latency of  $2t \cdot 2(\eta + 1)$ . Note that two cycles of additional latency per syndrome are introduced due to the pipeline registers included to maintain critical path limited to the sum of one multiplier, one adder and one multiplexor. Therefore the total latency of the syndrome computation unit is  $N + 1 + 2t \cdot 2(\eta + 1)$ .

KES units require  $t + 1$  coefficients for the discrepancy computation and  $t + 1$  coefficients must be updated during  $2t$  iterations. Hence,  $2(t + 1) \cdot 2t$  clock cycles are required for the computation of one value of  $\lambda(x)$ . Since the resources during the KES for the odd and even test vectors computations are shared, the last  $t + 1$  coefficients of the odd (or even) polynomials cannot be multiplexed, hence extra cycles are needed to complete the computation of odd (or even)  $\lambda(x)$ . In addition, the KES unit must perform  $t$  additional iterations to compute the  $\omega(x)$ . During the calculation of the coefficients for  $\omega(x)$  only the discrepancy computation must be run and there is no need to update the coefficients. Therefore, the last iteration requires  $t - 1$  cycles, resulting a latency of  $t \cdot 2(t + 1) - (t + 1)$  cycles. The total latency of KES unit is  $[2(t + 1) \cdot 2t + t + 1 + t \cdot 2(t + 1) - (t + 1)] = 6t \cdot (t + 1)$ . The latency of the Chien search is  $K + 1$  cycles. The latency Forney's algorithm equals to the length of the message  $N + 1$ .

For our implementation of a (255,239)RS code,  $t = (255 - 239)/2 = 8$ ,  $N = 255$  and  $K = 239$ . In addition, the latency of the proposed LCC architecture is analyzed for  $\eta = 3$  and  $\eta = 4$  because for high-rate codes these are common values. With  $\eta > 4$  coding gain compared to  $\eta = 4$  is negligible and complexity is prohibitive for hardware implementations. With  $\eta < 3$  coding gain is not remarkable and does not justify and increase of area. Therefore, the latency of different blocks could be evaluated as follows:

(i) Syndrome computation:  $N + 1 + 2t \cdot 2(\eta + 1) = 384$  clock cycles with  $\eta = 3$  and  $N + 1 + 2t \cdot 2(\eta + 1) = 416$  clock cycles with  $\eta = 4$

(ii) KES:  $6t \cdot (t + 1) = 432$  clock cycles

(iii) Chien search:  $K + 1 = 240$  clock cycles

(iv) Forney's algorithm:  $N + 1 = 256$  clock cycles

Since the block with the maximum latency establishes the latency of the decoder, latency of the complete LCC decoder with  $\eta = 3$  and  $\eta = 4$  is 432 clock cycles, which is the same as the one of KES block. Interestingly, the critical path of the LCC decoder is one multiplier, one adder and one 2-to-1 multiplexor regardless of the value of  $t$  or  $\eta$ . Therefore, the maximum usable frequency remains invariant of  $t$  and  $\eta$ .

On the other hand, Table 4.2 includes the hardware resources required by the proposed decoder. As it can be seen, area increases exponentially with  $\eta$  and linearly with  $t$ . The proposed arithmetic and control resources of this LCC architecture are independent from  $N$  and  $K$ . Just the memories required to store the received sequence between the different processors depend on the length of the codeword.

#### 4.2.6 FPGA and ASIC results

The proposed LCC architecture for a (255, 239)RS code with  $\eta = 3$  has been modelled in VHDL and implemented in a Virtex-V FPGA device. It is found to involve 5399 bit-registers and 5114 LUTs in 2527 slices. The maximum clock frequency is found to be 150.5MHz which is equivalent to a throughput of  $(255 \times 8 \times 150.5)/432 = 710.69$ Mbps.

The same VHDL model of the proposed decoder has been synthesized with the Synopsis tool as well, using the SAED 90nm standard cell library. The synthesis result shows that the proposed decoder has area equivalent to 31138 XOR gates and the data arrival time of 1.24ns. Since, the proposed decoder for  $\eta = 3$  has latency of 432 clock cycles; we can estimate its throughput to be  $(255 \times 8)/(432 \times 1.124ns) = 4.2$ Gbps for ASIC implementation.

**Table 4.2:** Theoretical estimation of the area-complexity of the proposed LCC decoder

	Syndromes	KES	Chien Search	Forney's algorithm	Total ( $\eta = 3$ )	Total ( $\eta = 4$ )
GF Mult	2	$3 \cdot 2^{\eta-1}$	0	1	15	30
GF Adders	$2t + 2^\eta$	$3 \cdot 2^{\eta-1}$	$t \cdot 2^\eta$	$t + 1$	109	193
GF Const Mult	$2t + 1$	0	$t \cdot 2^\eta$	$t$	89	153
GF Inv	1	0	0	1	2	2
Reg.(Byte)	$2t \cdot (2^\eta + 1) + 4\eta + 3$	$(5t + 3) \cdot 2^\eta$	$(4t + 1) \cdot 2^\eta$	$3t$	783	1511
MUX 2-1(Byte)	$4t \cdot 2^\eta + 3$	$9 + 2^{\eta-1}$	$t \cdot 2^\eta$	$t$	255	499
RAM(Byte)	$N$	$N$	$N$	0	765	765

### 4.3 Comparisons with other works

In this subsection we compare the area-complexity of the proposed decoder architecture with other efficient decoders found in literature. All the architectures compared in this subsection are for a (255,239)RS code with  $\eta = 3$ . In Table 4.3 we have listed the complexity of Galois field multiplier, adder and inverter in terms of their equivalent *XOR* and *AND* gate-counts extracted from [64], [70], [73], [85], [86].

When the architecture and the algorithm described in this chapter were submitted for publication at the end of 2010, the most efficient LCC decoders were based on interpolation methods. To the best knowledge of the author, the architecture for a full LCC decoder based on the backward interpolation implementation of [64] proposed in [70] and a free-factorization architecture of LCC in [85] were the implementations with better area over ratio results. For this reason they are taken as references for comparison.

Comparing the results from [70] with the architecture of the proposed LCC decoder (Table 4.4), we find that the proposed architecture involves  $(48479 - 38520)/48479 = 20.5\%$  less area than the LCC re-encoded decoder with  $\eta = 3$  based on backward interpolation of [64]. Compared with the free-factorization based design of [85] with  $\eta = 3$ , the area of our proposed decoder is  $(39311 - 38520)/39311 = 2.1\%$  less, for the same (255,239)RS code.

The area-delay product of factorization-free decoder of [85] amounts to  $(39311 \times 660)$  units (because latency is 660 clock cycles and the estimated area is equivalent to 39311 *XOR* gates); whereas the area-delay product of our proposed decoder is  $(38520 \times 432)$  units. Since, the duration of clock periods are the same in both the structures, the proposed decoder provides nearly  $(39311 \times 660)/(38520 \times 432) - 1 = 56\%$  saving in area-delay product over that of [85]. Similarly, the area-delay products of re-encoded decoder is  $(48479 \times 660)$  units (because latency is 660 clock cycles and the estimated area is equivalent to 48479 *XOR* gates). The proposed



**Table 4.3:** Equivalence in *XOR* gates of hardware resources

GF(2 <sup>8</sup> ) Multipliers	GF(2 <sup>8</sup> ) Const. Mult.	GF(2 <sup>8</sup> ) Adders	Register (Bit)	Mux 2-1 (Bit)	RAM (Bit)	GF(2 <sup>8</sup> ) Inv	AND Gate
100 <i>XORs</i>	15 <i>XORs</i>	8 <i>XORs</i>	3 <i>XORs</i>	1 <i>XOR</i>	1 <i>XORs</i>	150 <i>XORs</i>	3/4 <i>XORs</i>

decoder therefore provides a saving of  $(48479 \times 660)/(38520 \times 432) - 1 = 92.3\%$  over the re-encoded decoder of [70].

After the publication of this proposal in 2011, several re-encoders and interpolation processor have been proposed, however just two complete LCC decoders have been published: i) the decoder based on interpolation and systematic re-encoder from [86]; and ii) the decoder in [87], which is based on the algorithm and architecture proposed in this chapter, and improves its performance. If we compare our proposal in terms of area-delay (or area over ratio efficiency) we obtain that it is  $(38520 \times 432)/(18948 \times 363) - 1 = 142\%$  less efficient than the one in [86]. Compared to [87], our proposal is  $(38520 \times 432)/(20233 \times 275) - 1 = 199\%$  less efficient. In addition, it is important to make comparisons between the improved decoder based on the LCC scheme from this thesis [87] and the decoder based on interpolation from [86]. In terms of efficiency, the decoder architecture for LCC based on HDD is  $(18948 \times 363)/(20233 \times 275) - 1 = 24\%$  more efficient than the one based on interpolation. Moreover, latency is reduced in a  $363 - 275/275 = 32\%$  with the architecture based on HDD, and as consequence throughput is 32% higher. Finally, it is remarkable that the architecture derived from HDD has almost achieved on-the-fly decoding, as each clock cycle one symbol is received and there are 255 symbols and the decoder has a latency of 275 clock cycles, so it is introducing just a delay of 15 clock cycles. The architecture that outperforms our proposal from this chapter introduces a more efficient processor to compute syndromes of the test-vectors. The architecture proposed in [87] obtains a throughput of 1.1Gbps on a Virtex V device, which is  $1.1\text{Gbps}/0.710\text{Gbps} = 1.55$  times more throughput than the result obtained in Section 4.2.6 for the same device.

Finally, we compare our LCC proposal with a conventional hard-decision decoder based on BM [85], included in Table 4.5. Both decoders have similar latency in terms of clock cycles with a slight improvement of  $(510 - 432)/510 = 15.3\%$  in our proposal. In terms of area, the soft-decision decoder requires just  $30959/14640 = 2.1$  times more area than the hard-decision one. This last result shows the efficiency of the architecture, since LCC decoder processes eight hard decision test-vectors, but only requires twice more hardware resources keeping similar latency. To sum up, we can conclude that LCC decoder represents an efficient way to obtain some extra coding gain keeping area resources relatively low with high throughput.

**Table 4.4:** XOR-gate complexity of LCC decoders for (255,239) RS code

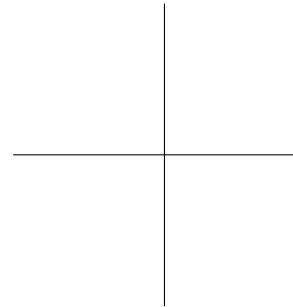
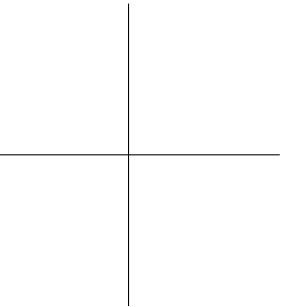
LCC decoder	GF(2 <sup>8</sup> ) Mult./ Const. Mult.	GF(2 <sup>8</sup> ) Inv.	GF(2 <sup>8</sup> ) Adders	MUX 2-1 (Bit)	RAM (Bits)	ROM (Bits)	Reg. (Bits)	Total xors	Latency clock cycles
[70], [64]	10200	0	1008	1317	12288	16928	6738	48479	660
[85]	7400	0	864	1109	8192	16928	4818	39311	660
[86]	2700/2340	2	1432	752	6736	704	3984	18948	363
[87]	5100/2415	1	1400	656	4080	0	6432	20233	275
This work	1500/1335	2	872	2040	6120	0	18792	30959	432

**Table 4.5:** XOR-gate complexity of a hard-decision decoder for (255,239) RS code

BM decoder	GF(2 <sup>8</sup> ) Mult.	GF(2 <sup>8</sup> ) Inv.	GF(2 <sup>8</sup> ) Adders	MUX 2-1 (Bit)	RAM (Bits)	ROM (Bits)	Reg. (Bits)	Total xors	Latency clock cycles
[85]	19	0	17	165	4096	6144	733	14640	510

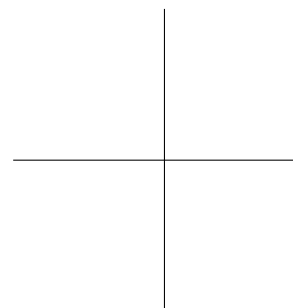
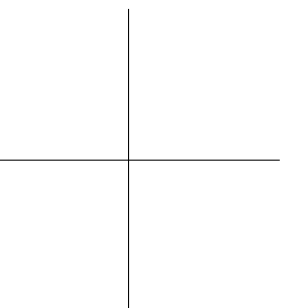
## 4.4 Conclusions

In this chapter, we propose a new scheme to implement low complexity Chase decoding. This new algorithm avoids complex solutions such as interpolation, re-encoder and factorization and makes use of hard-decision techniques to decode the different test-vectors. In addition, an algorithm based on binary tree mapping is applied in the syndrome computation to reduce complexity and the total number of operations. Moreover, a method to select the test-vectors based on decoding failure is also implemented in order to choose the correct decoded codeword. This algorithm does not introduce any performance loss and allows us to derive efficient high-throughput architectures. The architecture proposed in this chapter was modified by other authors obtaining lower area and higher-throughput. The improved architecture based on our algorithm shows that low complexity Chase decoding combined with hard decision decoding techniques can reach higher throughput and higher efficiency than interpolation based architectures. Hence, we can conclude that the algorithm proposed in this chapter outperforms interpolation methods from a VLSI perspective, without any performance degradation.



## Part III

# Conclusion and future works





## Chapter 5

# Conclusion and future works

The objective of this thesis was the design of high speed non-binary decoder architectures based on both NB-LDPC and Reed-Solomon codes, and in particular for high-rate codes. In this chapter the main conclusions of the thesis are exposed and the future lines of research are drawn.

### 5.1 Conclusions

In the first part of this manuscript, focused on NB-LDPC codes, we conclude that algorithms derived from Qary Sum-Product, like EMS and Min-max, involve high complexity. The most efficient hardware architectures that can be derived from the existing algorithms suffer from an extremely high number of clock cycles per iteration, which leads to low speed solutions. Other alternatives with higher degree of parallelism, such as the ones based on trellis, yield to very dense routing and large silicon area. Moreover, high-rate codes have an added difficulty to reach good area over speed ratio. For architectures based on algorithms such as forward-backward or bubble check, throughput decreases with the degree the check node,  $d_c$ , and  $d_c$  is very large for codes with rates over 0.8. For architectures based on parallel trellis area and routing increases also with  $d_c$ , turning them into inefficient solutions for high-rate codes over fields above GF(16).

The main objective of the research in this area is the proposal of alternatives to Qary Sum-Product, EMS and Min-max with lower complexity and better area over speed ratio. The most interesting candidates are symbol-flipping algorithms and in particular, for the high-rate codes, Generalized Bit-Flipping algorithm (GBFDA). Although the complexity of the decoder also depends on  $d_c$ , its check node performs only hard-decision operations that simplifies by  $q$  routing and area, being  $q$  the

size of the Galois Field. However, a direct mapping architecture of GBFDA suffers from elevated number of memory resources and reduced throughput, as GBFDA requires two times more iterations to converge. In addition, GBFDA has a performance loss of around 0.7dB compared to EMS algorithm. Hence, the aim of this part of the work was: i) reduce area requirements, especially the ones due to memories; ii) increase throughput; and iii) increase the coding gain of symbol flipping algorithms.

First, a modified tentative decoding algorithm for GBFDA (M-GBFDA) was proposed in order to reduce the amount of storage resources for a partial parallel architecture based on flooding schedule. This method consists in computing the hard-decision at the tentative decoding based on the extrinsic information, introducing a negligible performance loss of 0.05dB. For example, for the (837,723) NB-LDPC code, this modification on the algorithm reduces in 22% the total area compared to the original GBFDA.

On the other hand, throughput for M-GBFDA architecture is limited when clipping is applied as the technique to control the data growth. For the partial parallel architecture based on flooding schedule, a parallel implementation of the clipping would require too much area, so serial implementation is the only efficient solution. Unfortunately, serial implementation of clipping reduces in a factor  $q$  the throughput. To avoid clipping, an alternative data growth control technique, named blocking, was proposed. This technique not only increases throughput but also reduces complexity avoiding comparisons and subtractions. For a high-rate code over GF(32), the throughput for a partial parallel architecture based on M-GBFDA using blocking is nine times the one of clipping. The drawback of this solution is that introduces an error floor around a bit error rate of  $10^{-7}$ , which is enough for wireless communications but it makes mandatory the use of concatenation with other codes for systems such as optical communications.

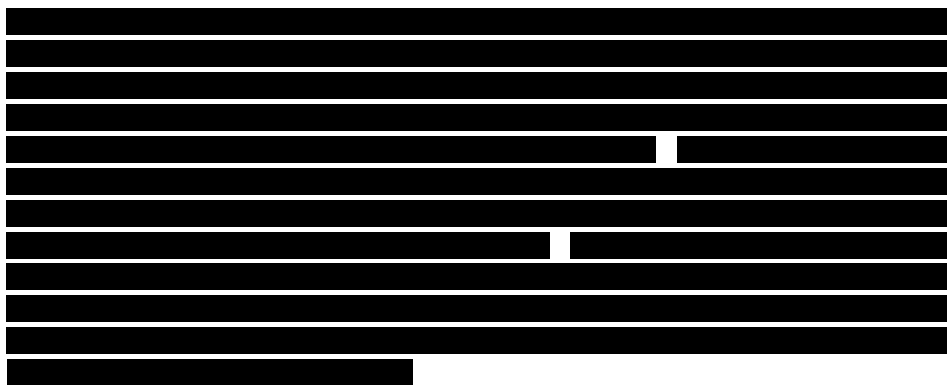
Comparing post place and route reports with synthesis reports we detected that there were differences in the maximum frequency achievable due to the excess of routing. To reduce routing and hence increase throughput we explored broadcasting techniques in order to reach similar results in both synthesis and post place and route processes. While half-broadcasting increases at least 1.4 times the throughput for a high-rate code over GF(32) when ASIC technology is applied, if the same design is routed in an FPGA device, the increase of speed is not good enough to balance the area overhead caused by this technique.

For the architectures based on M-GBFDA with flooding schedule we conclude that: i) the one based on clipping has a performance similar to GBFDA (only 0.05dB of difference) but its throughput is very reduced, even applying half-broadcasting; and ii) the one based on blocking has a very high throughput but suffers a performance degradation which introduces error floor. Based on this we decided to

explore serial schedule algorithms for symbol-flipping algorithms because they can reduce the number of iterations and therefore increase the throughput.

The first serial algorithm that was proposed, ES-GBFDA, has two improvements compared to a conventional layered/serial scheduling: i) it accumulates all the information from the previous iterations achieving a better convergence; and ii) stores the intrinsic information from the voting process in a separate variable from the channel information, reducing the number of bits required for the quantize model. For applying the serial update, ES-GBFDA reduces to half the number of iterations required. The derived architecture reduces memory requirements compared to M-GBFDA and reaches similar throughput to M-GBFDA with blocking, but without introducing any performance degradation, because implements a parallel version of clipping.

As the parallel version of clipping reduces the maximum frequency achievable and increases the number of arithmetic resources, a different initialization is proposed to avoid clipping. Furthermore, a second serial algorithm is proposed. This algorithm, named SES-GBFDA, is a simplified version of the first one, and only stores the information from the previous iteration, but scales the votes according to some statistical parameters. This scaling emulates the effect of storing the votes of all the iterations but at the same time reduces the number of bits required to quantize the intrinsic information to one. The architecture for SES-GBFDA has higher throughput than M-GBFDA and ES-GBFDA architectures, due to the alternative initialization and the reduction of wiring to one bit quantization; this proposal also reduces the total area to half compared to the architecture based on ES-GBFDA. SES-GBFDA architecture gets higher throughput and higher efficiency in terms of area over speed than previous solutions found in literature. We can conclude that with SES-GBFDA we have accomplished the objectives of getting a NB-LDPC with a very reduced area and a reasonable high throughput.



In the second part of the thesis, soft-decision Reed-Solomon algorithms are evaluated concluding that LCC is the most efficient one if high-rate codes and high

speed are the targets. Compared to other efficient soft-decision algorithms such as Koetter-Vardy's one, LCC reaches similar coding with lower number of operations. However, to the best knowledge of the author all the hardware approaches of LCC until the proposal included in this manuscript were based on interpolation and factorization, which still involves high complexity.

In this thesis, an LCC decoder based on hard-decision algorithms was designed. This decoder reduces the complexity of the soft-decision decoding for Reed-Solomon codes without introducing any performance degradation. A new architecture for this version of LCC decoding was also implemented, reaching high-throughput in a lower area than previous proposals. This architecture has been improved by other authors showing that the hard-decision scheme proposed here outperforms solutions based on interpolation in terms of hardware efficiency and under the same performance.

## 5.2 Future research lines

After the conclusions of the thesis, we think that some interesting future works in this area are:

For the NB-LDPC field:

- The design of binary LDPC decoders based on some of the improvements of symbols flipping for GBFDA in order to reduce more the complexity of these decoders.
- [REDACTED]
- [REDACTED]
- [REDACTED]
- The modification of Min-max check node update equations, inspired by T-EMS.
- [REDACTED]
- The implementation and study of NB-LDPC decoders combined with higher order modulations and different channels.

For Reed-Solomon soft-decision codes:



- The design of an LCC decoder based on FFT hard-decision decoders, as it have been shown that they can be implemented with half area and twice throughput.
- The proposal of a new algorithm similar to LCC based on the erasure decoders.
- The study of belief propagation algorithms for Reed-Solomon using NB-LDPC experience.

Finally, other works that involve both parts of the thesis are:

- Continue improving and modifying architectures and algorithms in order to achieve 25Gbps.
- Concatenate the different architectures of NB-LDPC decoders with the LCC for Reed-Solomon to see what is the best configuration in terms of coding gain and hardware efficiency for future optical communication systems.



# Bibliography

- [1] B. Zhou, J. Kang, S. Song, S. Lin, K. Abdel-Ghaffar, and M. Xu, "Construction of non-binary quasi-cyclic LDPC codes by arrays and array dispersions - [transactions papers]," *IEEE Transactions on Communications*, vol. 57, no. 6, pp. 1652–1662, June 2009.
- [2] R. Gallager, "Low-density parity-check codes," *Transactions on Information Theory, IRE*, vol. 8, no. 1, pp. 21–28, 1962.
- [3] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.
- [4] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.
- [5] 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, no. 3, pp. 404–412, 2002.
- [6] S.-H. Kang and I.-C. Park, "Loosely coupled memory-based decoding architecture for low density parity check codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 5, pp. 1045–1056, 2006.
- [7] Z. Wang, Z. Cui, and J. Sha, "VLSI design for low-density parity-check code decoding," *IEEE Circuits and Systems Magazine*, vol. 11, no. 1, pp. 52–69, 2011.
- [8] M. Davey and D. MacKay, "Low-density parity check codes over  $\text{GF}(q)$ ," *IEEE Communications Letters*, vol. 2, no. 6, pp. 165–167, June 1998.
- [9] L. Barnault and D. Declercq, "Fast decoding algorithm for LDPC over  $\text{GF}(q)$ ," in *Proceedings of the 2003 IEEE Information Theory Workshop*, April 2003, pp. 70–73.

- [10] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of LDPC codes over  $\text{GF}(q)$ ," in *2004 IEEE International Conference on Communications*, vol. 2, June 2004, pp. 772 – 776.
- [11] C. Spagnol, E. Popovici, and W. Marnane, "Hardware implementation of LDPC decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 12, pp. 2609 –2620, Dec. 2009.
- [12] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over  $\text{GF}$ ," *IEEE Transactions on Communications*, vol. 55, no. 4, pp. 633 –643, April 2007.
- [13] V. Savin, "Min-max decoding for non binary LDPC codes," in *IEEE International Symposium on Information Theory, 2008. ISIT 2008.*, July 2008, pp. 960 –964.
- [14] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Low-complexity decoding for non-binary LDPC codes in high order fields," *IEEE Transactions on Communications*, vol. 58, no. 5, pp. 1365 –1375, May 2010.
- [15] Y.-L. Ueng, K.-H. Liao, H.-C. Chou, and C.-J. Yang, "A high-throughput trellis-based layered decoding architecture for non-binary LDPC codes using max-log-QSPA," *IEEE Transactions on Signal Processing*, vol. 61, no. 11, pp. 2940–2951, 2013.
- [16] E. Boutillon and L. Conde-Canencia, "Bubble check: a simplified algorithm for elementary check node processing in extended min-sum non-binary LDPC decoders," *Electronics Letters*, vol. 46, no. 9, pp. 633 –634, April 2010.
- [17] X. Zhang and F. Cai, "Reduced-complexity decoder architecture for non-binary LDPC codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 7, pp. 1229 –1238, July 2011.
- [18] X. Chen, S. Lin, and V. Akella, "Efficient configurable decoder architecture for nonbinary quasi-cyclic LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 1, pp. 188 –197, Jan. 2012.
- [19] E. Li, K. Gunnam, and D. Declercq, "Trellis based Extended Min-Sum for decoding nonbinary LDPC codes," in *8th International Symposium on Wireless Communication Systems (ISWCS), 2011*, Nov. 2011, pp. 46 –50.
- [20] X. Ma, K. Zhang, H. Chen, and B. Bai, "Low complexity X-EMS algorithms for nonbinary LDPC codes," *IEEE Transactions on Communications*, vol. 60, no. 1, pp. 9 –13, January 2012.
- [21] D. Project. (2010, Jun.) Final publishable summary. [Online]. Available: [http://www.ict-davinci-codes.eu/project/deliverables/FR\\_www.pdf](http://www.ict-davinci-codes.eu/project/deliverables/FR_www.pdf)

- [22] A. Darabiha, A. Carusone, and F. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 1, pp. 74–78, Jan. 2008.
- [23] J. Lin, J. Sha, Z. Wang, and L. Li, "Efficient decoder design for nonbinary quasicyclic LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 5, pp. 1071–1082, May 2010.
- [24] F. Cai and X. Zhang, "Relaxed Min-Max decoder architectures for Nonbinary Low-Density Parity-Check codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2013.
- [25] E. Li, "Decodeurs haute performance et faible complexite pour les codes LDPC binaires et non-binaires," Ph.D. dissertation, l'Universite de Cergy-Pontoise, Ecole Nationale Superieure de l'electronique et de ses Applications, December 2012.
- [26] D. Zhao, X. Ma, C. Chen, and B. Bai, "A low complexity decoding algorithm for majority-logic decodable nonbinary LDPC codes," *IEEE Communications Letters*, vol. 14, no. 11, pp. 1062–1064, November 2010.
- [27] C. Chen, B. Bai, X. Wang, and M. Xu, "Nonbinary LDPC codes constructed based on a cyclic MDS code and a low-complexity nonbinary message-passing decoding algorithm," *IEEE Communications Letters*, vol. 14, no. 3, pp. 239–241, March 2010.
- [28] B. Zhou, J. Kang, Y. Y. Tai, Q. Huang, and S. Lin, "High performance non-binary quasi-cyclic LDPC codes on euclidean geometries," in *IEEE Military Communications Conference, 2007. MILCOM 2007.*, Oct. 2007, pp. 1–8.
- [29] B. Liu, J. Gao, G. Dou, and W. Tao, "Weighted symbol-flipping decoding for nonbinary LDPC codes," in *Second International Conference on Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010*, vol. 1, April 2010, pp. 223–226.
- [30] —, "Majority decision based weighted symbol-flipping decoding for nonbinary LDPC codes," in *2nd International Conference on Future Computer and Communication (ICFCC), 2010*, vol. 3, May 2010, pp. V3–6–V3–10.
- [31] S. El Hassani, M. Hamon, and P. Penard, "A comparison study of binary and non-binary LDPC codes decoding," in *International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2010*, 2010, pp. 355–359.
- [32] V. S. Ganepola, R. Carrasco, I. Wassell, and S. Le-Goff, "Performance study of non-binary LDPC codes over  $GF(q)$ ," in *6th International Symposium on Communication Systems, Networks and Digital Signal Processing, 2008. CNS-DSP 2008.*, 2008, pp. 585–589.

- [33] X. Zhang and F. Cai, "Efficient partial-parallel decoder architecture for quasi-cyclic nonbinary LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 2, pp. 402–414, 2011.
- [34] D. Chang, F. Yu, Z. Xiao, Y. Li, N. Stojanovic, C. Xie, X. Shi, X. Xu, and Q. Xiong, "FPGA verification of a single QC-LDPC code for 100 gb/s optical systems without error floor down to BER of  $10^{-15}$ ," in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference*, 2011, pp. 1–3.
- [35] M. Magarini, R. J. Essiambre, B. Basch, A. Ashikhmin, G. Kramer, and A. de Lind van Wijngaarden, "Concatenated coded modulation for optical communications systems," *IEEE Photonics Technology Letters*, vol. 22, no. 16, pp. 1244–1246, 2010.
- [36] I. Djordjevic, L. Xu, and T. Wang, "On the reverse concatenated coded-modulation for ultra-high-speed optical transport," in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference*, 2011, pp. 1–3.
- [37] M. Arabaci, I. Djordjevic, R. Saunders, and R. Marcocchia, "Rate-adaptive non-binary-LDPC-coded polarization-multiplexed multilevel modulation with coherent detection for optically-routed networks," in *11th International Conference on Transparent Optical Networks, 2009. ICTON '09.*, 2009, pp. 1–4.
- [38] M. Arabaci, I. Djordjevic, L. Xu, and T. Wang, "Four-dimensional nonbinary LDPC-coded modulation schemes for ultra-high-speed optical fiber communication," *IEEE Photonics Technology Letters*, vol. 23, no. 18, pp. 1280–1282, 2011.
- [39] H. Song and J. Cruz, "Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording," *IEEE Transactions on Magnetics*, vol. 39, no. 2, pp. 1081–1087, 2003.
- [40] C. Chen, B. Bai, X. Ma, and X. Wang, "A symbol-reliability based message-passing decoding algorithm for nonbinary LDPC codes over finite fields," in *6th International Symposium on Turbo Codes and Iterative Information Processing (ISTC), 2010*, Sept. 2010, pp. 251–255.
- [41] C.-S. Choi, H. Lee, N. Kaneda, and Y.-K. Chen, "Concatenated non-binary LDPC and HD-FEC codes for 100Gb/s optical transport systems," in *IEEE International Symposium on Circuits and Systems (ISCAS), 2012*, May 2012, pp. 1783–1786.
- [42] X. Zhang, F. Cai, and S. Lin, "Low-complexity reliability-based message-passing decoder architectures for non-binary LDPC codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 11, pp. 1938–1950, Nov. 2012.

- [43] F. Garcia-Herrero, M. Canet, J. Valls, and M. Flanagan, "Serial symbol-reliability based algorithm for decoding non-binary LDPC codes," *IEEE Communications Letters*, vol. 16, no. 6, pp. 909–912, June 2012.
- [44] L. Amaru, M. Martina, and G. Masera, "High speed architectures for finding the first two maximum/minimum values," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 12, pp. 2342–2346, Dec. 2012.
- [45] A. Voicila, F. Verdier, D. Declercq, M. Fossorier, and P. Urard, "Architecture of a low-complexity non-binary LDPC decoder for high order fields," in *International Symposium on Communications and Information Technologies, 2007. ISCIT '07.*, Oct. 2007, pp. 1201–1206.
- [46] X. Zhang and F. Cai, "Reduced-complexity extended Min-sum check node processing for non-binary LDPC decoding," in *53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2010*, Aug. 2010, pp. 737–740.
- [47] —, "Reduced-latency scheduling scheme for min-max non-binary LDPC decoding," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2010*, Dec. 2010, pp. 414–417.
- [48] —, "Reduced-complexity check node processing for non-binary LDPC decoding," in *IEEE Workshop on Signal Processing Systems (SIPS), 2010*, Oct. 2010, pp. 70–75.
- [49] —, "Partial-parallel decoder architecture for quasi-cyclic non-binary LDPC codes," in *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP), 2010*, March 2010, pp. 1506–1509.
- [50] —, "Efficient partial-parallel decoder architecture for quasi-cyclic nonbinary LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 2, pp. 402–414, Feb. 2011.
- [51] J. Lin, J. Sha, Z. Wang, and L. Li, "An efficient VLSI architecture for non-binary LDPC decoders," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 1, pp. 51–55, Jan. 2010.
- [52] S. Zhou, J. Sha, L. Li, and Z. Wang, "Layered decoding for non-binary LDPC codes," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, June 2010, pp. 481–484.
- [53] Y.-L. Ueng, C.-Y. Leong, C.-J. Yang, C.-C. Cheng, K.-H. Liao, and S.-W. Chen, "An efficient layered decoding architecture for nonbinary QC-LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 2, pp. 385–398, Feb. 2012.

- [54] J. Lin and Z. Yan, "Efficient shuffled decoder architecture for nonbinary quasi-cyclic LDPC codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 9, pp. 1756–1761, Sept. 2013.
- [55] X. Chen and C.-L. Wang, "High-throughput efficient non-binary LDPC decoder based on the Simplified Min-Sum algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 11, pp. 2784–2794, Nov. 2012.
- [56] H. Zhong, W. Xu, N. Xie, and T. Zhang, "Area-efficient min-sum decoder design for high-rate quasi-cyclic low-density parity-check codes in magnetic recording," *IEEE Transactions on Magnetics*, vol. 43, no. 12, pp. 4117–4122, 2007.
- [57] R. E. Blahut, *Algebraic Codes on Lines, Planes, and Curves, An Engineering Approach*. Urbana-Champaign, University of Illinois, 2008.
- [58] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Transactions on Information Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.
- [59] A. Vardy and Y. Be'ery, "Bit-level soft-decision decoding of Reed-Solomon codes," *IEEE Transactions on Communications*, vol. 39, no. 3, pp. 440–444, Mar 1991.
- [60] M. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, Sep 1995.
- [61] J. Jiang and K. Narayanan, "Iterative soft-input soft-output decoding of Reed-Solomon codes by adapting the parity-check matrix," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3746–3756, Aug. 2006.
- [62] R. Nielsen, "Decoding AG-codes beyond half the minimum distance," Ph.D. dissertation, University of Denmark, 1998.
- [63] K. Lee and M. O'Sullivan, "An interpolation algorithm using Grobner bases for soft-decision decoding of Reed-Solomon codes," in *IEEE International Symposium on Information Theory, 2006*, July 2006, pp. 2032–2036.
- [64] J. Zhu, X. Zhang, and Z. Wang, "Backward interpolation architecture for algebraic soft-decision Reed-Solomon decoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 11, pp. 1602–1615, Nov. 2009.
- [65] R. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance," *IEEE Transactions on Information Theory*, vol. 46, no. 1, pp. 246–257, Jan. 2000.



- [66] W. Gross, F. Kschischang, and P. Gulak, "Architecture and implementation of an interpolation processor for soft-decision Reed-Solomon decoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 3, pp. 309–318, March 2007.
- [67] W. Gross, F. Kschischang, R. Koetter, and R. Gulak, "A VLSI architecture for interpolation in soft-decision list decoding of Reed-Solomon codes," in *IEEE Workshop on Signal Processing Systems, 2002. (SIPS '02).*, Oct. 2002, pp. 39–44.
- [68] J. Bellorado, "Low-complexity soft decoding algorithms for Reed-Solomon codes," Ph.D. dissertation, Harvard University, 2006.
- [69] J. Jiang and K. Narayanan, "Algebraic soft-decision decoding of Reed-Solomon codes using bit-level soft information," *IEEE Transactions on Information Theory*, vol. 54, no. 9, pp. 3907–3928, Sept. 2008.
- [70] J. Zhu and X. Zhang, "Factorization-free low-complexity Chase soft-decision decoding of Reed-Solomon codes," in *IEEE International Symposium on Circuits and Systems, 2009. ISCAS 2009*, May 2009, pp. 2677–2680.
- [71] F. Garcia-Herrero, M. Canet, J. Valls, and P. Meher, "High-throughput interpolator architecture for low-complexity Chase decoding of RS codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 3, pp. 568–573, March 2012.
- [72] J. Zhu and X. Zhang, "Efficient VLSI architecture for soft-decision decoding of Reed-Solomon codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 10, pp. 3050–3062, Nov. 2008.
- [73] X. Zhang and J. Zhu, "High-throughput interpolation architecture for algebraic soft-decision Reed-Solomon decoding," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 3, pp. 581–591, March 2010.
- [74] —, "Hardware complexities of algebraic soft-decision Reed-Solomon decoders and comparisons," in *Information Theory and Applications Workshop (ITA), 2010*, Feb. 2010, pp. 1–10.
- [75] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 170–182, Jan 1972.
- [76] T. Moon, *Error Correction Coding: Mathematical Methods And Algorithms*. John Wiley & Sons Inc, 2004.
- [77] R. Blahut, *Theory and practice of Error-Control Codes*. Addison-Wesley, 1983.

- [78] D. Sarwate and N. Shanbhag, “High-speed architectures for Reed-Solomon decoders,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 5, pp. 641 –655, Oct. 2001.
- [79] K. Seth, K. Viswajith, S. Srinivasan, and V. Kamakoti, “Ultra folded high-speed architectures for Reed Solomon decoders,” in *19th International Conference on VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design.*, Jan. 2006.
- [80] I. Reed and M. Shih, “VLSI design of inverse-free Berlekamp-Massey algorithm,” *IEE Proceedings Computers and Digital Techniques*, vol. 138, no. 5, pp. 295 – 298, Sep. 1991.
- [81] B. Yuan, Z. Wang, L. Li, M. Gao, J. Sha, and C. Zhang, “Area-efficient Reed-Solomon decoder design for optical communications,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 6, pp. 469 –473, June 2009.
- [82] H. Lee, “A high-speed low-complexity Reed-Solomon decoder for optical communications,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 52, no. 8, pp. 461 – 465, Aug. 2005.
- [83] —, “High-speed VLSI architecture for parallel Reed-Solomon decoder,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 2, pp. 288 –294, April 2003.
- [84] B. Sklar, *Digital Communications: Fundamentals and Applications*. Prentice-Hall, 2001.
- [85] X. Zhang, “High-speed VLSI architecture for low-complexity Chase soft-decision Reed-Solomon decoding,” in *Information Theory and Applications Workshop, 2009*, Feb. 2009, pp. 422 –430.
- [86] X. Zhang and Y. Zheng, “Systematically re-encoded algebraic soft-decision Reed-Solomon decoder,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 6, pp. 376 –380, June 2012.
- [87] W. Zhang, H. Wang, and B. Pan, “Reduced-complexity LCC Reed-Solomon decoder based on unified syndrome computation,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 974 – 978, May 2013.