UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Design and implementation of decoders for error correction in high-speed communication systems

17th of May 2017

Author: Joan Marc Català Pérez

Thesis supervisors: Dr. Javier Valls Coquillat
Dr. Francisco Miguel García Herrero

Abstract

This thesis is focused on the design and implementation of binary low-density parity-check (LDPC) code decoders for high-speed modern communication systems. The basic of LDPC codes and the performance and bottlenecks, in terms of complexity and hardware efficiency, of the main soft-decision and hard-decision decoding algorithms (such as Min-Sum, Optimized 2-bit Min-Sum and Reliability-based iterative Majority-Logic) are analyzed. The complexity and performance of those algorithms are improved to allow efficient hardware architectures.

A new decoding algorithm called One-Minimum Min-Sum is proposed. It reduces considerably the complexity of the check node update equations of the Min-Sum algorithm. The second minimum is estimated from the first minimum value by a means of a linear approximation that allows a dynamic adjustment. The Optimized 2-bit Min-Sum algorithm is modified to initialize it with the complete LLR values and to introduce the extrinsic information in the messages sent from the variable nodes. Its variable node equation is reformulated to reduce its complexity. Both algorithms were tested for the (2048,1723) RS-based LDPC code and (16129,15372) LDPC code using an FPGA-based hardware emulator. They exhibit BER performance very close to Min-Sum algorithm and do not introduce early error-floor.

In order to show the hardware advantages of the proposed algorithms, hardware decoders were implemented in a 90 nm CMOS process and FPGA devices based on two types of architectures: full-parallel and partial-parallel one with horizontal layered schedule. The results show that the decoders are more area-time efficient than other published decoders and that the low-complexity of the Modified Optimized 2-bit Min-Sum allows the implementation of 10 Gbps decoders in current FPGA devices.

Finally, a new hard-decision decoding algorithm, the Historical-Extrinsic Reliability-Based Iterative Decoder, is presented. This algorithm introduces the new idea of considering hard-decision votes as soft-decision to compute the extrinsic information of previous iterations. It is suitable for high-rate codes

and improves the BER performance of the previous RBI-MLGD algorithms, with similar complexity.

Resumen

Esta tesis se ha centrado en el diseño e implementación de decodificadores binarios basados en códigos de comprobación de paridad de baja densidad (LDPC) válidos para los sistemas de comunicación modernos de alta velocidad. Los conceptos básicos de códigos LDPC, sus prestaciones y cuellos de botella, en términos de complejidad y eficiencia *hardware*, fueron analizados para los principales algoritmos de decisión *soft* y decisión *hard* (como Min-Sum, Optimized 2-bit Min-Sum y Reliability-based iterative Majority-Logic). La complejidad y prestaciones de estos algoritmos se han mejorado para conseguir arquitecturas *hardware* eficientes.

Se ha propuesto un nuevo algoritmo de decodificación llamado One-Minimum Min-Sum. Éste reduce considerablemente la complejidad de las ecuaciones de actualización del nodo de comprobación del algoritmo Min-Sum. El segundo mínimo se ha estimado a partir del valor del primer mínimo por medio de una aproximación lineal, la cuál permite un ajuste dinámico. El algoritmo Optimized 2-bit Min-Sum se ha modificado para ser inicializado con los valores LLR e introducir la información extrínseca en los mensajes enviados desde los nodos variables. La ecuación del nodo variable de este algoritmo ha sido reformulada para reducir su complejidad. Ambos algoritmos fueron probados para el código (2048,1723) RS-based LDPC y para el código (16129,15372) LDPC utilizando un emulador *hardware* implementado en un dispositivo FPGA. Éstos han alcanzado unas prestaciones de BER muy cercanas a las del algoritmo Min-Sum evitando, además, la aparición temprana del fenómeno denominado suelo del error.

Con el objetivo de mostrar las ventajas *hardware* de los algoritmos propuestos, los decodificadores se implementaron en *hardware* utilizando tecnología CMOS de 90 nm y en dispositivos FPGA basados en dos tipos de arquitecturas: completamente paralela y parcialmente paralela utilizando el método de actualización por capas horizontales. Los resultados muestran que los decodificadores propuestos e implementados son más eficientes en área-tiempo que otros decodificadores publicados y que la baja complejidad del algoritmo Modified Optimized 2-bit Min-Sum permite la implementación de decodificadores en los dispositivos FPGA actuales consiguiendo una tasa de 10 Gbps.

Finalmente, se ha presentado un nuevo algoritmo de decodificación de decisión *hard*, el Historical-Extrinsic Reliability-Based Iterative Decoder. Este algoritmo introduce la nueva idea de considerar los votos de decisión *hard* como decisión soft para calcular la información extrínseca de iteracions anteriores. Este algoritmo es adecuado para códigos de alta velocidad y mejora el rendimiento BER de los algoritmos RBI-MLGD anteriores, con una complejidad similar.

Resum

Aquesta tesi s'ha centrat en el disseny i implementació de descodificadors binaris basats en codis de comprovació de paritat de baixa densitat (LDPC) vàlids per als sistemes de comunicació moderns d'alta velocitat. Els conceptes bàsics de codis LDPC, les seues prestacions i colls de botella, en termes de complexitat i eficiència *hardware*, van ser analitzats pels principals algoritmes de decisió *soft* i decisió *hard* (com el Min-Sum, Optimized 2-bit Min-Sum y Reliability-based iterative Majority-Logic). La complexitat i prestacions d'aquests algoritmes s'han millorat per aconseguir arquitectures *hardware* eficients.

S'ha proposat un nou algoritme de descodificació anomenat One-Minimum Min-Sum. Aquest redueix considerablement la complexitat de les equacions d'actualització del node de comprovació del algoritme Min-Sum. El segon mínim s'ha estimat a partir del valor del primer mínim per mitjà d'una aproximació lineal, la qual permet un ajust dinàmic. L'algoritme Optimized 2-bit Min-Sum s'ha modificat per ser inicialitzat amb els valors LLR i introduir la informació extrínseca en els missatges enviats des dels nodes variables. L'equació del node variable d'aquest algoritme ha sigut reformulada per reduir la seva complexitat. Tots dos algoritmes van ser provats per al codi (2048,1723) RS-based LDPC i per al codi (16129,15372) LDPC utilitzant un emulador *hardware* implementat en un dispositiu FPGA. Aquests han aconseguit unes prestacions BER molt properes a les del algoritme Min-Sum evitant, a més, l'aparició primerenca del fenomen denominat sòl de l'error.

Per tal de mostrar els avantatges *hardware* dels algoritmes proposats, els descodificadors es varen implementar en *hardware* utilitzan una tecnologia CMOS d'uns 90 nm i en dispositius FPGA basats en dos tipus d'arquitectures: completament paral·lela i parcialment paral·lela utilitzant el mètode d'actualització per capes horitzontals. Els resultats mostren que els descodificadors proposats i implementats són més eficients en àrea-temps que altres descodificadors publicats i que la baixa complexitat del algoritme Modified Optimized 2-bit Min-Sum permet la implementació de decodificadors en els dispositius FPGA actuals obtenint una taxa de 10 Gbps.

Finalment, s'ha presentat un nou algoritme de descodificació de decisió *hard*, el Historical-Extrinsic Reliability-Based Iterative Decoder. Aquest algoritme presenta la nova idea de considerar els vots de decisió *hard* com decisió soft per calcular la informació extrínseca d'iteracions anteriors. Aquest algoritme és adequat per als codis d'alta taxa i millora el rendiment BER dels algoritmes RBI-MLGD anteriors, amb una complexitat similar.

Acknowledgments

Firstly, I would like to thank my supervisor Dr. Javier Valls, not only for having directed my Ph.D. degree, my Master's thesis and the Bachelor degree project, but for his support and unconditional help both during my stay at the university and afterwards.

Besides, my deeply gratitude to Dr. Francisco Miguel García for his unconditional help both as supervisor and friend. Thanks to the motivation of Javier and Francisco this work was finalized.

Secondly, thanks to the research group GISED, specially my co-workers Julian, Jesus and Ferran with whom I shared this experience.

Thanks to the Spanish Government for providing financial help with a FPI grant (Grant No. BES-2012-052130).

Finally, I would like to thank my parents and my brother for teaching me that with hard work anything can be done, and to my wife Adriana and my son Jan for cheering me up with their smiles every single day. Without them, it would have been impossible to develop this work.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ASIC Application-Specific Integrated Circuit.

AWGN Additive White Gaussian Noise

BER Bit Error Rate

BPSK Binary Phase-Shift Keying

CMOS Complementary Metal-Oxide-Semiconductor

CN Check Node

CNU Check Node Update

ECC Error Control Coding

FEC Forward Error Correction

FER Frame Error Rate

FIFO First In, First Out

FPGA Field Programmable Gate Array

GF Galois Field

HDD Hard-Decision Decoder

HDL Hardware Description Language

HE-RBID Historical-Extrinsic Reliability-Based Iterative Decoding

LDPC Low-Density Parity-Check

LLR Log-Likelihood Ratio

LUT Lookup Table

MLGD Majority-Logic Decoding

MRBI Modified Reliability-Based Iterative

MSA Min-Sum Algorithm

MSD Min-Sum Decoding

OMO-MSA One-Minimum-Only Min-Sum Algorithm

RAM Random-Access Memory

RBI Reliability-Based Iterative

ROM Read-Only Memory

RS Reed-Solomon

RTL Resistor-Transistor Logic

SNR Signal-to-Noise Ratio

SoC System-on-Chip

SPA Sum-Product Algorithm

VHDL VHSIC Hardware Description Language

VLSI Very Large Scale Integration

VN Variable Node

VNU Variable Node Update

Preface

Nowadays, error correction codes (ECC) is involved in almost every communication and storage system such as long-haul networks, space data link protocols [1], digital TV, wireless communication [2], optical storage systems (*e.g.*, BluRay, DVD), magnetic storage systems, flash memories [3], etc. For this reason, it is important that hardware designers know the performance of these codes in terms of both error-correction capability and physical implementation parameters (*i.e.*, the silicon area and throughput of the derived decoder) in order to choose the code and the hardware architecture that fits better with the constraints of different scenarios.

Since 1948, when Claude Shannon established a limit for any communication channel, many different codes have been proposed, some of the most relevant are: Hamming (1950) [4], Reed-Muller (1945) [5], Reed-Solomon (1960) [6], Bose-Chaudhuri-Hocquengham (1960) [7], low-density parity-check (LDPC) (1962) [8], Turbo codes (1993) [9], Polar codes (2009) [10], etc. Although in some cases these codes achieve an error correction that is close to the channel capacity, the complexity of the decoding process and the limitations of the technology did not allow to implement their decoders in hardware. As an example, the first hardware implementations of Reed-Solomon and LDPC decoders started about 30 and 40 years, respectively, after these codes were discovered. On the other hand, these implementations belong to decoding algorithms that are sub-optimal [11], in other words, algorithms that are adapted to be hardware friendly and allow a practical tradeoff between silicon area, throughput and error correction capability. However, with the improvement of processors, communication systems and storage resources, three constraints force us to search for better decoding algorithms and implementations: i) the technology scaling, which entails lowering the operating voltages and increasing the integration densities, and hence the number of errors increases; ii) the increase of speed in both communication and processing, which requires high throughput subsystems that do not slow down the whole application and iii) the need for reducing the power consumption, extremely important for mobile devices, which depend on the battery life. So, the main challenge in ECC

is the design of a decoder with the maximum error correction capability and a throughput large enough to not become the bottleneck of the system.

Among all error correcting codes, LDPC codes are the most promising to overcome the previously mentioned difficulties as they meet the requirements of good error correction capability close to the Shannon limit, and reasonable complexity for high-throuhgput decoding architectures.

This thesis is focused on reducing the complexity of different decoders based on soft and hard decision algorithms including: the Min-Sum decoding algorithm, with the proposal of a new decoder called One-Minimum Min-Sum (OMO-MS); a new Reliability-Base Iterative Decoder (HE-RBID) with an improvement in the error-rate performance; a Modified Optimized 2-bit Min-Sum decoder with less complexity than other 2-bit LDPC decoders.

Objectives

This work continues with the line of research started in the Digital Communication Laboratory with a previous Ph.D. thesis [33] in which a first approximation to LDPC codes was done in order to: i) evaluate the behaviour of different code constructions; ii) analyze different decoding algorithms; and iii) evaluate the impact of quantization in the bit error rate performance. Taking all this knowledge as starting point, this thesis is focused on the proposal of new forward error correction (FEC) algorithms and architectures for LDPC codes that reduce complexity and improve error correction performance in high-speed systems. Therefore, the specific objectives of this thesis are:

1. To review the state of the art in LDPC, soft-decision and hard-decision decoding algorithms with the aim of detecting the main bottlenecks in the existing algorithms and decoder architectures.

2. To reduce the check node processor complexity for soft-decision decoders.

3. To reduce wiring congestion, especially for high-rate codes, with the aim of increasing the maximum frequency of the derived architectures.

4. To reduce the number of bits of the messages exchanged between the check node and the variable node processors in order to save storage resources and hence to save silicon area.

5. To improve, reduce or eliminate the early performance degradation that hard-decision LDPC decoders introduce for codes with low d_v and non-EG constructions.

Methodology

The objectives of this thesis were achieved following the methodology described below.

First, a study of the state of the art in coding theory was done, focused in FEC and specially in binary LDPC codes. After this, the main decoding algorithms were modeled using MATLAB software. The first software model is based on floating-point and the second one applies a fixed-point precision model. This models were taken as references/boundaries for our contributions.

After the previous step, modifications in the original algorithms were performed in order to reduce the number of operations without introducing a significant error correction degradation. This process was iterative. Modified versions of the algorithms were modeled in MATLAB until a good tradeoff between the coding gain in the waterfall region and the complexity was obtained.

Once verified the correct operation of the algorithms modeled in MATLAB, for some of the proposed algorithms, an architecture was implemented in VHDL using the computer software Quartus II Altera, ISE Xilinx and ASIC tools. Comparing the output of the decoder implemented in VHDL with the output of the fixed-point models, the proper functioning of the system was verified with the use of VHDL language.

After the verification, the algorithm was implemented in hardware and its behaviour was tested with a decoder emulator. The performance results were also compared to the performance obtained with the software model of the emulator. The emulator also allowed to explore the error-floor region which is more time consuming and in some cases cannot be studied with software models.

Finally, the implementation results (area and speed) of the proposed solution were compared to the ones found in literature. This process was repeated until a significant improvement in the state-of-the-art was reached.

Contributions

The contributions of this thesis are listed below:

1. A complete tutorial based on the state of the art to divulge the best practices in the design of efficient LDPC decoders for different scenarios and an analysis of the best soft-decision algorithms and architectures.
2. The One-Minimum Min-Sum Decoding Algorithm (OMO-MSA) that reduces to half the complexity of the check node update equations in low-density parity-check (LDPC) decoding algorithms with negligible performance loss.

3. A Modified Optimized 2-bit Min-Sum Decoding Algorithm (MO2-BIT-MSA) and an architecture that improves the bit-error-rate performance of the previous Optimized 2-bit Min-Sum Decoding Algorithm with less complexity, and allows the implementation of very fast decoders in FPGA devices.

4. The Historical-Extrinsic Reliability-Based Iterative Decoder (HE-RBID) that computes the extrinsic information of previous iterations of the variable node improving the bit-error-rate performance of the previous RBI-MLGD with less complexity.

Some of these contributions have been published or submitted to journals and conferences:

- International Journals

  1. J.M. Català, F. Garcia-Herrero, J.Valls, K. Liu and S. Lin Life Fellow, IEEE, "Reliability-Based Iterative Decoding Algorithm for LDPC Codes With Low Variable-Node Degree," IEEE Communications Letters, Vol.18, No.12, pp.2065-2068, Dec.2014
  2. K. Gunnam, J.M. Català and F. Garcia-Herrero, "Algorithms and VLSI Architectures for Low-Density Parity-Check Codes: Part 1 - Low-Complexity Iterative Decoding," IEEE Solid-State Circuits Magazine, Vol.8, No.4, pp.57-63, Nov.2016
  3. K. Gunnam, J.M. Català and F. Garcia-Herrero, "Algorithms and VLSI Architectures for Low-Density Parity-Check Codes: Part 2 - Efficient Coding Architectures," IEEE Solid-State Circuits Magazine, Vol.9, No.1, pp.23-28, Jan.2017

- National Conferences

  1. J.M. Català, F. Garcia-Herrero, J.Valls, K. Liu and S. Lin Life Fellow, IEEE, "Reliability-Based Iterative Decoding Algorithm for LDPC Codes With Low Variable-Node Degree," Congreso XXIX Simposium Nacional de la Unión Científica Internacional de Radio, Valencia, Spain, Jun.2014

Thesis structure

This memory is divided in five chapters. The first one briefly summarizes the basics of the codes under study, and reviews the state of the art, identifying the bottlenecks and the most interesting topics of research from the author's perspective. The second chapter deals with the reduction of complexity of the check node of the conventional Min-Sum algorithm. The third chapter is focused on the improvement of a 2-bit non uniform quantized Min-Sum algorithm. The fourth

chapter introduces a new hard-decision decoding algorithm that improves the coding gain of high-rate codes. In these chapters, first the algorithmic novelties and their impact on the coding gain are detailed and, then, the derived architectures or complexity analysis are presented. Each chapter includes its own conclusions at the end and comparisons with the most efficient results found in literature. Conclusions and topics for future work are outlined in the last chapter.

Chapter 1

Background concepts for low-density parity-check decoders

This first chapter reviews the basics of LDPC codes, the soft-decision iterative decoding algorithms and hard-decision iterative ones. It is organized into six sections. The first and second section introduce LDPC codes and general decoding schedules. The third section shows the main existing architectures for LDPC decoders. The fourth section describes the main soft-decision algorithms based on belief propagation, which required to understand better the soft-decision contributions of this thesis: i) Sum-Product, ii) scaled Min-Sum, iii) Optimized 2-bit Min-Sum, and v) performance analysis of the previously mentioned algorithms. The fifth section is focused on hard-decision iterative decoding algorithms beginning with an introduction and a description of the following three reliability-based iterative decoding algorithms: i) Reliability-Based Iterative Majority-Logic Decoding algorithm (RBI-MLGD), ii) Modified Reliability-Based Iterative Majority-Logic Decoding algorithm (MRBI-MLGD) and iii) Reliability-Based Iterative Min-Sum Decoding algorithm (RBI-MSD) and ending with a performance comparation of the previously mentioned algorithms. In the final section the conclusions of this chapter are outlined.

1.1. LDPC codes

Let us define an LDPC code by its parity check matrix H. This parity check matrix can be described as a sparse $M \times N$ matrix, where: i) each one of the M rows represents a parity check equation over $GF(2^q)$; and ii) the columns in H correspond to each one of the received symbols. For the binary case, with $q = 1$, the symbols are equal to bits, so N also represents the codeword length in terms of bits. The codeword, denoted by $\mathbf{c}$, can be computed by means of a generator matrix G that satisfies the following equation $G \times H^T = 0$. The codeword is equal to $\mathbf{c} = G \times b$, where b is the K-bit [1] information vector. The code rate can be computed as K/N.

LDPC codes can also be represented by a Tanner Graph, which is a bipartite graph with two different kind of nodes: check nodes and variable nodes. Each check node m_x (squares in Fig. 1.1) corresponds to one of the parity check equations in H. The variable nodes, n_x (circles in Fig. 1.1, also called bit nodes) correspond to each one of the columns in H. An edge connects a check node and a variable node if and only if the corresponding row and column share a non-zero element in H. The number of non-zero elements in a row (*i.e.*, the number of variable nodes connected to a check node) is the degree of the check node (d_c) and the number of non-zero elements in a column (*i.e.*, the number of check nodes connected to a variable node) is the degree of the variable node (d_v). In Fig. 1.1 and Fig. 1.2 the two representations of the same code, the Tanner graph and the parity-check matrix, can be compared. In this example the degree of the check node is four and the degree of the variable node is three, as d_c and d_v are limited by the maximum d_c (d_v) in all the check node (variable node).

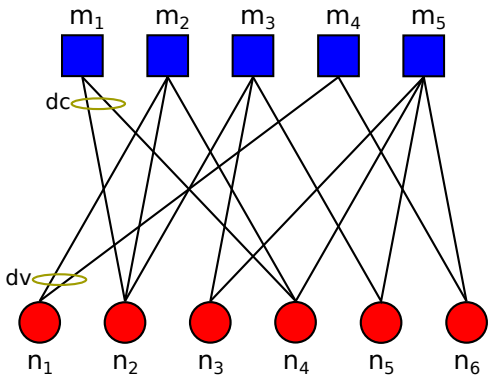

Figure 1.1: Representation of the parity check matrix H with a Tanner Graph.

[1] The parity check matrix H is of range K, over 2^q

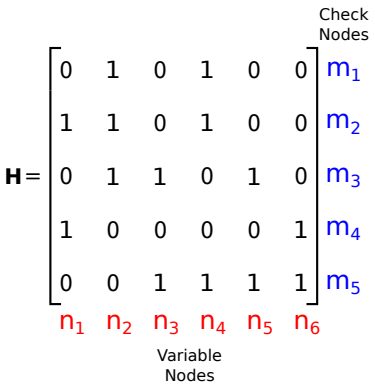

Figure 1.2: Parity check matrix H.

The Tanner graph representation is useful to understand the general decoding procedure that is explained in the next section.

1.2. Message-passing decoding schedules

The received vector at the decoder is $\mathbf{y} = \mathbf{c} + \mathbf{e}$, where $\mathbf{e}$ is the error vector introduced by the channel. The channel decoder transforms the received vector $\mathbf{y}$ into a soft-decision vector $\mathbf{L}$, which is loaded in each variable node. The complete system can be found in Fig. 1.3.

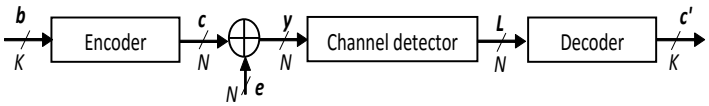

Figure 1.3: Simplified diagram of channel coding and decoding.

LDPC decoding algorithms are based on the message passing between check and variable nodes. Each variable node sends information (messages) to its connected check nodes. Once a check node has received all the messages from its neighbours (connected variable nodes), it computes the parity check equation and modifies the output messages depending on whether the equation was satisfied or not. The variable nodes receive the updated messages from the connected check nodes and update their values. After the update of the variable node information the process starts again. This schedule can be repeated until a maximum number of iterations is reached or until all the equations are satisfied.

It is easy to deduce that with each iteration the decoder tries to converge to the correct codeword, so the number of errors in the estimated one, **c'**, is reduced, as can be seen in Fig. 1.4.

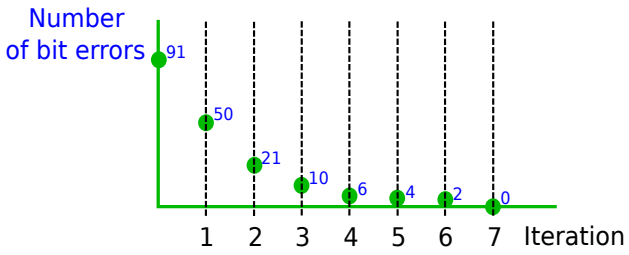

Figure 1.4: Evolution of the number of errors with the iterations.

Related to this, in each iteration the number of satisfied parity check equations increases, as can be seen in Fig. 1.5, or in other words, the syndrome weight (syndromes not equal to zero) decreases.

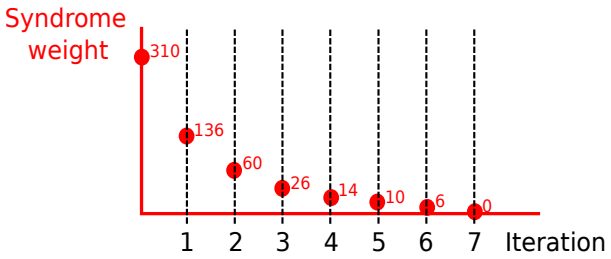

Figure 1.5: Evolution of the syndrome weight with the iterations.

The syndromes are equal to the evaluation of the parity check equations. In the following equations the syndromes of the parity check matrix in Fig. 1.2 are represented.

$$
\begin{aligned}
s_1 &= n_2 \oplus n_4 \\
s_2 &= n_1 \oplus n_2 \oplus n_4 \\
s_3 &= n_2 \oplus n_3 \oplus n_5 \\
s_4 &= n_1 \oplus n_6 \\
s_5 &= n_3 \oplus n_4 \oplus n_5 \oplus n_6
\end{aligned}
$$

Depending on how variable node messages are initialized and the nature (amount of channel information) of the messages exchanged in the bipartite graph, two types of decoders can be defined: hard-decision and soft-decision decoders. The soft-decision decoders include a measure of the reliability of the information. The hard-decision decoders only provide a result '0' or '1' without a measure of the bit's reliability.

Soft-decision decoders are analyzed in Sections 1.4; hard-decision decoders will be described in Sections 1.5.

With independence of the information exchanged between nodes the decoding process/schedule is as follows. In the first step of a soft-decision decoder for LDPC codes, variable nodes n_i are initialized with one metric of the reliability of the corresponding received bit (y_i). The most common metric is the log-likelihood ratio (LLR), which can be calculated with $L_n(y_i) = log(\frac{P(y_i=0)}{P(y_i=1)})$, using the outputs of a channel detector. For the Binary Phase Shift Keying (BPSK), LLR can be calculated as $L_n(y_i) = 2 \cdot y_i/\sigma^2$, which simplifies the computational load of the decoder, where σ^2 is the noise variance assuming that the information is transmitted by a channel with Additive White Gaussian Noise (AWGN).

When the iterative process starts each variable node, n, sends all its messages, $Q^{(i)}_{n,m}$, which include the channel information of the associated bit to the connected check nodes, m. The reliability metrics are recomputed based on the parity constraints at each check node. The updated information $R^{(i)}_{m,n}$ returns to the neighbouring variable nodes. Each variable node updates its decision for the estimated codeword based on the channel information from the initialization and the extrinsic information from all the neighbouring check nodes. The process is repeated until it converges to a valid codeword or until a time limit is reached (maximum number of iterations).

In Fig. 1.6 and Fig. 1.7 the process for one code with three check nodes and seven variable nodes is presented. This example is focused on the fourth bit ($n = 3$) and the second check node ($m = 1$).

First, the variable node is initialized with the LLR for the variable node $n = 3$, *i.e.*, L_3. This variable node sends the information to the connected check nodes, the message that is sent to the check node $m = 0$ is $Q^{(0)}_{3,0}$. For check node $m = 1$ is the message $Q^{(0)}_{3,1}$ and for check node $m = 2$ the message is $Q^{(0)}_{3,2}$ (Fig. 1.6b). Each check node updates its messages and sends them back to the variable nodes. The messages are $R^{(0)}_{0,3}$, $R^{(0)}_{1,3}$ and $R^{(0)}_{2,3}$ (Fig. 1.6c).

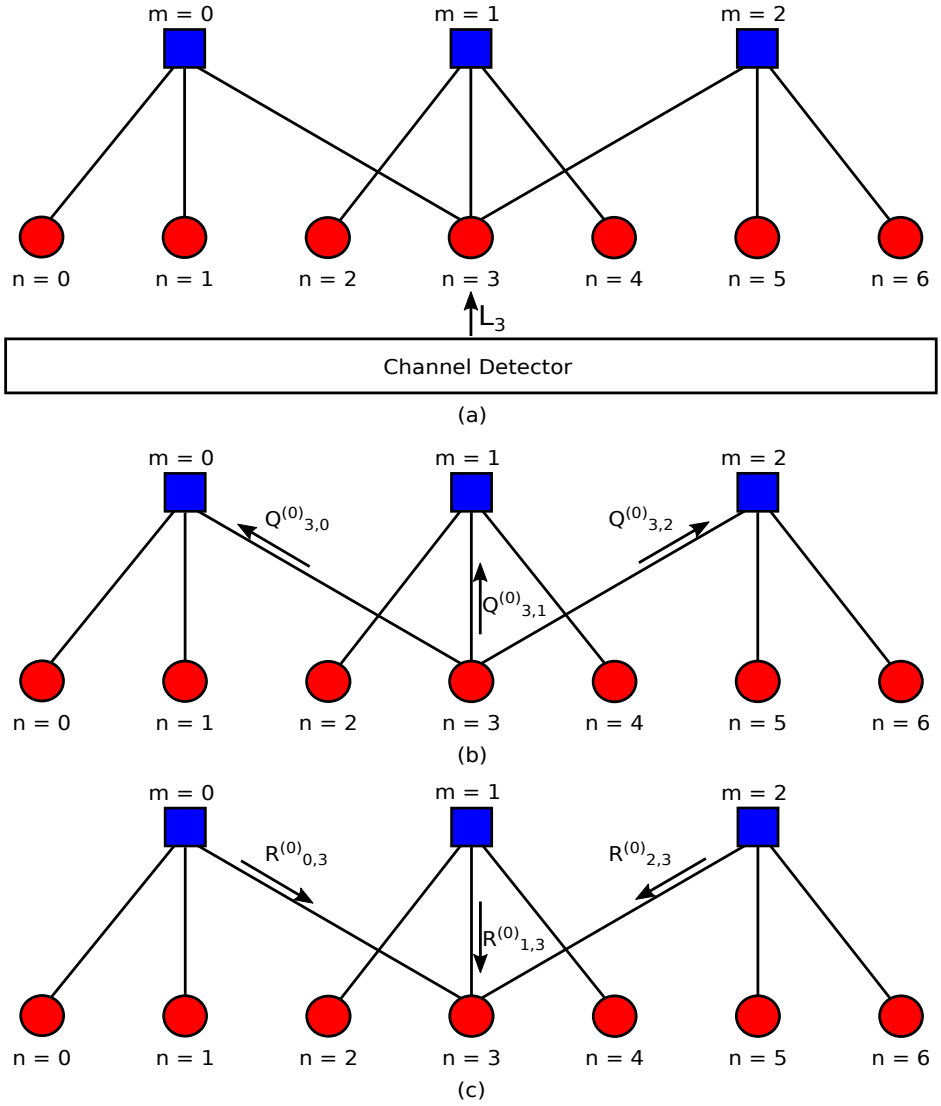

Figure 1.6: (a) Step 1 of a message passing algorithm. (b) Step 2 of a message passing algorithm. (c) Step 3 of a message passing algorithm.

In the second iteration, the variable node $n = 3$ only uses the extrinsic information to update the messages that are going to be sent to the connected check nodes. As an example, to compute the message for $m = 0$, the variable node only processes $R^{(0)}_{1,3}$ and $R^{(0)}_{2,3}$ to calculate $Q^{(1)}_{3,0}$ (Fig. 1.7d). In order to compute the message for $m = 1$, $Q^{(1)}_{3,1}$, the variable node only processes $R^{(0)}_{0,3}$ and $R^{(0)}_{2,3}$ (Fig. 1.7e) and to

calculate the message for $m = 2$, $Q^{(1)}_{3,2}$, the variable node only processes $R^{(0)}_{0,3}$ and $R^{(0)}_{1,3}$ (Fig. 1.7f).

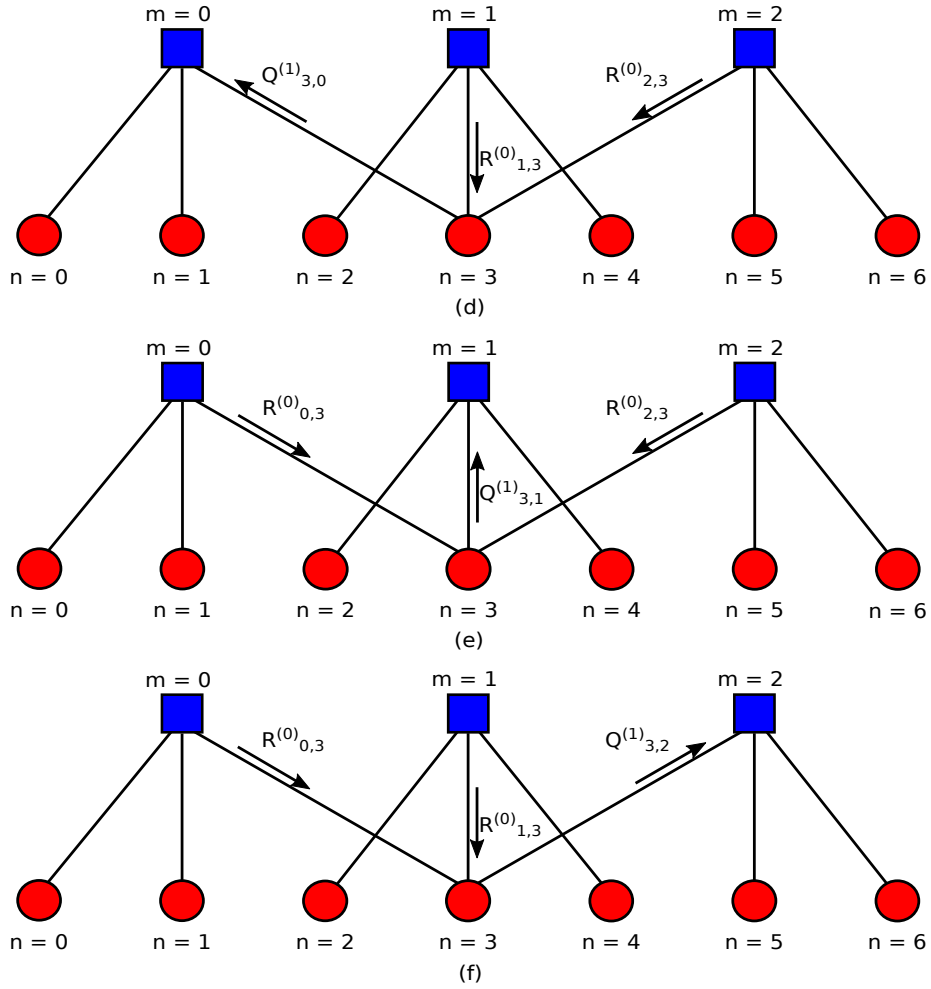

Figure 1.7: (d) Step 4 of a message passing algorithm. (e) Step 5 of a message passing algorithm. (f) Step 6 of a message passing algorithm.

To update the information of one check node, the use of its own messages from previous iterations is avoided to eliminate the possibility of feeding back any error introduced by this node. For this reason, one check node only "trusts" in the

information updated in the variable node that processed the messages which come from its neighbours, in other words, the extrinsic information.

This message exchange can be performed in two different ways: flooding (Fig. 1.8) and layered (Fig. 1.9). The flooding schedule first updates all the variable nodes and then all the check nodes or viceversa, this is the first scheduling method proposed in literature.

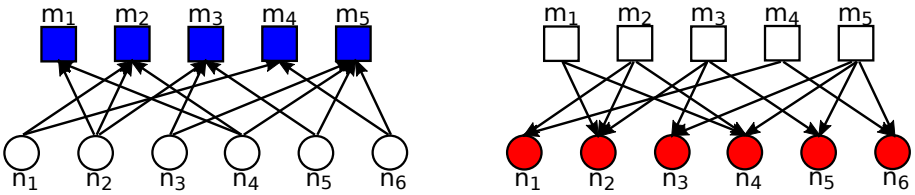

Figure 1.8: Tanner graph representation for one iteration with flooding or parallel scheduling.

To speed up the convergence of the decoder this basic schedule can be varied. Instead of computing first all the variable nodes and then all the check nodes, a layered scheduling can be applied. This layered scheduling consists on updating one check (variable) node at a time and, after each update, computing the new messages of the connected variable (check) nodes. If the update is performed row wise (updating each check node and all the variable nodes connected to this updated check node at a time) the schedule is called horizontal layered, if the update is performed column wise (updating each variable node and all the check nodes connected to this updated variable node at a time), the schedule is called vertical layered or shuffled.

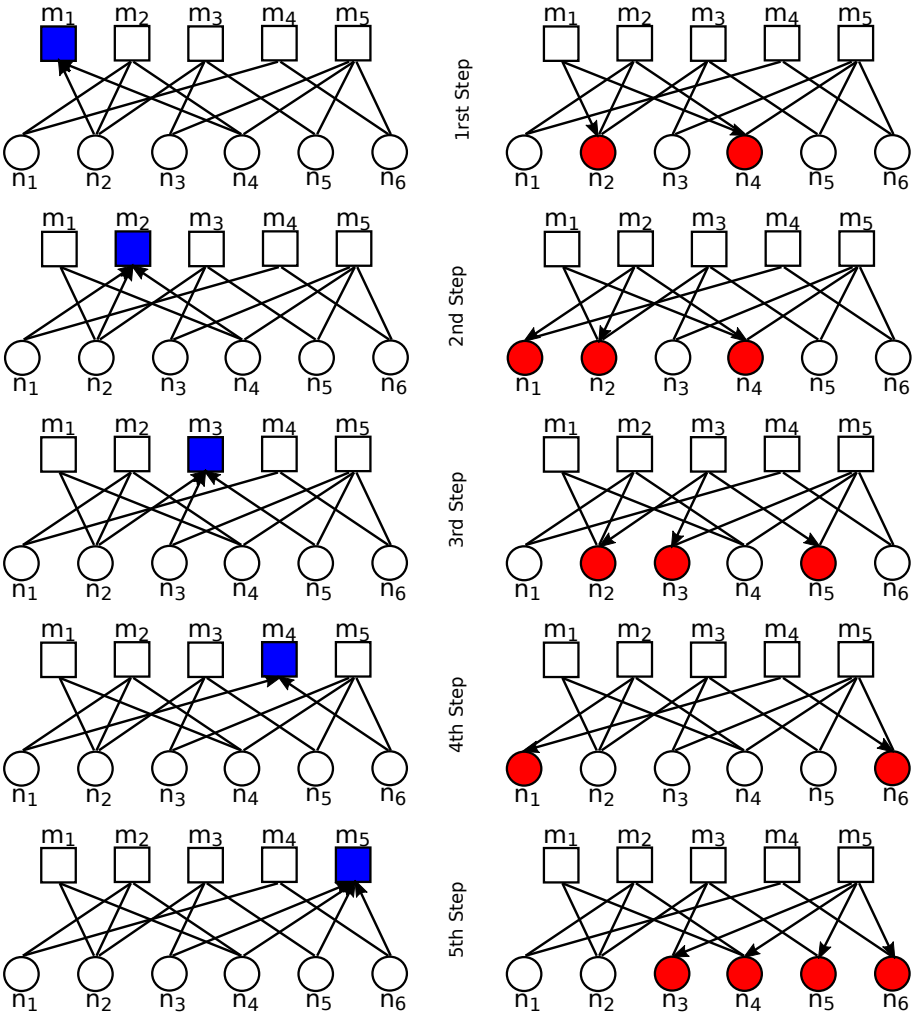

Figure 1.9: Tanner graph representation for one iteration with serial layered scheduling.

This layered scheduling converges faster because the nodes process the most recently updated information when updating their messages. The size of the layer, *i.e.*, the processed check or variable nodes can be between one and the maximum number of rows or columns, respectively as shown in Fig. 1.10.

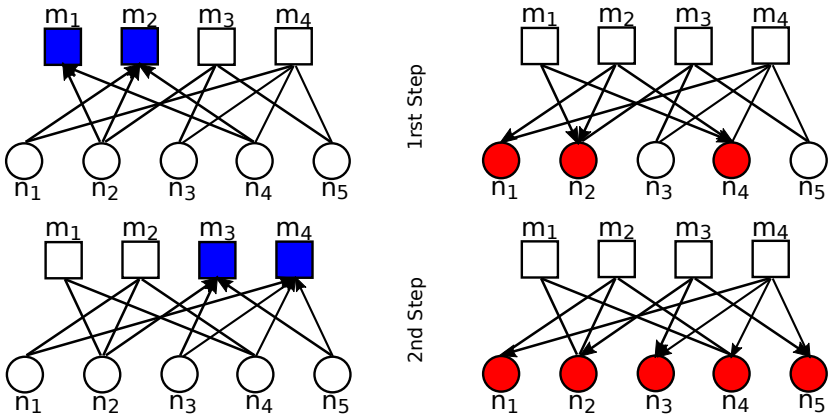

Figure 1.10: Tanner graph representation for one iteration with serial layered scheduling, updating 2 check nodes at a time.

1.3. Hardware architectures for LDPC decoders

A general diagram of a LDPC decoder architecture can be found in Fig. 1.11, where a general idea of the main blocks of the decoder is depicted. We found two types of processors, the check node units (CNUs) and the variable node units (VNUs) that compute the equations described in the following sections to perform the operations of CN and VN algorithms. The rest of the decoder is formed by memories to store: i) $R_{m,n}^{(i)}$ messages from check node to the variable node units; ii) $Q_{n,m}^{(i)}$ messages from variable node to the check node units; and iii) memories to store the estimated codeword reliability values Q_n. These memories can be implemented with RAM (double or single port) or registers depending on the depth of the memories and the selected architecture. Finally, some ROM memory is required to store the connections between check nodes and variable nodes (location of non-zero elements). About 80% of the decoder's area is allocated to memory resources.

Regarding the speed of the decoder, the general throughput equation is shown in Eq. 1.1.

$$Throughput = \frac{N \times f_{clk}}{\#iter \times (\frac{M}{\#CNU} \times \#clk_{CNU} + \frac{N}{\#VNU} \times \#clk_{VNU} + pipeline)} \tag{1.1}$$

In this equation, f_{clk} corresponds to the operating clock frequency of the decoder, $\#iter$ is the number of iterations of the algorithm, $\#CNU$ is the number of CNU

processors, $\#VNU$ is the number of VNU processors, $\#clk_{CNU}$ is the number of CNU's clock cycles, $\#clk_{VNU}$ is the number VNU's clock cycles and *pipeline* corresponds to the latency in the CNU and VNU due to the pipeline stages.

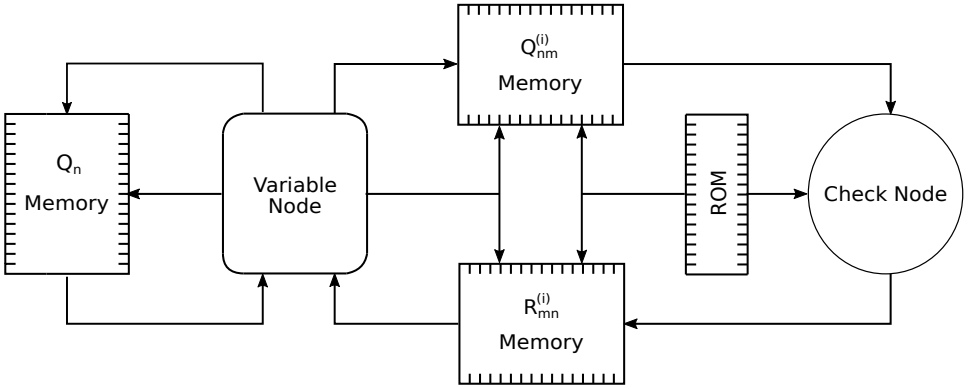

Figure 1.11: General diagram of a LDPC decoder architecture.

The three basic kinds of architectures are:

1. **Fully-parallel** architecture: it implements M check node processors and N variable node processors. It maps each row and each column of the parity check matrix H to a different processing unit which operates in parallel, so all the connections of the Tanner graph are physically implemented. The problem of implementing all the processing units in parallel is that the wiring between processors is considerably large, including at least $M \times d_c$ connections between processors. As the wiring congestion increases with the codeword length, it becomes the main limitation of the maximum achievable throughput in most of the cases. On the other hand, the main advantage of this kind of architecture is that it only needs a single cycle per message-passing iteration, so this decoders are energy efficient, but due to this, it implements the memory units for the message exchange in registers increasing about three times the required area. The main purpose of this kind of architecture is the implementation of the flooding schedule. As in this case $\#CNU = M$ and $\#VNU = N$ are implemented, and the clock cycles in each one is 1; the throughput of the fully-parallel architecture can be calculated as is shown in Eq. 1.2.

$$Throughput = \frac{N \times f_{clk}}{\#iter \times 2} \tag{1.2}$$

2. **Fully-serial** architecture: it consists of only one processing unit or core and one memory block. This processing unit calculates a single check or

variable node per cycle and it stores messages in a memory to be used in the following calculations. This kind of architecture consumes the minimum arithmetic resources and area as it increases the number of shared elements, minimizing the decoder area. Also the reached operating frequency is higher, as wiring is greatly reduced, due to the fact that the depth of the critical path is minimized. The main problem is that throughput is dramatically reduced to hundreds of kbps or tens of Mbps due to the increase of the number of clock cycles. Thus fully-serial architectures are not recommended for systems where latency is a critical parameter. The throughput of the fully-serial architecture can be calculated as is shown in Eq. 1.3.

$$Throughput = \frac{N \times f_{clk}}{\#iter \times ((M + N + pipeline) \times X)} \tag{1.3}$$

In Eq. 1.3, X represents the number of times that the pipeline stages of the decoder must be emptied per iteration to avoid memory conflicts. The reduction of this equation is due to the fact that one CNU ($\#CNU = 1$) and VNU ($\#VNU = 1$) are implemented each time.

3. **Partially parallel** architecture: it is a mix between fully-serial and fully-parallel architectures. This architecture implements several check node unit processors which compute the d_c input messages from the connected variable nodes, these messages are read from several banks of RAM memory ($Q^{(i)}_{n,m}$ memory). When the output messages of the check node processors are available, they are stored in a different RAM memory from which the check node inputs were read ($R^{(i)}_{m,n}$ memory). The variable node processor computes d_v messages in parallel, these messages are read from $R^{(i)}_{m,n}$ memory and the outputs are written in $Q^{(i)}_{n,m}$ memory. It is important to remark two issues of this architecture. First, the variable nodes (check nodes) cannot be computed until $R^{(i)}_{m,n}$ ($Q^{(i)}_{n,m}$) memory is completely updated. The second issue is that, depending on how we select the check nodes and/or variable nodes that we want to update each clock cycle, we may experience some conflicts, trying to write messages in the same address of the memories. These memory conflicts can be avoided by reordering the parity check matrix for the computation or selecting only the check nodes that do not share any variable node.

   In addition, using quasi-cyclic codes such as the ones in [11], message passing memories are eliminated too and only the use of registers are required to store the partial state and final state of CNU in addition to the FIFO for Q sign memory and a memory for storing channel LLR and hard decision values. Partially parallel architecture allows to obtain between hundreds of Mbps to tens of Gbps and is now widely used for smaller length LDPC codes in storage applications. In this case, the throughput of the partially-parallel

architecture depends on the level of parallelization/serialization of the architecture in Eq. 1.1.

1.4. Soft-decision decoding algorithms

1.4.1. Sum-Product algorithm

Belief propagation algorithm is used either in information theory or in artificial intelligence. The feature of this decoder is that the messages passed along the edges are probabilities or beliefs.

The first belief propagation algorithm proposed in literature for an LDPC decoder is known as the sum-product algorithm (SPA) [15].

In the first stage of iteration i, the magnitude of the messages are computed, as observed in Fig. 1.12. These magnitudes represent the reliability of the output messages. The idea beneath the following equation is to search for the less reliable message, which will limit the reliability of the decisions at the check node. In other words, the output message (estimated codeword bit) will be as reliable as the least reliable input message. For this reason, as the extrinsic information is used ($n' \in N(m)\backslash n$), only the least reliable message within the messages of neighbour nodes is looked for, not its own message. Note that the computation of the reliability in the check node requires a product and a *tanh()*, which are computationally complex operations.

$$\kappa_{m,n}^{(i)} = \phi(|R_{m,n}^{(i)}|) = \phi(|Q_{n',m}^{(i-1)}|) \quad where \quad \phi(x) = \prod_{n' \in N(m)\backslash n} tanh(\frac{1}{2}x), \quad x \geq 0$$

The second stage of the check node computation evaluates the parity check node equation and estimates the value of the received bits. The algorithm maps the sign of the value of the hard decision bit. According to equation $L_n(y_i) = log(\frac{P(y_i=0}{P(y_i=1)})$ and assuming binary phase-shift keying (BPSK) modulation, if the received bit is more likely to be zero the sign is positive; if the received bit is more likely to be one the sign is negative. So, it can be concluded that the logic one is mapped to the negative sign and the logic zero is mapped to the positive sign. By multiplying all the d_c signs from the d_c input messages the parity check equation can be evaluated. If the result is positive (logic zero), the equation is satisfied and the estimated output bits (and their corresponding signs) are equal to the input bits (which are the incoming bits/messages from the variable node). On the other hand, if the result is negative (logic one), the parity check equation is not satisfied and the estimated bits may be flipped compared to the input bits. In the same way as the magnitude, the bits are flipped considering the extrinsic information.

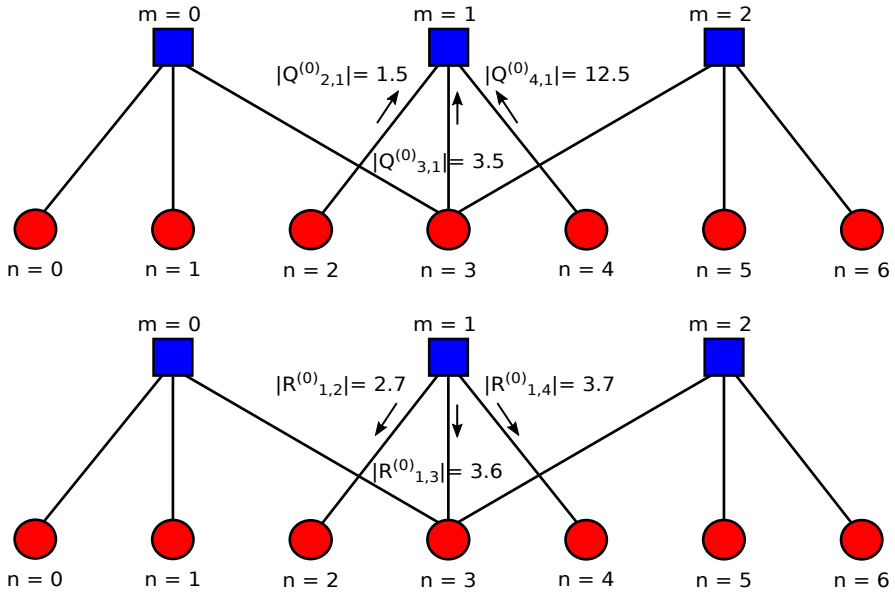

Figure 1.12: Check node magnitude processing for Sum-Product algorithm.

Hence, the signs that come from the neighbour nodes are multiplied to estimate the value of one node. Note that the estimated value of one node is equal to the multiplication of the neighbours (because multiplying the messages with the same sign will always satisfy the equation). An example can be found in Fig. 1.13.

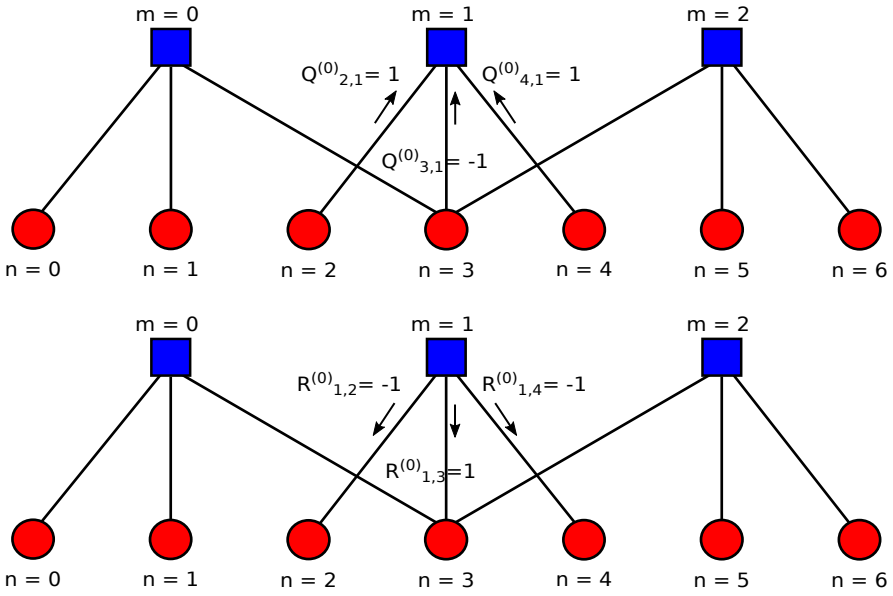

Figure 1.13: Check node sign processing for Sum-Product algorithm.

This process can be summarized with this equation, where *sgn* is the function that extracts the sign of the messages:

$$\sigma_{m,n}^{(i)} = (\prod_{n' \in N(m) \backslash n} sgn(Q_{n',m}^{(i-1)}))$$

After computing the reliability of the output messages (magnitude) and the hard decision (sign), the output messages from the check node can be processed as:

$$R_{m,n}^{(i)} = \sigma_{m,n}^{(i)} \phi^{-1}(\kappa_{m,n}^{(i)})$$

At the variable node the output messages from the check node are added to the channel information, ignoring the contribution of the own node according to the next equation:

$$Q_{m,n}^{(i)} = L_n + \sum_{m' \in M(n) \backslash m} R_{m',n}^{(i)}$$

Finally, when all the $\sigma_{m,n}^{(i)}$ are positive or the maximum number of iterations is reached, all the messages from the check nodes are added at the variable nodes.

At the end of each decoding iteration, hard decision is performed to estimate the received codeword. The hard decision process consists on deciding whether the recibe symbol is $1 \in GF(2)$ or $0 \in GF(2)$. Otherwise, check node and variable node operation are computed again.

$$Q_n = L_n + \sum_{m \in M(n)} R_{m,n}^{(i)}$$

$$\hat{x}_n = \begin{cases} 1, & Q_n < 0 \\ 0, & Q_n \geq 0 \end{cases}$$

where Q_n contains the reliability value of each bit in the tentative decoding codeword. These values are calculated as the addition of the channel information and the incoming check-to-variable messages, without excluding the intrinsic information (note that it is different compared to the calculation of $Q_{m,n}$).

Algorithm 1 Sum-Product decoding algorithm.

Input : $Q_n^{(0)} = L_n$, with $n \in \{0, ..., N-1\}$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update
1 $\kappa_{m,n}^{(i)} = \phi(|Q_{n',m}^{i-1}|), \quad \phi(x) = -log(tanh(\frac{1}{2}x)), x \geq 0$
$\sigma_{m,n}^{(i)} = (\prod_{n' \in N(m) \setminus n} sgn(Q_{n',m}^{(i-1)})$
$R_{m,n}^{(i)} = \sigma_{m,n}^{(i)} \phi^{-1}(\kappa_{m,n}^{(i)})$
Variable-node update
2 $Q_{m,n}^{(i)} = L_n + \sum_{m' \in M(n) \setminus m} R_{m',n}^{(i)}$
Tentative decoding
3 $Q_n^{(i)} = L_n + \sum_{m \in M(n)} R_{m,n}^{(i)}$
4 $\hat{x}_n^{(i)} = \begin{cases} 1, & Q_n^{(i)} < 0 \\ 0, & Q_n^{(i)} \geq 0, \quad n \in \{0, ..., N-1\} \end{cases}$
5 $s_m = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_n^{(i)} h_{m,n} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$
if $(s^{(i)} = 0)$ **then** {SKIP}
end
Output : $\hat{x}$

Sum-Product decoding algorithm provides the best performance among all the decoding algorithms for LDPC codes, but it requires high computational

complexity due to the number of operations performed in the CN for the calculation of *tanh()* and the product.

1.4.1.1. Example 1. Sum-Product algorithm

Assume that a bit sequence of seven zeros is transmitted on a channel with errors. At the receiver, when we read these bits, assume that we have access to the analog level of the received bit. The magnitude of the LLR correlates with the magnitude of the received level. We take the positive sign as a logic 0 and the negative sign as a logic 1.

In this example LLR values are computed as $L_0(y_0) = +15 = n_0$, $L_1(y_1) = +15 = n_1$, $L_2(y_2) = +15 = n_2$, $L_3(y_3) = -1 = n_3$, $L_4(y_4) = +15 = n_4$, $L_5(y_5) = +15 = n_5$ and $L_6(y_6) = +15 = n_6$.

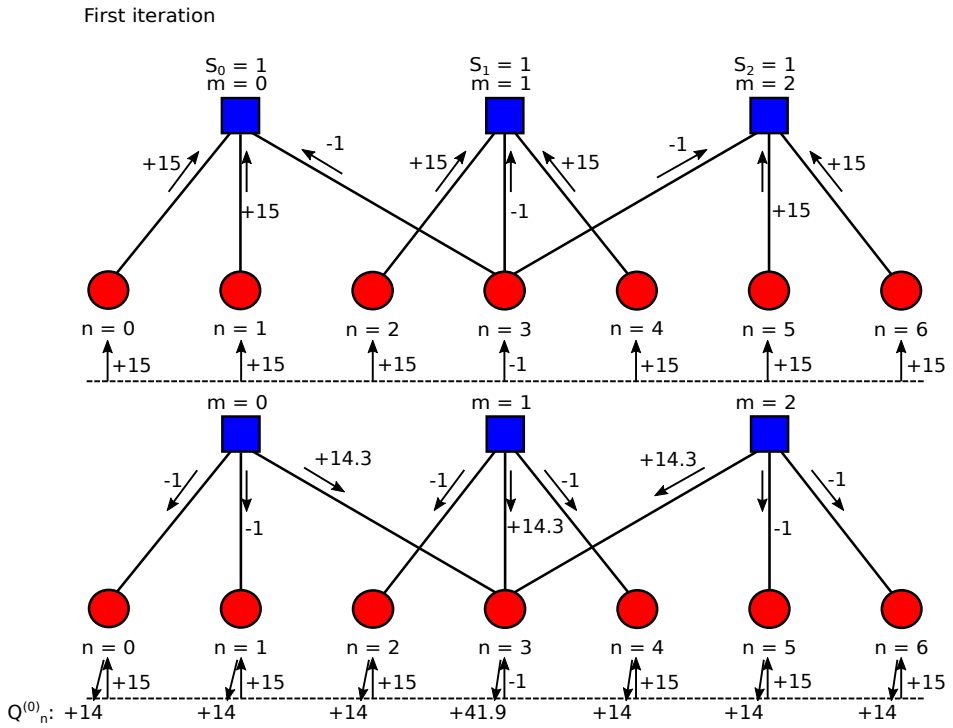

Figure 1.14: SPA example. First iteration.

First, the variable nodes load these received LLR values. After this, messages are sent from variable nodes to check nodes, computing the syndromes. Applying the syndrome equation, the following results are obtained:

$$s_0 = \hat{x}_0^{(0)} \oplus \hat{x}_1^{(0)} \oplus \hat{x}_3^{(0)} = 0 \oplus 0 \oplus 1 = 1$$
$$s_1 = \hat{x}_2^{(0)} \oplus \hat{x}_3^{(0)} \oplus \hat{x}_4^{(0)} = 0 \oplus 1 \oplus 0 = 1$$
$$s_2 = \hat{x}_3^{(0)} \oplus \hat{x}_5^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 \oplus 0 = 1$$

where

$$\hat{x}_n^{(0)} = \begin{cases} 1, & Q_n^{(0)} < 0 \\ 0, & Q_n^{(0)} \geq 0 \end{cases}$$

Once the syndromes are computed, the value that will satisfy each of the parity-check equations is computed. Considering that the incoming messages from the variable node can have an error (processing only the information from the neighbours), the following equation is applied to determine the sign of the message from the check node to the variable node:

$$sgn(R_{m,n}^{(i)}) = sgn(Q_{n,m}^{(i-1)}) \oplus s_m$$

So applying this equation in this example:

$$
\begin{aligned}
sgn(R_{0,0}^{(0)}) &= sgn(Q_{0,0}^{(0)}) \oplus s_0 = 0 \oplus 1 = 1 \\
sgn(R_{0,1}^{(0)}) &= sgn(Q_{1,0}^{(0)}) \oplus s_0 = 0 \oplus 1 = 1 \\
sgn(R_{0,3}^{(0)}) &= sgn(Q_{3,0}^{(0)}) \oplus s_0 = 1 \oplus 1 = 0 \\
sgn(R_{1,2}^{(0)}) &= sgn(Q_{2,1}^{(0)}) \oplus s_1 = 0 \oplus 1 = 1 \\
sgn(R_{1,3}^{(0)}) &= sgn(Q_{3,1}^{(0)}) \oplus s_1 = 1 \oplus 1 = 0 \\
sgn(R_{1,4}^{(0)}) &= sgn(Q_{4,1}^{(0)}) \oplus s_1 = 0 \oplus 1 = 1 \\
sgn(R_{2,3}^{(0)}) &= sgn(Q_{3,2}^{(0)}) \oplus s_2 = 1 \oplus 1 = 0 \\
sgn(R_{2,5}^{(0)}) &= sgn(Q_{5,2}^{(0)}) \oplus s_2 = 0 \oplus 1 = 1 \\
sgn(R_{2,6}^{(0)}) &= sgn(Q_{6,2}^{(0)}) \oplus s_2 = 0 \oplus 1 = 1
\end{aligned}
$$

The previous results can be compared to the sign of the message exchange in Fig. 1.14. The computation of the magnitude of the messages is done as follows:

$$
\begin{aligned}
|R_{0,0}^{(0)}| &= \phi^{-1}(\phi(|Q_{0,0}^{(0)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(1/2)) = 1 \\
|R_{0,1}^{(0)}| &= \phi^{-1}(\phi(|Q_{1,0}^{(0)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(1/2)) = 1 \\
|R_{0,3}^{(0)}| &= \phi^{-1}(\phi(|Q_{3,0}^{(0)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(15/2)) = 14.3 \\
|R_{1,2}^{(0)}| &= \phi^{-1}(\phi(|Q_{2,1}^{(0)}|)) = 2 \cdot arctanh(tanh(1/2) \cdot tanh(15/2)) = 1 \\
|R_{1,3}^{(0)}| &= \phi^{-1}(\phi(|Q_{3,1}^{(0)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(15/2)) = 14.3 \\
|R_{1,4}^{(0)}| &= \phi^{-1}(\phi(|Q_{4,1}^{(0)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(1/2)) = 1 \\
|R_{2,3}^{(0)}| &= \phi^{-1}(\phi(|Q_{3,2}^{(0)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(15/2)) = 14.3 \\
|R_{2,5}^{(0)}| &= \phi^{-1}(\phi(|Q_{5,2}^{(0)}|)) = 2 \cdot arctanh(tanh(1/2) \cdot tanh(15/2)) = 1 \\
|R_{2,6}^{(0)}| &= \phi^{-1}(\phi(|Q_{6,2}^{(0)}|)) = 2 \cdot arctanh(tanh(1/2) \cdot tanh(15/2)) = 1
\end{aligned}
$$

where $\phi(x) = \prod_{n' \in N(m) \setminus n} tanh(\frac{1}{2}x), \quad x \geq 0$

Finally, the tentative decoding is computed by adding the LLR value received from the channel to the incoming check-to-variable node messages as follows:

$$
\begin{aligned}
Q_0^{(0)} &= L_0(y_0) + R_{0,0}^{(0)} = 15 + (-1) = +14 \\
Q_1^{(0)} &= L_1(y_1) + R_{0,1}^{(0)} = 15 + (-1) = +14 \\
Q_2^{(0)} &= L_2(y_2) + R_{1,2}^{(0)} = 15 + (-1) = +14 \\
Q_3^{(0)} &= L_3(y_3) + R_{0,3}^{(0)} + R_{1,3}^{(0)} + R_{2,3}^{(0)} = -1 + 14.3 + 14.3 + 14.3 = +41.9 \\
Q_4^{(0)} &= L_4(y_4) + R_{1,4}^{(0)} = 15 + (-1) = +14 \\
Q_5^{(0)} &= L_5(y_5) + R_{2,5}^{(0)} = 15 + (-1) = +14 \\
Q_6^{(0)} &= L_6(y_6) + R_{2,6}^{(0)} = 15 + (-1) = +14
\end{aligned}
$$

The variable-to-check messages obtained are:

$$
\begin{aligned}
Q_{0,0}^{(0)} &= L_0(y_0) = +15 \\
Q_{1,0}^{(0)} &= L_1(y_1) = +15 \\
Q_{2,0}^{(0)} &= L_2(y_2) = +15 \\
Q_{3,0}^{(0)} &= L_3(y_3) + R_{1,3}^{(0)} + R_{2,3}^{(0)} = -1 + 14.3 + 14.3 = +27.6 \\
Q_{3,1}^{(0)} &= L_3(y_3) + R_{0,3}^{(0)} + R_{2,3}^{(0)} = -1 + 14.3 + 14.3 = +27.6 \\
Q_{3,2}^{(0)} &= L_3(y_3) + R_{0,3}^{(0)} + R_{1,3}^{(0)} = -1 + 14.3 + 14.3 = +27.6 \\
Q_{4,1}^{(0)} &= L_4(y_4) = +15 \\
Q_{5,2}^{(0)} &= L_5(y_5) = +15 \\
Q_{6,3}^{(0)} &= L_6(y_6) = +15
\end{aligned}
$$

Performing a second iteration:

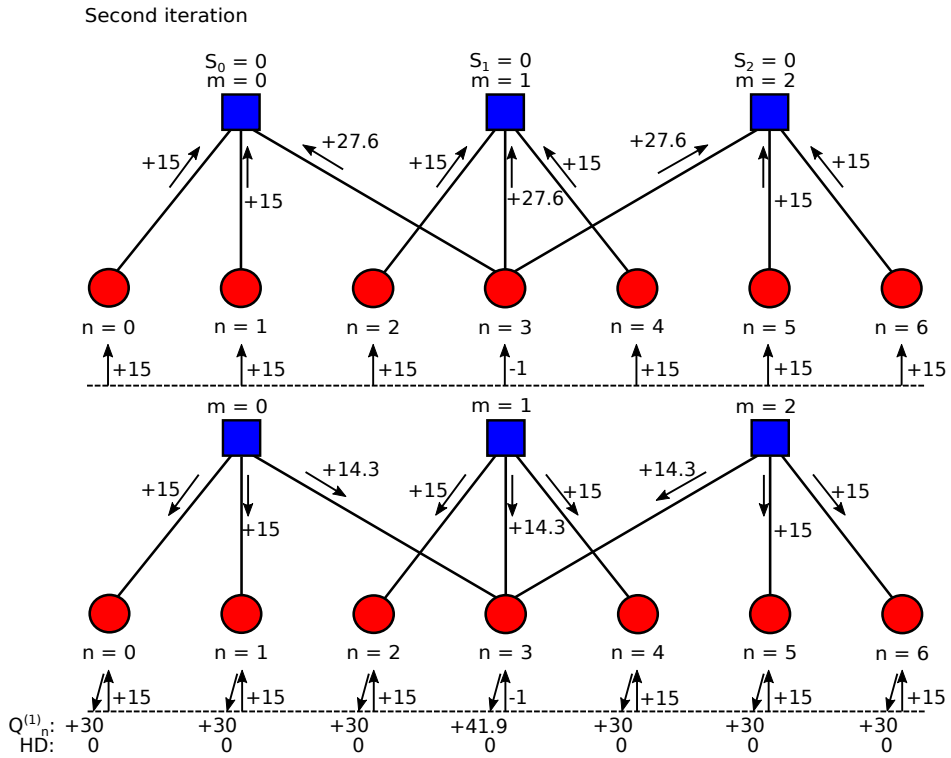

Figure 1.15: SPA example. Second iteration.

The messages computed in each variable node are sent from variable nodes to check nodes, computing the syndromes:

$$s_0 = \hat{x}_0^{(1)} \oplus \hat{x}_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 \oplus 0 = 0$$
$$s_1 = \hat{x}_2^{(1)} \oplus \hat{x}_3^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 \oplus 0 = 0$$
$$s_2 = \hat{x}_3^{(1)} \oplus \hat{x}_5^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 \oplus 0 = 0$$

where

$$\hat{x}_n^{(1)} = \begin{cases} 1, & Q_n^{(1)} < 0 \\ 0, & Q_n^{(1)} \geq 0 \end{cases}$$

Once the syndromes are computed, the value that will satisfy each of the parity-check equations is computed. On the one hand, the sign is calculated:

$$
\begin{aligned}
sgn(R_{0,0}^{(1)}) &= sgn(Q_{0,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0 \\
sgn(R_{0,1}^{(1)}) &= sgn(Q_{1,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0 \\
sgn(R_{0,3}^{(1)}) &= sgn(Q_{3,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0 \\
sgn(R_{1,2}^{(1)}) &= sgn(Q_{2,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0 \\
sgn(R_{1,3}^{(1)}) &= sgn(Q_{3,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0 \\
sgn(R_{1,4}^{(1)}) &= sgn(Q_{4,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0 \\
sgn(R_{2,3}^{(1)}) &= sgn(Q_{3,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0 \\
sgn(R_{2,5}^{(1)}) &= sgn(Q_{5,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0 \\
sgn(R_{2,6}^{(1)}) &= sgn(Q_{6,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0
\end{aligned}
$$

On the other hand, the computation of the magnitude of the messages is done as follows:

$$
\begin{aligned}
|R_{0,0}^{(1)}| &= \phi^{-1}(\phi(|Q_{0,0}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(27.6/2)) = 15 \\
|R_{0,1}^{(1)}| &= \phi^{-1}(\phi(|Q_{1,0}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(27.6/2)) = 15 \\
|R_{0,3}^{(1)}| &= \phi^{-1}(\phi(|Q_{3,0}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(15/2)) = 14.3 \\
|R_{1,2}^{(1)}| &= \phi^{-1}(\phi(|Q_{2,1}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(27.6/2)) = 15 \\
|R_{1,3}^{(1)}| &= \phi^{-1}(\phi(|Q_{3,1}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(15/2)) = 14.3 \\
|R_{1,4}^{(1)}| &= \phi^{-1}(\phi(|Q_{4,1}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(27.6/2)) = 15 \\
|R_{2,3}^{(1)}| &= \phi^{-1}(\phi(|Q_{3,2}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(15/2)) = 14.3 \\
|R_{2,5}^{(1)}| &= \phi^{-1}(\phi(|Q_{5,2}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(27.6/2)) = 15 \\
|R_{2,6}^{(1)}| &= \phi^{-1}(\phi(|Q_{6,2}^{(1)}|)) = 2 \cdot arctanh(tanh(15/2) \cdot tanh(27.6/2)) = 15
\end{aligned}
$$

where $\phi(x) = \prod_{n' \in N(m) \setminus n} tanh(\frac{1}{2}x), \quad x \geq 0$

In this second iteration the reliability values for computing the tentative decoding are:

$$
\begin{aligned}
Q_0^{(1)} &= L_0(y_0) + R_{0,0}^{(1)} = 15 + 15 = +30 \\
Q_1^{(1)} &= L_1(y_1) + R_{0,1}^{(1)} = 15 + 15 = +30 \\
Q_2^{(1)} &= L_2(y_2) + R_{1,2}^{(1)} = 15 + 15 = +30 \\
Q_3^{(1)} &= L_3(y_3) + R_{0,3}^{(1)} + R_{1,3}^{(1)} + R_{2,3}^{(1)} = -1 + 14.3 + 14.3 + 14.3 = +41.9 \\
Q_4^{(1)} &= L_4(y_4) + R_{1,4}^{(1)} = 15 + 15 = +30 \\
Q_5^{(1)} &= L_5(y_5) + R_{2,5}^{(1)} = 15 + 15 = +30 \\
Q_6^{(1)} &= L_6(y_6) + R_{2,6}^{(1)} = 15 + 15 = +30
\end{aligned}
$$

The variable-to-check messages obtained are:

$$
\begin{aligned}
Q_{0,0}^{(1)} &= L_0(y_0) = +15 \\
Q_{1,0}^{(1)} &= L_1(y_1) = +15 \\
Q_{2,1}^{(1)} &= L_2(y_2) = +15 \\
Q_{3,0}^{(1)} &= L_3(y_3) + R_{1,3}^{(1)} + R_{2,3}^{(1)} = -1 + 14.3 + 14.3 = +27.6 \\
Q_{3,1}^{(1)} &= L_3(y_3) + R_{0,3}^{(1)} + R_{2,3}^{(1)} = -1 + 14.3 + 14.3 = +27.6 \\
Q_{3,2}^{(1)} &= L_3(y_3) + R_{0,3}^{(1)} + R_{1,3}^{(1)} = -1 + 14.3 + 14.3 = +27.6 \\
Q_{4,1}^{(1)} &= L_4(y_4) = +15 \\
Q_{5,2}^{(1)} &= L_5(y_5) = +15 \\
Q_{6,2}^{(1)} &= L_6(y_6) = +15
\end{aligned}
$$

Finally, it can be checked that all the syndromes are satisfied and, hence, the code word is corrected. The error in variable node $n = 3$ has been corrected, flipping the sign of the node from negative to positive. All the Q_n messages are now positive, and all the hard-decision messages are zero, as shown in Fig. 1.15.

1.4.2. Scaled Min-Sum algorithm

The Min-Sum algorithm (MSA) [16] - [17] is a reduced complexity decoding algorithm which simplifies operations in the check node compared to the SPA. Note that *tanh* and product operations are eliminated in MSA.

In the first stage of iteration i, the magnitudes of the messages are computed as follows:

$$\kappa_{m,n}^{(i)} = |R_{m,n}^{(i)}| = \min_{n' \in N(m) \backslash n} |Q_{n',m}^{(i-1)}|$$

The complexity of the calculation of the minimum magnitude can be greatly reduced by applying the technique of the value reuse. It can be easily demonstrated that the magnitude of the output messages, in $(d_c - 1)$ cases, is the absolute minimum of the d_c input messages. Only for the output message of the node that sends the least reliable message, the magnitude will be the second least reliable magnitude within the d_c input messages. This computation is named as value reuse ($\kappa_{m,n}^{(i)}$) because only one search of the first minimum magnitude and the second minimum magnitude is required to compute the d_c output messages, not d_c minimum searches.

$$\kappa_{m,n}^{(i)} = |R_{m,n}^{(i)}| = \begin{cases} \min |Q_{n',m}^{(i-1)}|, if \min\limits_{n' \in N(m)} |Q_{n',m}^{(i-1)}| \neq |Q_{n,m}^{(i-1)}| \\ \min_2 |Q_{n',m}^{(i-1)}|, if \min\limits_{n' \in N(m)} |Q_{n',m}^{(i-1)}| = |Q_{n,m}^{(i-1)}| \end{cases}$$

The process of the check node computation in the second phase is similar to the SPA. Therefore, the output messages from the check node can be processed as:

$$R_{m,n}^{(i)} = \sigma_{m,n}^{(i)} \kappa_{m,n}^{(i)}$$

Finally, the output messages from the check node are computed as:

$$Q_{m,n}^{(i)} = L_n + \alpha \sum_{m' \in M(n) \backslash m} R_{m',n}^{(i)}$$

To estimate the received codeword, Q_n is used instead of $Q_{m,n}$.

The Min-Sum algorithm overestimates the messages at the check node, so a scaling factor or an offset correction factor should be applied to $R_{m,n}^{(i)}$ messages and/or $Q_{m,n}^{(i)}$ messages for better performance, that is, to get a performance closer to Sum-Product algorithm.

Algorithm 2 Min-Sum decoding algorithm.

Input : $Q_n^{(0)} = L_n,$ with $n \in \{0, ..., N-1\}$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update
1 $\kappa_{m,n}^{(i)} = \min_{n' \in N(m) \backslash n} |Q_{n',m}^{(i-1)}|$
$\sigma_{m,n}^{(i)} = (\prod_{n' \in N(m) \backslash n} sgn(Q_{n',m}^{(i-1)})$
$R_{m,n}^{(i)} = \sigma_{m,n}^{(i)} \kappa_{m,n}^{(i)}$
Variable-node update
2 $Q_{m,n}^{(i)} = L_n + \alpha \sum_{m' \in M(n) \backslash m} R_{m',n}^{(i)}$
Tentative decoding
3 $Q_n^{(i)} = L_n + \alpha \sum_{m \in M(n)} R_{m,n}^{(i)}$
4 $\hat{x}_n^{(i)} = \begin{cases} 1, & Q_n^{(i)} < 0 \\ 0, & Q_n^{(i)} \geq 0, \quad n \in \{0, ..., N-1\} \end{cases}$
5 $s_m = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_n^{(i)} h_{m,n} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$
if $(s^{(i)} = 0)$ **then** {SKIP}
end
Output : $\hat{x}$

With the calculation of the two minimums, Min-Sum decoding algorithm reduces complexity considerably by means of simplifying the number of operations performed in the CN. But on the contrary, a slightly worse performance is obtained compared to Sum-Product decoding algorithm. Although this algorithm reduces the complexity compared to the previous one; it still has high complexity in the CN because it requires a double tree to find the minimum in the derived hardware implementations, and the magnitudes are quantized with more than two bits. This quantization of more than two bits makes the exchange of messages larger. This two facts have a great impact on the arithmetic resources and the wiring congestion of the derived architectures.

1.4.2.1. Example 2. Scaled Min-Sum algorithm

In this example we assume the same conditions as in Example 1, with the same input values (LLR) and, in this case, we use a scale factor $\alpha = 0.75$.

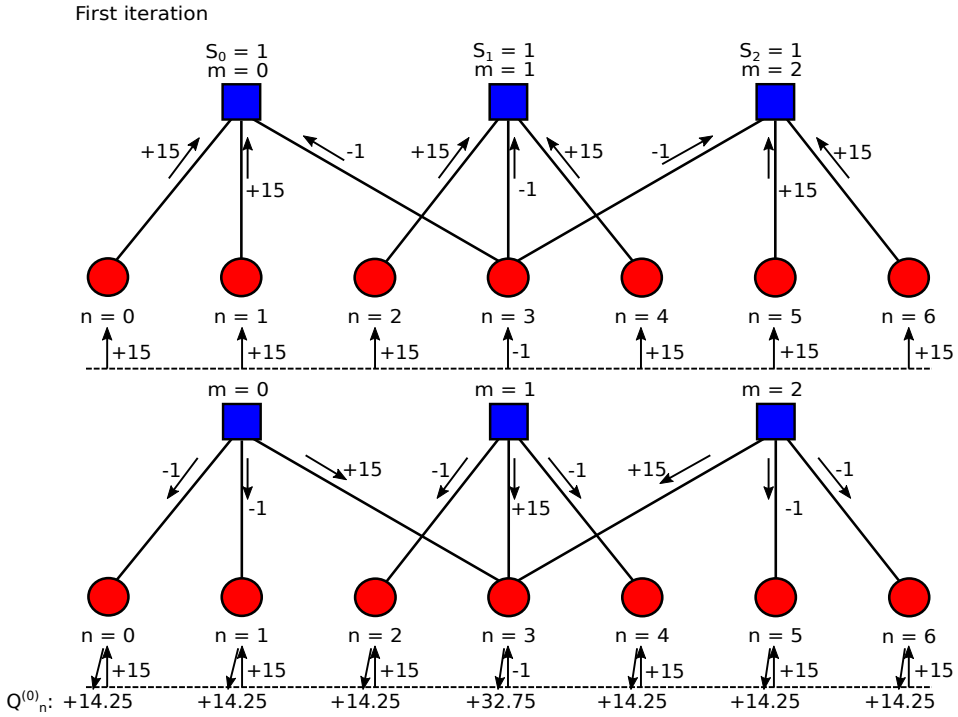

Figure 1.16: MSA example. First iteration.

Like in the previous example, first the LLR values are loaded in the variable nodes. Once this is done, messages are sent from variable nodes to check nodes, computing the syndromes:

$$s_0 = \hat{x}_0^{(0)} \oplus \hat{x}_1^{(0)} \oplus \hat{x}_3^{(0)} = 0 \oplus 0 \oplus 1 = 1$$
$$s_1 = \hat{x}_2^{(0)} \oplus \hat{x}_3^{(0)} \oplus \hat{x}_4^{(0)} = 0 \oplus 1 \oplus 0 = 1$$
$$s_2 = \hat{x}_3^{(0)} \oplus \hat{x}_5^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 \oplus 0 = 1$$

where

$$\hat{x}_n^{(0)} = \begin{cases} 1, & Q_n^{(0)} < 0 \\ 0, & Q_n^{(0)} \geq 0 \end{cases}$$

With the syndromes computed, the value that will satisfy each of the parity-check equations is calculated. Considering that the incoming messages from the variable node can have an error (processing only the information from the neighbours), the following equation is applied to determine the sign of the message from the check node to the variable node:

$$sgn(R_{m,n}^{(i)}) = sgn(Q_{n,m}^{(i-1)}) \oplus s_m$$

So the signs are computed as:

$$\begin{aligned}
sgn(R_{0,0}^{(0)}) &= sgn(Q_{0,0}^{(0)}) \oplus s_0 = 0 \oplus 1 = 1 \\
sgn(R_{0,1}^{(0)}) &= sgn(Q_{1,0}^{(0)}) \oplus s_0 = 0 \oplus 1 = 1 \\
sgn(R_{0,3}^{(0)}) &= sgn(Q_{3,0}^{(0)}) \oplus s_0 = 1 \oplus 1 = 0 \\
sgn(R_{1,2}^{(0)}) &= sgn(Q_{2,1}^{(0)}) \oplus s_1 = 0 \oplus 1 = 1 \\
sgn(R_{1,3}^{(0)}) &= sgn(Q_{3,1}^{(0)}) \oplus s_1 = 1 \oplus 1 = 0 \\
sgn(R_{1,4}^{(0)}) &= sgn(Q_{4,1}^{(0)}) \oplus s_1 = 0 \oplus 1 = 1 \\
sgn(R_{2,3}^{(0)}) &= sgn(Q_{3,2}^{(0)}) \oplus s_2 = 1 \oplus 1 = 0 \\
sgn(R_{2,5}^{(0)}) &= sgn(Q_{5,2}^{(0)}) \oplus s_2 = 0 \oplus 1 = 1 \\
sgn(R_{2,6}^{(0)}) &= sgn(Q_{6,2}^{(0)}) \oplus s_2 = 0 \oplus 1 = 1
\end{aligned}$$

We can check the previous results with the sign of the message exchange in Fig. 1.16. Following with the computation of the magnitude of the messages, in this case, the minimum of the absolute value of the neighbour messages is selected as the magnitude of the check-node outputs:

$$|R_{0,0}^{(0)}| = min(|Q_{1,0}^{(0)}|, |Q_{3,0}^{(0)}|) = 1$$
$$|R_{0,1}^{(0)}| = min(|Q_{0,0}^{(0)}|, |Q_{3,0}^{(0)}|) = 1$$
$$|R_{0,3}^{(0)}| = min(|Q_{0,0}^{(0)}|, |Q_{1,0}^{(0)}|) = 15$$
$$|R_{1,2}^{(0)}| = min(|Q_{3,1}^{(0)}|, |Q_{4,1}^{(0)}|) = 1$$
$$|R_{1,3}^{(0)}| = min(|Q_{2,1}^{(0)}|, |Q_{4,1}^{(0)}|) = 15$$
$$|R_{1,4}^{(0)}| = min(|Q_{2,1}^{(0)}|, |Q_{3,1}^{(0)}|) = 1$$
$$|R_{2,3}^{(0)}| = min(|Q_{5,2}^{(0)}|, |Q_{6,2}^{(0)}|) = 15$$
$$|R_{2,5}^{(0)}| = min(|Q_{3,2}^{(0)}|, |Q_{6,2}^{(0)}|) = 1$$
$$|R_{2,6}^{(0)}| = min(|Q_{3,2}^{(0)}|, |Q_{5,2}^{(0)}|) = 1$$

Finally, the tentative decoding is computed by adding the LLR value received from the channel to the incoming check-to-variable node messages as follows:

$$Q_0^{(0)} = L_0(y_0) + (\alpha \cdot R_{0,0}^{(0)}) = 15 + (0.75 \cdot (-1)) = +14.25$$
$$Q_1^{(0)} = L_1(y_1) + (\alpha \cdot R_{0,1}^{(0)}) = 15 + (0.75 \cdot (-1)) = +14.25$$
$$Q_2^{(0)} = L_2(y_2) + (\alpha \cdot R_{1,2}^{(0)}) = 15 + (0.75 \cdot (-1)) = +14.25$$
$$Q_3^{(0)} = L_3(y_3) + (\alpha \cdot (R_{0,3}^{(0)} + R_{1,3}^{(0)} + R_{2,3}^{(0)})) = (-1) + (0.75 \cdot (15 + 15 + 15)) = +32.75$$
$$Q_4^{(0)} = L_4(y_4) + (\alpha \cdot R_{1,4}^{(0)}) = 15 + (0.75 \cdot (-1)) = +14.25$$
$$Q_5^{(0)} = L_5(y_5) + (\alpha \cdot R_{2,5}^{(0)}) = 15 + (0.75 \cdot (-1)) = +14.25$$
$$Q_6^{(0)} = L_6(y_6) + (\alpha \cdot R_{2,6}^{(0)}) = 15 + (0.75 \cdot (-1)) = +14.25$$

The variable-to-check messages obtained are:

$$Q_{0,0}^{(0)} = L_0(y_0) = +15$$
$$Q_{1,0}^{(0)} = L_1(y_1) = +15$$
$$Q_{2,1}^{(0)} = L_2(y_2) = +15$$
$$Q_{3,0}^{(0)} = L_3(y_3) + (\alpha \cdot (R_{1,3}^{(0)} + R_{2,3}^{(0)})) = (-1) + (0.75 \cdot (15 + 15)) = +21.5$$
$$Q_{3,1}^{(0)} = L_3(y_3) + (\alpha \cdot (R_{0,3}^{(0)} + R_{2,3}^{(0)})) = (-1) + (0.75 \cdot (15 + 15)) = +21.5$$
$$Q_{3,2}^{(0)} = L_3(y_3) + (\alpha \cdot (R_{0,3}^{(0)} + R_{1,3}^{(0)})) = (-1) + (0.75 \cdot (15 + 15)) = +21.5$$
$$Q_{4,1}^{(0)} = L_4(y_4) = +15$$
$$Q_{5,2}^{(0)} = L_5(y_5) = +15$$
$$Q_{6,2}^{(0)} = L_6(y_6) = +15$$

Performing a second iteration:

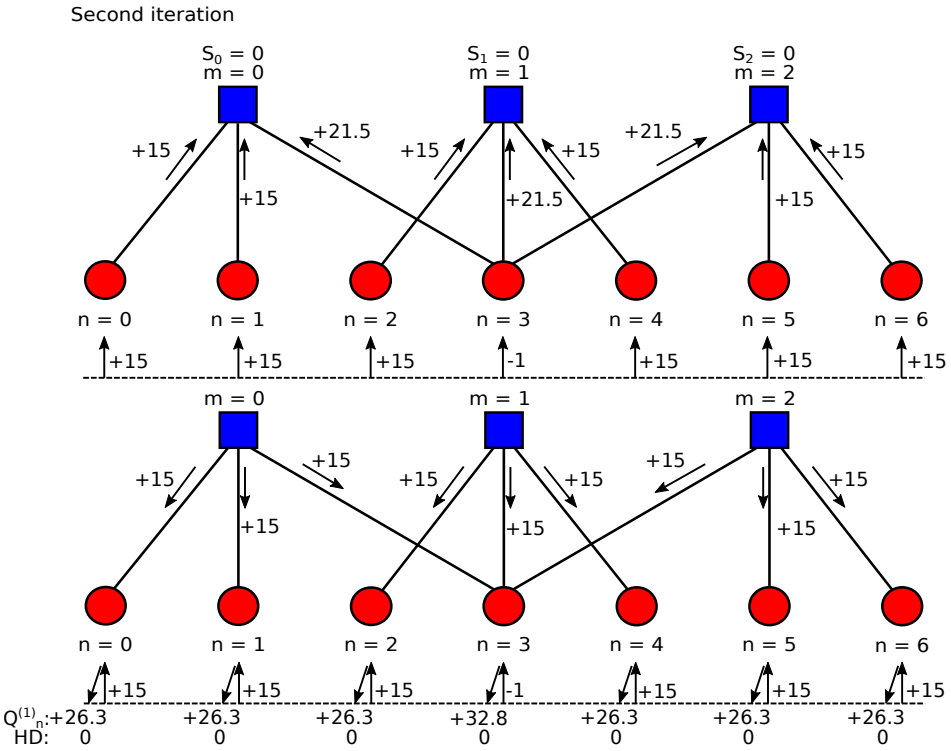

Figure 1.17: MSA example. Second iteration.

The messages computed in each variable node are sent from variable nodes to check nodes, computing the syndromes:

$$s_0 = \hat{x}_0^{(1)} \oplus \hat{x}_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 \oplus 0 = 0$$
$$s_1 = \hat{x}_2^{(1)} \oplus \hat{x}_3^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 \oplus 0 = 0$$
$$s_2 = \hat{x}_3^{(1)} \oplus \hat{x}_5^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 \oplus 0 = 0$$

where

$$\hat{x}_n^{(1)} = \begin{cases} 1, & Q_n^{(1)} < 0 \\ 0, & Q_n^{(1)} \geq 0 \end{cases}$$

Once the syndromes are computed, the value that will satisfy each of the parity-check equations is processed. On the one hand, the sign is calculated:

$$
\begin{aligned}
sgn(R_{0,0}^{(1)}) &= sgn(Q_{0,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0 \\
sgn(R_{0,1}^{(1)}) &= sgn(Q_{1,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0 \\
sgn(R_{0,3}^{(1)}) &= sgn(Q_{3,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0 \\
sgn(R_{1,2}^{(1)}) &= sgn(Q_{2,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0 \\
sgn(R_{1,3}^{(1)}) &= sgn(Q_{3,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0 \\
sgn(R_{1,4}^{(1)}) &= sgn(Q_{4,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0 \\
sgn(R_{2,3}^{(1)}) &= sgn(Q_{3,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0 \\
sgn(R_{2,5}^{(1)}) &= sgn(Q_{5,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0 \\
sgn(R_{2,6}^{(1)}) &= sgn(Q_{6,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0
\end{aligned}
$$

On the other hand, the magnitude of the messages is computed:

$$
\begin{aligned}
|R_{0,0}^{(1)}| &= min(|Q_{1,0}^{(1)}|, |Q_{3,0}^{(1)}|) = 15 \\
|R_{0,1}^{(1)}| &= min(|Q_{0,0}^{(1)}|, |Q_{3,0}^{(1)}|) = 15 \\
|R_{0,3}^{(1)}| &= min(|Q_{0,0}^{(1)}|, |Q_{1,0}^{(1)}|) = 15 \\
|R_{1,2}^{(1)}| &= min(|Q_{3,1}^{(1)}|, |Q_{4,1}^{(1)}|) = 15 \\
|R_{1,3}^{(1)}| &= min(|Q_{2,1}^{(1)}|, |Q_{4,1}^{(1)}|) = 15 \\
|R_{1,4}^{(1)}| &= min(|Q_{2,1}^{(1)}|, |Q_{3,1}^{(1)}|) = 15 \\
|R_{2,3}^{(1)}| &= min(|Q_{5,2}^{(1)}|, |Q_{6,2}^{(1)}|) = 15 \\
|R_{2,5}^{(1)}| &= min(|Q_{3,2}^{(1)}|, |Q_{6,2}^{(1)}|) = 15 \\
|R_{2,6}^{(1)}| &= min(|Q_{3,2}^{(1)}|, |Q_{5,2}^{(1)}|) = 15
\end{aligned}
$$

In this second iteration, the reliability values for computing the tentative decoding are:

$$Q_0^{(1)} = L_0(y_0) + (\alpha \cdot R_{0,0}^{(1)}) = 15 + (0.75 \cdot 15) = +26.3$$
$$Q_1^{(1)} = L_1(y_1) + (\alpha \cdot R_{0,1}^{(1)}) = 15 + (0.75 \cdot 15) = +26.3$$
$$Q_2^{(1)} = L_2(y_2) + (\alpha \cdot R_{1,2}^{(1)}) = 15 + (0.75 \cdot 15) = +26.3$$
$$Q_3^{(1)} = L_3(y_3) + (\alpha \cdot (R_{0,3}^{(1)} + R_{1,3}^{(1)} + R_{2,3}^{(1)})) = (-1) + (0.75 \cdot (15 + 15 + 15)) = +32.8$$
$$Q_4^{(1)} = L_4(y_4) + (\alpha \cdot R_{1,4}^{(1)}) = 15 + (0.75 \cdot (-1)) = +26.3$$
$$Q_5^{(1)} = L_5(y_5) + (\alpha \cdot R_{2,5}^{(1)}) = 15 + (0.75 \cdot (-1)) = +26.3$$
$$Q_6^{(1)} = L_6(y_6) + (\alpha \cdot R_{2,6}^{(1)}) = 15 + (0.75 \cdot (-1)) = +26.3$$

The variable-to-check messages obtained are:

$$Q_{0,0}^{(1)} = L_0(y_0) = +15$$
$$Q_{1,0}^{(1)} = L_1(y_1) = +15$$
$$Q_{2,1}^{(1)} = L_2(y_2) = +15$$
$$Q_{3,0}^{(1)} = L_3(y_3) + (\alpha \cdot (R_{1,3}^{(1)} + R_{2,3}^{(1)})) = (-1) + (0.75 \cdot (15 + 15)) = +21.5$$
$$Q_{3,1}^{(1)} = L_3(y_3) + (\alpha \cdot (R_{0,3}^{(1)} + R_{2,3}^{(1)})) = (-1) + (0.75 \cdot (15 + 15)) = +21.5$$
$$Q_{3,2}^{(1)} = L_3(y_3) + (\alpha \cdot (R_{0,3}^{(1)} + R_{1,3}^{(1)})) = (-1) + (0.75 \cdot (15 + 15)) = +21.5$$
$$Q_{4,1}^{(1)} = L_4(y_4) = +15$$
$$Q_{5,2}^{(1)} = L_5(y_5) = +15$$
$$Q_{6,2}^{(1)} = L_6(y_6) = +15$$

Finally, it can be checked that all the syndromes are satisfied and, hence, the code word is corrected. The error in variable node $n = 3$ has been corrected, flipping the sign of the node from negative to positive. All the Q_n messages are now positive, and all the hard-decision messages are zero, as shown in Fig. 1.17.

1.4.3. Optimized 2-bit Min-Sum algorithm

An optimised 2-bit Min-Sum decoding algorithm was proposed in [18] in order to reduce the complexity and the routing congestion of Min-Sum decoding algorithm. This algorithm works as Min-Sum algorithm with the only difference that each message sent from one node to another is 2 bits $b_s b_m$. These 2 bits are composed of the hard decision of a received soft message b_s, and the hard decision confidence b_m, where $b_m = 1$ represents a high confidence and $b_m = 0$ represents a low confidence. Fig. 1.18 and Fig. 1.19 show the SPA and MSA routing congestion and Optimized 2-bit MSA routing congestion, respectively.

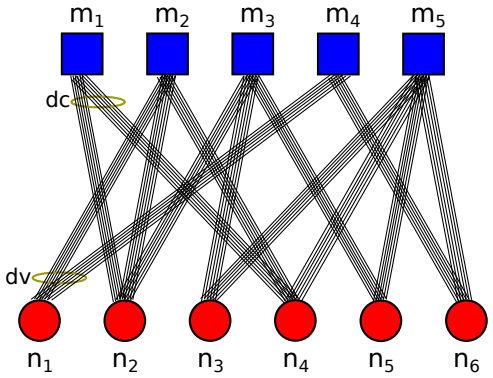

Figure 1.18: 6-bit message SPA and MSA interconnection.

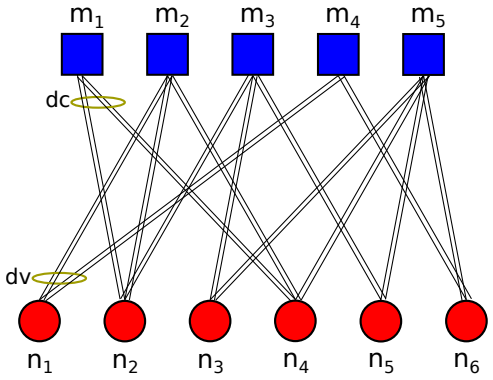

Figure 1.19: 2-bit message Optimized 2-bit min-sum decoding algorithms interconnection.

Defining y as the input message and T_y as an input threshold. Bit b_s value is equivalent to the sign of input message y and bit b_m value is obtained doing a comparation with T_y threshold as follows:

$$b_s b_m = \begin{cases} 01, & if \quad y > T_y \\ 00, & if \quad T_y \geq y \geq 0 \\ 10, & if \quad 0 > y \geq -T_y \\ 11, & if \quad y < -T_y \end{cases}$$

In this case, the check-to-variable and variable-to-check messages updating phases are calculated as:

$$R_{m,n}^{(i)} = \prod_{n' \in N(m) \backslash n} sign(Q_{n',m}^{(i-1)}) \min_{n' \in N(m) \backslash n} |Q_{n',m}^{(i-1)}|$$

$$Q_{m,n}^{(i)} = Q_n = g(f(L_n) + \alpha \sum_{m \in M(n)} f(R_{m,n}^{(i)}))$$

This algorithm has two functions which convert a 2-bit message into an integer and an integer into a 2-bit message. These functions are $f(\cdot)$ and $g(\cdot)$, respectively. α is the scaling factor used in this algorithm to improve the bit-error rate.

Three steps are needed to compute variable-to-check messages:

1. The input 2-bit data L_n or $R_{m',n}$ is converted through function $f(\cdot)$ to an integer number, which represents the confidence level. The converted integer is set to W_L if $b_m = 0$, low confidence, and to W_H if $b_m = 1$, high confidence.

$$f(b_s b_m) = \begin{cases} W_L, & if \quad b_s b_m = 00 \\ W_H, & if \quad b_s b_m = 01 \\ -W_L, & if \quad b_s b_m = 10 \\ -W_H, & if \quad b_s b_m = 11 \end{cases}$$

2. The addition of all the check nodes connected to the evaluated variable node is performed to obtain the total confidence level.

3. The result from point 2 is turned back into a 2-bit message through function $g(\cdot)$, doing a comparation with threshold, T_L, designed to decide the new bit value and whether its confidence level is high or low:

$$b_s b_m = g(x) = \begin{cases} 00, & if \quad T_L > x \geq 0 \\ 01, & if \quad x \geq T_L \\ 10, & if \quad 0 > x > -T_L \\ 11, & if \quad x \leq -T_L \end{cases}$$

Finally the Q_n is calulated. These values are calculated as the addition of the channel information and the incoming check-to-variable messages, without excluding the intrinsic information (note that in this case $Q_n = Q_{m,n}$).

Algorithm 3 Optimized 2 bit Min-Sum decoding algorithm.

Input : $Q_n^{(0)} = L_n = b_s b_m$, where
$b_s = sign(L_n)$
$b_m = \begin{cases} 1, & if|L_n| > T_y \\ 0, & otherwise \end{cases}$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update

1 $R_{m,n}^{(i)} = \prod_{n' \in N(m) \backslash n} sign(Q_{n',m}^{(i-1)}) \min_{n' \in N(m) \backslash n} |Q_{n',m}^{(i-1)}|$

Variable-node update

2 $Q_n^{(i)} = Q_{m,n}^{(i)} = g(f(L_n) + \alpha \sum_{m \in M(n)} f(R_{m,n}^{(i)}))$

Tentative decoding

3 $\hat{x}_n^{(i)} = \begin{cases} 1, & Q_n^{(i)} < 0 \\ 0, & Q_n^{(i)} \geq 0, \quad n \in \{0, ..., N-1\} \end{cases}$

4 $s_m = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_n^{(i)} h_{m,n} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$

if $(s^{(i)} = 0)$ **then** {SKIP}
end
Output : $\hat{x}$

Optimized 2-bit Min-Sum decoding algorithm significantly reduces the complexity, the hardware resources and the wiring congestion due to the use of only 2 bits for each message. The equations for the check-to-variable and variable-to-check require very simple combinatorial logic computations, which means a reduction of logic gates in both computational units. In addition, using 2 bits for message exchange, a large number of memory resources are reduced compared to SPA and MSA. It should be pointed out that Optimized 2 bit Min-Sum decoding algorithm is not well suited for decoding LDPC codes with very low column weights (2 or 3) because of the large performance loss compared to SPA.

1.4.3.1. Example 3. Optimized 2-bit Min-Sum algorithm

In this example we assume the same conditions and input values (LLR) as the previous examples. Also, we use a scale factor $\alpha = 0.75$ and $T_Y = 1.5$, $W_H = 5$, $W_L = 1$ and $T_L = 3$.

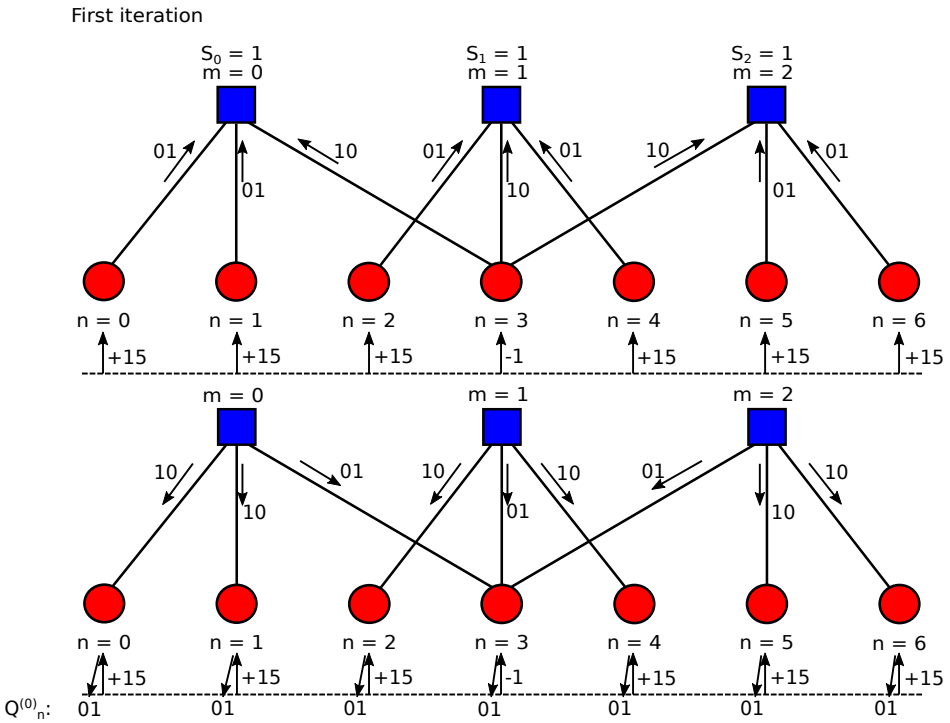

Figure 1.20: MS2-bit example. First iteration.

Like in the previous examples, variable nodes are loaded with the received LLR values. Then, messages are sent from variable nodes to check nodes, computing the syndromes:

$$s_0 = \hat{x}_0^{(0)} \oplus \hat{x}_1^{(0)} \oplus \hat{x}_3^{(0)} = 0 \oplus 0 \oplus 1 = 1$$
$$s_1 = \hat{x}_2^{(0)} \oplus \hat{x}_3^{(0)} \oplus \hat{x}_4^{(0)} = 0 \oplus 1 \oplus 0 = 1$$
$$s_2 = \hat{x}_3^{(0)} \oplus \hat{x}_5^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 \oplus 0 = 1$$

where

$$\hat{x}_n^{(0)} = \begin{cases} 1, & Q_n^{(0)} < 0 \\ 0, & Q_n^{(0)} \geq 0 \end{cases}$$

After that, the value that will satisfy each of the parity-check equations is computed. Considering that the incoming messages from the variable node can have an error (processing only the information from the neighbours), the sign of the message from the check node to the variable node is computed using:

$$sgn(R_{m,n}^{(i)}) = sgn(Q_{n,m}^{(i-1)}) \oplus S_m$$

Therefore, by applying this equation we obtain:

$$\begin{aligned}
sgn(R_{0,0}^{(0)}) &= sgn(Q_{0,0}^{(0)}) \oplus S_0 = 0 \oplus 1 = 1 \\
sgn(R_{0,1}^{(0)}) &= sgn(Q_{1,0}^{(0)}) \oplus S_0 = 0 \oplus 1 = 1 \\
sgn(R_{0,3}^{(0)}) &= sgn(Q_{3,0}^{(0)}) \oplus S_0 = 1 \oplus 1 = 0 \\
sgn(R_{1,2}^{(0)}) &= sgn(Q_{2,1}^{(0)}) \oplus S_1 = 0 \oplus 1 = 1 \\
sgn(R_{1,3}^{(0)}) &= sgn(Q_{3,1}^{(0)}) \oplus S_1 = 1 \oplus 1 = 0 \\
sgn(R_{1,4}^{(0)}) &= sgn(Q_{4,1}^{(0)}) \oplus S_1 = 0 \oplus 1 = 1 \\
sgn(R_{2,3}^{(0)}) &= sgn(Q_{3,2}^{(0)}) \oplus S_2 = 1 \oplus 1 = 0 \\
sgn(R_{2,5}^{(0)}) &= sgn(Q_{5,2}^{(0)}) \oplus S_2 = 0 \oplus 1 = 1 \\
sgn(R_{2,6}^{(0)}) &= sgn(Q_{6,2}^{(0)}) \oplus S_2 = 0 \oplus 1 = 1
\end{aligned}$$

In Fig. 1.20, we can see that the signs of the messages exchanged are the same as the previous results. The magnitude of the check-node outputs can be computed as the minimum of the absolute value of the neighbour messages:

$$
\begin{aligned}
|R_{0,0}^{(0)}| &= min(|Q_{1,0}^{(0)}|, |Q_{3,0}^{(0)}|) = 0 \\
|R_{0,1}^{(0)}| &= min(|Q_{0,0}^{(0)}|, |Q_{3,0}^{(0)}|) = 0 \\
|R_{0,3}^{(0)}| &= min(|Q_{0,0}^{(0)}|, |Q_{1,0}^{(0)}|) = 0 \\
|R_{1,2}^{(0)}| &= min(|Q_{3,1}^{(0)}|, |Q_{4,1}^{(0)}|) = 0 \\
|R_{1,3}^{(0)}| &= min(|Q_{2,1}^{(0)}|, |Q_{4,1}^{(0)}|) = 0 \\
|R_{1,4}^{(0)}| &= min(|Q_{2,1}^{(0)}|, |Q_{3,1}^{(0)}|) = 0 \\
|R_{2,3}^{(0)}| &= min(|Q_{5,2}^{(0)}|, |Q_{6,2}^{(0)}|) = 0 \\
|R_{2,5}^{(0)}| &= min(|Q_{3,2}^{(0)}|, |Q_{6,2}^{(0)}|) = 0 \\
|R_{2,6}^{(0)}| &= min(|Q_{3,2}^{(0)}|, |Q_{5,2}^{(0)}|) = 0
\end{aligned}
$$

Finally, the tentative decoding is computed by adding the LLR value received from the channel to the incoming check-to-variable node messages as follows:

$$
\begin{aligned}
Q_0^{(0)} &= g(f(L_0(y_0)) + (\alpha \cdot f(R_{0,0}^{(0)})) = g(5 + (0.75 \cdot (-1))) = g(+4.25) = 01 \\
Q_1^{(0)} &= g(f(L_1(y_1)) + (\alpha \cdot f(R_{0,1}^{(0)})) = g(5 + (0.75 \cdot (-1))) = g(+4.25) = 01 \\
Q_2^{(0)} &= g(f(L_2(y_2)) + (\alpha \cdot f(R_{1,2}^{(0)})) = g(5 + (0.75 \cdot (-1))) = g(+4.25) = 01 \\
Q_3^{(0)} &= g(f(L_3(y_3)) + (\alpha \cdot f(R_{0,3}^{(0)} + R_{1,3}^{(0)} + R_{2,3}^{(0)}))) = g(-1 + (0.75 \cdot (5+5+5))) = g(+10.25) = 01 \\
Q_4^{(0)} &= g(f(L_4(y_4)) + (\alpha \cdot f(R_{1,4}^{(0)})) = g(5 + (0.75 \cdot (-1))) = g(+4.25) = 01 \\
Q_5^{(0)} &= g(f(L_5(y_5)) + (\alpha \cdot f(R_{2,5}^{(0)})) = g(5 + (0.75 \cdot (-1))) = g(+4.25) = 01 \\
Q_6^{(0)} &= g(f(L_6(y_6)) + (\alpha \cdot f(R_{2,6}^{(0)})) = g(5 + (0.75 \cdot (-1))) = g(+4.25) = 01
\end{aligned}
$$

$$
where \quad f(x) = \begin{cases} W_L, & if \quad x = 00 \\ W_H, & if \quad x = 01 \\ -W_L, & if \quad x = 10 \\ -W_H, & if \quad x = 11 \end{cases}
$$

$$and \quad g(x) = \begin{cases} 00, & if \quad T_L > x \geq 0 \\ 01, & if \quad x \geq T_L \\ 10, & if \quad 0 > x > -T_L \\ 11, & if \quad x \leq -T_L \end{cases}$$

Performing a second iteration:

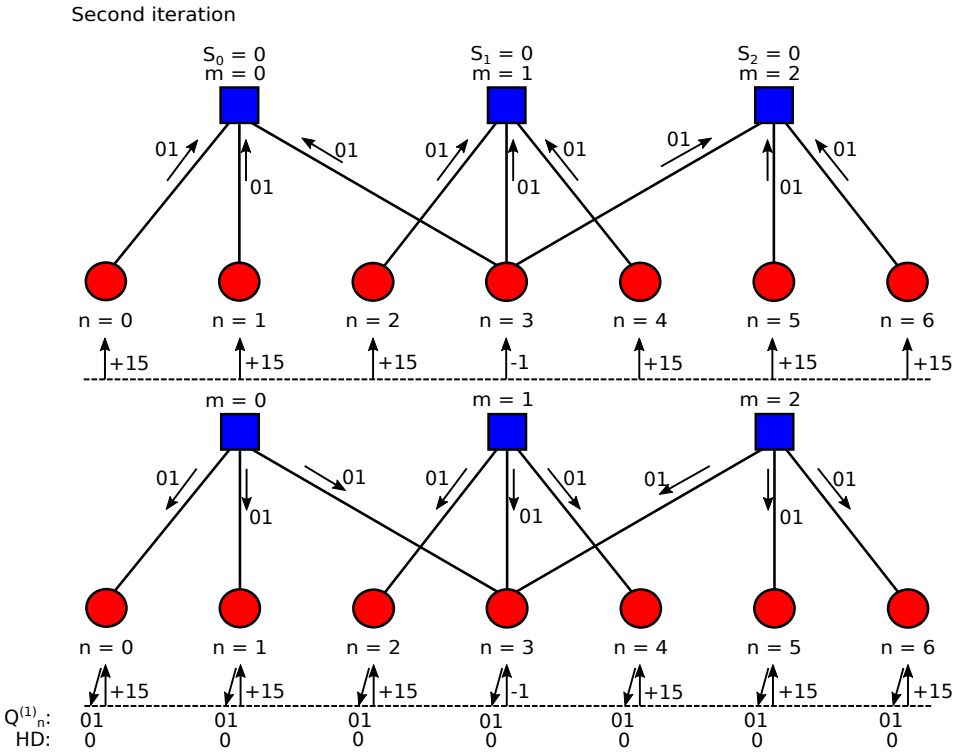

Figure 1.21: MS2-bit example. Second iteration $T_Y = 1.5$.

The messages computed in each variable node are sent from variable nodes to check nodes, computing the syndromes:

$$s_0 = \hat{x}_0^{(1)} \oplus \hat{x}_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 \oplus 0 = 0$$
$$s_1 = \hat{x}_2^{(1)} \oplus \hat{x}_3^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 \oplus 0 = 0$$
$$s_2 = \hat{x}_3^{(1)} \oplus \hat{x}_5^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 \oplus 0 = 0$$

where

$$\hat{x}_n^{(1)} = \begin{cases} 1, & Q_n^{(1)} < 0 \\ 0, & Q_n^{(1)} \geq 0 \end{cases}$$

Once the syndromes are calculated, the value that will satisfy each of the parity-check equations is computed. On the one hand, the sign is processed:

$$sgn(R_{0,0}^{(1)}) = sgn(Q_{0,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0$$
$$sgn(R_{0,1}^{(1)}) = sgn(Q_{1,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0$$
$$sgn(R_{0,3}^{(1)}) = sgn(Q_{3,0}^{(1)}) \oplus s_0 = 0 \oplus 0 = 0$$
$$sgn(R_{1,2}^{(1)}) = sgn(Q_{2,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0$$
$$sgn(R_{1,3}^{(1)}) = sgn(Q_{3,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0$$
$$sgn(R_{1,4}^{(1)}) = sgn(Q_{4,1}^{(1)}) \oplus s_1 = 0 \oplus 0 = 0$$
$$sgn(R_{2,3}^{(1)}) = sgn(Q_{3,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0$$
$$sgn(R_{2,5}^{(1)}) = sgn(Q_{5,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0$$
$$sgn(R_{2,6}^{(1)}) = sgn(Q_{6,2}^{(1)}) \oplus s_2 = 0 \oplus 0 = 0$$

On the other hand, the computation of the magnitude of the messages is done as follows:

$$|R_{0,0}^{(1)}| = min(|Q_{1,0}^{(1)}|, |Q_{3,0}^{(1)}|) = 1$$
$$|R_{0,1}^{(1)}| = min(|Q_{0,0}^{(1)}|, |Q_{3,0}^{(1)}|) = 1$$
$$|R_{0,3}^{(1)}| = min(|Q_{0,0}^{(1)}|, |Q_{1,0}^{(1)}|) = 1$$
$$|R_{1,2}^{(1)}| = min(|Q_{3,1}^{(1)}|, |Q_{4,1}^{(1)}|) = 1$$
$$|R_{1,3}^{(1)}| = min(|Q_{2,1}^{(1)}|, |Q_{4,1}^{(1)}|) = 1$$
$$|R_{1,4}^{(1)}| = min(|Q_{2,1}^{(1)}|, |Q_{3,1}^{(1)}|) = 1$$
$$|R_{2,3}^{(1)}| = min(|Q_{5,2}^{(1)}|, |Q_{6,2}^{(1)}|) = 1$$
$$|R_{2,5}^{(1)}| = min(|Q_{3,2}^{(1)}|, |Q_{6,2}^{(1)}|) = 1$$
$$|R_{2,6}^{(1)}| = min(|Q_{3,2}^{(1)}|, |Q_{5,2}^{(1)}|) = 1$$

In this second iteration the reliability values for computing the tentative decoding are:

$$Q_0^{(1)} = g(f(L_0(y_0)) + (\alpha \cdot f(R_{0,0}^{(1)})) = g(5 + (0.75 \cdot (5))) = g(+8.75) = 01$$
$$Q_1^{(1)} = g(f(L_1(y_1)) + (\alpha \cdot f(R_{0,1}^{(1)})) = g(5 + (0.75 \cdot (5))) = g(+8.75) = 01$$
$$Q_2^{(1)} = g(f(L_2(y_2)) + (\alpha \cdot f(R_{1,2}^{(1)})) = g(5 + (0.75 \cdot (5))) = g(+8.75) = 01$$
$$Q_3^{(1)} = g(f(L_3(y_3)) + (\alpha \cdot f((R_{0,3}^{(1)} + R_{1,3}^{(1)} + R_{2,3}^{(1)}))) = g(-1 + (0.75 \cdot (5+5+5))) = g(+10.25) = 01$$
$$Q_4^{(1)} = g(f(L_4(y_4)) + (\alpha \cdot f(R_{1,4}^{(1)})) = g(5 + (0.75 \cdot (5))) = g(+8.75) = 01$$
$$Q_5^{(1)} = g(f(L_5(y_5)) + (\alpha \cdot f(R_{2,5}^{(1)})) = g(5 + (0.75 \cdot (5))) = g(+8.75) = 01$$
$$Q_6^{(1)} = g(f(L_6(y_6)) + (\alpha \cdot f(R_{2,6}^{(1)})) = g(5 + (0.75 \cdot (5))) = g(+8.75) = 01$$

$$where \quad f(x) = \begin{cases} W_L, & if \quad x = 00 \\ W_H, & if \quad x = 01 \\ -W_L, & if \quad x = 10 \\ -W_H, & if \quad x = 11 \end{cases}$$

$$and \quad g(x) = \begin{cases} 00, & if \quad T_L > x \geq 0 \\ 01, & if \quad x \geq T_L \\ 10, & if \quad 0 > x > -T_L \\ 11, & if \quad x \leq -T_L \end{cases}$$

Finally, it can be checked that all the syndromes are satisfied and, hence, the code word is corrected. The error in variable node $n = 3$ has been corrected, flipping the sign of the node from negative to positive. All the Q_n messages are now positive, and all the hard-decision messages are zero, as shown in Fig. 1.21.

If instead of using $T_Y = 1.5$ we would have used $T_Y = 3/8$, the sign of the variable node three would not have been corrected as shown in Fig. 1.22. The reason is that because this threshold T_Y is not optimal for this code as shown in Fig. 1.22.

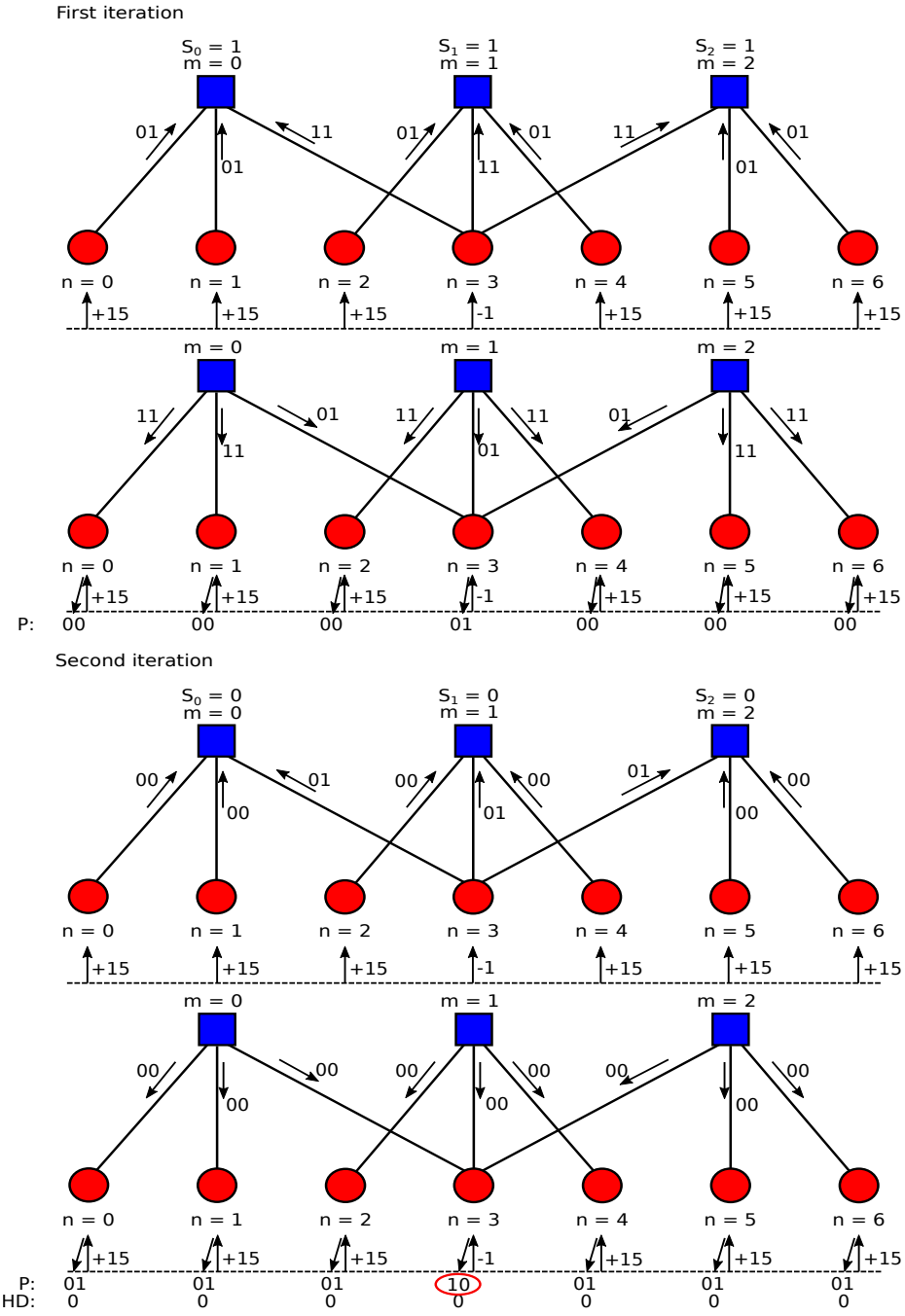

Figure 1.22: MS2-bit example. Second iteration $T_Y = 3/8$.

1.4.4. Performance comparison of soft-decision LDPC decoders

Fig. 1.23 shows the BER performance of the previous algorithms: SPA, MSA and Optimized 2-bit MSA. The code used to this simulation is the quasi-cyclic (2048, 1723) RS-LDPC with $d_v = 6$ and $d_c = 32$ adopted for 10GBASE-T. BPSK modulation and AWGN channel are assumed and the maximum number of iterations is set to 30.

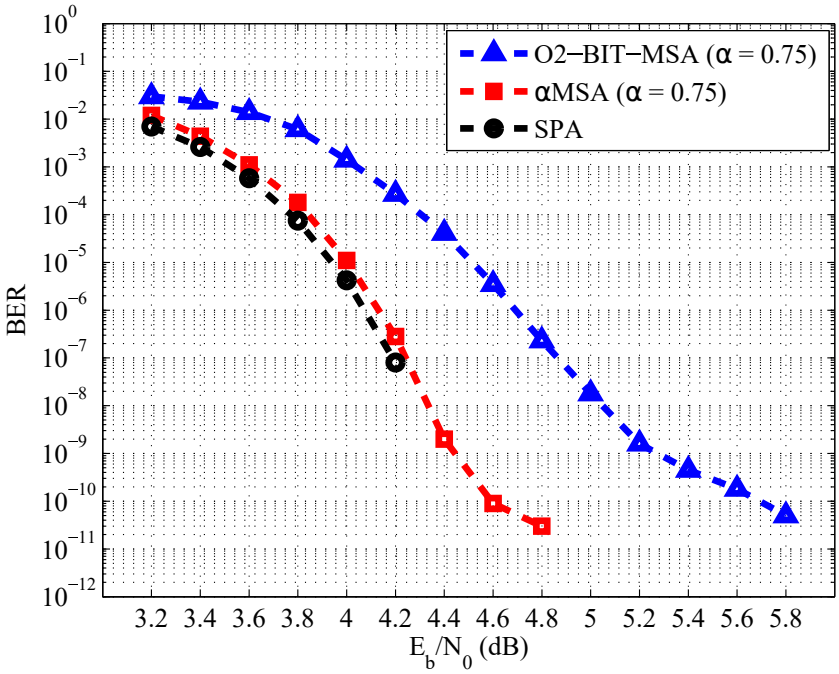

Figure 1.23: SPA, MSA and Optimized 2-bits MSA performance.

In Fig. 1.23 two regions can be observed. The first one is called *waterfall* region, which indicates the decoding quality of the algorithm (algorithm performance). The second one is in the lower part of the curve where the slope decreases, called *error floor*, which indicates that the algorithm has reached a point at which it is not able to increase the rate of corrected errors even with larger E_b/N_0. This error floor can be caused by the structure of the code or by the decoding algorithms.

On the one hand, the best result obtained in the waterfall region has been with SPA, at the expense of the highest complexity for the CNU, as explained before. In Fig. 1.23, SPA outperforms MSA by 0.1 dB and Optimized 2-bit MSA by 0.7

dB at BER 10^{-7}. Also Optimized 2-bit MSA scheme has a performance loss of 0.5 dB from MSA at BER 10^{-7} and note that its slope is worse than the others.

On the other hand, as can be observed in Fig. 1.23, both MSA and Optimized 2-bit MSA have error floor at BER 10^{-10} and at BER 10^{-9}, respectively.

1.5. Hard-decision decoding algorithms

Reliability-based iterative majority-logic decoding (RBI-MLGD) algorithms [29], [30] for LDPC codes represent a very interesting type of alternatives to decoders derived from the SPA [15] due to serveral reasons. The first one is that majority-logic decoders do not require soft information based on channel information at each iteration. Unlike SPA or MSA [16], [17], majority-logic algorithms do not compute soft-decision information at the check-node, they only perform hard-decision operations which reduce significantly the overall complexity. The second reason is that the message exchanged between VN and CN is of just one bit: the bit estimation done in the different edges. This advantage has a great impact when the decoder is implemented in hardware, as more than 80% of the delay and hence 80% of the speed is limited by routing [31], and routing is proportional to the number of bits of the exchanged messages.

Despite the previously mentioned advantages, majority-logic decoders have also some serious limitations: i) the BER performance and; ii) the degree distribution of the code to be decoded. For codes with a construction method based on Euclidean Geometry (EG codes) [11], the performance is very close to SPA and MSA [29], as EG ones are majority-logic decodable codes. However, when codes with different kinds of construction methods [8] - [10] are decoded with RBI-MLGD algorithms (even when the degree distribution is similar to an EG code) a notable performance loss in the BER curve is introduced. This performance loss can be greater than 1.5 dB compared to MSA. On the other hand, the degree distribution is crucial if RBI-MLGD algorithms are to be used. If the variable-node degree (d_v) is not large enough (at least $d_v = 10$), the RBI-MLGD is not able to correct most of the errors, introducing an early degradation at a BER higher than 10^{-5} [30], which makes it impractical for most modern communication or storage systems. The two previous problems prevent majority-logic algorithms from being applied in many systems, due to the fact that the hardware implementations of decoders usually prefer low d_v.

1.5.1. Reliability-Based Iterative Majority-Logic algorithm

RBI-MLGD reaches an efficient trade-off between decoding performance and computational complexity. This algorithm performs well for structured majority-logic decodable LDPC codes with low decoding complexity.

In Algorithm 4 the RBI-MLGD from [29] is described. This algorithm is divided into three main parts: i) initialization; ii) CN update; and iii) VN update.

Algorithm 4 RBI-MLGD.

Input : $Q_n^{(0)} = \varrho(y_n)$, with $n \in \{0, ..., N-1\}$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update
1 $\hat{x}_n^{(i)} = \begin{cases} 1, & Q_n^{(i)} \geq 0 \\ 0, & otherwise, \end{cases} \quad n \in \{0, ..., N-1\}$
2 $s_m^{(i)} = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_n^{(i)} h_{mn} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$
3 $\sigma_{mn}^{(i)} = \sum_{n' \in N_m \setminus n} \oplus \hat{x}_{n'}^{(i)} = s_m^{(i)} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$
Variable-node update
4 $R_n^{(i)} = \sum_{m \in M_n} (2\sigma_{mn}^{(i)} - 1), \quad n \in \{0, ..., N-1\}$
5 $Q_n^{(i+1)} = Q_n^{(i)} + R_n^{(i)}, \quad n \in \{0, ..., N-1\}$
Tentative decoding
if $(s^{(i)} = 0)$ **then** {SKIP}
end
Output : $\hat{x}$

For the initialization, the RBI-MLGD quantizes the received vectors by using the ϱ function. The ϱ function saturates the maximum amplitudes of the received sequence and normalizes them by performing a division by Δ (for more details we refer to [30]). The output of the ϱ function is taken as reliability information to initialize the variable-nodes at iteration zero, $Q_n^{(0)}$.

During the check-node update, the syndromes of the M parity check equations are computed. If the parity check equations are satisfied ($s_m = 0$ in Step 2) the new estimated bits are equal to the old ones ($\sigma_{mn} = \hat{x}_n$, Step 3), otherwise, the estimated bits are flipped ($\sigma_{mn} = s_m \oplus \hat{x}_n = 1 \oplus \hat{x}_n$, Step 3). These new estimated bits (σ_{mn}) can be interpreted as candidates in the voting process performed by the majority-logic at the variable-nodes.

At the variable-node, σ_{mn} is transformed from bits to votes. According to Step 4 in Algorithm 4, the amplitude of the vote will be -1 if $\sigma_{mn} = 0$, and it will be +1 if

$\sigma_{mn} = 1$. Depending on the majority vote, R_n will have the same sign as $\hat{x}_n$ or not. The recount of the votes (R_n) is added to Q_n, which stores the channel information ($\varrho(y_n)$) and the votes in the previous iterations (Step 5). The purpose of the votes is trying to correct or validate the channel information ($\varrho(y_m)$) depending on the number of parity check-nodes that are unsatisfied. In this case the VN passes full messages (the incoming check-to-variable messages for each VN added to the value received from the channel for that VN) (Q_n) without excluding the value of the CN to which the message is sent.

The algorithm stops when a maximum number of iterations is reached or all the syndromes are equal to zero (tentative decoding step).

The first important idea derived from RBI-MLGD is that the message-passing of this algorithm is binary. As it can be seen, the information from check-node to variable-node are the candidate bits σ_{mn}, and the variable-node to check-node messages are the hard-decision bits $\hat{x}_n$. This is important to keep complexity low and reduce routing (increase speed) in hardware implementations [31].

The second important idea that can be extracted from Algorithm 4 is that in the iterative process two different domains are mixed: the soft-information from the channel and the votes. The votes, which are mapped in the case of RBI-MLGD algorithm to +1 and -1, are an abstract representation of the reliability of a decision based on the check-node information. In some cases, this uncorrelation between the vote and the channel information introduces performance loss.

1.5.1.1. Example 5. Reliability-Based Iterative Majority-Logic algorithm

Let us assume that a bit sequence of seven zeros is transmitted on a channel with errors. The magnitude of the LLR is correlated with the magnitude of the received level. We take the positive sign as a logic 0 and the negative sign as a logic 1.

In this example LLR values are computed as $L_0(y_0) = -15 = n_0$, $L_1(y_1) = -15 = n_1$, $L_2(y_2) = -15 = n_2$, $L_3(y_3) = +1 = n_3$, $L_4(y_4) = -15 = n_4$, $L_5(y_5) = -15 = n_5$ and $L_6(y_6) = -15 = n_6$.

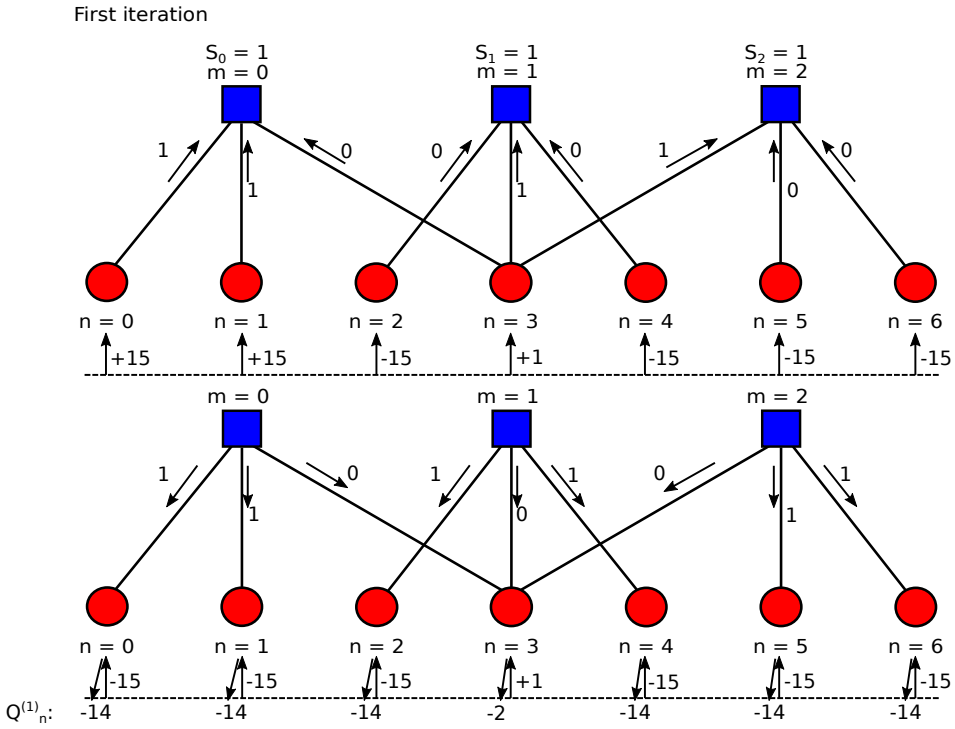

Figure 1.24: RBI-MLGD example. First iteration.

First, the variable nodes load these received LLR values. After this, messages are sent from VNs to CNs, and the syndromes are computed. Applying the syndrome equation the following results are obtained:

$$s_0 = \hat{x}_0^{(0)} \oplus \hat{x}_1^{(0)} \oplus \hat{x}_3^{(0)} = 0 \oplus 0 \oplus 1 = 1$$
$$s_1 = \hat{x}_2^{(0)} \oplus \hat{x}_3^{(0)} \oplus \hat{x}_4^{(0)} = 0 \oplus 1 \oplus 0 = 1$$
$$s_2 = \hat{x}_3^{(0)} \oplus \hat{x}_5^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 \oplus 0 = 1$$

where

$$\hat{x}_n^{(0)} = \begin{cases} 1, & Q_n^{(0)} \geq 0 \\ 0, & otherwise \end{cases}$$

Once the syndromes are computed, the value that will satisfy each of the parity-check equations is computed. Considering that the incoming messages from the VN can have an error (processing only the information from the neighbours), the following equation is applied to determine the reliability of the message from the check node to the variable node:

$$\sigma_{m,n}^{(i)} = s_m^{(i)} \oplus \hat{x}_n^{(i)}, m \in 0, ..., M-1$$

So applying this equation in this example we obtain:

$$\begin{aligned}
\sigma_{0,0}^{(0)} &= s_0^{(0)} \oplus \hat{x}_0^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{0,1}^{(0)} &= s_0^{(0)} \oplus \hat{x}_1^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{0,3}^{(0)} &= s_0^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{1,2}^{(0)} &= s_1^{(0)} \oplus \hat{x}_2^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{1,3}^{(0)} &= s_1^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{1,4}^{(0)} &= s_1^{(0)} \oplus \hat{x}_4^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{2,3}^{(0)} &= s_2^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{2,5}^{(0)} &= s_2^{(0)} \oplus \hat{x}_5^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{2,6}^{(0)} &= s_2^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 = 1
\end{aligned}$$

The previous results can be compared to the reliability of the message exchange in Fig. 1.24.

Finally, the tentative decoding is computed by adding the value received from the channel to the incoming check-to-variable node messages as follows:

$$Q_0^{(1)} = Q_0^{(0)} + (2 \cdot \sigma_{0,0}^{(0)} - 1) = (-15) + 1 = -14$$
$$Q_1^{(1)} = Q_1^{(0)} + (2 \cdot \sigma_{0,1}^{(0)} - 1) = (-15) + 1 = -14$$
$$Q_2^{(1)} = Q_2^{(0)} + (2 \cdot \sigma_{1,2}^{(0)} - 1) = (-15) + 1 = -14$$
$$Q_3^{(1)} = Q_3^{(0)} + (2 \cdot \sigma_{0,3}^{(0)} - 1) + (2 \cdot \sigma_{1,3}^{(0)} - 1) + (2 \cdot \sigma_{2,3}^{(0)} - 1) = 1 + (-1) + (-1) + (-1) = -2$$
$$Q_4^{(1)} = Q_4^{(0)} + (2 \cdot \sigma_{1,4}^{(0)} - 1) = (-15) + 1 = -14$$
$$Q_5^{(1)} = Q_5^{(0)} + (2 \cdot \sigma_{2,5}^{(0)} - 1) = (-15) + 1 = -14$$
$$Q_6^{(1)} = Q_6^{(0)} + (2 \cdot \sigma_{2,6}^{(0)} - 1) = (-15) + 1 = -14$$

Performing a second iteration:

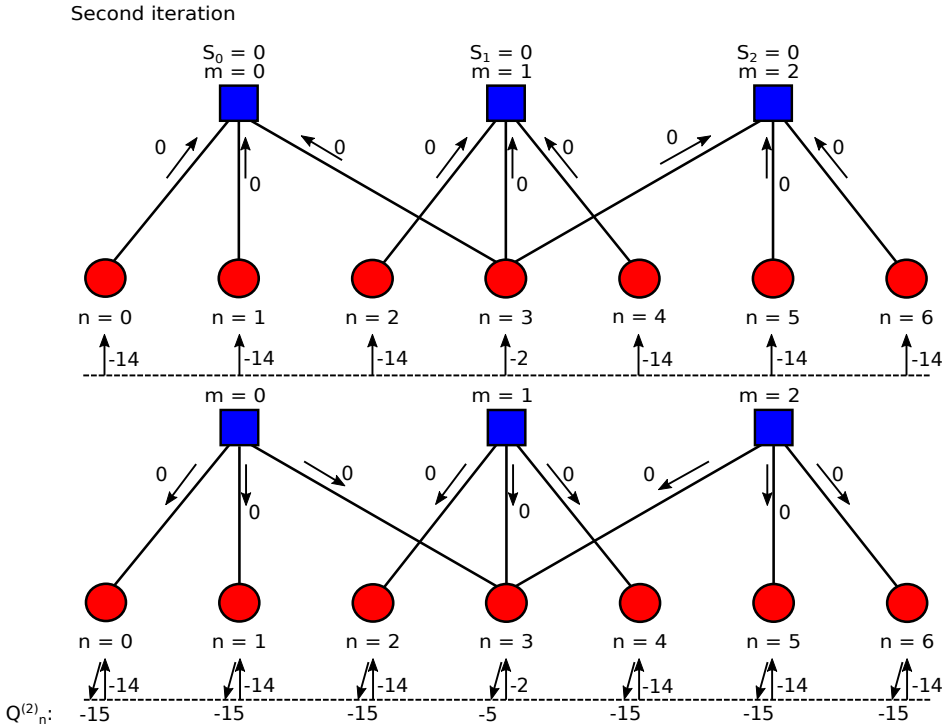

Figure 1.25: RBI-MLGD example. Second iteration.

The messages computed in each VN are sent from VNs to CNs, the syndromes are computed.

$$
\begin{aligned}
s_0 &= \hat{x}_0^{(1)} \oplus \hat{x}_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 \oplus 0 = 0 \\
s_1 &= \hat{x}_2^{(1)} \oplus \hat{x}_3^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 \oplus 0 = 0 \\
s_2 &= \hat{x}_3^{(1)} \oplus \hat{x}_5^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 \oplus 0 = 0
\end{aligned}
$$

where

$$
\hat{x}_n^{(1)} = \begin{cases} 1, & Q_n^{(1)} \geq 0 \\ 0, & otherwise \end{cases}
$$

Once the syndromes are computed, the value that will satisfy each of the parity-check equations is computed. The computation of the magnitude of the messages is done as follows:

$$
\begin{aligned}
\sigma_{0,0}^{(1)} &= s_0^{(1)} \oplus \hat{x}_0^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{0,1}^{(1)} &= s_0^{(1)} \oplus \hat{x}_1^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{0,3}^{(1)} &= s_0^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{1,2}^{(1)} &= s_1^{(1)} \oplus \hat{x}_2^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{1,3}^{(1)} &= s_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{1,4}^{(1)} &= s_1^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{2,3}^{(1)} &= s_2^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{2,5}^{(1)} &= s_2^{(1)} \oplus \hat{x}_5^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{2,6}^{(1)} &= s_2^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 = 0
\end{aligned}
$$

In this second iteration the reliability values for computing the tentative decoding are:

$$Q_0^{(2)} = Q_0^{(1)} + (2 \cdot \sigma_{0,0}^{(1)} - 1) = (-14) + (-1) = -15$$
$$Q_1^{(2)} = Q_1^{(1)} + (2 \cdot \sigma_{0,1}^{(1)} - 1) = (-14) + (-1) = -15$$
$$Q_2^{(2)} = Q_2^{(1)} + (2 \cdot \sigma_{1,2}^{(1)} - 1) = (-14) + (-1) = -15$$
$$Q_3^{(2)} = Q_3^{(1)} + (2 \cdot \sigma_{0,3}^{(1)} - 1) + (2 \cdot \sigma_{1,3}^{(1)} - 1) + (2 \cdot \sigma_{2,3}^{(1)} - 1) = (-2) + (-1) + (-1) + (-1) = -5$$
$$Q_4^{(2)} = Q_4^{(1)} + (2 \cdot \sigma_{1,4}^{(1)} - 1) = (-14) + (-1) = -15$$
$$Q_5^{(2)} = Q_5^{(1)} + (2 \cdot \sigma_{2,5}^{(1)} - 1) = (-14) + (-1) = -15$$
$$Q_6^{(2)} = Q_6^{(1)} + (2 \cdot \sigma_{2,6}^{(1)} - 1) = (-14) + (-1) = -15$$

Finally, it can be checked that all the syndromes are satisfied and, hence, the code word is corrected. The error in variable node $n = 3$ has been corrected, flipping the sign of the node from positive to negative. All the Q_n messages are now negative, and all the hard-decision messages are zero, as shown in Fig. 1.25.

1.5.2. Modified Reliability-Based Iterative Majority-Logic algorithm

Due to the performance loss introduced by the uncorrelation between the vote and the channel information, authors from [30] proposed a modified RBI-MLGD (MRBI-MLGD) algorithm trying to solve this problem.

The differences between MRBI-MLGD and RBI-MLGD are located at the variable-node update equations. In order to match the concept of the votes with the information from the channel, a scaling factor α is introduced in Step 4 [30]. This scaling factor compensates the differences between the channel information and the constant amplitude of the votes, trying to balance the importance of the hard-decision information (σ_{mn} from the check-node) and the soft-decision (received y sequence), Eq. 1.4.

$$R_n^{(i)} = [\alpha R_n^{(i)}] = [\alpha \sum_{m \in M_n} (2\sigma_{mn}^{(i)} - 1)] \tag{1.4}$$

Moreover, Step 5 is also modified as shown in Eq. 1.5.

$$Q_n^{(i+1)} = Q_n^{(0)} + R_n^{(i)}, \quad n \in 0, ..., N-1, \tag{1.5}$$

where Q_n contains the reliability value of each bit in the tentative decoding codeword.

This modification introduces an improvement of the coding gain as the ratio "votes-soft input" is closer to the one found in the min-sum algorithm, this is because it mimics the update performed in turbo decoders in which the output of a variable-node is $\varrho(y_m)$ plus the information from the check-node [30].

Algorithm 5 MRBI-MLGD.

Input : $Q_n^{(0)} = \varrho(y_n)$, with $n \in \{0, ..., N-1\}$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update

1 $\hat{x}_n^{(i)} = \begin{cases} 1, & Q_n^{(i)} \geq 0 \\ 0, & otherwise, \end{cases} \quad n \in \{0, ..., N-1\}$

2 $s_m^{(i)} = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_n^{(i)} h_{mn} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$

3 $\sigma_{mn}^{(i)} = \sum_{n' \in N_m \backslash n} \oplus \hat{x}_{n'}^{(i)} = s_m^{(i)} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$

Variable-node update

4 $R_n^{(i)} = [\alpha R_n^{(i)}] = [\alpha \sum_{m \in M_n} (2\sigma_{mn}^{(i)} - 1)], \quad n \in \{0, ..., N-1\}$

5 $Q_n^{(i+1)} = Q_n^{(0)} + R_n^{(i)}, \quad n \in \{0, ..., N-1\}$

Tentative decoding
if $(s^{(i)} = 0)$ **then** {SKIP}
end
Output : $\hat{x}$

Unfortunately, MRBI-MLGD does not solve the problem of working with codes with a degree distribution of low d_v. As it is shown in [30], MRBI-MLGD algorithm introduces an important degradation in performance, not being able to reach a bit error rate below 10^{-3} when the degree of variable-node is low.

1.5.2.1. Example 6. Modified Reliability-Based Iterative Majority-Logic algorithm

In this example we assume the same condition as in Example 5, with the same input values (LLR) and in this case we use a scale factor $\alpha = 0.5$.

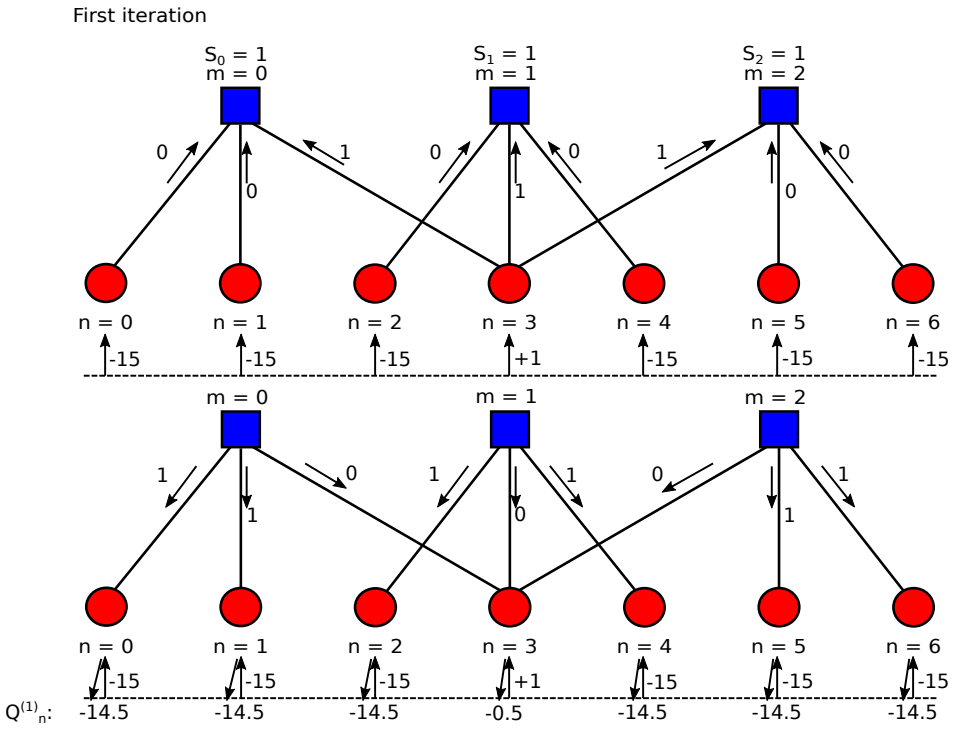

Figure 1.26: MRBI-MLGD example. First iteration.

Like in the previous example, first the LLR values are loaded in the VNs. Once this is done, messages are sent from VNs to CNs and the syndromes are computed:

$$s_0 = \hat{x}_0^{(0)} \oplus \hat{x}_1^{(0)} \oplus \hat{x}_3^{(0)} = 0 \oplus 0 \oplus 1 = 1$$
$$s_1 = \hat{x}_2^{(0)} \oplus \hat{x}_3^{(0)} \oplus \hat{x}_4^{(0)} = 0 \oplus 1 \oplus 0 = 1$$
$$s_2 = \hat{x}_3^{(0)} \oplus \hat{x}_5^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 \oplus 0 = 1$$

where

$$\hat{x}_n^{(0)} = \begin{cases} 1, & Q_n^{(0)} \geq 0 \\ 0, & otherwise \end{cases}$$

With the syndromes computed, the value that will satisfy each of the parity-check equations is calculated. Considering that the incoming messages from the VN can

have an error (processing only the information from the neighbours), the following equation is applied to determine the magnitude of the messages from the CN to VN:

$$\sigma_{m,n}^{(i)} = s_m^{(i)} \oplus \hat{x}_n^{(i)}, m \in 0, ..., M-1$$

So the magnitude is computed as:

$$\begin{aligned}
\sigma_{0,0}^{(0)} &= s_0^{(0)} \oplus \hat{x}_0^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{0,1}^{(0)} &= s_0^{(0)} \oplus \hat{x}_1^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{0,3}^{(0)} &= s_0^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{1,2}^{(0)} &= s_1^{(0)} \oplus \hat{x}_2^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{1,3}^{(0)} &= s_1^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{1,4}^{(0)} &= s_1^{(0)} \oplus \hat{x}_4^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{2,3}^{(0)} &= s_2^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{2,5}^{(0)} &= s_2^{(0)} \oplus \hat{x}_5^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{2,6}^{(0)} &= s_2^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 = 1
\end{aligned}$$

We can check the previous results with the magnitude of the message exchange in Fig. 1.26.

Finally, the tentative decoding is computed by adding the value received from the channel to the incoming check-to-variable node messages as follows:

$$Q_0^{(1)} = Q_0^{(0)} + (\alpha \cdot (2 \cdot \sigma_{0,0}^{(0)} - 1)) = (-15) + (0.5 \cdot 1) = -14.5$$
$$Q_1^{(1)} = Q_1^{(0)} + (\alpha \cdot (2 \cdot \sigma_{0,1}^{(0)} - 1)) = (-15) + (0.5 \cdot 1) = -14.5$$
$$Q_2^{(1)} = Q_2^{(0)} + (\alpha \cdot (2 \cdot \sigma_{1,2}^{(0)} - 1)) = (-15) + (0.5 \cdot 1) = -14.5$$
$$Q_3^{(1)} = Q_3^{(0)} + (\alpha \cdot ((2 \cdot \sigma_{0,3}^{(0)} - 1) + (2 \cdot \sigma_{1,3}^{(0)} - 1) + (2 \cdot \sigma_{2,3}^{(0)} - 1))) = 1 + (0.5 \cdot ((-1) + (-1) + (-1))) = -0.5$$
$$Q_4^{(1)} = Q_4^{(0)} + (\alpha \cdot (2 \cdot \sigma_{1,4}^{(0)} - 1)) = (-15) + (0.5 \cdot 1) = -14.5$$
$$Q_5^{(1)} = Q_5^{(0)} + (\alpha \cdot (2 \cdot \sigma_{2,5}^{(0)} - 1)) = (-15) + (0.5 \cdot 1) = -14.5$$
$$Q_6^{(1)} = Q_6^{(0)} + (\alpha \cdot (2 \cdot \sigma_{2,6}^{(0)} - 1)) = (-15) + (0.5 \cdot 1) = -14.5$$

Performing a second iteration:

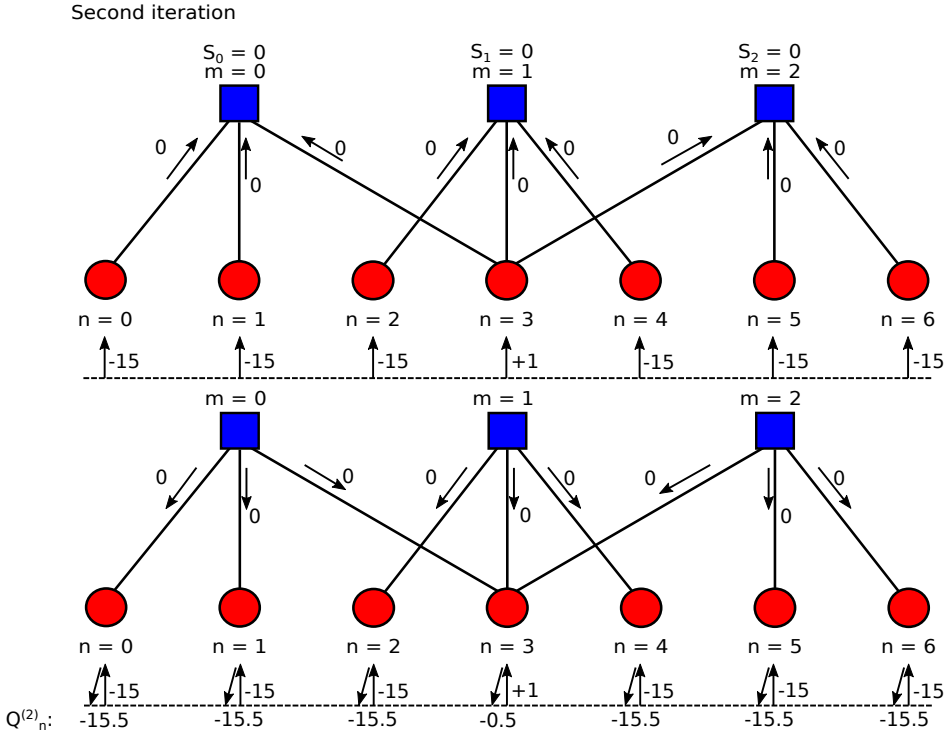

Figure 1.27: MRBI-MLGD example. Second iteration.

The messages computed in each VN are sent from VNs to CNs, computing the syndromes:

$$
\begin{aligned}
s_0 &= \hat{x}_0^{(1)} \oplus \hat{x}_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 \oplus 0 = 0 \\
s_1 &= \hat{x}_2^{(1)} \oplus \hat{x}_3^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 \oplus 0 = 0 \\
s_2 &= \hat{x}_3^{(1)} \oplus \hat{x}_5^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 \oplus 0 = 0
\end{aligned}
$$

where

$$
\hat{x}_n^{(1)} = \begin{cases} 1, & Q_n^{(1)} \geq 0 \\ 0, & otherwise \end{cases}
$$

Once the syndromes are computed, the value that will satisfy each of the parity-check equations is processed. The computation of the magnitude of the messages is done as follows:

$$
\begin{aligned}
\sigma_{0,0}^{(1)} &= s_0^{(1)} \oplus \hat{x}_0^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{0,1}^{(1)} &= s_0^{(1)} \oplus \hat{x}_1^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{0,3}^{(1)} &= s_0^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{1,2}^{(1)} &= s_1^{(1)} \oplus \hat{x}_2^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{1,3}^{(1)} &= s_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{1,4}^{(1)} &= s_1^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{2,3}^{(1)} &= s_2^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{2,5}^{(1)} &= s_2^{(1)} \oplus \hat{x}_5^{(1)} = 0 \oplus 0 = 0 \\
\sigma_{2,6}^{(1)} &= s_2^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 = 0
\end{aligned}
$$

In this second iteration the reliability values for computing the tentative decoding are:

$$Q_0^{(2)} = Q_0^{(0)} + (\alpha \cdot (2 \cdot \sigma_{0,0}^{(1)} - 1)) = (-15) + (0.5 \cdot (-1)) = -15.5$$
$$Q_1^{(2)} = Q_1^{(0)} + (\alpha \cdot (2 \cdot \sigma_{0,1}^{(1)} - 1)) = (-15) + (0.5 \cdot (-1)) = -15.5$$
$$Q_2^{(2)} = Q_2^{(0)} + (\alpha \cdot (2 \cdot \sigma_{1,2}^{(1)} - 1)) = (-15) + (0.5 \cdot (-1)) = -15.5$$
$$Q_3^{(2)} = Q_3^{(0)} + (\alpha \cdot ((2 \cdot \sigma_{0,3}^{(1)} - 1) + (2 \cdot \sigma_{1,3}^{(1)} - 1) + (2 \cdot \sigma_{2,3}^{(1)} - 1))) = 1 + (0.5 \cdot ((-1) + (-1) + (-1))) = -0.5$$
$$Q_4^{(2)} = Q_4^{(0)} + (\alpha \cdot (2 \cdot \sigma_{1,4}^{(1)} - 1)) = (-15) + (0.5 \cdot (-1)) = -15.5$$
$$Q_5^{(2)} = Q_5^{(0)} + (\alpha \cdot (2 \cdot \sigma_{2,5}^{(1)} - 1)) = (-15) + (0.5 \cdot (-1)) = -15.5$$
$$Q_6^{(2)} = Q_6^{(0)} + (\alpha \cdot (2 \cdot \sigma_{2,6}^{(1)} - 1)) = (-15) + (0.5 \cdot (-1)) = -15.5$$

Finally, it can be checked that all the syndromes are satisfied and, hence, the code word is corrected. The error in VN $n = 3$ has been corrected, flipping the sign of the node from positive to negative. All the Q_n messages are now negative, and all the hard-decision messages are zero, as shown in Fig. 1.27.

1.5.3. Reliability-Based Iterative Min-Sum algorithm

Reliability-based iterative min-sum decoding algorithm (RBI-MSD) is the alternative proposed in [30] to avoid an early performance degradation. This algorithm introduces soft-information and transforms the one-bit message-passing decoder into an integer message-passing decoder increasing its complexity. This algorithm also uses a scaling factor α to modify the total extrinsic information to compensate the performance degradation as shown in Eq. 1.6:

$$R_n^{\sim(i)} = [\alpha \sum_{m \in M_n} U_{mn}^{(i)}], \tag{1.6}$$

where $U_{mn}^{(i)}$ is the extrinsic message passed from the m-th check node to the n-th variable node at the i-th iteration. The integer message $U_{mn}^{(i)}$ plays a role equivalent to $2\sigma_{mn}^{(i)} - 1$ in the RBI-MLGD and MRBI-MLGD algorithms:

$$U_{mn}^{(i)} = (2\sigma_{mn}^{(i)} - 1) \min_{n' \in N_m \backslash n} |Q_{n'}^{(i)}|, \tag{1.7}$$

being $Q_{n'}^{(i)}$ the reliability message of $v_{n'}$, which is always clipped into the interval [-(2^{b-1} - 1),(2^{b-1} - 1)].

Algorithm 6 RBI-MSD.

Input : $Q_n^{(0)} = \varrho(y_n)$, with $n \in \{0, ..., N-1\}$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update
1 $\hat{x}_n^{(i)} = \begin{cases} 1, & Q_n^{(i)} \geq 0 \\ 0, & otherwise, \end{cases} \quad n \in \{0, ..., N-1\}$
2 $s_m^{(i)} = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_n^{(i)} h_{mn} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$
3 $\sigma_{mn}^{(i)} = \sum_{n' \in N_m \setminus n} \oplus \hat{x}_{n'}^{(i)} = s_m^{(i)} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$
4 $U_{mn}^{(i)} = (2\sigma_{mn}^{(i)} - 1) \min_{n' \in N_m \setminus n} |Q_{n'}^{(i)}|$
Variable-node update
5 $R_n^{\sim(i)} = [\alpha \sum_{m \in M_n} U_{mn}^{(i)}], \quad n \in \{0, ..., N-1\}$
6 $Q_n^{(i+1)} = Q_n^{(0)} + R_n^{\sim(i)}, \quad n \in \{0, ..., N-1\}$
Tentative decoding
if $(s^{(i)} = 0)$ **then** {SKIP}
end
Output : $\hat{x}$

The RBI-MSD algorithm differs from Min-Sum decoding algorithm in that RBI-MSD passes full messages (the incoming check-to-variable messages for each VN added to the value received from the channel for that VN) (Q_n) without excluding the value of the CN to which the message is directed.

1.5.3.1. Example 7. Reliability-Based Iterative Min-Sum algorithm

In this example we assume the same conditions and input values from the previous examples. Also we use a scale factor $\alpha = 0.5$.

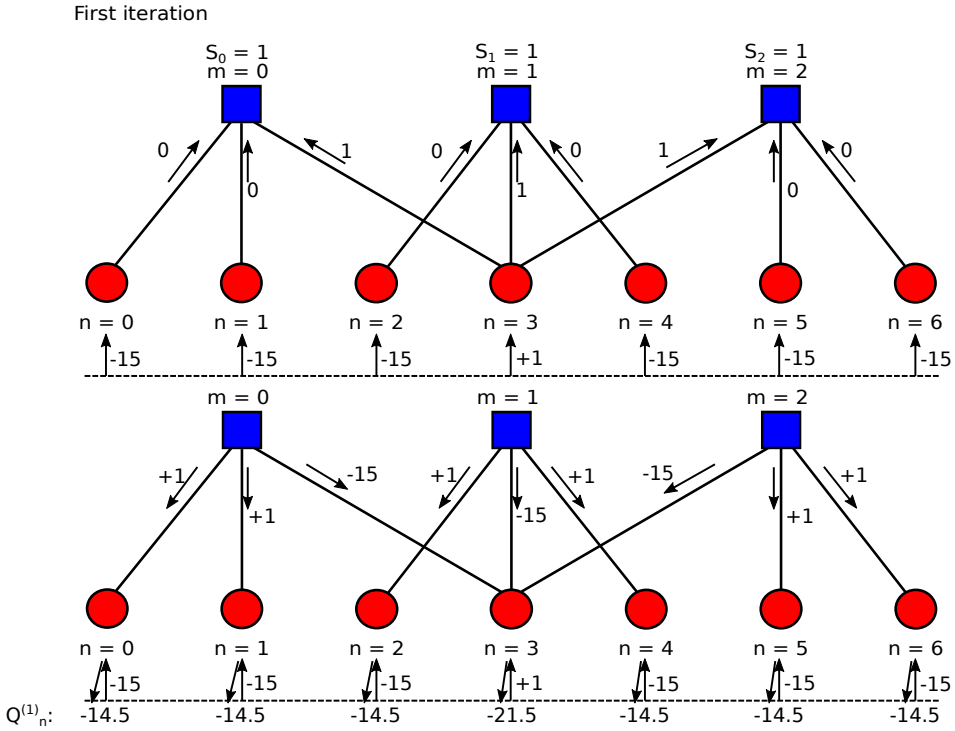

Figure 1.28: RBI-MSD example. First iteration.

Like in the previous examples, VNs are loaded with received LLR values. Then messages are sent from VNs to CNs, computing the syndromes:

$$s_0 = \hat{x}_0^{(0)} \oplus \hat{x}_1^{(0)} \oplus \hat{x}_3^{(0)} = 0 \oplus 0 \oplus 1 = 1$$
$$s_1 = \hat{x}_2^{(0)} \oplus \hat{x}_3^{(0)} \oplus \hat{x}_4^{(0)} = 0 \oplus 1 \oplus 0 = 1$$
$$s_2 = \hat{x}_3^{(0)} \oplus \hat{x}_5^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 \oplus 0 = 1$$

where

$$\hat{x}_n^{(0)} = \begin{cases} 1, & Q_n^{(0)} \geq 0 \\ 0, & otherwise \end{cases}$$

After that, the value that will satisfy each of the parity-check equations is computed. Considering that the incoming messages from the VN can have an error

(processing only the information from the neighbours), the sign of the messages from the CN to VN are computed using:

$$\sigma_{m,n}^{(i)} = s_m^{(i)} \oplus \hat{x}_n^{(i)}, m \in 0, ..., M-1$$

Therefore, by applying this equation we obtain:

$$\begin{aligned}
\sigma_{0,0}^{(0)} &= s_0^{(0)} \oplus \hat{x}_0^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{0,1}^{(0)} &= s_0^{(0)} \oplus \hat{x}_1^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{0,3}^{(0)} &= s_0^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{1,2}^{(0)} &= s_1^{(0)} \oplus \hat{x}_2^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{1,3}^{(0)} &= s_1^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{1,4}^{(0)} &= s_1^{(0)} \oplus \hat{x}_4^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{2,3}^{(0)} &= s_2^{(0)} \oplus \hat{x}_3^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{2,5}^{(0)} &= s_2^{(0)} \oplus \hat{x}_5^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{2,6}^{(0)} &= s_2^{(0)} \oplus \hat{x}_6^{(0)} = 1 \oplus 0 = 1
\end{aligned}$$

In Fig. 1.28, we can see that the signs of the messages exchanged are the same as the previous results. The magnitude of the CN outputs can be computed as the minimum of the absolute value of the neighbours messages:

$$
\begin{aligned}
U_{0,0}^{(0)} &= min(|Q_{1,0}^{(0)}|, |Q_{3,0}^{(0)}| = 1 \\
U_{0,1}^{(0)} &= min(|Q_{0,0}^{(0)}|, |Q_{3,0}^{(0)}| = 1 \\
U_{0,3}^{(0)} &= min(|Q_{0,0}^{(0)}|, |Q_{1,0}^{(0)}| = 15 \\
U_{1,2}^{(0)} &= min(|Q_{3,1}^{(0)}|, |Q_{4,1}^{(0)}| = 1 \\
U_{1,3}^{(0)} &= min(|Q_{2,1}^{(0)}|, |Q_{4,1}^{(0)}| = 15 \\
U_{1,4}^{(0)} &= min(|Q_{2,1}^{(0)}|, |Q_{3,1}^{(0)}| = 1 \\
U_{2,3}^{(0)} &= min(|Q_{5,2}^{(0)}|, |Q_{6,3}^{(0)}| = 15 \\
U_{2,5}^{(0)} &= min(|Q_{3,2}^{(0)}|, |Q_{6,2}^{(0)}| = 1 \\
U_{2,6}^{(0)} &= min(|Q_{3,2}^{(0)}|, |Q_{5,2}^{(0)}| = 1
\end{aligned}
$$

Finally, the tentative decoding is computed by adding the value received from the channel to the incoming check-to-variable node messages as follows:

$$
\begin{aligned}
Q_0^{(1)} &= Q_0^{(0)} + (\alpha \cdot U_{0,0}^{(0)}) = (-15) + (0.5 \cdot 1) = -14.5 \\
Q_1^{(1)} &= Q_1^{(0)} + (\alpha \cdot U_{0,1}^{(0)}) = (-15) + (0.5 \cdot 1) = -14.5 \\
Q_2^{(1)} &= Q_2^{(0)} + (\alpha \cdot U_{1,2}^{(0)}) = (-15) + (0.5 \cdot 1) = -14.5 \\
Q_3^{(1)} = Q_3^{(0)} + (\alpha \cdot (U_{0,3}^{(0)} + U_{1,3}^{(0)} + U_{2,3}^{(0)})) &= 1 + (0.5 \cdot ((-15) + (-15) + (-15))) = -21.5 \\
Q_4^{(1)} &= Q_4^{(0)} + (\alpha \cdot U_{1,4}^{(0)}) = (-15) + (0.5 \cdot 1) = -14.5 \\
Q_5^{(1)} &= Q_5^{(0)} + (\alpha \cdot U_{2,5}^{(0)}) = (-15) + (0.5 \cdot 1) = -14.5 \\
Q_6^{(1)} &= Q_6^{(0)} + (\alpha \cdot U_{2,6}^{(0)}) = (-15) + (0.5 \cdot 1) = -14.5
\end{aligned}
$$

Performing a second iteration:

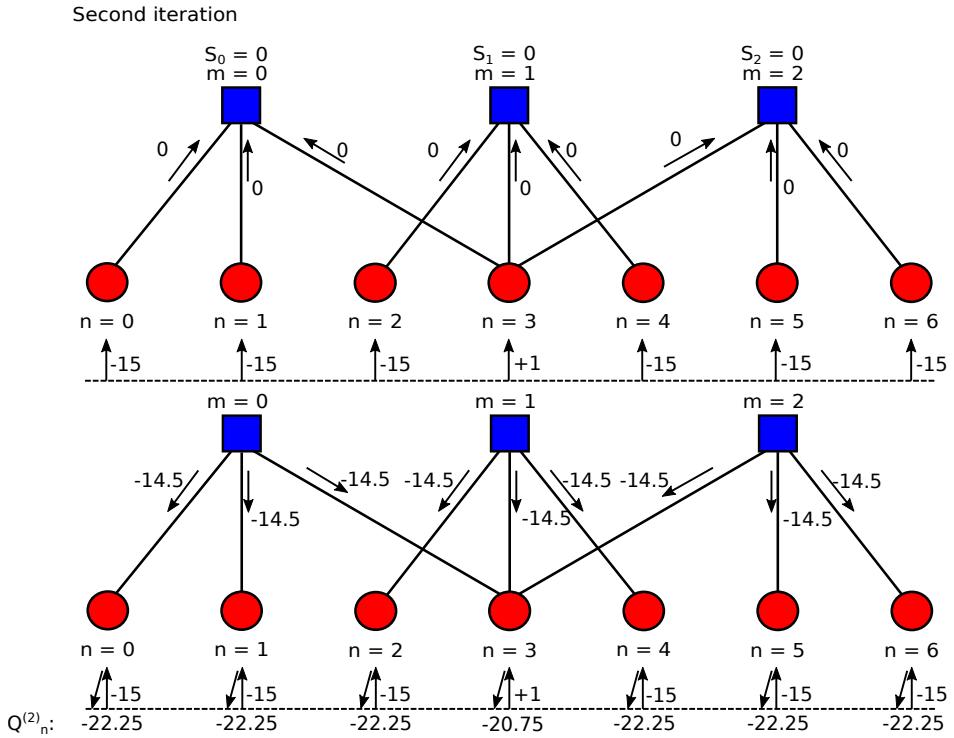

Figure 1.29: RBI-MSD example. Second iteration.

The messages computed in each VN are sent from VNs to CNs, computing the syndromes:

$$s_0 = \hat{x}_0^{(1)} \oplus \hat{x}_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 \oplus 0 = 0$$
$$s_1 = \hat{x}_2^{(1)} \oplus \hat{x}_3^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 \oplus 0 = 0$$
$$s_2 = \hat{x}_3^{(1)} \oplus \hat{x}_5^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 \oplus 0 = 0$$

where

$$\hat{x}_n^{(1)} = \begin{cases} 1, & Q_n^{(1)} \geq 0 \\ 0, & otherwise \end{cases}$$

Once the syndromes are computed, the value that will satisfy each of the parity-check equations is processed. The computation of the magnitude of the messages is done as follows:

$$\sigma_{0,0}^{(1)} = s_0^{(1)} \oplus \hat{x}_0^{(1)} = 0 \oplus 0 = 0$$
$$\sigma_{0,1}^{(1)} = s_0^{(1)} \oplus \hat{x}_1^{(1)} = 0 \oplus 0 = 0$$
$$\sigma_{0,3}^{(1)} = s_0^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0$$
$$\sigma_{1,2}^{(1)} = s_1^{(1)} \oplus \hat{x}_2^{(1)} = 0 \oplus 0 = 0$$
$$\sigma_{1,3}^{(1)} = s_1^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0$$
$$\sigma_{1,4}^{(1)} = s_1^{(1)} \oplus \hat{x}_4^{(1)} = 0 \oplus 0 = 0$$
$$\sigma_{2,3}^{(1)} = s_2^{(1)} \oplus \hat{x}_3^{(1)} = 0 \oplus 0 = 0$$
$$\sigma_{2,5}^{(1)} = s_2^{(1)} \oplus \hat{x}_5^{(1)} = 0 \oplus 0 = 0$$
$$\sigma_{2,6}^{(1)} = s_2^{(1)} \oplus \hat{x}_6^{(1)} = 0 \oplus 0 = 0$$

On the other hand, the computation of the magnitude of the messages is done as follows:

$$U_{0,0}^{(1)} = min(|Q_{1,0}^{(1)}|, |Q_{3,0}^{(1)}| = 14.5$$
$$U_{0,1}^{(1)} = min(|Q_{0,0}^{(1)}|, |Q_{3,0}^{(1)}| = 14.5$$
$$U_{0,3}^{(1)} = min(|Q_{0,0}^{(1)}|, |Q_{1,0}^{(1)}| = 14.5$$
$$U_{1,2}^{(1)} = min(|Q_{3,1}^{(1)}|, |Q_{4,1}^{(1)}| = 14.5$$
$$U_{1,3}^{(1)} = min(|Q_{2,1}^{(1)}|, |Q_{4,1}^{(1)}| = 14.5$$
$$U_{1,4}^{(1)} = min(|Q_{2,1}^{(1)}|, |Q_{3,1}^{(1)}| = 14.5$$
$$U_{2,3}^{(1)} = min(|Q_{5,2}^{(1)}|, |Q_{6,3}^{(1)}| = 14.5$$
$$U_{2,5}^{(1)} = min(|Q_{3,2}^{(1)}|, |Q_{6,2}^{(1)}| = 14.5$$
$$U_{2,6}^{(1)} = min(|Q_{3,2}^{(1)}|, |Q_{5,2}^{(1)}| = 14.5$$

In this second iteration the reliability values for computing the tentative decoding are:

$$Q_0^{(2)} = Q_0^{(0)} + (\alpha \cdot U_{0,0}^{(1)}) = (-15) + (0.5 \cdot (-14.5)) = -22.25$$
$$Q_1^{(2)} = Q_1^{(0)} + (\alpha \cdot U_{0,1}^{(1)}) = (-15) + (0.5 \cdot (-14.5)) = -22.25$$
$$Q_2^{(2)} = Q_2^{(0)} + (\alpha \cdot U_{1,2}^{(1)}) = (-15) + (0.5 \cdot (-14.5)) = -22.25$$
$$Q_3^{(2)} = Q_3^{(0)} + (\alpha \cdot (U_{0,3}^{(1)} + U_{1,3}^{(1)} + U_{2,3}^{(1)})) = 1 + (0.5 \cdot ((-14.5) + (-14.5) + (-14.5))) = -20.75$$
$$Q_4^{(2)} = Q_4^{(0)} + (\alpha \cdot U_{1,4}^{(1)}) = (-15) + (0.5 \cdot (-14.5)) = -22.25$$
$$Q_5^{(2)} = Q_5^{(0)} + (\alpha \cdot U_{2,5}^{(1)}) = (-15) + (0.5 \cdot (-14.5)) = -22.25$$
$$Q_6^{(2)} = Q_6^{(0)} + (\alpha \cdot U_{2,6}^{(1)}) = (-15) + (0.5 \cdot (-14.5)) = -22.25$$

Finally, it can be checked that all the syndromes are satisfied and, hence, the code word is corrected. The error in VN $n = 3$ has been corrected, flipping the sign of the node from positive to negative. All the Q_n messages are now negative, and all the hard-decision messages are zero, as shown in Fig. 1.29.

Note that the magnitudes obtained after the second iteration are much larger than those obtained in the previous hard-decision examples, which indicates that the values obtained in this case are more reliable, so this algorithm converges faster than the previous ones.

1.5.4. Performance comparison of RBI decoders

Fig. 1.30 shows the BER performace of the previous algorithms RBI-MLGD, MRBI-MLGD, RBI-MSD and MSA. The code used in this simulation is the (2304,2048) Algebraic LDPC code with $d_c = 36$ and $d_v = 6$ and the maximum number of iterations is set to 20.

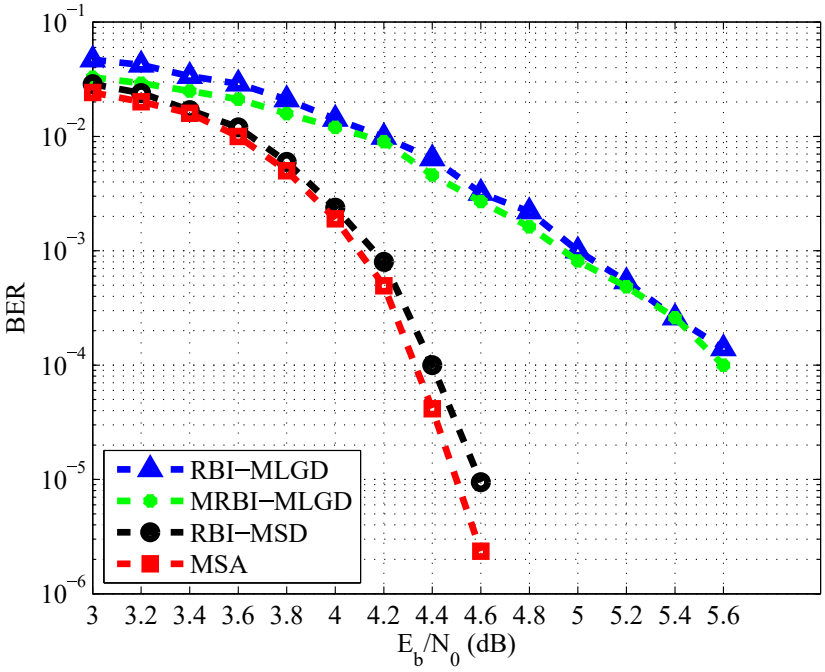

Figure 1.30: RBI-MLGD, MRBI-MLGD, RBI-MSD and MSA performance for the (2304,2048) Algebraic LDPC codes.

In Fig. 1.30 it can be observed that RBI-MSD outperforms RBI-MLGD and MRBI-MLGD by 1.2 dB at BER 10^{-4} at the expense of increasing complexity. In addition RBI-MSD has a performance loss of 0.1 dB with respect to MSA at BER 10^{-5}. As for RBI-MLGD and MRBI-MLGD, RBI-MLGD scheme has a performance loss $\approx$ 0.1 dB compared to MRBI-MLGD following both of them the same slope.

1.6. Conclusions

In this chapter we have reviewed the basics of LDPCs needed to understand the contribution of this thesis. The message passing decoding schedules have been introduced and the three basic hardware architectures for LDPC decoders, fully-parallel, fully-serial and partially-parallel, have been analyzed. Three soft-decision decoding algorithms, SPA, MSA and Optimized 2-bit Min-Sum, and three hard-decision ones, RBI-MLGD, MRBI-MLGD and RBI-MSD, have been presented and their performance comparison have been included.

For the architectural implementation, it is concluded that the fully-parallel architecture is energy efficient due to the fact that it only needs a single cycle per message-passing iteration, at the expenses of a high wiring congestion and an increase of the area due to the use of registers. With the use of fully-serial architecture, routing congestion is greatly reduced and area is minimized obtaining in this case an increase of the number of clock cycles (throughput is dramatically reduced). So, the use of this architecture is only for systems in which latency is not a critical parameter. With the aim of obtaining an architecture with the improvements of the previous architectures, partially-parallel architecture was introduced, obtaining a balance of the benefits of the two previous architectures by partitioning H into rowwise and columnwise groupings so that a set of check node and variable node updates can be done per cycle.

From the previous analysis it can be concluded that the Sum-Product decoding algorithm provides the best performance but requires the highest computational complexity. In order to reduce the arithmetic resources of SPA, scaled Min-Sum decoding algorithm was proposed. However, scaled Min-Sum causes slightly performance degradations and its complexity is still moderated. With the aim of reducing complexity and routing congestion of MSA, Optimized 2-bit Min-Sum decoding algorithm was proposed, which reduces the message-passing to two bits and gets improvements in routing but with a small loss in coding gain.

Among the three hard-decision decoding algorithms presented, it can be concluded that the RBI-MSD achieves the best decoding performance but it has high complexity due to it is an integer message-passing algorithm which contains soft information compared to binary message-passing algorithms as RBI-MLGD and MRBI-MLGD. In addition RBI-MSD and MRBI-MLGD use a scaling factor to compensate the performance degradation and try to balance the differences between the channel information and the constant amplitude of the votes. Moreover, the RBI-MLGD uses the previouly updated messages $Q_n^{(i)}$ in the iterative procedures while MRBI-MLGD and RBI-MSD use the initial messages $Q_n^{(0)}$ in the iterative procedure.

Chapter 2

One-Minimum-Only Min-Sum Algorithm (OMO-MSA)

2.1. Introduction

Since the first decoding algorithm, SPA, appeared, several proposals have been published in order to reduce or solve the check node complexity. Among these proposals, the one that has a better trade-off between complexity and coding gain is MSA.

As we have seen in the previous chapter, to compute the reliability of the results obtained from the parity check equation, the check node messages, $R_{m,n}^{(i)}$, are calculated as

$$R_{m,n}^{(i)} = \sigma_{m,n}^{(i)} \kappa_{m,n}^{(i)} = \prod_{n' \in N(m) \setminus n} sgn(Q_{n',m}^{(i-1)}) \min_{n' \in N(m) \setminus n} |Q_{n',m}^{(i-1)}|.$$

From the two parts that compose this equation, the first one, the sign calculation, is not computationally complex, since the size of the data is one bit and the operation to be performed is a simple logical operation.

However, although the MSA has reduced the check node complexity, it remains a bottleneck of processing due to the second part of this equation, the magnitude of the messages:

$$\kappa_{m,n}^{(i)} = \min_{n' \in N(m) \setminus n} |Q_{n',m}^{(i-1)}|.$$

The complexity in the calculation of the magnitude is due to the size of the operands, which is greater than one bit. The number of operations in this case depends on d_c, which can be large, especially for very high-rate codes. In order to know the number of comparisons that performs each check node (CN), the case of a CN with $d_c = 8$ is analyzed, following the example of Fig. 2.1.

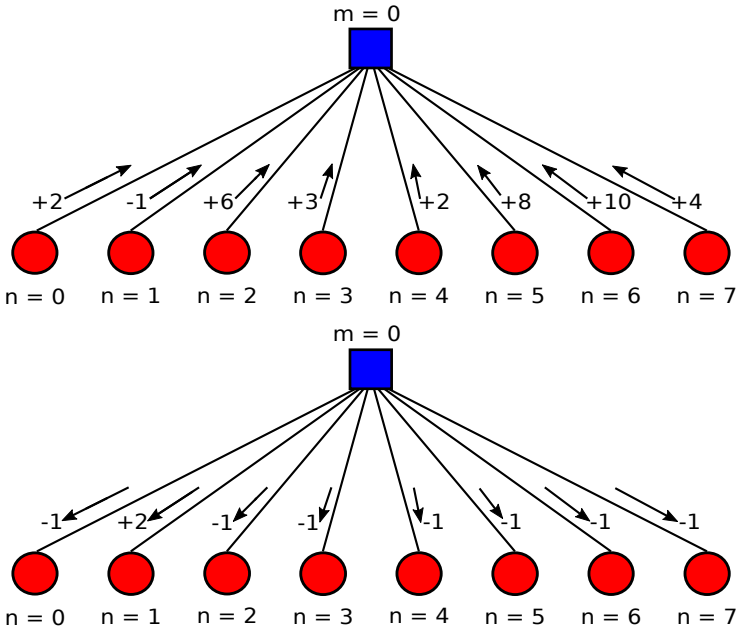

Figure 2.1: MSA with $d_c = 8$.

Applying the $\sigma_{m,n}^{(i)} = \prod_{n' \in N(m) \setminus n} sgn(Q_{n',m}^{(i-1)})$ equation of the sign calculation, we obtain:

$$\sigma_{0,0}^{(0)} = sgn(Q_{1,0}^{(0)}) \oplus sgn(Q_{2,0}^{(0)}) \oplus sgn(Q_{3,0}^{(0)}) \oplus sgn(Q_{4,0}^{(0)}) \oplus sgn(Q_{5,0}^{(0)}) \oplus sgn(Q_{6,0}^{(0)}) \oplus sgn(Q_{7,0}^{(0)}) = 1$$
$$\sigma_{0,1}^{(0)} = sgn(Q_{0,0}^{(0)}) \oplus sgn(Q_{2,0}^{(0)}) \oplus sgn(Q_{3,0}^{(0)}) \oplus sgn(Q_{4,0}^{(0)}) \oplus sgn(Q_{5,0}^{(0)}) \oplus sgn(Q_{6,0}^{(0)}) \oplus sgn(Q_{7,0}^{(0)}) = 0$$
$$\sigma_{0,2}^{(0)} = sgn(Q_{0,0}^{(0)}) \oplus sgn(Q_{1,0}^{(0)}) \oplus sgn(Q_{3,0}^{(0)}) \oplus sgn(Q_{4,0}^{(0)}) \oplus sgn(Q_{5,0}^{(0)}) \oplus sgn(Q_{6,0}^{(0)}) \oplus sgn(Q_{7,0}^{(0)}) = 1$$
$$\sigma_{0,3}^{(0)} = sgn(Q_{0,0}^{(0)}) \oplus sgn(Q_{1,0}^{(0)}) \oplus sgn(Q_{2,0}^{(0)}) \oplus sgn(Q_{4,0}^{(0)}) \oplus sgn(Q_{5,0}^{(0)}) \oplus sgn(Q_{6,0}^{(0)}) \oplus sgn(Q_{7,0}^{(0)}) = 1$$
$$\sigma_{0,4}^{(0)} = sgn(Q_{0,0}^{(0)}) \oplus sgn(Q_{1,0}^{(0)}) \oplus sgn(Q_{2,0}^{(0)}) \oplus sgn(Q_{3,0}^{(0)}) \oplus sgn(Q_{5,0}^{(0)}) \oplus sgn(Q_{6,0}^{(0)}) \oplus sgn(Q_{7,0}^{(0)}) = 1$$
$$\sigma_{0,5}^{(0)} = sgn(Q_{0,0}^{(0)}) \oplus sgn(Q_{1,0}^{(0)}) \oplus sgn(Q_{2,0}^{(0)}) \oplus sgn(Q_{3,0}^{(0)}) \oplus sgn(Q_{4,0}^{(0)}) \oplus sgn(Q_{6,0}^{(0)}) \oplus sgn(Q_{7,0}^{(0)}) = 1$$
$$\sigma_{0,6}^{(0)} = sgn(Q_{0,0}^{(0)}) \oplus sgn(Q_{1,0}^{(0)}) \oplus sgn(Q_{2,0}^{(0)}) \oplus sgn(Q_{3,0}^{(0)}) \oplus sgn(Q_{4,0}^{(0)}) \oplus sgn(Q_{5,0}^{(0)}) \oplus sgn(Q_{7,0}^{(0)}) = 1$$
$$\sigma_{0,7}^{(0)} = sgn(Q_{0,0}^{(0)}) \oplus sgn(Q_{1,0}^{(0)}) \oplus sgn(Q_{2,0}^{(0)}) \oplus sgn(Q_{3,0}^{(0)}) \oplus sgn(Q_{4,0}^{(0)}) \oplus sgn(Q_{5,0}^{(0)}) \oplus sgn(Q_{6,0}^{(0)}) = 1$$

As can be seen, the operation performed is a logic xor with a 1-bit operand. Following with the computation of the magnitude of the messages $\kappa_{m,n}^{(i)} = \min_{n' \in N(m) \setminus n} |Q_{n',m}^{(i-1)}|$:

$$\kappa_{0,0}^{(0)} = min(|Q_{0,1}^{(0)}|, |Q_{0,2}^{(0)}|, |Q_{0,3}^{(0)}|, |Q_{0,4}^{(0)}|, |Q_{0,5}^{(0)}|, |Q_{0,6}^{(0)}|, |Q_{0,7}^{(0)}|) = 1$$
$$\kappa_{0,1}^{(0)} = min(|Q_{0,0}^{(0)}|, |Q_{0,2}^{(0)}|, |Q_{0,3}^{(0)}|, |Q_{0,4}^{(0)}|, |Q_{0,5}^{(0)}|, |Q_{0,6}^{(0)}|, |Q_{0,7}^{(0)}|) = 2$$
$$\kappa_{0,2}^{(0)} = min(|Q_{0,0}^{(0)}|, |Q_{0,1}^{(0)}|, |Q_{0,3}^{(0)}|, |Q_{0,4}^{(0)}|, |Q_{0,5}^{(0)}|, |Q_{0,6}^{(0)}|, |Q_{0,7}^{(0)}|) = 1$$
$$\kappa_{0,3}^{(0)} = min(|Q_{0,0}^{(0)}|, |Q_{0,1}^{(0)}|, |Q_{0,2}^{(0)}|, |Q_{0,4}^{(0)}|, |Q_{0,5}^{(0)}|, |Q_{0,6}^{(0)}|, |Q_{0,7}^{(0)}|) = 1$$
$$\kappa_{0,4}^{(0)} = min(|Q_{0,0}^{(0)}|, |Q_{0,1}^{(0)}|, |Q_{0,2}^{(0)}|, |Q_{0,3}^{(0)}|, |Q_{0,5}^{(0)}|, |Q_{0,6}^{(0)}|, |Q_{0,7}^{(0)}|) = 1$$
$$\kappa_{0,5}^{(0)} = min(|Q_{0,0}^{(0)}|, |Q_{0,1}^{(0)}|, |Q_{0,2}^{(0)}|, |Q_{0,3}^{(0)}|, |Q_{0,4}^{(0)}|, |Q_{0,6}^{(0)}|, |Q_{0,7}^{(0)}|) = 1$$
$$\kappa_{0,6}^{(0)} = min(|Q_{0,0}^{(0)}|, |Q_{0,1}^{(0)}|, |Q_{0,2}^{(0)}|, |Q_{0,3}^{(0)}|, |Q_{0,4}^{(0)}|, |Q_{0,5}^{(0)}|, |Q_{0,7}^{(0)}|) = 1$$
$$\kappa_{0,7}^{(0)} = min(|Q_{0,0}^{(0)}|, |Q_{0,1}^{(0)}|, |Q_{0,2}^{(0)}|, |Q_{0,3}^{(0)}|, |Q_{0,4}^{(0)}|, |Q_{0,5}^{(0)}|, |Q_{0,6}^{(0)}|) = 1$$

As can be concluded, in the magnitude calculation $7 \times 8 = 56$ comparisons are performed to search the minimum (7 comparisons in each of the 8 VNs, remember that the intrinsic information of the node is not computed), so the number of comparisons are $d_c \times (d_c - 1)$.

The previous example shows the first method used in the literature to search the magnitudes and their assignment to each VN. This search is improved by the value-reuse technique (Chapter 1), which reduces the search of the magnitudes to a search of two minimums, firstly searching for min_1, then searching for min_2 and, finally, assigning to each VN min_1 or min_2. In this latter method, the number of comparisons are reduced to $(2 \times d_c - 1) + d_c = 3 \times d_c - 1$. Even with this value-reuse technique the complexity is still too high.

As our goal is to reduce the complexity of calculating the magnitude, three different options can be analyzed:

1. To avoid or simplify the way of calculating min_1.
2. To avoid or simplify the way of calculating min_2.
3. To avoid or simplify the way of calculating min_1 and min_2.

Analyzing the three options, it is observed that min_1 approximation (first option) would produce a greater impact since it is sent to $d_c - 1$ nodes, as it can be seen in Fig. 2.2.

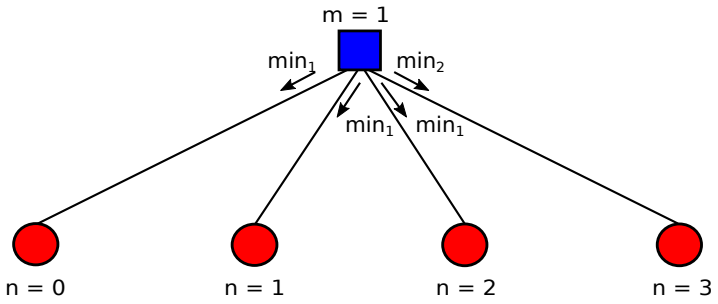

Figure 2.2: Tanner Graph.

Therefore, we discard this case because an erroneous calculation of this approach would greatly affect the decoding. The approximation of the min_2 would have less impact because the min_2 is only sent to one node. In addition, it would reduce the complexity since $d_c - 1$ comparisons are eliminated. The third option will be discussed in the next chapter with the 2-bit decoders.

Therefore, starting from the hypothesis that estimating the second minimum will have a smaller impact than estimating the first minimum, several methods to approximate the second minimum at the check node have been recently proposed.

The first one is the single-minimum MSA from [21] - [22], in which the second minimum is computed by means of applying a constant correction factor on the

first minimum. This technique has the lowest complexity but, unfortunately, it introduces some important performance degradation in the error-floor region due to the use of a constant correction factor applied to the absolute minimum in order to estimate the second one. This happens because the difference between min_1 and min_2 is not constant through the iterations and for different signal-to-noise ratio values, as it is shown in Fig. 2.3 and Fig. 2.4, respectively.

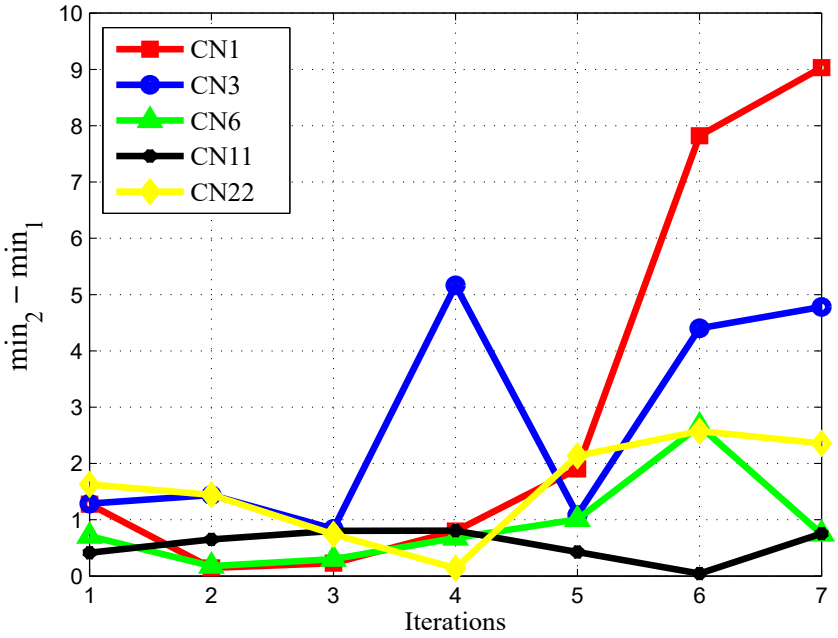

Figure 2.3: Difference between min_1 and min_2 through the iterations. Where CNn corresponds to the n-th CN equation from the (2048,1723) LDPC code.

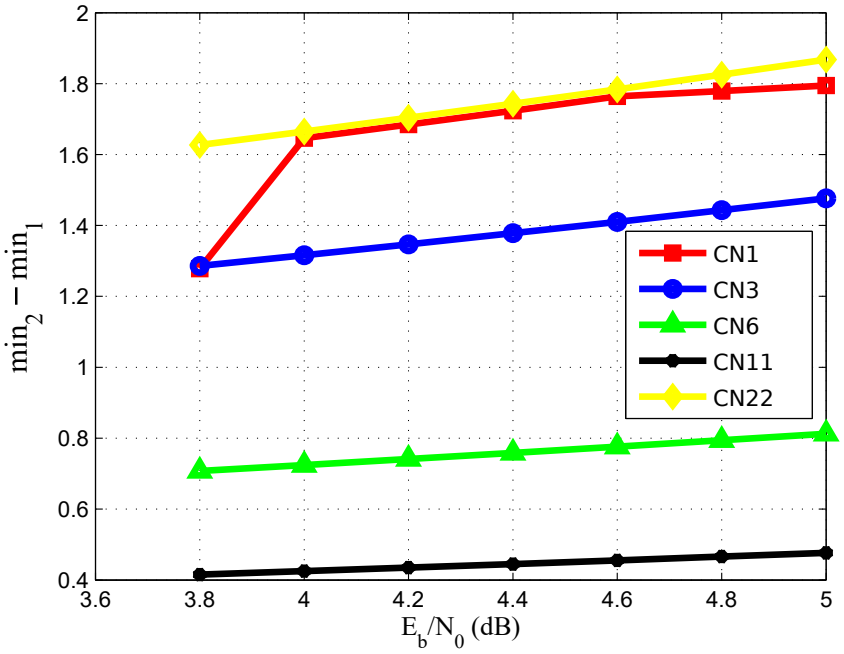

Figure 2.4: Difference between min_1 and min_2 through the different signal-to-noise ratio values. Where CNn corresponds to the n-th CN equation from the (2048,1723) LDPC code.

The second method to approximate min_2 tries to overcome the previously mentioned problem proposing an algorithm named as variable-weight MSA [23]. The variable-weight MSA avoids performance degradation by means of computing a correction factor different for each iteration or range of iterations. This algorithm requires an optimization of parameters that depends on the number of iteration, the value of the correction factor in the previous iterations and the signal-to-noise. The main disadvantage of the method proposed in [23] is how to obtain the correction factor. To compute the value of the correction factor for iteration i in an heuristic way, the values of $i+1$ and $i-1$ have to be considered, because as authors from [23] indicate, the decision made on a correction factor for one iteration will be influenced by correction factors from previous iterations. Moreover, the optimization process in [23] needs to be repeated for each signal-to-noise ratio, and each LDPC code. Another disadvantage of this process is that the channel status should be known at any time in order to change the weights.

Although performance degradation is avoided, the min_2 calculation is still dependent on constant values and not on other parameters, this keeps the variation of the difference between the two minimums and the iterations/signal-to-noise.

In order to solve the previous problems we proposed a novel approximation to compute the second minimum without increasing the complexity or sacrificing error correction performance. This new estimation combines the simplicity form [21] - [22], with the adaptability from [23], as the proposed method does not use the same correction factor in different iterations. Moreover, this approximation does not require a thorough off-line optimization (finding the best combination of parameters for each iteration and signal-to-noise ratio), because with only two constant factors per code the second minimum is estimated dynamically. Also, our solution does not degrade the correction performance and allows a faster optimization of the parameters and higher area-throughput efficiency in the derived hardware implementations, as will be seen in the following sections.

2.2. Only one minimum check node approximation

The main idea of the approximation proposed in this thesis is to provide an estimator that allows a dynamic adjustment of the correction factor, which is adapted automatically to the iteration number and the signal-to-noise ratio. We call this approach one minimum only Min-Sum algorithm (OMO-MSA).

The min_1 is the least reliable estimator of all received messages and the min_2 is the second least reliable element. So, the information that gives us the difference between min_1 and min_2 is the difference in reliability between the calculation of some messages and others. Therefore, our aim is to obtain a measure of the less reliable message in comparison to the other messages, and due to this we have decided to use another message other than min_2 to make the estimation of the min_2.

Let us define the sets $N0(m)$ and $N1(m)$ as $N0(m) \cup N1(m) = N(m)$, where the cardinality of $N0(m)$ and $N1(m)$ is $\sharp N0(m) = \sharp N1(m) = d_c/2$.

Let define min_1 as the absolute minimum of $N(M)$ as

$$min_1 = min(\min_{n' \in N0(m)}(|Q_{n',m}|), \min_{n' \in N1(m)}(|Q_{n',m}|)). \tag{2.1}$$

From the previous equation, the complementary operation,

$$min_2^{'''} = max(\min_{n' \in N0(m)}(|Q_{n',m}|), \min_{n' \in N1(m)}(|Q_{n',m}|)), \tag{2.2}$$

will satisfy[1] that in a 50% of the cases min_2''' is equal to the second minimum (min_2), $(min_1 \in N0(m) \wedge min_2 \in N1(m)) \vee (min_1 \in N1(m) \wedge min_2 \in N0(m))$, and in the rest of cases, the value is larger, $(min_1, min_2 \in N0(m)) \vee (min_1, min_2 \in N1(m))$.

For example, having two sets with the same number of elements $d_c/2 = 4$, two cases can be found, as shown in Fig. 2.5 and 2.6.

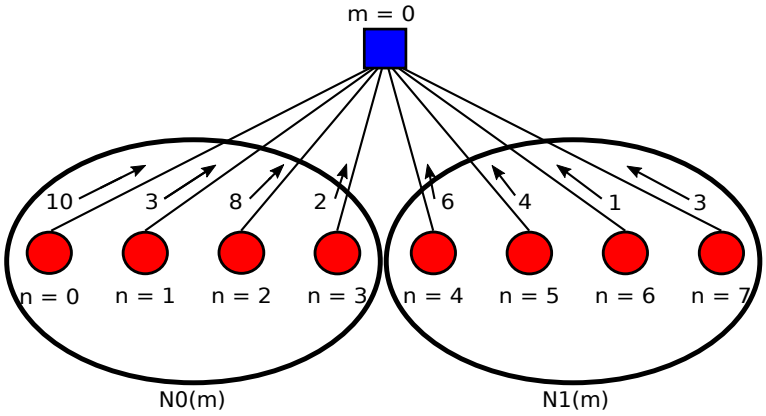

Figure 2.5: min_1 and min_2 are located in different sets.

In Figure 2.5, it can be analyzed the first scenario where $min_1 = 1$ is in the $N1(m)$ and $min_2 = 2$ is in the $N0(m)$ set:$(min_1 \in N0(m) \wedge min_2 \in N1(m)) \vee (min_1 \in N1(m) \wedge min_2 \in N0(m))$. So, in this case $min_2''' \in N0(m) = min_2$.

[1]Assuming, without loss of generality, the equiprobability of the distribution of the d_c messages

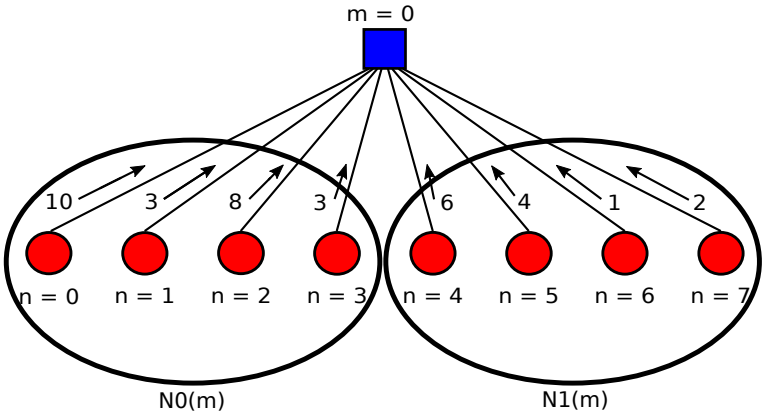

Figure 2.6: min_1 and min_2 are located in the same set.

On the other hand, in Fig. 2.6 we can see the case in which min_1 and min_2 are in the same subset $N1(m)$: $(min_1, min_2 \in N0(m)) \vee (\ min_1, min_2 \in N1(m))$. In this case the $min_2^{'''}$ has a magnitude larger than the min_2.

The scenarios from Fig. 2.5 and Fig. 2.6 can be generalized concluding than $min_2^{'''}$ is an upper bound, or overestimation, of min_2. This upper bound provides useful information of the distance between min_1 and the rest of the messages at the check node, but it cannot be used as a final approximation because we would be assuming that there is more distance between the less reliable and a more reliable value, and we would be making an incorrect estimation of the channel. So, min_2 is between min_1 and $min_2^{'''}$, and this distance may depend on the code, the channel and other variables such as the SNR. Due to this fact, we would need to have some degree of flexibility to estimate min_2, therefore, we propose to make a linear approximation as we explain next.

To compensate the $min_2^{'''}$ value, a second estimator is computed using min_1 as a lower bound. The min_1 value is multiplied by a correction factor, α_2, similarly to the one in [21] to modify its magnitude. Combining both approximations, we obtain $min_2^* = min_1 \cdot \alpha_2 + min_2^{'''} \cdot \gamma$, which is similar to an average of two bound values. Both α_2 and γ allow us to weight each one of the approximations depending on the code. It is important to remark that min_2^* auto-adjusts the correction value applied to min_1,

$$\chi = \alpha_2 + \frac{\min_2^{'''}}{\min_1} \cdot \gamma,$$

because the estimation includes a reference of the magnitude of the rest of the messages. So $min_2^* = min_1 \cdot \chi$.

For example, with the input values to the check node shown in Fig. 2.7, first, the search for the min_1 is performed in both sets $N0(m)$ and $N1(m)$. Observing Fig. 2.7 it is concluded that $min_1 = -0.61$. Once min_1 is obtained, we choose $min_2^{'''}$, in this example $min_2^{'''} = -4.94$. With the values calculated above and using an $\alpha_2 = 0.75$ and $\gamma = 1$, we apply the correction factor as follows:

$$min_2^* = min_1 \cdot \chi = min_1 \cdot \alpha_2 + min_2^{'''} \cdot \gamma = (-0.61) \cdot 0.75 + -4.94 \cdot 1 = -5.4$$

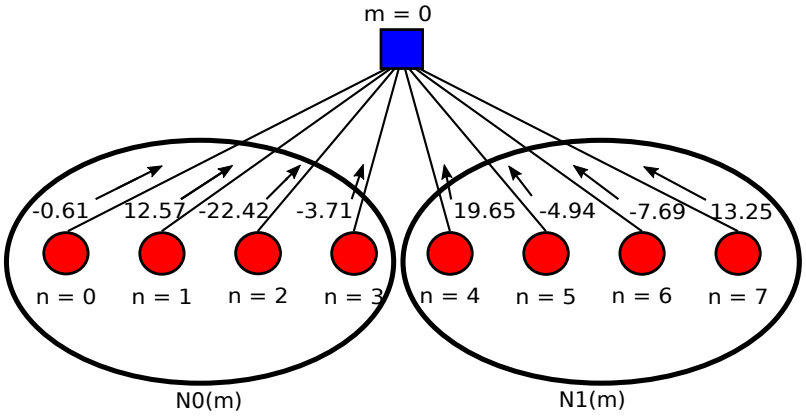

Figure 2.7: CN inputs.

To calculate the values of α_2 and γ, an optimization is done depending on the signal-to-noise ratio and the iterations, but having a quotient, it was self-tuning. Then, we chose an intermediate point between the slope of the waterfall and the area of error floor, first ensuring that the slope of waterfall is as close as possible to the waterfall slope of the original MSA (without approximation). Once at this point, we perform the optimization of the parameters. It is important to remark that this proposal only needs to optimize two parameters. No further searches with different iteration values or signal-to-noise ratios are required as far as there is only one constant value of α_2 and γ for each code, and these values only change with the code.

To sum up this process only has to be applied once, as follows:

1. Look for the min_1 and $min_2^{'''}$ in $N0(m)$ and $N1(m)$.
2. Multiply by two constants α_2 and γ.
3. Perform an addition.

This correction factor does not depend on d_c and it has no complexity, in addition we save $d_c - 1$ operations.

2.3. Only one minimum check node architecture

For the hardware implementation of MSA's CNU, two minimum values are searched among the d_c inputs using a two minimum comparator tree, whose hardware complexity is:

$$N_{Comparators} = (d_c - 1) + (2 \cdot (\frac{d_c}{2} - 1)).$$

As for multiplexers:

$$N_{Multiplexers} = d_c + (4 \cdot (\frac{d_c}{2} - 1)).$$

Fig. 2.8 shows the MSA CNU architecture and the comparator blocks used in the different stages of the tree. The pipeline registers are drawn as dashed lines. The critical path is made of two multiplexors and one comparator and is determined by the module $R2_0$.

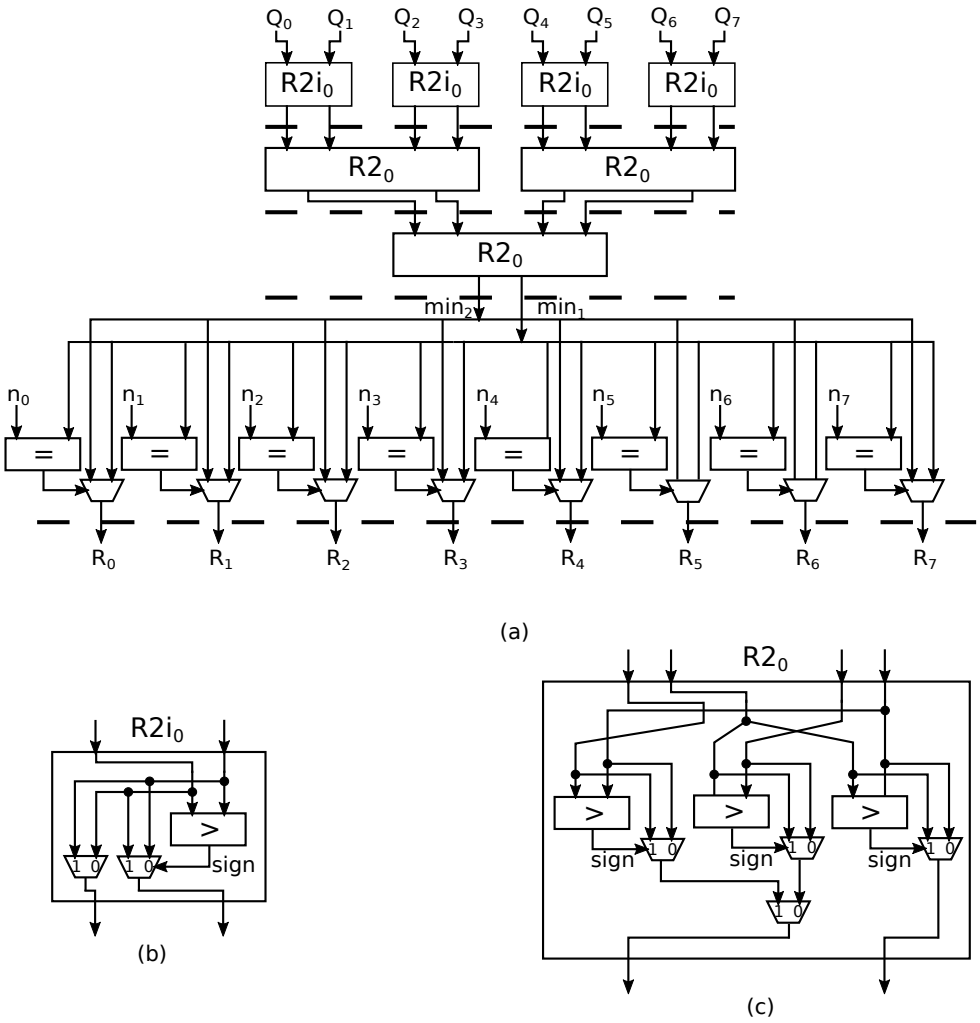

Figure 2.8: (a) MSA CNU architecture. (b) Initial Radix-2 block ($R2i_0$). (c) Radix-2 block ($R2_0$).

As regarding OMO, in terms of hardware, only one adder to sum $min_1 \cdot \alpha_2$ and $min_2''' \cdot \gamma$ are required as shown in Fig. 2.9. No extra control or memories are needed. The multiplication by α_2 and γ is wired as the factors are selected as powers of two. This will have a great impact on area compared to MSA decoders as we will show in next section. As in the previous figure, pipeline registers are drawn as dashed lines and the critical path is reduced in one multiplexor as well as the wiring conexions are almost half.

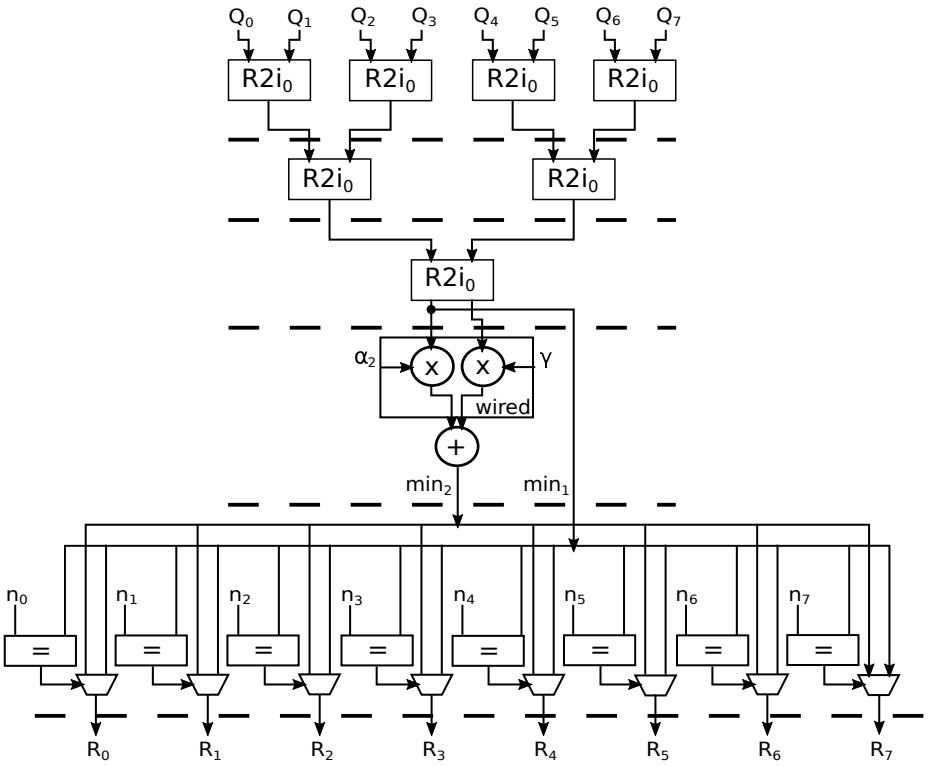

Figure 2.9: OMO CNU architecture.

The number of comparators and multiplexers are shown in Eq. 2.3 and Eq. 2.4 respectively.

$$N_{Comparators} = d_c - 1. \tag{2.3}$$

$$N_{Multiplexers} = 2 \cdot (d_c - 1). \tag{2.4}$$

The critical path is reduced with respect to MSA due to the elimination of Radix-2 blocks ($R2_0$).

Figures 2.10, 2.11 and 2.12 compare the number of comparators, multiplexers and registers, respectively, of MSA and OMO-MSA, for different values of d_c (from 2 to 64).

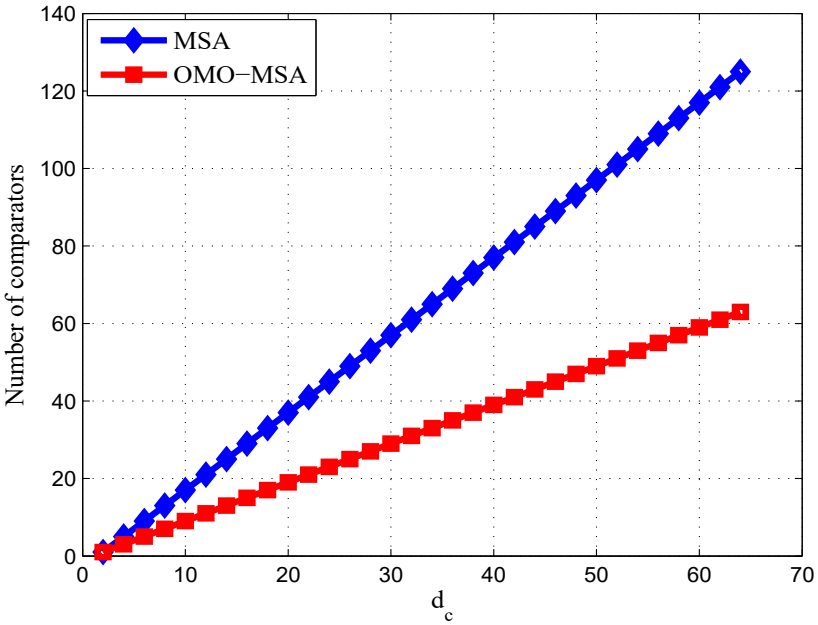

Figure 2.10: MSA and OMO-MSA number of comparators.

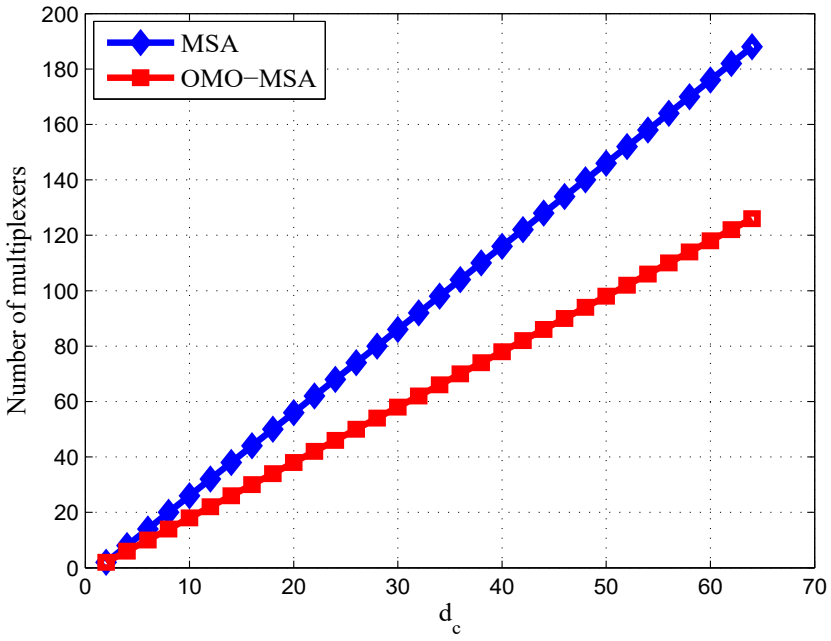

Figure 2.11: MSA and OMO-MSA number of multiplexers.

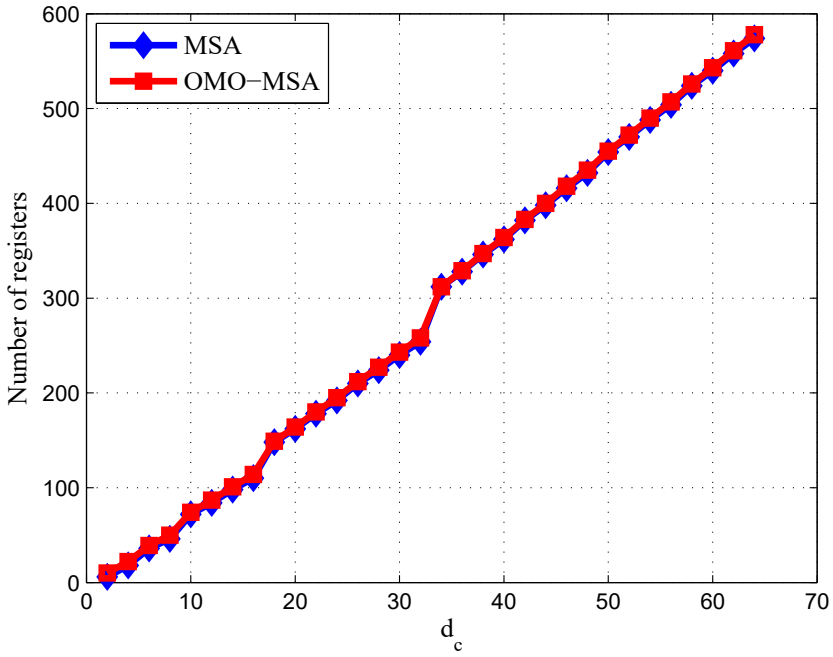

Figure 2.12: MSA and OMO-MSA number of registers.

2.4. Error correction performance

Firstly, performance tests were performed using a computer cluster and we obtained the results for the waterfall region ($BER \geq 10^{-8}$). But, as our objective was also to verify that this algorithm still worked properly in the error-floor region, where the probability of error is very low ($BER < 10^{-8}$), and it was not feasible to obtain results in a reasonable time with the computer network, we used a hardware emulator implemented in an FPGA [27]. With the use of this emulator with a decoding rate of 1.35Gbps (1 core), configurable up to 4 cores, we were able to obtain the results between $BER < 10^{-8}$ and $BER > 10^{-15}$.

Fig. 2.13 shows the blocks of the hardware emulator. It is composed of the following blocks:

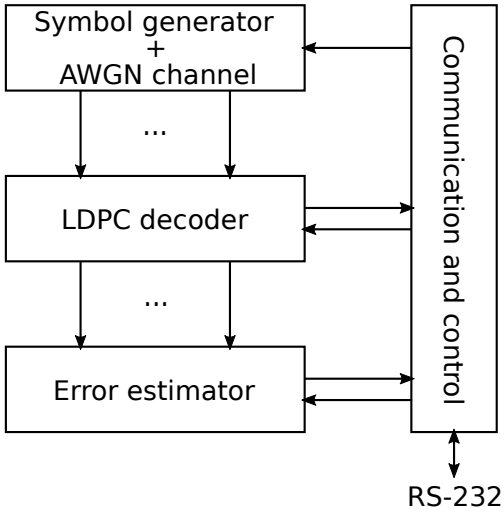

Figure 2.13: Block diagram of the hardware emulator.

1. Symbol generator. In this block the symbols are generated from a codeword stored in a ROM. The bits of the codeword are modulated in BPSK and AWGN is added in order to emulate the AWGN channel.

2. LDPC decoder. Here the decoder is implemented using a partially parallel architecture based on memories, as described in the previous chapter.

3. Error estimator. In this block the number of erroneous bits per iteration is calculated by comparing the output of the decoder with the codeword transmitted, which is stored in a ROM.

4. Communication and control. This block is responsible for the internal control of the hardware emulator and the communication with the software interface.

Fig. 2.14 shows the BER performance of the scaled MSA and the OMO-MSA quantized with seven bits for the (2048, 1723) LDPC code, applied to the IEEE 802.3an standard, with parameters $d_c = 32$, $d_v = 6$. These simulations have been obtained using the FPGA-based emulator. Both decoders include the scaling factor α, that provides the best performance.

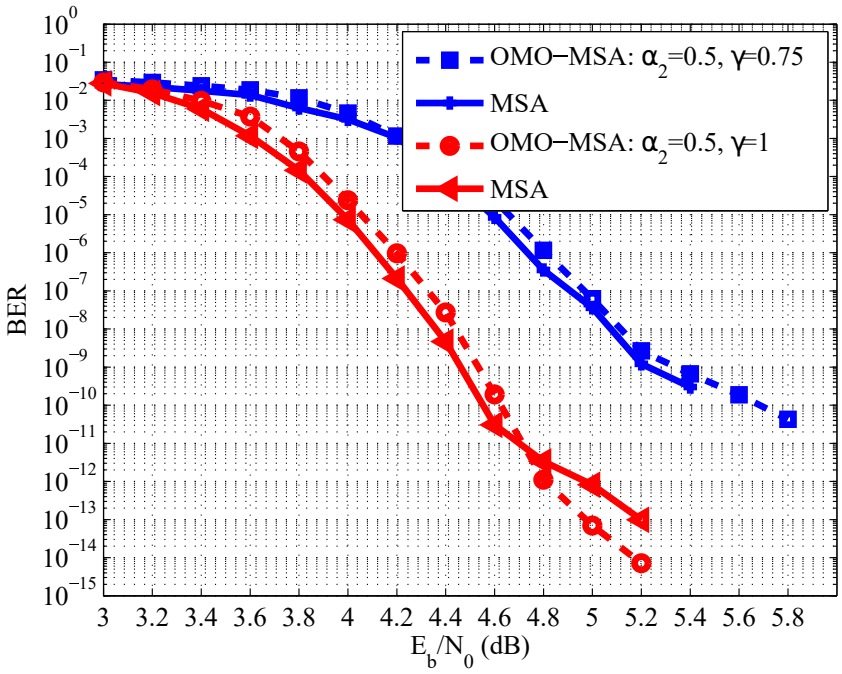

Figure 2.14: BER performance of MSA and OMO-MSA (seven-bit quantization) for the (2048, 1723) LDPC code in red lines and (2304, 2048) LDPC code in blue lines.

It can be seen that our approach has similar performance in the waterfall region, with just 0.08 dB of performance loss up to $BER = 10^{-11}$, and in the error floor region, from $BER = 10^{-11}$ to $BER = 10^{-14}$, slightly improves the performance. This negligible differences are due to the use of hardware friendly coefficients (powers of two) and also to the quantized model. Similar behaviour can be observed in Fig. 2.14 for the (2304, 2048) LDPC code with parameters $d_c = 36$, $d_v = 6$ constructed using the method proposed in [10]. For this code, both the waterfall and the error floor region have almost the same performance, with a difference of less than 0.05 dB, showing that the proposed method to approximate the second minimum at the check node introduces only a small performance degradation compared to the MSA with full processing.

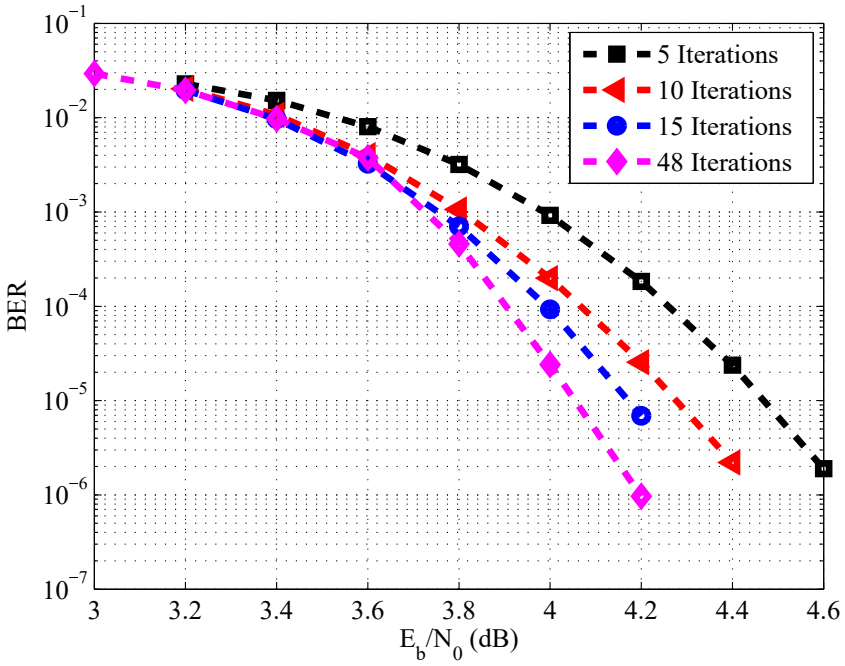

Figure 2.15: OMO-MSA BER performance at 5, 10, 15 and 48 iteration for the (2048,1723) LDPC code.

In Fig. 2.15 OMO-MSA with 5, 10, 15 and 48 iterations for the (2048,1723) LDPC code is represented. As can be seen a performance loss between 0.1 dBs and 0.15 dBs at BER 10^{-5} and 0.1 dBs at BER 10^{-5} is obtained with 15 and 10 iterations, respectively, compared to OMO-MSA with 48 iterations. Between 48 and 5 iterations it can be observed a difference of 0.4 dBs at BER 10^{-5}.

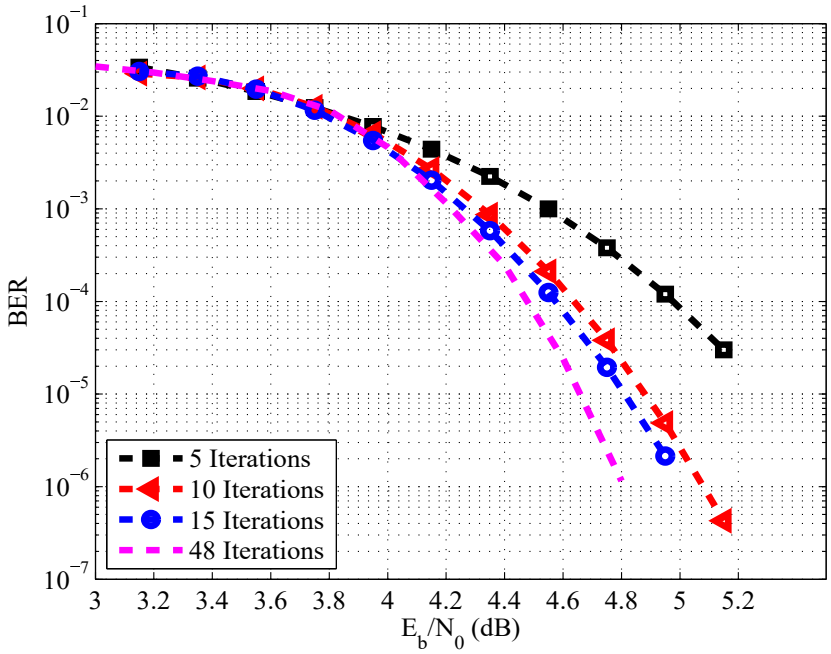

Figure 2.16: OMO-MSA BER performance at 5, 10, 15 and 48 iteration for the (2304,2048) LDPC code.

In Fig. 2.16 OMO-MSA with 5, 10, 15 and 48 iterations for the (2304,2048) LDPC code is represented. As can be seen there is a difference in performance of 0.15 dBs between OMO-MSA with 48 and OMO-MSA with 15 iterations and 0.2 dBs between OMO-MSA with 48 and OMO-MSA with 10 iterations at BER 10^{-5}. Between 48 and 5 iterations it can be observed a difference of 0.5 dBs at BER 10^{-4}.

Finally, the (16129, 15372) LDPC code with $d_c = 127$ and $d_v = 6$ proposed in [28] is analyzed. This code has a special interest due to its very high rate and its good behaviour in the error floor region, which is remarkable compared to the rest of high-rate codes. The implemented emulator that includes the HDL implementation of the decoder showed how our approximation and the initial hypothesis is confirmed as the OMO decoder does not introduce any performance degradation in the error floor region, Fig. 2.17. In addition, the performance loss in the waterfall is almost negligible (0.1 dB).

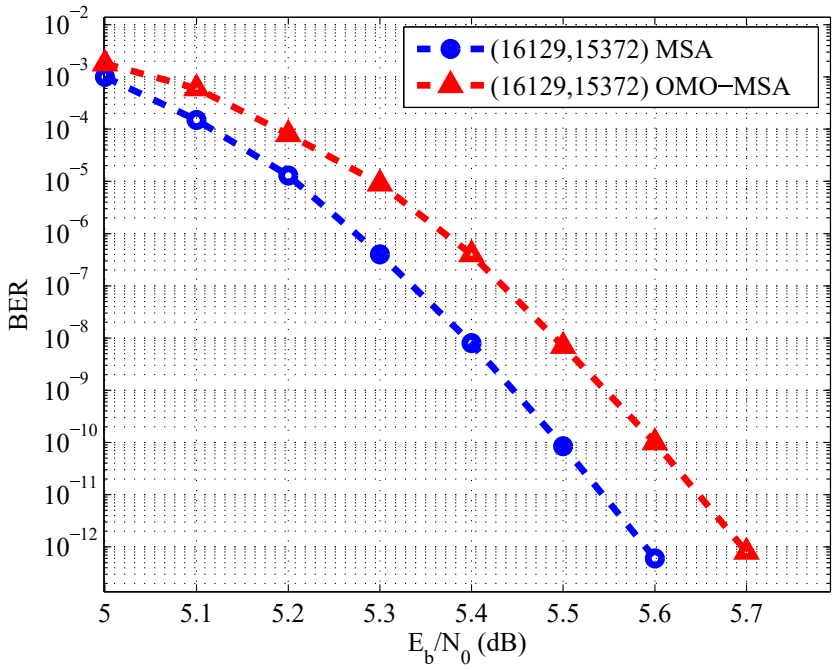

Figure 2.17: BER performance of MSA and OMO-MSA for the (16129,15372) LDPC codes.

On the other hand, another issue that was taken into account is that working with finite precision there is more probability that many values are repeated, so more than one min_1 can be found among the input messages. This will affect to the performance of the decoder. In order to test different scenarios we performed the following test if more than one min_1 was detected:

1. Test called OMO1. We assigned the min_1 value to all the outputs. In this test we considered that there was not enough precision to detect the least reliable element and, therefore, we assumed that all the incoming messages were unreliable.

2. Test called OMO2. We assigned the min_2 value to all the outputs. In this case we considered that there was enough precision to detect the least reliable element, so we assumed that all incoming messages were reliable. In Fig. 2.18, the results of this test are worse in the waterfall region due to the fact that we assumed that with several unreliable messages in the input, reliable outputs were obtained.

3. Test called OMO3. We assigned the min_1 value if the corresponding input has a value different to min_1 and we assigned the min_2 value if the corresponding input was equal to min_1. In this third test, we considered that there was an intermediate precision to detect the least reliable element, therefore, we assumed that incoming messages could be as reliable as unreliable.

Finally, Test 1 (OMO1) was chosen because of its better performance in the waterfall region as in the error-floor region Fig. 2.18, and because of the small changes in the check node architectures.

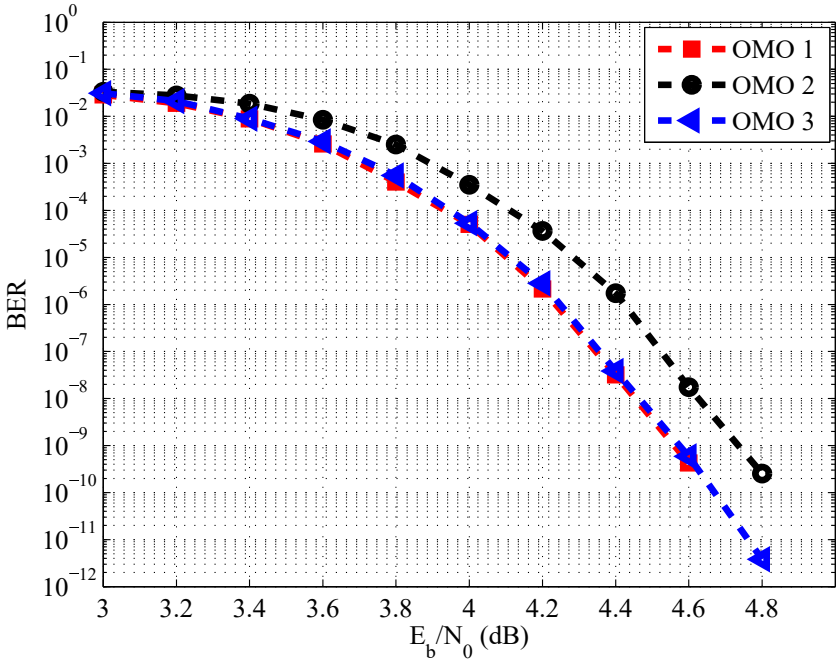

Figure 2.18: BER performance of OMO 1, OMO 2 and OMO 3 test for the (2048,1723) LDPC codes.

2.5. Hardware results and comparisons

The proposed OMO-MSA was implemented following the layered architecture from [23] for the (2048, 1723) LDPC code and the (16129,15372) LDPC code. As both codes have a $d_v = 6$, the layered architecture was dimensioned to compute one iteration in 6 clock cycles. It means that 64 and 762 check nodes were parallelized for the (2048,1723) code and (16129,15372) code, respectively. VHDL was used for the description of the hardware. Cadence RTL Compiler and SOC encounter tools were employed for synthesis and place and route, respectively, with a 90 nm CMOS process of nine layers with standard cells and operating conditions of 25°C and 1.2 V. The obtained results are summarized in Table 2.1 together with the most efficient MSA decoders found in literature, implemented for the same codes and under the same technology.

Decoder	Area (mm^2)	Frequency (MHz)	Throughput (Gbps)	Ratio throughput/ area (Gbps/mm^2)
Proposed OMO decoder	3.15	132	11.2	3.55
Variable-weight MSA [23]	3.84	226	12.8	3.34
Split-row MSA [24]	9.68	156	29.64	3
MSA [23]	4.84	121	7.1	1.47
MSA [25]	5.35	137	7.79	1.45
MSA [26]	9.2	348	11.85	1.28

Table 2.1: ASIC implementation results of OMO and different MSA decoders for the (2048, 1723) LDPC code.

For the (2048,1723) LDPC, results show how the new approximation obtains the smallest area in all cases. Compared to other approximations of the second minimum [23] we can see that our proposal saves 18% of area due to the fact that it does not require to store and control the use of different correction factors for the different range iterations. If we compare to regular MSA decoders, area is reduced between 41% [25] and 66% [24] depending on the degree of parallelism of the decoder. This work reaches similar throughput to the approximation introduced in [23] and the decoder from [26]; and improves by 44% the throughput of [25].

In terms of efficiency, measured as throughput/area, our proposal is slightly more efficient than the approximation from [23] and it does not require a complex optimization of parameters that changes with iterations or with the signal-to-noise ratio, so it reduces the time of design without sacrificing hardware advantages. On the other hand, the OMO-MSA increases between 18% [24] and 177% [26] the efficiency of the MSA decoder, depending on the design.

If we compare exactly the same decoder with and without the approximation (MSA decoder from [23]), we can see that the area saving is about 35% and the maximum frequency is increased about 10%, because of the reduction of the critical path in the minimum finder. The throughput is also increased, making the total efficiency in terms of throughput/area ratio two times larger by introducing this novel estimation of the second minimum in the decoder.

Table 2.2 shows the results of the implementation of the same layered decoder using MSA and the proposed OMO check node for the (16129,15372) LDPC. It can be seen how the same decoder with conventional MSA check node has an area overhead of 10.8% compared to the OMO decoder. In addition, frequency is increased in 28.9% and the total throughput is improved in a 28.9%. This proposal is 1.736 times smaller than previous decoders and 1.44 times more efficient than MSA [23] . It is important to remark that simulations showed that this decoders does not introduce any performance degradation for BER $> 10^{-16}$ and the coding gain in the waterfall is almost the same.

Decoder	Area (mm^2)	Frequency (MHz)	Throughput (Gbps) (15it)/(10it)/(5it)	Ratio throughput/ area (Gbps/mm^2)
OMO-MSA	14.173	48	8.6/12.9/25.8	0.6067
MSA	15.909	37.23	6.67/10/20	0.4192

Table 2.2: ASIC implementation results of decoders for the (16129, 15372) LDPC code.

Finally, it is worth mentioning that the proposed approximation can be applied to other architectures.

2.6. Conclusions

In this chapter a new approximation to estimate the second minimum of an LDPC check node processor is introduced. This algorithm reduces not only the complexity of the check node update hardware but also the complexity of the design parameters compared to other lossless approximation. This proposal makes independent the required correction coefficients of the iterations or the signal-to-noise ratio, allowing the decoder to automatically adjust the second minimum estimator without adding extra complexity. The derived architecture improves the efficiency of the MSA by 18% in terms of throughput/area ratio.

Chapter 3

Modified-Optimized 2-Bit Min-Sum Algorithm (MO2-BIT-MSA

3.1. Introduction

In the previous chapter it was concluded that the difference between min_1 and min_2 is more important than the absolute value of the magnitude. Taking into account this fact, in this chapter min_1 and min_2 values are estimated with the smallest number of quantization bits. The complexity of the decoding process and the wiring congestion will be reduced with respect to the two main algorithms, SPA and MSA, when the number of quantized bits are reduced, as will be seen in the next section.

The smallest number of bits to quantize the value of the min_1 and min_2 is one bit, but this case corresponds to a hard decision algorithm which will be analyzed in the next chapter. Therefore, for the soft-decision algorithm we are going to address in this chapter, the number of bits to quantize these two values is 2 bits.

The two main methods of quantization are: uniform and non-uniform. The first one uses the same distance between the reconstruction levels and it does not make any assumption about the nature of the signal to be quantized, resulting in a large number of quantization errors for soft-decoding when the number of levels is small. As regarding the second one, the distance between reconstruction levels is not always the same, this is, it is a non-linear quantification. This method of quantification studies the nature of the signal to obtain an optimal quantification and to minimize the errors. So under the same number

of bits, non-uniform quantification achieves better performance than uniform quantification for soft-decision decoders, as will be shown later.

As in this case our objective is to distinguish the less reliable input values from the most reliable ones, a threshold of reliability is defined. To avoid losing accuracy of information and, therefore, to minimize errors in the system, non-uniform quantification is the most appropiate method in this case.

To the best knowledge of the authors the algorithm O2-BIT-MSA proposed in [18], summarized in Chapter 1, is the most efficient one to reduce the quantization to two bits. However, it has two weak points. The first and main weakness of this algorithm is that the BER performance is deteriorated at high SNR, which results in an early error floor. The second drawback is that the algorithm has several interrelated parameters and this fact makes its design difficult and time consuming. In this chapter we propose a modification of this algorithm that we called Modified Optimized 2-bit Min-Sum Algorithm (MO2-bit-MSA). The new algorithm has a BER performance very close to the MSA and maintains its low complexity. On the other hand, a method to simplify the selection of the parameters and to reduce the design time is proposed. Finally, to show the advantage of its low complexity, several hardware architectures were implemented in FPGA and a 90 nm CMOS process and compared with MSA implementations.

3.2. Channel quantization

The performance of the decoder is directly proportional to the amount of input information (LLR). The loss of information (accuracy) due to the quantization can lead to the high occurrence of less reliable input values, degradating the performance of the system.

In [18], the input messages from the channel y (LLR) are quantized to a value that can be coded with 2 bits depending on a input threshold T_y, as seen in Eq. 3.1:

$$b_s b_m = \begin{cases} 01, & if \quad y > T_y \\ 00, & if \quad T_y \geq y \geq 0 \\ 10, & if \quad 0 > y \geq -T_y \\ 11, & if \quad y < -T_y \end{cases} \tag{3.1}$$

where bit b_s corresponds to the sign of the input message y and bit b_m is the result of the comparation with the threshold T_y, which means that $b_m = 1$ is a high reliable bit and $b_m = 0$ is a low reliable one. Therefore, the choice of a reliable or less reliable value depends on T_y, which can result in a worse performance in the case of not having enough accuracy.

With the aim of improving the performance of [18], the first step is to remove the non-uniform 2-bit input quantification and this allows more input accuracy as shown in Algorithm 7.

Algorithm 7 1st modification of Optimized 2 bit Min-Sum decoding algorithm.

Input : $Q_n^{(0)} = Ln$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update
1 $R_{m,n}^{(i)} = \prod_{n' \in N(m) \backslash n} sign(Q_{n',m}^{(i-1)}) \min_{n' \in N(m) \backslash n} |Q_{n',m}^{(i-1)}|$
Variable-node update
2 $Q_n^{(i)} = Q_{m,n}^{(i)} = g(L_n + \alpha \sum_{m \in M(n)} f(R_{m,n}^{(i)}))$
Tentative decoding
3 $\hat{x}_n^{(i)} = \begin{cases} 1, & Q_n^{(i)} < 0, \\ 0, & Q_n^{(i)} \geq 0, \end{cases} \quad n \in \{0, ..., N-1\}$
4 $s_m = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_n^{(i)} h_{m,n} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_n^{(i)}, \quad m \in \{0, ..., M-1\}$
if $(s^{(i)} = 0)$ **then** {SKIP}
end
Output : $\hat{x}$

3.3. Optimization of parameters

As we saw in the introduction of the Optimized 2-bit min-sum decoding algorithm of chapter 1 or in [18], the coding gain of this algorithm depends on the values of five parameters (W_H, W_L, T_L, T_y and α). In the previous subsection it was concluded that with the elimination of the input threshold T_y, it was possible to improve the performance, therefore, only four of the previous five parameters can be taken into account.

After the analysis of the relation among these four parameters, we conclude that only the parameters W_H, W_L and T_L depend on each other, whereas the value of α does not depend on the previous parameters, since it depends on the relation of proportionality between the channel messages and the internal messages interchanged in the decoding process.

Although the number of dependent parameters was reduced, the selection of the optimal values for these three interrelated parameters is a difficult and time-consuming process that involves many simulations, which are done as follows. The values of one parameter are swept and the other two are kept fixed, then the

results are observed, and the process is repeated for the remaining parameters until an optimal combination is found.

To avoid this time-consuming process and to facilitate the implementation of this algorithm with different LDPC codes, a method to select the values of the parameters and to reduce the amount of simulations is proposed in this thesis.

From the Algorithm 7, we call TR_n (Total Reliability of a VN) to the term obtained by adding the reliability of all the messages taking into account their signs:

$$TR_n = \sum_{m \in M(n)} f(R_{m,n}^{(i)}), \tag{3.2}$$

where the $f(\cdot)$ function was defined in Chapter 1. Let's suppose that the target code is the (2048,1723) RS-LDPC with $d_v = 6$ and $d_c = 32$ adopted for 10GBASE-T. As a VN processes only $d_v = 6$ messages and each message has a possible reliability value of W_H or W_L, where $W_H > W_L$, there is a finite set of possible total reliability values TR_n. From this set, we take the subset that accomplishes $TR_n = |xW_H \pm yW_L|$ with $x + y = d_v = 6$ (x and y are natural numbers), as the values with highest and lowest reliability are included and is big enough to represent the behaviour of the complete set. This subset is enumerated in Table 3.1 for the case of the (2048,1723) LDPC code.

Total reliability in one VN
$\|6W_H\|$
$\|5W_H \pm 1W_L\|$
$\|4W_H \pm 2W_L\|$
$\|3W_H \pm 3W_L\|$
$\|2W_H \pm 4W_L\|$
$\|1W_H \pm 5W_L\|$
$\|6W_L\|$

Table 3.1: Possible reliability values obtained by adding the reliability of each message in a variable node of $d_v = 6$.

In order to guarantee that the decoding process works properly, the combination of messages more reliable must have high reliability. Thus, the values of W_H and W_L should accomplish the following conditions:

1. $|6W_H|$ is the highest reliability value.
2. $|1W_H - 5W_L|$ is the lowest reliability value.
3. $|aW_H \pm yW_L| > |bW_H \pm yW_L|$ if $a > b$, $a + y = b + y = 6$,

where a and b are natural numbers. Fig. 3.1 shows graphically how the subset TR_n must be sorted according the previous conditions. As can be seen in the right-hand side column of Fig. 3.1, there are several values of TR_n whose relative order is impossible to know *a priori* (*e.g.* $|2W_H - 4W_L|$, $|1W_H + 5W_L|$ and $|6W_L|$).

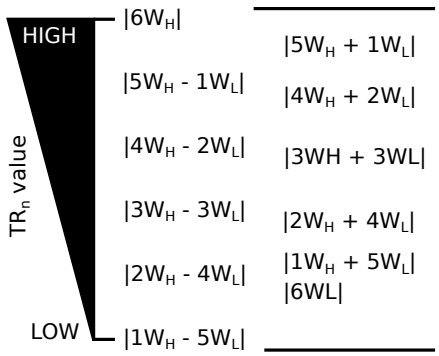

Figure 3.1: Sorted reliability values of the selected subset of TR_n.

In order to check the values of W_H and W_L that result in a sorted subset as indicated previously, different cases were analyzed assuming $W_L = 1$ and with W_H from $W_H = 2W_L$ to $W_H = 7W_L$. The sorted subset of each case is shown in Table 3.2. Note that it was concluded that the difference between min_1 and min_2 is more important than the absolute value of their magnitude, so to fix the parameter W_L to 1 will not incur any performance penalization. On the other hand, the parameter α will be used to stablish a proportionality between the magnitudes of the incoming channel and the internal messages.

Analyzing the different cases, it is observed in Table 3.2 that the cases 2 and 3 do not accomplish the established conditions. In case 2, $|2W_H - 4W_L| < |W_H - 5W_L|$ and, in case 3, $|2W_H - 4W_L| = |W_H - 5W_L|$. So these cases are discarted. On the other hand, Tabla 3.2 guide us to choose the value for the threshold T_L. As can be seen, it has to be decided how to divide the table in two parts to generate the outputs of the variable node, which is composed of two bits: the sign bit and the reliability bit. This is the task performed by function $g(\cdot)$ explained in Section 1.4.3. The value of the reliability bit is chosen as follows: 0 for the combinations with reliability lower than T_L, and 1 otherwise.

From now on, several simulations were performed to find the best cases and it was found that the best combination was $W_H = 5W_L$ with $|2W_H - 4W_L|$ as the limit stablished decides the output reliability, as shown in Fig. 3.2.

Reliability order	x	$\pm y$	case 7 W_H=7W_L	case 6 W_H=6W_L	case 5 W_H=5W_L	Reliability bit of $g(\cdot)$
11 (highest)	6	0	42	36	30	1
10	5	1	36	31	26	1
9	5	-1	34	29	24	1
8	4	2	30	26	22	1
7	4	-2	26	22	18	1
6	3	3	24	21	18	1
5	2	4	18	16	14	1
4	3	-3	18	15	12	1
3	1	5	12	11	10	1
2	0	6	6	6	6	0
1	2	-4	10	8	6	0
0 (lowest)	1	-5	2	1	0	0

Reliability order	x	$\pm y$	case 4 W_H=4W_L	case 3 W_H=3W_L	case 2 W_H=2W_L	Reliability bit of $g(\cdot)$
11 (highest)	6	0	24	18	12	1
10	5	1	21	16	11	1
9	5	-1	19	14	9	1
8	4	2	18	14	10	1
7	4	-2	14	10	6	1
6	3	3	15	12	9	1
5	2	4	12	10	8	1
4	3	-3	9	6	3	1
3	1	5	9	8	7	1
2	0	6	6	6	6	0
1	2	-4	4	2	0	0
0 (lowest)	1	-5	1	2	3	0

Table 3.2: TR_n values and output reliability bit at variable node for different cases.

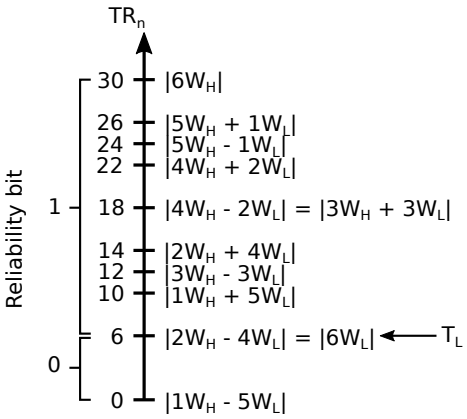

Figure 3.2: Graphical representation of the boundary between the two output reliability bit values at variable node.

Therefore, to obtain the optimal value of each of the 3 constants: W_H, W_L and T_L, we have the following equations:

$$W_L = 1, W_H = 5W_L = 5 \tag{3.3}$$

$$T_L = (2W_H - 4W_L) \cdot \alpha \tag{3.4}$$

In Fig. 3.3 BER performance of O2-BIT-MSA from [18] and O2-BIT-MSA with optimum parameters are plotted for the (2048,1723) RS-LDPC code with 48 iterations. Applying the above equations, the optimum parameters are: $W_H = 5$, $W_L = 1$, $T_L = 3$ and $\alpha = 0.5$.

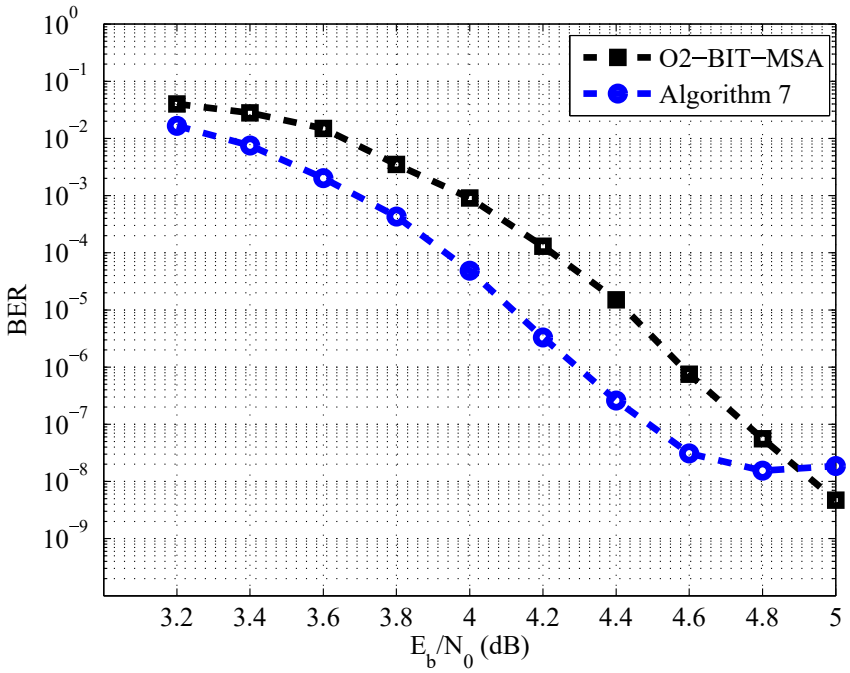

Figure 3.3: Original O2-BIT-MSA with $T_y = 3/8$ and algorithm 7 with optimum parameters BER performance. $W_H = 5$, $W_L = 1$ and $T_L = 3$.

As can be seen, the proposed modification described in Algorithm 7 using the optimized parameters exhibits a 0.3 dBs of coding gain with respect to the O2-BIT-MSA. On the other hand, it also can be seen that the O2-BIT-MSA exhibits worse error correction performance at high E_b/N_0 values: the error floor appears before and more strongly manifested. This problem will be solved introducing extrinsic information on the algorithm, as explained in the next section.

3.4. Inclusion of the extrinsic information

As explained in Section 1.2 of Chapter 1, in the process of message passing, when the information of one CN or VN is updated, it is avoided to use its own messages (intrinsic information) from the previous iterations to eliminate the possibility of feeding back any error introduced by this node.

Analyzing the proposal of [18], it is observed that when VNs update their messages (Eq. 3.5), each message sent to the CNs takes into account the intrinsic information from the previous iterations. Therefore, the VNs feed back any previously produced error, which results in a deterioration of performace.

$$Q_{mn}^{(i)} = g(L_n + \alpha \sum_{m \in M(n)} f(R_{mn}^{(i)})) \tag{3.5}$$

In order to avoid this situation, another modification of the original algorithm is proposed. It consists of updating the messages taking into account only the incoming messages from the neighbours, *i.e.* using only extrinsic information. This modification is named MO2-BIT-MSA and in Algorithm 7, the Eq. 3.5 is now rewritten as shown in Eq. 3.6.

$$Q_{mn}^{(i)} = g(L_n + \alpha \sum_{m' \in M(n) \setminus m} f(R_{m'n}^{(i)})) \tag{3.6}$$

Therefore, the MO2-BIT-MSA algorithm, which includes the extrinsic information and does not use the T_y threshold, is reformulated as shown in Algorithm 8.

Algorithm 8 Modified Optimized 2 bit Min-Sum decoding algorithm.

Input : $Q_n^{(0)} = Ln$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update
1 $R_{m,n}^{(i)} = \prod_{n' \in N(m) \setminus n} sign(Q_{n',m}^{(i-1)}) \min_{n' \in N(m) \setminus n} |Q_{n',m}^{(i-1)}|$
Variable-node update
2 $Q_{m,n}^{(i)} = g(L_n + \alpha \sum_{m' \in M(n) \setminus m} f(R_{m',n}^{(i)}))$
Tentative decoding
3 $Q_n^{(i)} = g(L_n + \alpha \sum_{m' \in M(n)} f(R_{m,n}^{(i)}))$
4 $\hat{x}_n^{(i)} = \begin{cases} 1, & Q_n^{(i)} < 0, \\ 0, & Q_n^{(i)} \geq 0, \end{cases} \quad n \in \{0, ..., N-1\}$
5 $s_m = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_n^{(i)} h_{m,n} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_n^{(i)}, \quad m \in (0, ..., M-1)$
if $(s^{(i)} = 0)$ **then** SKIP
end
Output : $\hat{x}$

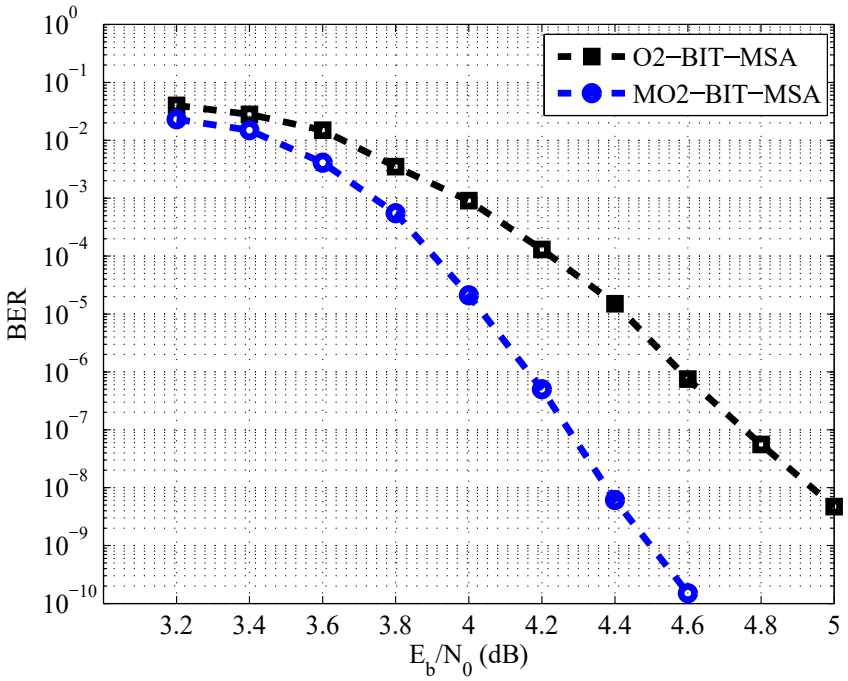

Figure 3.4: BER performance of O2-BIT-MSA with $T_y = 3/8$ and MO2-BIT-MSA. $W_H = 5$, $W_L = 1$ and $T_L = 3$.

Fig. 3.4 shows the BER performance of the original O2-BIT-MSA and MO2-BIT-MSA. Both of them are simulated using the optimum parameters obtained in previous sections. The code used for this simulation is the (2048,1723) RS-LDPC with $d_v = 6$ and $d_c = 32$ using a maximum number of iterations of 48 as in [18]. Original O2-BIT-MSA from [18] was simulated using the message from channel as input instead of using LLRs. On the other hand, MO2-BIT-MSA was simulated using the LLRs as input, since as our goal is to let the greater accuracy at the input, the input was not limited to the range -1, 1 from the BPSK modulation. As can be seen in Fig. 3.4, the error correction performance of MO2-BIT-MSA with $W_H = 5$, $W_L = 1$ and $T_L = 3$ is substantially improved. It outperforms in 0.6 dBs the performance of O2-BIT-MSA at BER 10^{-8}.

3.5. Error correction performance comparison with MSA

Fig. 3.5 shows the BER performance of the MO2-BIT-MSA and MSA with an scaling factor α (αMSA). The code used for this simulation is the quasi-cyclic (2048,1723) RS-LDPC with $d_v = 6$ and $d_c = 32$ adopted for 10GBASE-T. BPSK modulation and AWGN channel are assumed and the maximum number of iterations used is 48. These simulations were obtained using the FPGA-based emulator [27] in order to see how the proposed algorithm behaves at very high E_b/N_o values.

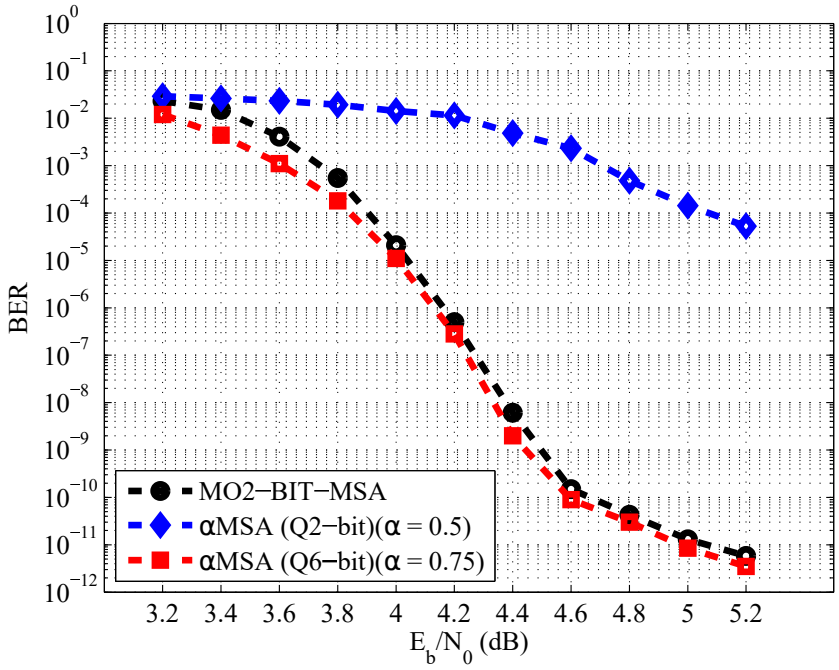

Figure 3.5: MO2-BIT-MSA, MSA (2-bit) and MSA (6-bit) BER performance for the (2048,1723) LDPC codes. $W_H = 5$, $W_L = 1$ and $T_L = 3$.

As can be seen in Fig. 3.5, if the MSA is quantified with only 2 bits to reduce its complexity, its performance is drastically deteriorated, and it loses nearly all its correction capability. On the other hand, the proposed algorithm has a behaviour similar to the scaled MSA with 6 bits, with a performance loss lower than 0.01 dBs. Futhermore, it follows the same error-floor pattern as the MSA, *i.e.* both error-floors appear at a BER 10^{-10}.

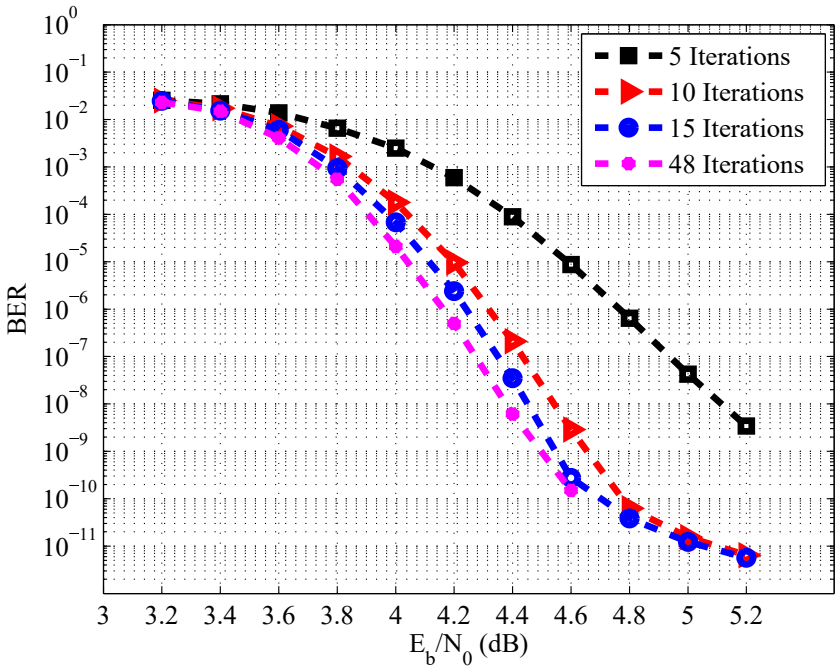

Figure 3.6: MO2-BIT-MSA BER performance at 5, 10, 15 and 48 iteration for the (2048,1723) LDPC code.

In Fig. 3.6 the BER performance of MO2-BIT-MSA with 5, 10, 15 and 48 iterations is represented. As can be seen, MO2-BIT-MSA with 15 and 10 iterations are very close to the MO2-BIT-MSA obtained with 48 iterations, with a difference of 0.1 dBs and 0.2 dBs, respectively, at BER $= 10^{-7}$. It can be observed a performance loss of 0.7 dBs between MO2-BIT-MSA with 48 and 5 iterations at BER 10^{-7}.

The MO2-BIT-MSA was also tested for the (16129,15372) LDPC code. Fig. 3.7 shows its BER performance with $W_H = 7$, $W_L = 1$ and $T_L = 3$, together with the one of the MSA and OMO-MSA for the same code. It can be seen that it exhibits a performance loss of only 0.2 dBs for BER $= 10^{-12}$ with respect the MSA and, despite its low number of bits in the interchanged messages, it does not degradate the performance at high SNR.

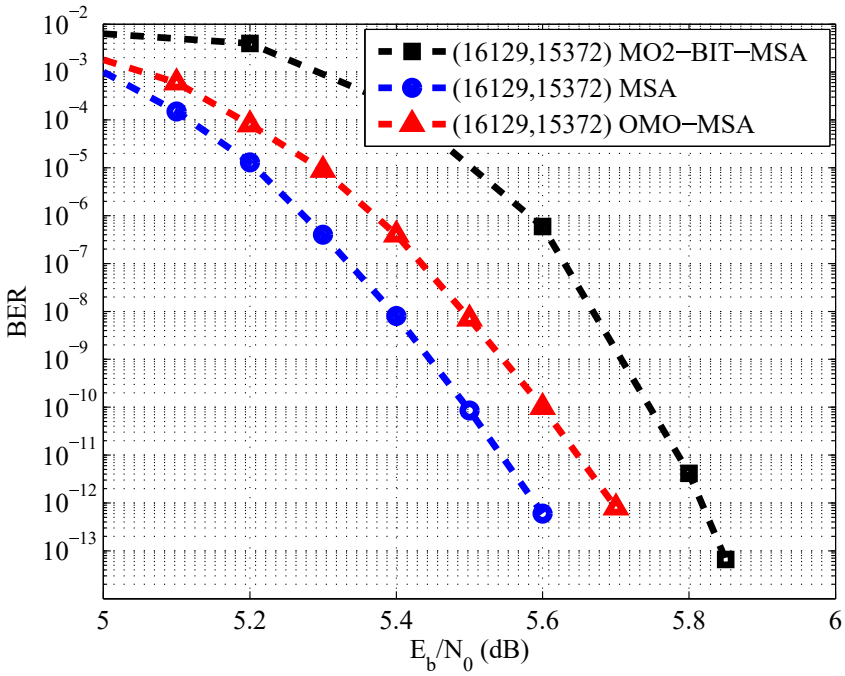

Figure 3.7: BER performance of MSA, OMO-MSA and MO2-BIT-MSA with $W_H = 7$, $W_L = 1$ and $T_L = 3$ for the (16129,15372) LDPC codes.

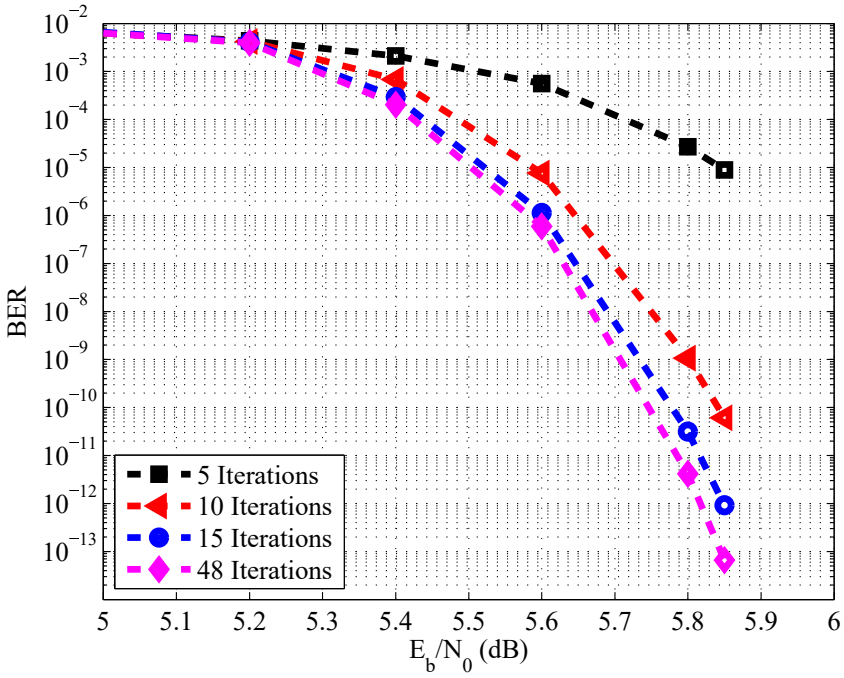

Figure 3.8: MO2-BIT-MSA BER performance at 5, 10, 15 and 48 iteration for the (16129,15372) LDPC code.

As can be seen in Fig. 3.8 a performance loss of less than 0.05 dBs at BER of 10^{-12} and less than 0.1 dBs at BER 10^{-10} is obtained with 15 and 10 iterations, respectively, compared to MO2-BIT-MSA with 48 iterations. The performance difference between MO2-BIT-MSA with 5 iterations and MO2-BIT-MSA with 48 iterations is 0.35 dBs at BER of 10^{-5}.

3.6. Modified Optimized 2-bit MSA architecture

For the hardware implementation of MO2-BIT-MSA, first a fully-parallel architecture was implemented, then, the pipeline interleaved technique was applied to process several codes in parallel.

3.6.1. MO2-BIT-MSA fully-parallel architecture

A fully-parallel architecture implements M CNUs and N VNUs connected as indicated by the Tanner graph of the code parity matrix. Due to the fact that M and N are usually big numbers, a careful design of these blocks have to be done, minimizing their hardware components, as they will be multiplied by M or N if belong to the CNUs or VNUs, respectively. In this section the design of a fully-parallel decoder architecture suitable for FPGA implementation for the (2048,1723) RS-LDPC code is detailed, although it can be easily generalized for other codes. This code has a parity matrix of $M \times N = 384 \times 2048$, and the degrees of variable and check nodes are $d_v = 6$ and $d_c = 32$, respectively.

The designed architecture uses the half-broadcast technique from [31] applied to the CNUs. It reduces the number of wires that are sent from the CNUs to the VNUs. So, instead of sending different messages from a CNU to each VNU, only one message (composed of sign, min_1 and min_2) is sent, *i.e.* only 3 bits are delivered from each CNU. On the other hand, the VNUs have to store their own messages, which have to be used to determine if the incoming messages are the min_1 or min_2.

The scheme of a CNU is depicted in Fig. 3.9. CNU performs very simple logic operations due to the fact that the messages have only 2 bits: the sign bit and the reliability bit. This unit is only composed of a tree of 1-bit logic functions ($xors$) to compute the parity check of the 32 (d_c) input signs and a tree of 6-input and 2-output logic functions to compute min_1 and min_2 from 32 (d_c) reliability input bits. It was decided to divide the function to calculate the min_1 and min_2 into 6-input functions, to match with the LUT size of the FPGA device used in the implementation. The pipeline registers are drawn as dashed lines in the scheme.

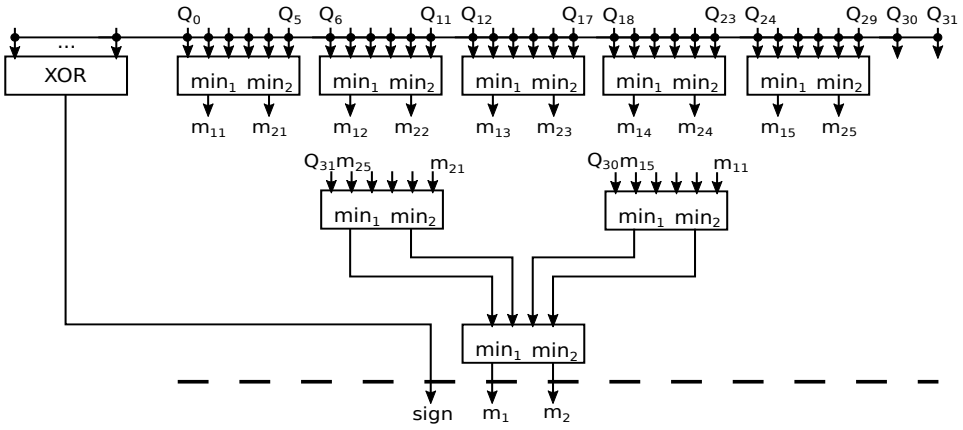

Figure 3.9: MO2-BIT-MSA CNU architecture.

The VNU has to compute $d_v = 6$ output messages and a decision bit according the following equations (which are rewritten here for the sake of readability):

$$Q_n^{(i)} = g(L_n + \alpha \sum_{m \in M(n)} f(R_{m,n}^{(i)})) \tag{3.7}$$

$$Q_{m,n}^{(i)} = g(L_n + \alpha \sum_{m' \in M(n) \backslash m} f(R_{m',n}^{(i)})), \tag{3.8}$$

being the function $g(\cdot)$:

$$g(x) = \begin{cases} b_s = sign(x) \\ b_m = \begin{cases} 1, & if|x| > T_L \\ 0, & otherwise \end{cases} \end{cases}$$

Fig. 3.10 shows the scheme of an VNU architecture that implements the previous equations. The first stage is composed of 6 (d_v) 5-input LUTs (CSF), which perform the compare-and-select operations, required to decide if the incoming message reliability value is the min_1 or min_2, together with the application of the $f(\cdot)$ function, shown in Table 3.3. The output of the CSF LUTs are added to obtain the total reliability (TR_n) of the VNU. The extrinsic values are computed subtracting the output of each CSF from the TR_n value. Then, they are scaled by α, added to the LLR value and processed by the $g(\cdot)$ function. On the other hand, the TR_n value is scaled by α and added to the LLR input to obtain the decision bit. Finally, 6 (d_v) multiplexers are used to propagate the information of the LLR in the initialization process, without taking into account the initial values of the $f(\cdot)$ and $g(\cdot)$ functions. As can be seen in Fig. 3.10, the VNU cell requires $3 \cdot d_v = 18$ adders, $(d_v + 1) = 7$ multipliers by a constant value, $d_v = 6$ $f(\cdot)$ fucntions, $(d_v + 1) = 7$ $g(\cdot)$ functions and $d_v = 6$ 2-bit 2-to-1 multiplexers. Its critical path is 1 $f(\cdot)$ function, 5 adders, 1 multiplier by a constant and 1 multiplexor.

$f(\cdot)$		$g(\cdot)$	
Message from memory	Integer for addition	Addition result	Message to memory
00	W_L	$T_L > x \geq 0$	00
01	W_H	$x \geq T_L$	01
10	$-W_L$	$0 > x > -T_L$	10
11	$-W_H$	$x \leq -T_L$	11

Table 3.3: Function $f(\cdot)$ and function $g(\cdot)$.

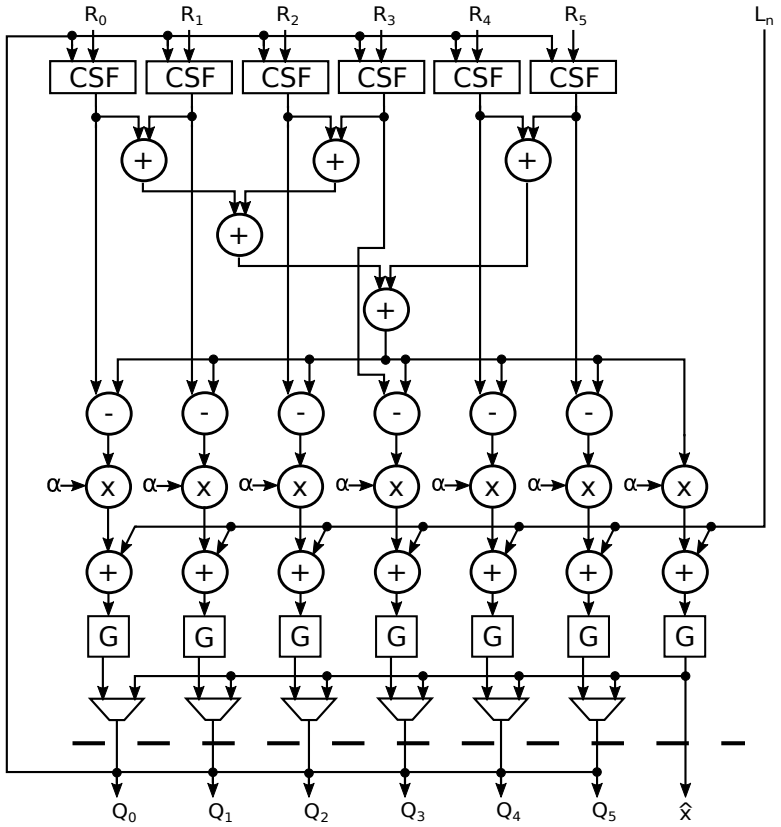

Figure 3.10: MO2-BIT-MSA VNU architecture.

The complexity of the VNU can be reduced if the equations 3.7 and 3.8 are rewritten taking α as common factor:

$$Q_n^{(i)} = g(\alpha(\frac{L_n}{\alpha} + \sum_{m \in M(n)} f(R_{m,n}^{(i)}))) \tag{3.9}$$

$$Q_{m,n}^{(i)} = g(\alpha(\frac{L_n}{\alpha} + \sum_{m' \in M(n) \setminus m} f(R_{m',n}^{(i)}))), \tag{3.10}$$

On the other hand, the common factor α can be removed if the function $g(\cdot)$ is modified scaling by α the threshold value of T_L, as shown in the final equations:

$$Q_n^{(i)} = g^{'}(\frac{L_n}{\alpha} + \sum_{m \in M(n)} f(R_{m,n}^{(i)})) \tag{3.11}$$

$$Q_{m,n}^{(i)} = g^{'}(\frac{L_n}{\alpha} + \sum_{m' \in M(n) \backslash m} f(R_{m',n}^{(i)})), \tag{3.12}$$

where the new function $g'(\cdot)$ is defined as:

$$g^{'}(x) = \begin{cases} b_s = sign(x) \\ b_m = \begin{cases} 1, & if|x| > \frac{T_L}{\alpha} \\ 0, & otherwise \end{cases} \end{cases}$$

The new implementation scheme of the VNU is shown in Fig. 3.11. As can be seen, now it requires $2 \cdot d_v = 12$ adders, 1 multiplier by a constant value, $d_v = 6$ $f(\cdot)$ functions, $(d_v + 1) = 7$ $g(\cdot)$ functions and $d_v = 6$ 2-bit 2-to-1 multiplexers. So, d_v adders and d_v multipliers by constant are saved. Futhermore, the critical path is reduced in one adder stage and 1 multiplier by a constant.

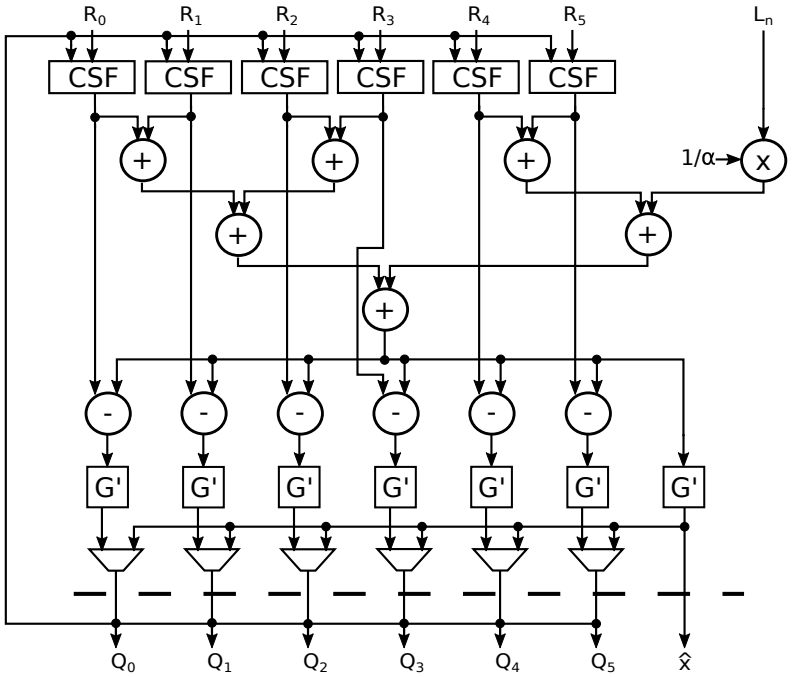

Figure 3.11: MO2-BIT-MSA VNU architecture.

Finally, as the latest FPGA devices (Xilinx Serie-7 and Altera Cyclone IV and Arria devices) allow the implementation of n-bit ternary adders ($S = A + B + C$) using n LUTs and the carry-propagation logic (*i.e.* the same hardware cost of a n-bit 2-operand adder), the scheme of Fig. 3.11 can be optimized for FPGA implementation regrouping the additions in groups of three, as shown in Fig. 3.12. In this case, the VNU saves 3 adders and reduces the critical path in one adder stage with respect to the VNU of Fig. 3.11. This is the scheme used in the implemented architecture.

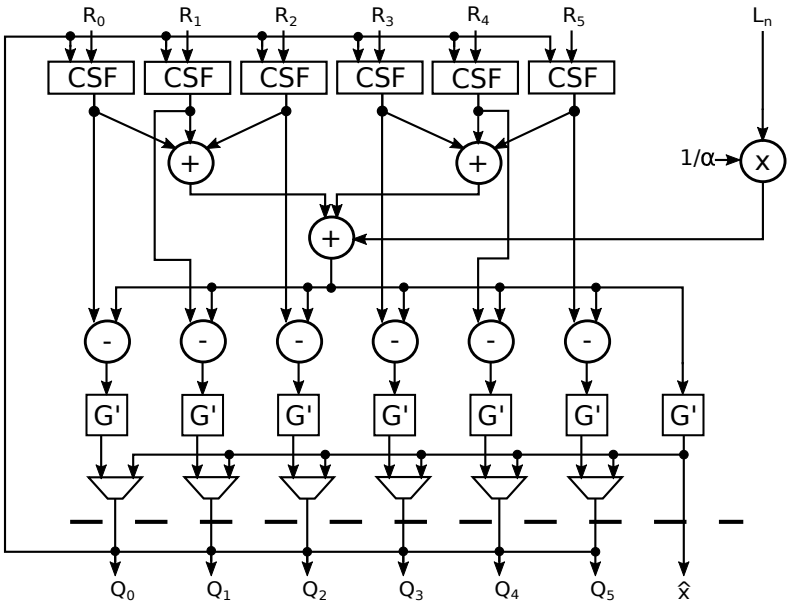

Figure 3.12: MO2-BIT-MSA VNU architecture.

3.6.2. MO2-BIT-MSA fully-parallel pipeline interleaved

As discussed in Section 1.2, Chapter 1, the flooding schedule first updates all the VNs and, then, all the CNs. The parallel architecture presented in the previous section only requires to have registered the outputs of the VNUs, so each iteration of the algorithm is computed in one clock cycle (this case is represented in Fig. 3.13.a, where the thick black line represents the registers). If both, the VNUs and CNUs, are implemented with registered outputs (as in Fig. 3.13.b), each iteration is computed in two clock cycles: in a clock cycle only all the CNUs are enabled and all the VNUs are disabled not performing any computation, or *vice versa*. This extra pipeline stage included in the CNUs outputs does not involve any throughput advantage: the working clock frequency could be as much doubled (this term is in the numerator of the throughput equation Eq. 3.13), but it is completely compensated by the two clock cycles required by each iteration (which is in the denominator of the throughput equation). However, these clock cycles of inactivity of each processors can be exploited by decoding 2 independent received frames (codes) at the same time.

This technique, called pipeline interleaved, inserts registers into the architecture with the objective to execute multiple processes or threads concurrently like

multithreading technique in computer programming. In this case this technique is used to decode several codes concurrently, without interference between them.

The schemes to process 2 and 4 interleaved codes are shown in Fig. 3.13.b and Fig. 3.13.c, respectively. The architecture to decode 2 codes requires to register the outputs of all VNUs and CNUs, while the architecture to decode 4 codes has also the inputs of all VNUs and CNUs registered. Note that it could seem that the delays among registers are not equalized in the topology to decode 4 codes of Fig. 3.13.c, as the paths between VNUS outputs and CNUs inputs or between CNUs outputs and VNUs inputs are only composed of wires and not involve any computation. However, due to the very high wiring congestion between CNUs and VNUs, their propagation delays can be as long as the ones of the CNUs and VNUs.

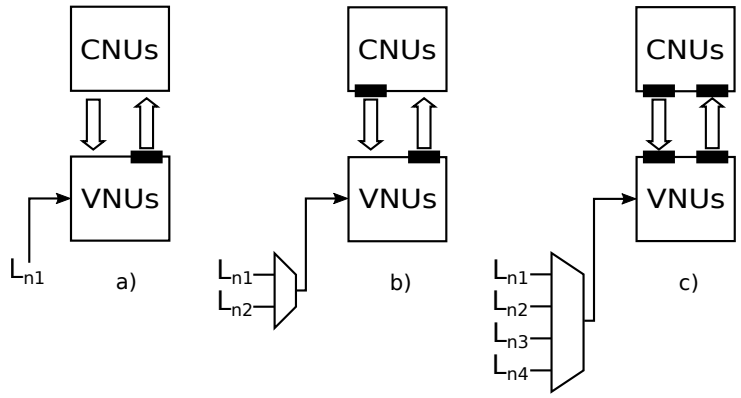

Figure 3.13: MO2-BIT-MSA fully-parallel pipeline interleaved. a) without pipeline interleaved. b) pipeline interleaved for 2 codes. c) pipeline interleaved for 4 codes.

In the case of processing 2 codes concurrently, initially code 1 ($C1$) enters into the VNU to be processed, at the next clock cycle $C1$ enters into the CNU and the code 2 ($C2$) enters into the VNU. So while a code is being processed in one unit (CNU or VNU) the other is processed in the other unit (CNU or VNU) as shown in Fig. 3.14:

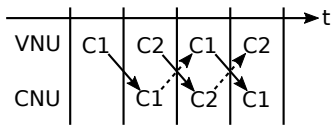

Figure 3.14: MO2-BIT-MSA fully-parallel pipeline interleaved for 2 codes.

In the case of processing 4 codes concurrently, the process is as follows. In the first clock cyle, $C1$ is processed in the VNU, next $C1$ waits for one clock cycle while $C2$ is processed in VNU. At the next clock cycle, $C1$ is processed in CNU, $C2$ waits for one clock cycle and $C3$ is processed in VNU. Finally, at the next clock cycle, $C1$ waits for a one clock cycle, $C2$ is processed into CNU, $C3$ waits for a one clock cycle and $C4$ is processed into VNU as observed in Fig. 3.15.

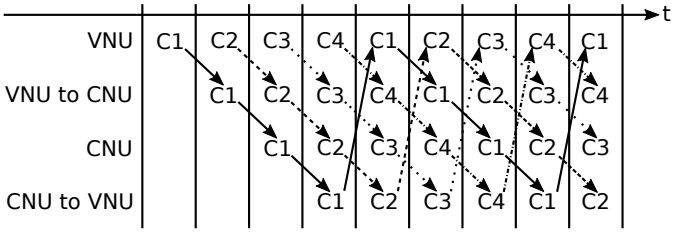

Figure 3.15: MO2-BIT-MSA fully-parallel pipeline interleaved for 4 codes.

Due to the fact that half broadcasting is used in the fully-parallel architecture, the VNU stores its own generated messages and are used to select between the min_1 or min_2 of the incoming messages (this is represented by the feedback line of the VNU scheme of Fig. 3.16). In order to properly complete the design of the pipeline interleaved fully-parallel architecture, the feedback lines of the VNUs must be delayed to compensate the extra delays added in the VNUs and CNUs. Therefore, one extra delay and three extra delays have to be included in the feedback lines of the VNUs for the 2-code and 4-code pipeline interleaved architectures, as shown in Fig. 3.17 and Fig. 3.18, respectively.

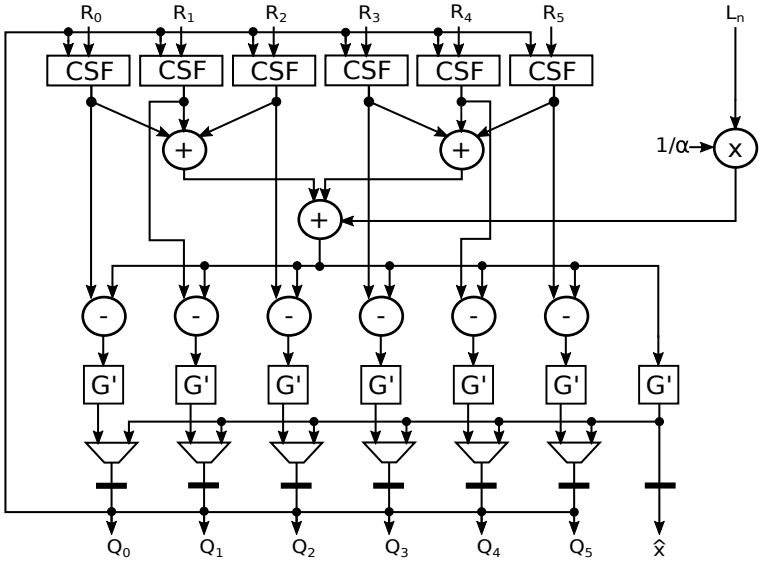

Figure 3.16: MO2-BIT-MSA VNU fully-parallel pipeline interleaved architecture for 1 codes.

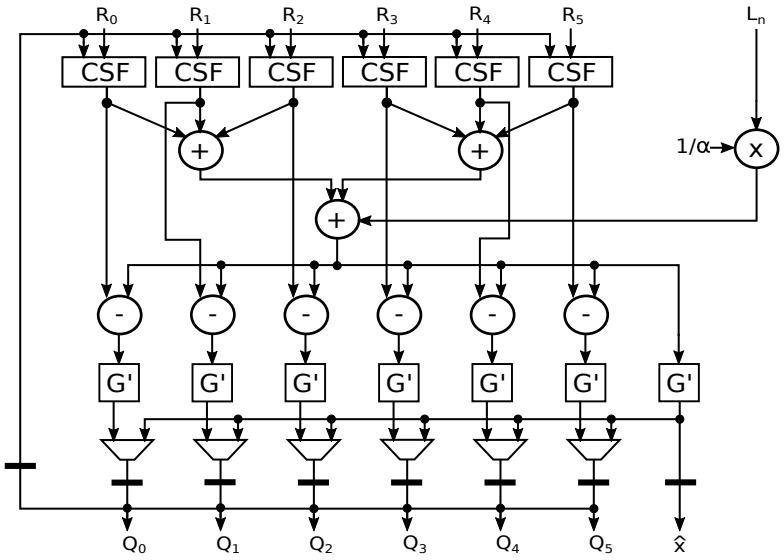

Figure 3.17: MO2-BIT-MSA VNU fully-parallel pipeline interleaved architecture for 2 codes.

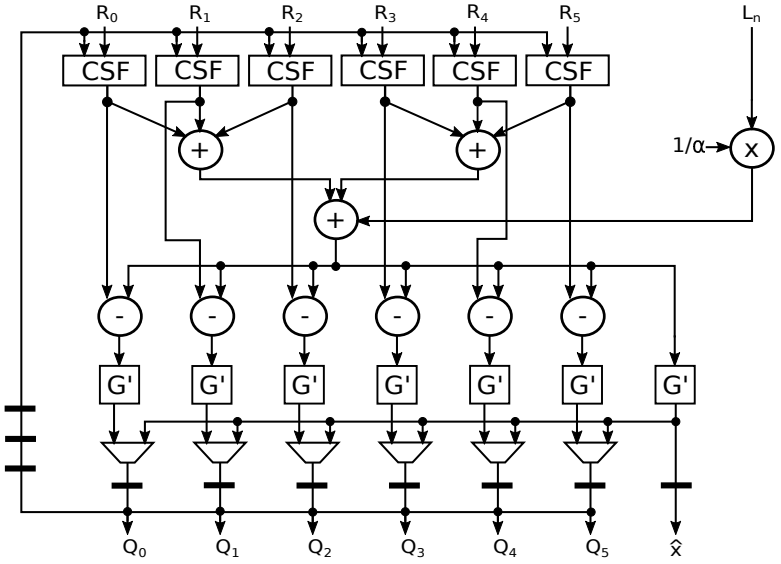

Figure 3.18: MO2-BIT-MSA VNU fully-parallel pipeline interleaved architecture for 4 codes.

Pipeline interleaved technique can be useful to improve the throughput of the parallel decoder if the additional registers included in the architecture shorten the critical path, increasing its working frequency. The throughput of the parallel architecture is

$$T = \frac{N \times f_{clk}}{\#clk \times \#iter}, \tag{3.13}$$

where N is the code length, f_{clk} the clock frequency, $\#clk$ is the number of clock cycles per iteration and $\#iter$ the number of iterations. When a pipeline interleaved of q codes, where $q = \#clk$, are used the throughput equation is given by:

$$T = \frac{q \times N \times f_{clk}}{\#clk \times \#iter} = \frac{N \times f_{clk}}{\#iter}. \tag{3.14}$$

As can be observed from the previous equation, the throughput directly raises as the term $\#clk$ is not any more in the denominator. Futhermore, the throughput also profits from the increase of the working clock frequency, f_{clk}, due to the extra pipeline stages.

3.6.3. Layered MO2-BIT-MSA

A special case of partial-parallel architecture suitable to implement Gbps decoders is the one that performs the message-passing using the horizontal layered schedule [34].

In the general case of a conventional MSA, a partial parallel architecture with horizontal layered schedule is derived as follows. The VNs always updates their soft-output information, SO_n, as :

$$SO_n = Q_n = L_n + \sum_{m \in M(n)} R_{m,n} \tag{3.15}$$

The layered architecture computes the soft-output information in three steps:

1. $Q_{m,n} = SO_n^{old} - R_{m,n}^{old}$
2. $R_{m,n}^{new} = F_{MS}(Q_{m,n})$, $n \in N(m)$
3. $SO_n^{new} = R_{m,n}^{new} + Q_{m,n}$,

where F_{MS} is the function that performs the MSA in the CNs. These three steps are computed with the layered processor (LP), as depicted in Fig. 3.19, which is the basic cell of the partial-parallel architecture with horizontal layered scheduling.

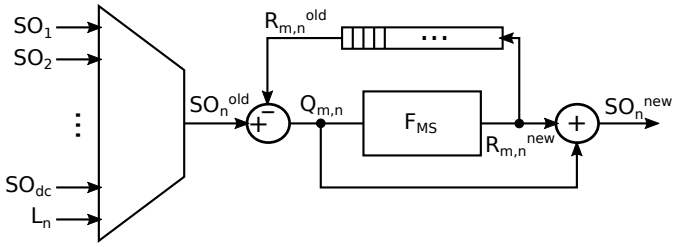

Figure 3.19: Scheme of the layered processor for MSA.

Next, we show the equations needed to derive the same layered architecture for the MO2-BIT-MSA algorithm. In this algorithm, the soft-output information, SO_n, is computed as:

$$SO_n = L_n + \sum_{m \in M(n)} f(R_{m,n}), \tag{3.16}$$

and the steps required to obtain the new soft-output value in one VN are:

1. $EI_{m,n} = SO_n^{old} - f(R_{m,n}^{old})$
2. $Q_{m,n} = g(EI_{m,n})$
3. $R_{m,n}^{new} = F_{MS}(Q_{m,n})$, $n \in N(m)$
4. $SO_n^{new} = f(R_{m,n}^{new}) + EI_{m,n}$.

From these equations the scheme of the layered processor is shown in Fig. 3.20.

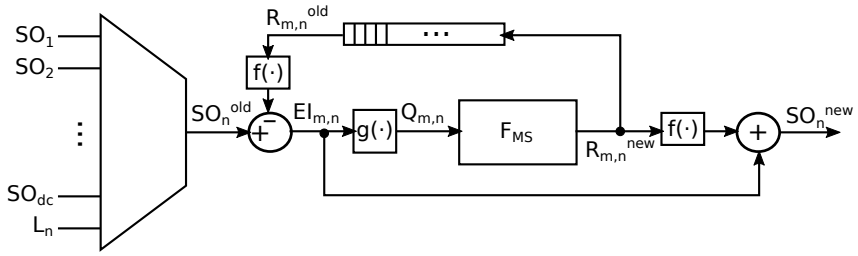

Figure 3.20: Scheme of the layered processor for MO2-BIT-MSA.

The (16129,15372) LDPC is too large to implement a fully-parallel decoder. Therefore, we used the layered processor of Fig. 3.20 to implement a partial parallel decoder for this code. It has a parity matrix of $M \times N = 762 \times 16129$, and the degrees of variable and check nodes are $d_v = 6$ and $d_c = 127$, respectively. The architecture computes in parallel one layer using 127 layered processors as the one of Fig. 3.20, so one iteration is completed in 6 clock cycles.

3.7. Hardware results and comparisons

The decoder architectures presented in previous sections for the (2048,1723) RS-LDPC code were modelled with VHDL and implemented in the Arria 10 10AX115-GES Altera device. This device was selected after many tries with Xilinx Virtex-7 devices. The main disadvantage of large Xilinx FPGA devices is that they are composed of several logic regions mounted on a passive silicon interposer with fast but limited number of connections among them. For example, the Virtex-7 T2000 device (the largest of the Virtex-7 family) has 13270 interconnections among logic regions, what makes impossible to route full-parallel LDPC decoders, which are characterized by a very high number of connections. The Arria 10 10AX115-GES device is the biggest one available in the Quartus Prime v.16 software, which was used to obtain the results. This device has 427200 ALMs (adaptive Logic Modules) composed of 2 ALUTs (Altera Look-Up Tables) and 4 FF (Flip-Flops), *i.e.* it contains a total of 854400 ALUTs and 1708800 FFs.

Table 3.4 shows the results of the implementation of the full-parallel architectures. It gives the number of ALMs, LUTs and FFs, the number of clock cycles per iteration ($\#cycles$), the maximum clock frequency (f_{clk}), and the throughput for 15, 10 and 5 iterations of the algorithm. Three versions of the fully-parallel decoder are evaluated: 1) convetional decoder architecture with 1 clock cycle per iteration ($v1\_co$), where the output of the VNUs are registered; 2) decoder with 2 clock cycles per iteration ($v1\_c1$), where both the outputs of the VNUs and CNUs are registered; and 3) decoder with 4 clock cycles per iteration ($v2\_c2$) with inputs and outputs of the VNUs and CNUs registered.

From the results of Table 3.4 it can be concluded that the faster decoder is achieved by registering both the output of the VNUs and CNUs. It reaches 11 Gbps with 10 iterations.

vr_cr	#ALMs	#ALUTs	#FFs	f_{clk}	#cycles	Throughput (Gbps) (15it)/(10it)/(5it)
$v1\_c0$	183016	324940	48838	48	1	6.5/9.8/19.6
$v1\_c1$	181501	326148	53170	109	2	7.4/11.1/22.3
$v1\_c2$	181626	326150	139312	167	4	5.7/8.5/17.1

Table 3.4: Results of the implementation of full-parallel MO2-BIT-MSA decoders for the (2048,1723) LDPC code.

Table 3.5 presents the results of the implementation of full-parallel pipeline-interleaved decoders for the cases of interleaving $\#int\_cod = 2$ and 4 codes. As can be seen, this technique is suitable to increase the throughput of the decoder. The throughput is basically doubled when two codes are interleaved and trebled with 4 interleaved codes.

#int_cod	#ALMs	#ALUTs	#FFs	f_{clk}	#cycles	Throughput (Gbps) (15it)/(10it)/(5it)
2	193803	326150	88976	115	2	15.7/23.5/47.1
4	212331	338438	245799	161	4	21.9/32.9/65.9

Table 3.5: Results of the implementation of full-parallel pipeline interleaved MO2-BIT-MSA decoders for the (2048,1723) LDPC code.

For comparison purposes, a fully-parallel decoder and a layered decoder for the MSA algorithm with 6-bit messages were implemented in the same Arria 10 device. The results are detailed in Table 3.6. The fully-parallel decoder exceeds the capacity of the device, so only the synthesis stage could be completed. It requires nearly three times more ALUTs than the proposed architectures. On the other

hand, the layered uses 26% less number of ALUTs but reaches a throughput 11 times lower.

MS_version	#ALMs	#ALUTs	#FFs	f_{clk}	#cycles	Throughput (Gbps) (15it)/(10it)/(5it)
FULL-PAR	631938	1125108	118936	-	2	-/-/-
LAYERED	167214	239335	36062	30.1	6	0.7/1/2.1

Table 3.6: Results of the implementation of 6-bit MSA decoders for the (2048,1723) LDPC code.

A layered decoder architecture was modelled with VHDL for the (16129,15372) LDPC code and implemented in a 90 nm CMOS process of nine layers with standard cells and operating conditions of 25°C and 1.2 V using Cadence RTL Compiler and SOC encounter tools. The obtained place and route results are summarized in Table 3.7. For the sake of readability, the results of MSA and OMO-MSA given in Chapter 2 for the same LDPC code and implemented in the same process are rewritten in this table.

Decoder	Area (mm^2)	Frequency (MHz)	Throughput (Gbps) (15it)/(10it)/(5it)	Ratio throughput/area (Gbps/mm^2)
MO2-BIT-MSA	8.154	70.7	12.67/19/38	1.55
OMO-MSA	14.173	48	8.6/12.9/25.8	0.6067
MSA [20]	15.909	37.23	6.67/10/20	0.4192

Table 3.7: Results of the ASIC implementation for the (16129,15372) LDPC code decoders.

For the (16129,15372) LDPC code, we can see from Table 3.7 that the MO2-BIT-MSA full layered decoder reduces the area in 42% and 48%, and increases the throughput in 47% and 90% with respect to OMO-MSA and MSA, respectively. Futhermore, simulations made with a FPGA emulator showed that MO2-BIT-MSA does not introduce error-floor.

3.8. Conclusions

In this chapter the optimized 2-bit MSA algorithm is improved using the LLR as input and including the extrinsic information in the messages sent from the variable nodes to the check nodes. As the algorithm has several interrelated parameters, a method to reduce the simulation time needed to find their optimum values is presented. The resulting algorithm performs very close to the MSA, which is shown with the (2048,1723) RS-LDPC and (16129,15372) LDPC codes. Several fully-parallel architectures were implemented in an FPGA device for the (2048,1723) RS-LDPC code using our proposed VNU, which reduces the decoder complexity. A throughput 11 times higher than a layered MSA architecture is reached with only an ALUT area overhead of 36%. The pipeline-interleaving technique is applied successfully doubling the throughput when two codes are interleaved. A layered decoder for the (16129,15372) LDPC code was implemented in a 90 CMOS process reaching the double of throughput and half area with respect the layered MSA.

Chapter 4

Historical-extrinsic reliability-based iterative decoder (HE-RBID)

4.1. Introduction

As seen in Chapter 1, Section 1.5, iterative hard-decision algorithms are not able to correct most of the errors and introduce an early degradation at BER higher than 10^{-5} [30], which make them impractical for most modern communication or storage systems. For example, analog-to-digital converters for optical communications need very few bits in their operation and they consume a lot, in this case it would be more efficient to use a hard-decision decoder instead of a soft-decision decoder. Due to these real problems, we are interested in proposing a hard-decision decoder close to the performance of a soft-decision decoder.

In order to solve the degradation of performance due to a low d_v in high-rate codes, we propose a new reliability-based iterative majority-logic decoding (RBI-MLGD) algorithm that computes the extrinsic information of previous iterations of the varible node. This decoding algorithm is called historical-extrinsic reliability-based iterative decoder (HE-RBID) and improves the BER performace of the previous RBI-MLGD [29] and [30]. HE-RBID is particularly interesting for codes with a low degree of variable node, where traditional RBI-MLGD algorithms do not provide a good behaviour. HE-RBID ensures good performance without searching for the minimum at the check node and only by exchanging 1-bit messages between variable and check nodes.

4.2. HE-RBID algorithm

The more soft-decision is included in the check-node, the better performance in the waterfall region is obtained. This is the strategy followed by the RBI-MSD algorithm, which uses integer reliability information in the check-node updates. However, the lack of soft-decision information in the check-node update does not justify that RBI-MLGD algorithms introduce some early performance degradation when the degree of the variable-node is low. We assume that the degradation of performance of RBI-MLGD algorithms is due to the lack of extrinsic information in the variable-node update. Therefore, we introduce here the new idea of considering hard-decision votes as soft-decision and extract from them the extrinsic information at the variable-node (hard-decision extrinsic (H-Ex) computation).

The H-Ex method was tested with both variable-node updating rules: 1) the one followed by RBI-MLGD in which the historical-reliability data are taken into account by accumulating the votes of all the previous iterations; and 2) the one followed by MRBI-MLGD in which only the initial reliability is added to the newly updated sum of votes, emulating the turbo decoder update (Tu-DU). The scaling factor α is required in both options, non-Tu-DU (1) and Tu-DU (2), to maintain a balance between the voting information and the incoming soft input from the channel. As it will be shown in next Section, the H-Ex method with Tu-DU introduces an early degradation in the BER performance, so hereafter we will focus on H-Ex method with non-TU-DU.

The proposal of this thesis is that each hard-decision at the check-node is also a one-bit message as in the RBI-MLGD algorithms, but in this case is calculated based on the extrinsic information computed at the variable-node, $Q_{m,n}$ as in Eq. 4.1.

$$\hat{x}_{m,n}^{(i)} = \begin{cases} 1, & Q_{m,n}^{(i)} \geq 0, \quad m \in \{0, ..., M-1\}, \quad n \in N_m \\ 0, & otherwise \end{cases} \tag{4.1}$$

Due to the computation of the extrinsic information, the reliability of the hard-decision applied to calculate the check-node equations, as in Eq. 4.2 and Eq. 4.3, is increased.

$$s_m^{(i)} = \sum_{0 \leq n \leq N-1} \hat{x}_{m,n}^{(i)} h_{m,n} = \sum_{n \in \mathcal{N}_m} \hat{x}_{m,n}^{(i)}, \quad m \in \{0, ..., M-1\} \tag{4.2}$$

$$\sigma_{m,n}^{(i)} = \sum_{n' \in N_m \backslash n} \hat{x}_{m,n'}^{(i)} = s_m^{(i)} \oplus \hat{x}_{m,n}^{(i)}, \quad m \in \{0, ..., M-1\} \tag{4.3}$$

Moreover, the vote counting at the variable-node is extrinsic (H-Ex) as in Eq. 4.4, as for the edge $R_{m,n}$ we only consider the information from the sets of check nodes $m' \in M_n \backslash m$. So, if the edge (m, n) has one error, it does not affect to the edge itself.

$$R_{m,n}^{(i)} = [\alpha \sum_{m' \in M_n \backslash m} (2\sigma_{m',n}^{(i)} - 1)], \quad n \in \{0, ..., N-1\} \tag{4.4}$$

Finally, Eq. 4.5 considers the H-Ex information from all the iterations, in other words, non-Tu-DU is applied. Non-Tu-DU ensures that the decoder takes into account not only the information of the last vote, but also the historical information of the H-Ex votes of the rest of previous iterations. This makes the algorithm converge slower than algorithms derived from SPA but improves the performace for codes with low d_v. This non-Tu-DU allows us to partially overcome the absence of a high number of check-nodes to make a reliable decision in one iteration by "remembering" the decision of a low number of check-nodes in a larger number of iterations.

$$Q_{m,n}^{(i+1)} = Q_{m,n}^{(i)} + R_{m,n}^{(i)}, \quad m \in \{0, ..., M-1\}, \quad n \in N_m \tag{4.5}$$

This algorithm is named as Historical-Extrinsic Reliability-Based Iterative Decoding (HE-RBID) and replaces steps 1 to 5 of RBI-MLGD algorithm with Eq. 4.1 - Eq. 4.5 as can be seen in Algorithm 9.

To perform the tentative decoding, Q_n is approximated by means of $Q_{m,n}$, because with independence of the index $m \in M_n$ selected (step 6 Algorithm 9), if the condition for early termination is satisfied, all the variable nodes converge to the same solution in most of the cases, avoiding performance loss in the waterfall region.

Algorithm 9 HE-RBID.

Input : $Q_n^{(0)} = \varrho(y_n)$, with $n \in \{0, ..., N-1\}$
Iterative process
for $i = 0 \rightarrow It_{max} - 1$ **do**
Check-node update

$$\hat{x}_{mn}^{(i)} = \begin{cases} 1, & Q_{mn}^{(i)} \geq 0 \\ 0, & otherwise \end{cases}$$

$$s_m^{(i)} = \sum_{0 \leq n \leq N-1} \oplus \hat{x}_{mn}^{(i)} h_{mn} = \sum_{n \in \mathcal{N}_m} \oplus \hat{x}_{mn}^{(i)}, \quad m \in \{0, ..., M-1\}$$

$$\sigma_{mn}^{(i)} = \sum_{n' \in N \setminus n} \oplus \hat{x}_{mn'}^{(i)} = s_m^{(i)} \oplus \hat{x}_{mn}^{(i)}, \quad m \in \{0, ..., M-1\}$$

Variable-node update

$$R_{mn}^{(i)} = \alpha \sum_{m' \in M_n \setminus m} (2\sigma_{m'n}^{(i)} - 1), \quad n \in \{0, ..., N-1\}$$

$$Q_{mn}^{(i+1)} = Q_{mn}^{(i)} + R_{mn}^{(i)}, \quad m \in \{0, ..., M-1\}, \quad n \in N_m$$

Tentative decoding

$$Q_n^{(i+1)} = Q_{mn}^{(i+1)}, \quad rand(m) \in \{0, ..., M-1\}, \quad n \in N_m$$

$$\hat{x}_n^{(i+1)} = \begin{cases} 1, & Q_n^{(i+1)} \geq 0 \\ 0, & otherwise \end{cases}$$

if $(s^{(i)} = 0)$ **then** SKIP
end
Output : $\hat{x}$

1.6.4.1. Example 1. Historical-extrinsic reliability-based iterative decoder

Let us assume that a bit sequence of seven zeros is transmitted on a channel with errors. The magnitude of the LLR is correlated with the magnitude of the received level. We take the positive sign as a logic 0 and the negative sign as a logic 1.

In this example LLR values are computed as $L_0(y_0) = -15 = n_0$, $L_1(y_1) = -15 = n_1$, $L_2(y_2) = -15 = n_2$, $L_3(y_3) = +1 = n_3$, $L_4(y_4) = -15 = n_4$, $L_5(y_5) = -15 = n_5$ and $L_6(y_6) = -15 = n_6$ and in this case we use a scale factor $\alpha = 0.5$.

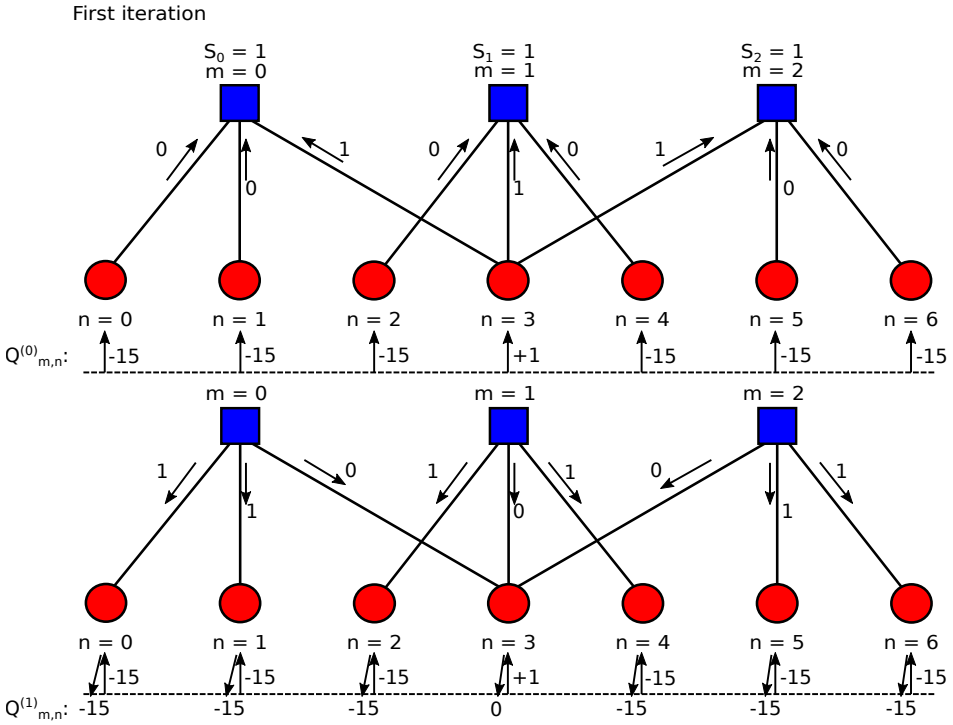

Figure 4.1: HE-RBID example. First iteration.

First, the LLR values are loaded in the VNs. Once this is done, messages are sent from VNs to CNs and the syndromes are computed:

$$s_0^{(0)} = \hat{x}_{0,0}^{(0)} \oplus \hat{x}_{0,1}^{(0)} \oplus \hat{x}_{0,3}^{(0)} = 0 \oplus 0 \oplus 1 = 1$$
$$s_1^{(0)} = \hat{x}_{1,2}^{(0)} \oplus \hat{x}_{1,3}^{(0)} \oplus \hat{x}_{1,4}^{(0)} = 0 \oplus 1 \oplus 0 = 1$$
$$s_2^{(0)} = \hat{x}_{2,3}^{(0)} \oplus \hat{x}_{2,5}^{(0)} \oplus \hat{x}_{2,6}^{(0)} = 1 \oplus 0 \oplus 0 = 1$$

where

$$\hat{x}_{m,n}^{(0)} = \begin{cases} 1, & Q_{m,n}^{(0)} \geq 0 \\ 0, & otherwise \end{cases}$$

With the syndromes computed, the value that will satisfy each of the parity-check equations is calculated. Considering that the incoming messages from the VN can

have an error (processing only the information from the neighbours), the following equation is applied to determine the magnitude of the messages from the CN to VN:

$$\sigma_{m,n}^{(i)} = s_m^{(i)} \oplus \hat{x}_{m,n}^{(i)}$$

So the magnitude is computed as:

$$\begin{aligned}
\sigma_{0,0}^{(0)} &= s_0^{(0)} \oplus \hat{x}_{0,0}^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{0,1}^{(0)} &= s_0^{(0)} \oplus \hat{x}_{0,1}^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{0,3}^{(0)} &= s_0^{(0)} \oplus \hat{x}_{0,3}^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{1,2}^{(0)} &= s_1^{(0)} \oplus \hat{x}_{1,2}^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{1,3}^{(0)} &= s_1^{(0)} \oplus \hat{x}_{1,3}^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{1,4}^{(0)} &= s_1^{(0)} \oplus \hat{x}_{1,4}^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{2,3}^{(0)} &= s_2^{(0)} \oplus \hat{x}_{2,3}^{(0)} = 1 \oplus 1 = 0 \\
\sigma_{2,5}^{(0)} &= s_2^{(0)} \oplus \hat{x}_{2,5}^{(0)} = 1 \oplus 0 = 1 \\
\sigma_{2,6}^{(0)} &= s_2^{(0)} \oplus \hat{x}_{2,6}^{(0)} = 1 \oplus 0 = 1
\end{aligned}$$

We can check the previous results with the magnitude of the message exchange in Fig. 4.1.

Finally, the tentative decoding is computed by adding the LLR value received from the channel to the incoming check-to-variable node messages as follows:

$$Q_{0,0}^{(1)} = Q_0^{(0)} = -15$$
$$Q_{0,1}^{(1)} = Q_1^{(0)} = -15$$
$$Q_{1,2}^{(1)} = Q_2^{(0)} = -15$$
$$Q_{0,3}^{(1)} = Q_3^{(0)} = Q_{0,3}^{(0)} + (\alpha \cdot ((2 \cdot \sigma_{1,3}^{(0)} - 1) + (2 \cdot \sigma_{2,3}^{(0)} - 1))) = 1 + (0.5 \cdot ((-1) + (-1))) = 0$$
$$Q_{1,3}^{(1)} = Q_{1,3}^{(0)} + (\alpha \cdot ((2 \cdot \sigma_{0,3}^{(0)} - 1) + (2 \cdot \sigma_{2,3}^{(0)} - 1))) = 1 + (0.5 \cdot ((-1) + (-1))) = 0$$
$$Q_{2,3}^{(1)} = Q_{2,3}^{(0)} + (\alpha \cdot ((2 \cdot \sigma_{0,3}^{(0)} - 1) + (2 \cdot \sigma_{1,3}^{(0)} - 1))) = 1 + (0.5 \cdot ((-1) + (-1))) = 0$$
$$Q_{1,4}^{(1)} = Q_4^{(0)} = -15$$
$$Q_{2,5}^{(1)} = Q_5^{(0)} = -15$$
$$Q_{2,6}^{(1)} = Q_6^{(0)} = -15$$

Performing a second iteration:

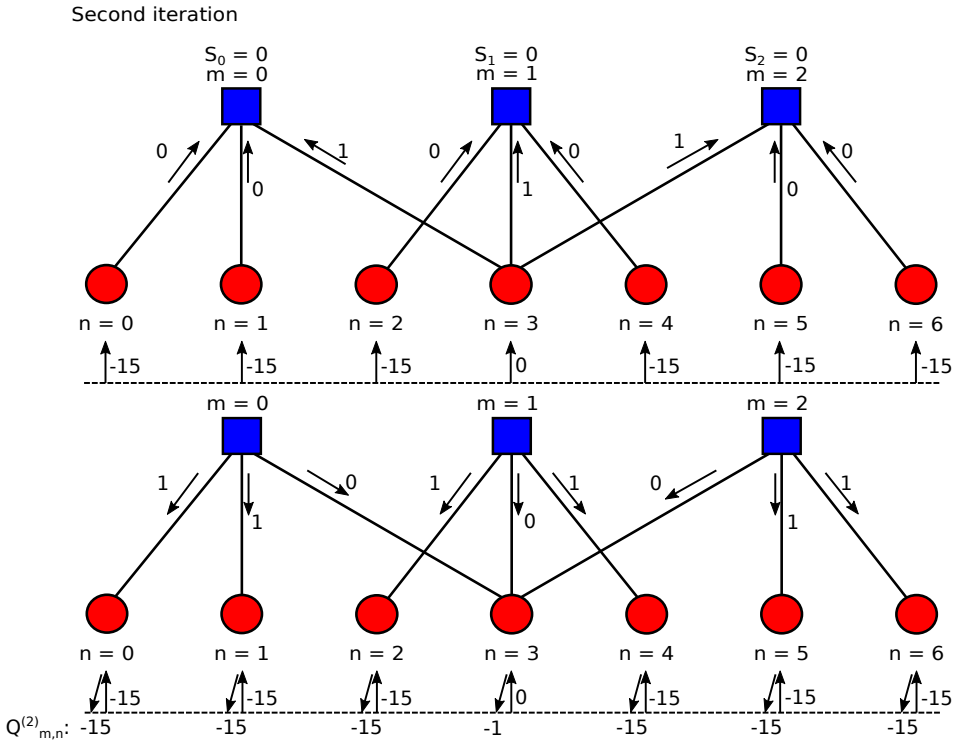

Figure 4.2: HE-RBID example. Second iteration.

The messages computed in each VN are sent from VNs to CNs, computing the syndromes:

$$s_0^{(1)} = \hat{x}_{0,0}^{(1)} \oplus \hat{x}_{0,1}^{(1)} \oplus \hat{x}_{0,3}^{(1)} = 0 \oplus 0 \oplus 1 = 1$$
$$s_1^{(1)} = \hat{x}_{1,2}^{(1)} \oplus \hat{x}_{1,3}^{(1)} \oplus \hat{x}_{1,4}^{(1)} = 0 \oplus 1 \oplus 0 = 1$$
$$s_2^{(1)} = \hat{x}_{2,3}^{(1)} \oplus \hat{x}_{2,5}^{(1)} \oplus \hat{x}_{2,6}^{(1)} = 1 \oplus 0 \oplus 0 = 1$$

where

$$\hat{x}_{m,n}^{(1)} = \begin{cases} 1, & Q_{m,n}^{(1)} \geq 0 \\ 0, & otherwise \end{cases}$$

Once the syndromes are computed, the value that will satisfy each of the parity-check equations is processed. The computation of the magnitude of the messages is done as follows:

$$\sigma_{0,0}^{(1)} = s_0^{(1)} \oplus \hat{x}_{0,0}^{(1)} = 1 \oplus 0 = 1$$
$$\sigma_{0,1}^{(1)} = s_0^{(1)} \oplus \hat{x}_{0,1}^{(1)} = 1 \oplus 0 = 1$$
$$\sigma_{0,3}^{(1)} = s_0^{(1)} \oplus \hat{x}_{0,3}^{(1)} = 1 \oplus 1 = 0$$
$$\sigma_{1,2}^{(1)} = s_1^{(1)} \oplus \hat{x}_{1,2}^{(1)} = 1 \oplus 0 = 1$$
$$\sigma_{1,3}^{(1)} = s_1^{(1)} \oplus \hat{x}_{1,3}^{(1)} = 1 \oplus 1 = 0$$
$$\sigma_{1,4}^{(1)} = s_1^{(1)} \oplus \hat{x}_{1,4}^{(1)} = 1 \oplus 0 = 1$$
$$\sigma_{2,3}^{(1)} = s_2^{(1)} \oplus \hat{x}_{2,3}^{(1)} = 1 \oplus 1 = 0$$
$$\sigma_{2,5}^{(1)} = s_2^{(1)} \oplus \hat{x}_{2,5}^{(1)} = 1 \oplus 0 = 1$$
$$\sigma_{2,6}^{(1)} = s_2^{(1)} \oplus \hat{x}_{2,6}^{(1)} = 1 \oplus 0 = 1$$

In this second iteration, the reliability values for computing the tentative decoding are:

$$Q_{0,0}^{(2)} = Q_0^{(1)} = -15$$
$$Q_{0,1}^{(2)} = Q_1^{(1)} = -15$$
$$Q_{1,2}^{(2)} = Q_2^{(1)} = -15$$
$$Q_{0,3}^{(2)} = Q_3^{(1)} = Q_{0,3}^{(1)} + (\alpha \cdot ((2 \cdot \sigma_{1,3}^{(1)} - 1) + (2 \cdot \sigma_{2,3}^{(0)} - 1))) = (0) + (0.5 \cdot ((-1) + (-1))) = -1$$
$$Q_{1,3}^{(2)} = Q_{1,3}^{(1)} + (\alpha \cdot ((2 \cdot \sigma_{0,3}^{(1)} - 1) + (2 \cdot \sigma_{2,3}^{(0)} - 1))) = (0) + (0.5 \cdot ((-1) + (-1))) = -1$$
$$Q_{2,3}^{(2)} = Q_{2,3}^{(1)} + (\alpha \cdot ((2 \cdot \sigma_{0,3}^{(1)} - 1) + (2 \cdot \sigma_{1,3}^{(0)} - 1))) = (0) + (0.5 \cdot ((-1) + (-1))) = -1$$
$$Q_{1,4}^{(2)} = Q_4^{(1)} = -15$$
$$Q_{2,5}^{(2)} = Q_5^{(1)} = -15$$
$$Q_{2,6}^{(2)} = Q_6^{(1)} = -15$$

Performing a third iteration:

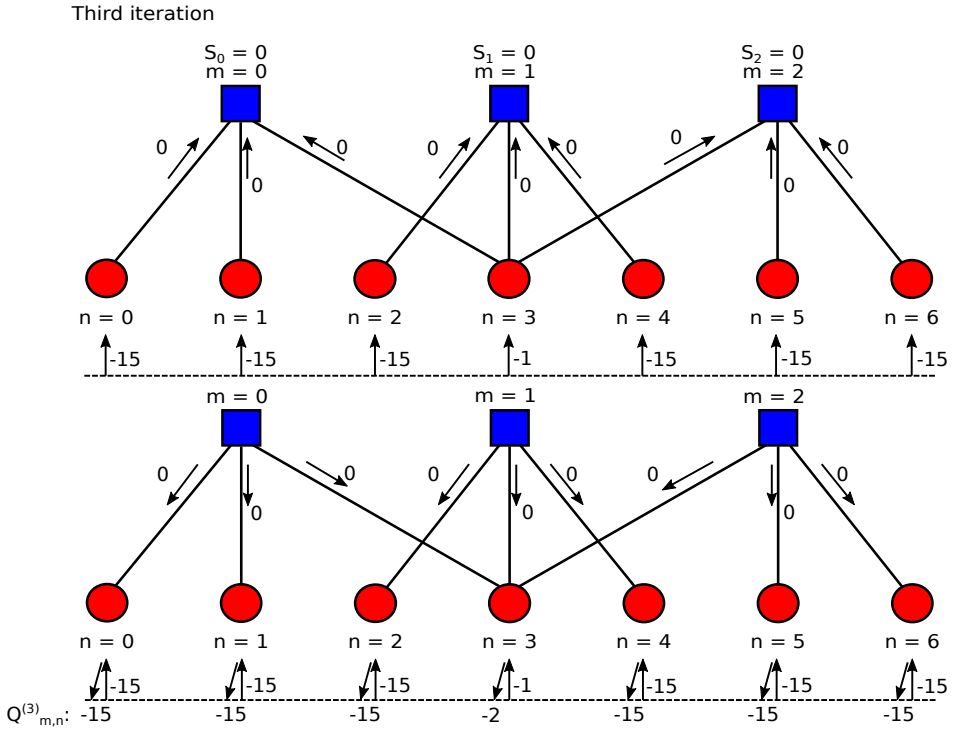

Figure 4.3: HE-RBID example. Third iteration.

Finally, it can be checked that all the syndromes are satisfied and, hence, the code word is corrected. The error in VN $n = 3$ has been corrected, flipping the sign of the node from positive to negative. All the $Q_{m,n}$ messages are now negative, and all the hard-decision messages are zero, as shown in Fig. 4.3. Note that the decisions taken by the algorithm require more iterations because the convergence is slower, which helps the algorithm not to diverge towards an incorrect solution in case of errors.

4.3. Error correction performance

Performance tests were performed using three different types of codes:

1. (2304,2048) algebraic LDPC code [10] with $d_c = 36$ and $d_v = 6$.
2. (2304,1920) LDPC code for Wimax [32] with $d_c = 20$ and $d_v = 4$.

3. (2048,1723) Reed-Solomon-based LDPC code [9] with $d_c = 32$ and $d_v = 6$.

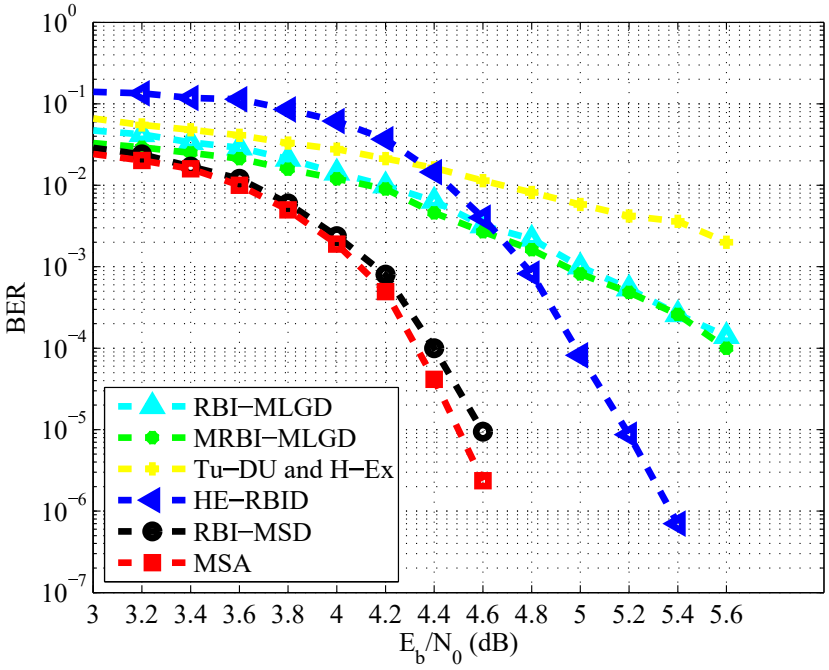

Figure 4.4: BER performance of several LDPC decoding algorithms for the (2304,2048) algebraic LDPC code (20 iterations).

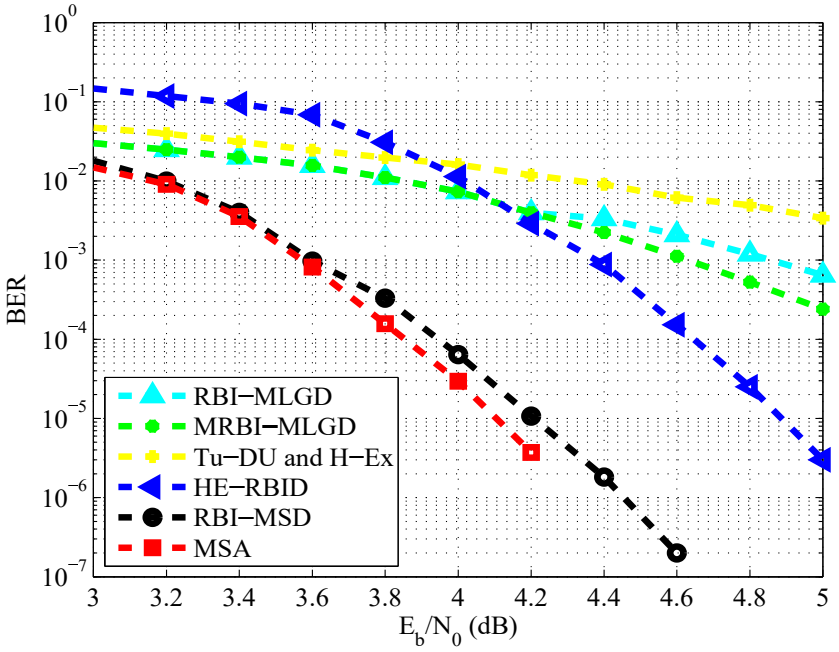

Figure 4.5: BER performance of several LDPC decoding algorithms for the (2304,1920) LDPC code for Wimax (20 iterations).

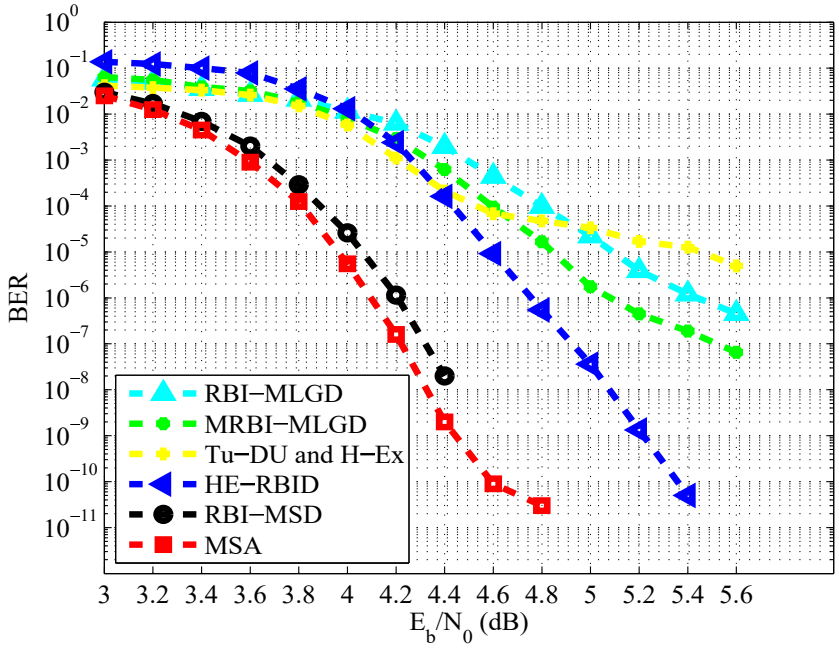

Figure 4.6: BER performance of several LDPC decoding algorithms for the (2048,1723) Reed-Solomon-based LDPC code (20 iterations).

For the (2304,2048) algebraic LDPC code (Fig. 4.4) and the (2304,1920) LDPC code for Wimax (Fig. 4.5) we can see that HE-RBID reduces the early degradation at least two orders of magnitude compared to (M)RBI-MLGD with a reduction of complexity compared to RBI-MSD and MSA, at a cost of 0.7 dB and 0.8 dB of performance loss in the waterfall region, respectively. With the (2048,1723) Reed-Solomon-based LDPC code, HE-RBID introduces a performance loss of 0.7 dB at BER $= 10^{-10}$ compared to MSA but with a complexity reduction (Fig. 4.6). If we compare HE-RBID to (M)RBI-MLGD, the first one reduces the early degradation at BER $= 10^{-6}$ to at least 10^{-10}. In all the cases a decoding algorithm that combines Tu-DU and H-Ex computation shows an early degradation that it is not practical for real applications.

4.4. Complexity analysis

In Table 4.1 we include the complexity of RBI-MLGD, MRBI-MLGD, HE-RBID and RBI-MSD algorithms in terms of number of operations per iteration. RBI-MLGD, MRBI-MLGD, HE-RBID and RBI-MSD involve the same number of binary operations; the main differences in complexity are in the number of real-number additions/subtractions and comparisons [1].

Algorithm	Binary Operation	Addition/ Subtraction	Comparison	Bits of Message Passing
(M)RBI -MLGD	$2 \times (d_c - 1) \times M + N$	$d_c \times M$	0	1
RBI -MSD	$2 \times (d_c - 1) \times M + N$	$d_c \times M$	$(d_c \times M) \times$ $\times (d_c - 2)$	q
HE-RBID -RBID	$2 \times (d_c - 1) \times M + N$	$2 \times d_c \times M$	0	1

Table 4.1: Complexity per iteration for different LDPC decoding algorithms.

RBI-MSD needs the processing of $d_c - 2$ comparisons at each edge of the code (remember that there are $d_c \times M$ edges) which is similar to the number of comparisons of MSA from [16]. In fact, RBI-MSD has less complexity compared to MSA with independence of the version [16], [17]. Due to this, if HE-RBID is less complex in terms of operations per iteration than RBI-MSD, it will be less complex than MSA, even with a greater reduction in complexity. HE-RBID duplicates the number of additions/subtractions because the H-Ex processing is required. Finally, it is important to remark once again that HE-RBID, RBI-MLGD and MRBI-MLGD only perform one-bit message passing, while RBI-MSD requires q-bit message passing, where usually $q = 8$ according to [30]. To have a reference of complexity of the mentioned algorithms, we enumerate the total number of operations per iteration (OPI) required for the codes:

1. (2304,2048) algebraic LDPC code:
   - $(M)RBID = 29KOPI$
   - $RBI - MSD = 342KOPI$
   - $HE - RBID(\alpha = 0.25) = 38KOPI$
2. (2304,1920) LDPC code for Wimax:
   - $(M)RBID = 24KOPI$

[1] Note that multiplications by the scaling factor α, which is constant, are omitted because they are negligible compared to the rest of the complexity.

- $RBI - MSD = 162KOPI$
- $HE - RBID(\alpha = 0.75) = 32KOPI$

3. (2048,1723) Reed-Solomon-based LDPC code:
   - $(M)RBID = 38KOPI$
   - $RBI - MSD = 406KOPI$
   - $HE - RBID(\alpha = 0.375) = 50KOPI$

From the previous results we see that HE-RBID is $(32K - 24K)/24k = 33\%$ more complex than (M)RBI-MLGD in the most restrictive case. On the other hand, RBI-MSD is heavily dependent on d_c. Our proposal is between five and eight times less complex than RBI-MSD for the (2304,1920) LDPC code for Wimax and the (2048,1723) Reed-Solomon-based LDPC code respectively.

Although the objective of this chapter is not to implement a hardware architecture but to analyze how to reduce the complexity of the decoding algorithm and to improve the error floor in RBI-MLGD algorithms, the following aspects can be easily deduced in terms of the derived architectures:

1. Area: Although HE-RBID duplicates the number of additions/substractions, as shown in Table 4.1, the number of bits of message passing is only one compared to q bits of RBI-MSD. So, the final complexity is reduced with respect to RBI-MSD and, also, the required area.
2. Throughput: The critical path of RBI-MSD and HE-RBID is the same, but since the HE-RBID has fewer bits to exchange between CNU and VNU, the routing will be smaller and, therefore, there will be less congestion and higher decoding speed.
3. Consumption: The consumption of an element is related to the transition from one logic level to another, therefore, although HE-RBID has a smaller number of operations with respect to RBI-MSD, it reduces the number of transitions, so it lowers the consumption. Also, since there are less bits used in each operation, consumption is further reduced. Therefore, using HE-RBID lower power is required.

4.5. Conclusions

In this chapter a new RBI-MLGD algorithm is proposed. This algorithm, named as HE-RBID, reduces between five and nine times the complexity of RBI-MSD at the cost of some performance loss (0.7-0.8 dB) in the waterfall. The proposed solution avoids the early performance degradation that RBI-MLGD and MRBI-MLGD introduce with codes of low d_v and non-EG construnction. HE-RBID computes

the extrinsic information from hard-decision votes of previous iterations at the varible node (hard-decision extrinsic (H-Ex) computation), while it does not increase complexity by more than 33%. In addition, the new proposal also improves the coding gain at the error floor region. HE-RBID uses the extrinsic information of previous iterations as soft-decision information without searching for the minimum at the check node like RBI-MSD, besides, our proposal is based on one-bit message-passing and so, no integer reliability is used in the CN, in contrast to RBI-MSD.

Chapter 5

Conclusions and future work

The objective of this thesis was the design of high-efficiency decoder architectures based on LDPC codes, and in particular for Min-sum and Reliability-based decoders. In this chapter the main conclusions of the thesis are exposed and the future lines of research are drawn.

5.1. Conclusions

Focused on the objective of improving LDPC decoders in order to obtain architectures with the best tradeoff between error correction performance and throughput-area ratio, the state of the art was reviewed with the aim of detecting the main bottlenecks in: i) algorithms complexity, ii) bit error rate performance and iii) hardware architectures. Derived from this analysis a tutorial guide was elaborated. Part of the guidelines included there are outlined in the following paragraphs.

First, the basics of LDPC codes and the message passing decoding schedules were introduced. Next, the three basic hardware architectures for LDPC decoders were analyzed, concluding that using a partially-parallel architecture, a balance between the benefits of fully-parallel (energetically efficient) and fully-serial (with very low routing congestion and area) architectures is obtained. Partially-parallel architecture improves efficiency by partitioning H into rowwise and columnwise groups so that a set of check-node and variable-node updates can be done per cycle.

SPA and scaled MSA were presented with their performance in Chapter 1. From the analysis of these two soft-decision decoding algorithms it was concluded that SPA provides the best performance at expenses of a high computational complexity

in the check node. In order to reduce this complexity the best proposal to achieve a better trade-off between complexity and coding gain is scaled MSA. However, scaled MSA causes a slightly performance degradation and its complexity is still moderated due to the number of operations in the calculation of the magnitude at the check-node, which depends on d_c especially large for very high-rate codes.

In Chapter 2, a new approximation to estimate the second minimum magnitude called One Minimum Only Min-Sum algorithm (OMO-MSA) is proposed. The OMO-MSA reduces the complexity of the check node update hardware using an estimated min_2 value, instead of computing it. The estimation is performed by a linear approximation tuned with two parameters and allows a dynamic adjustment of the correction factor, which is adapted automatically to the iteration number and to the signal-to-noise ratio. The design of the tuning parameters is quite simple compared to other lossless approximations. It was shown that our solution has a performance very close to MSA: three LDPC codes based on different construction methods were evaluated obtaining a performance loss lower than 0.1 dB in all the cases. The performance at high SNR values was also evaluated using an FPGA-based hardware emulator. It was demonstrated that the OMO simplification does not degrade the correction performance at high SNR. An interesting result is that for the (2048,1723) LDPC code used in the IEEE 803.an standard, the OMO algorithm has a 0.2 dB of coding gain with respect to MSA at BER = 10^{-13}. It was also shown that the reduction of complexity of the proposed OMO-MSA leads to a higher area-throughput efficiency in the hardware decoder implementations. A hardware OMO check node was derived and used to implement partial parallel decoders with horizontal layered schedule for the (2048,1723) LDPC code and the (16129,15372) LDPC code. The decoders were modelled in VHDL and implemented in a 90 nm CMOS process. The implementation results indicate that the decoder for the (2048,1723) LDPC code improves the efficiency of the MSA by 18% in terms of throughput/area ratio, and the one for the (16129,15372) LDPC code improves the throughput in 28.9% and reduces the area 1.7 times.

A third soft-decision decoding algorithm, Optimized 2-bit MSA (O2-BIT-MSA) was analyzed in Chapter 1. From previous analysis, it was concluded that Optimized 2-bit MSA reduces the routing congestion with the use of two bits in the message-passing but with a moderate loss in the coding gain with respect to SPA and MSA. Furthermore, the algorithm performance depends on five parameters, whose optimization is difficult and time-consuming, as requires many simulations. In order to overcome those weak points the Modified Optimized 2-bit Min-Sum Algorithm (MO2-BIT-MSA) is proposed in Chapter 3. The first modification introduced in the MO2-BIT-MSA is the use of the LLR in the variable nodes with uniform quantization of several bits instead of 2-bit non-uniform one. This has an important impact on the BER performance, while the routing congestion is not affected: the interchanged messages are kept with 2 bits. A method to find the

optimum values for the parameters of the algorithm was exposed, which helps to reduce the number of simulations. Using this new approach it is shown for the (2048,1723) LDPC code that a coding gain of 0.2 dBs with respect to O2-BIT-MSA is achieved, but the performance curve suffers of an early error floor that arises at BER $= 10^{-8}$. A second modification is applied to the MO2-BIT-MSA to overcome this problem: the extrinsic information is introduced in the variable node messages. It has a great impact on the performance, but it increases the number of messages sent from the variable nodes in a factor d_v. A coding gain improvement of 0.6 dB is achieved for the (2048,1723) LDPC code. In order to analyze its behaviour at high SNR values, the CNU and VNU of the MO2-BIT-MSA were modelled in VHDL and used in an FPGA-based hardware emulator. It is shown that the proposed algorithm performs very close to the MSA (with a performance loss of 0.01 dBs) and follows the same error floor pattern as the MSA. The algorithm was also evaluated for the (16129,15372) LDPC code, which does not exhibits error floor, using the hardware emulator. In this case a performance loss of 0.2 dB is obtained without the introduction of error floor. Two main features make the MO2-BIT-MSA to be attractive for high speed implementations in FPGA devices using a fully-parallel architecture: the extreme simplicity of the check node (it only requires logic operations) and the reduced number of connections between check and variable nodes. However, its variable node is more complex than the one of the conventional MSA. In order to reduce the complexity of the VNU, an algorithm transformation is proposed that saves 9 adders and 6 constant multipliers per VNU for the (2048,1723) LDPC code. Fully-parallel architectures were modelled in VHDL and implemented in an Altera Arria 10 device. A throughput of 11 Gbps is obtained with a decoder configured for 10 iterations (with 0.2 dB of performance loss). Fully-parallel pipeline-interleaved decoders were also modelled in VHDL and implemented in the same FPGA device. Throughputs with 10 iteration of 23 Gbps and 32 Gbps are obtained when 2 and 4 codes are interleaved. Under the best knowledge of the author, these are the highest throughputs reported for FPGA devices with the used code. On the other hand, a partial parallel architecture with horizontal layered scheduling was derived for the MO2-BIT-MSA. A decoder for the (16129,15372) LDPC code using this layered architecture was modelled in VHDL and implemented in a 90 nm CMOS process. The decoder has an area of 8.1 mm^2 and reaches 19 Gbps with 10 iterations, which is nearly the double of throughput with nearly half the area of a MSA with the same architecture.

After analyzing in Chapter 1 hard-decision LDPC decoders based on reliability-based iterative proposals, RBI-MLGD, MRBI-MLGD and RBI-MSD, it was concluded that the main limitation of these algorithms is its inability to improve the coding gain of high-rate codes (with low d_v) at the error-floor region. The only algorithm that does not degrade the error correction performance for high E_b/N_0 is the RBI-MSD, but it uses soft-decision information, which yields into an increase of complexity similar to other decoders like MSA. To solve this problem, avoiding soft-decision messages, the differences between RBI algorithms

and other algorithms like MSA were studied. Two deviations were detected: i) RBI algorithms do not perform the computation of the extrinsic information in the hard-decision messages and ii) there is a lack of historical information from the previous iterations because of the emulation of turbo-decoder updates (Tu-DU). For this reason, algorithms with different combinations of Tu-DU and non-Tu-DU configurations and with the inclusion of extrinsic information calculation were designed and tested. After comparing the results of the decoding algorithms with different kinds of codes based on several code construction methods (algebraic LDPC, Reed-Solomon codes, etc.), it was concluded that the best combination is the non-Tu-DU and the calculation of extrinsic information based on hard-decision messages. This configuration obtained the best results in the waterfall region and avoided the early error-floor degradation introduced by the other hard-decision LDPC decoders such as RBI-MLGD. Moreover, the tentative decoding process was simplified by means of approximating the estimation of the decoder codeword to the variable node messages. The combination of all this features was named as HE-RBID. The proposed HE-RBID from Chapter 4 reduces between five and nine times the complexity of RBID-MSD at a cost of a performance loss (0.7-0.8 dB) in the waterfall. In addition, HE-RBID avoids the early performance degradation that RBI-MLGD and MRBI-MLGD introduce with codes of low d_v and it improves the coding gain at the error-floor region with a complexity increment of 33% in the worst scenario. Finally, HE-RBID will allow to derive architectures with higher throughput, as the message passing is reduced to one bit and, hence, the global routing is greatly reduced and, area and consumption will be smaller due to the replacement of integer operations by binary computation.

5.2. Future research lines

In this last section we address some research lines that could be carried out on the basis of this thesis.

- Threshold split-row technique proposed in [24] was demonstrated to be effective reducing the routing congestion and the area of full-parallel decoders. This technique could be adapted to the OMO-MSA to implement faster decoders.

- The MO2-BIT-MSA has a performance very close to the MSA with very low complexity. That complexity reduction has made possible the implementation in an FPGA device of a 11 Gbps decoder for the (2048,1723) LDPC code. However, this code exhibits an error floor at BER = 10^{-10} that prevents its use in certain applications, like optical communications. One working line is to apply and develop techniques to reduce the error floor, like the ones in [35], without compromise the low complexity of the algorithm and to implement a 10 Gbps decoder with low error floor in an FPGA device.

- Recently, a new dimension of parallelism unrolling the iterations of the decoding algorithm has been proposed in [36] and [37] to design ultra-high throughput LDPC decoders. This technique has serious problems of routing congestion and power consumption. The MO2-BIT-MSA, thanks to its low complexity and reduced wiring between CNUs and VNUs, could be a good candidate to be implemented in a fully-parallel unrolled architecture.

- It was shown that the proposed HE-RBID algorithm exhibits good performance with high-rate codes while it keeps low complexity and very low wiring between CNUs and VNUs, but it was not shown how those advantages are translated to a hardware decoder. Therefore, a high-speed decoder architecture based on this algorithm could be developed and implemented.

Bibliography

[1] Consultative Commitee for Space Data Systems (2016). [Online]. Available: http://cwe.ccsds.org/

[2] IEEE. (2012). *IEEE Standard for Air Interface for Broadband Wireless Access Systems*. IEEE Standard 802.16-2012. [Online]. Available: http://standards.ieee.org/findstds/standard/802.16-2012.html

[3] Working Group's Status, (2016, Feb.). *IEEE Standards Working Group on Error Correction Coding for Non-volatile Memories*. IEEE P1890. [Online]. Available: https://ieee-sa.imeetcentral.com/p1890wgpub-lic/raw/Status/IEEE_Flash_Standard_Status.pdf

[4] R. W. Hamming,"The bell system technical journal" *Bell Syst. Tech. J.*, vol. XXVI, no. 2, pp. 147 –160, 1950.

[5] D. E. Muller, "Application of Boolean algebra to switching circuit design and to error detection," *Trans. IRE Profess. Group Electron. Computers*, vol. EC-3, no. 3, pp. 6 –12, Sept. 1954.

[6] I.S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Appl. Maths*, vol. 8, no. 2, pp. 300 –304, 1960.

[7] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Inf. Control*, vol. 3, no. 1, pp. 68 –79, Mar. 1960.

[8] R. Gallager, "Low-density parity-check codes." *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21 –28, Jan. 1962.

[9] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed base on Reed-Solomon codes with two information symbols," *IEEE Commun. Lett.*, vol. 7, no. 7, pp. 317 –319, Jul. 2003.

[10] 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," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1652 –1662, Jun. 2009.

[11] Q. Diao, W. Zhou, S. Lin, and K. Abdel-Ghaffar, "A transform approach for constructing quasi-cyclic Euclidean geometry LDPC codes," in *Proc. Inf. Theory Appl. Workshop*, Feb. 2012, pp. 204 –211.

[12] C. Berrou, A. Glavieux, and P. Thitimajshi-ma, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Int. Conf. Communications*, Geneva, 1993, vol. 2, pp. 1064 –1070.

[13] E. Arikan, "Channel polarization: A method for constructing capacity achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051 –3073, 2009.

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

[15] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498 –519, Feb. 1998.

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

[17] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-Complexity Decoding of LDPC Codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288 –1299, Aug 2005.

[18] Z. Cui and Z. Wang, "Improved low-complexity low-density parity-check decoding," *IET Communications*, vol. 2, no. 8, pp. 1061 –1068, Aug 2008.

[19] T. Mohsenin and B. Baas, "Split-Row: A reduced complexity, high throughput LDPC decoder architecture," in *Proc. ICCD*, 2006, pp. 320 –325.

[20] T. Mohsenin and B. Baas, "High-throughput LDPC decoders using a multiple split-row method," in *Proc. IEEE ICASSP*, Apr. 2007, vol. 2, pp. II-13-II-16.

[21] A. Darabiha, A. Carusone, and F. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *IEEE International Symposium on Circuits and Systems*, 2006. ISCAS 2006, May 2006, pp. 4 pp.

[22] C. Zhang, Z. Wang, J. Sha, L. Li, and J. Lin, "Flexible LDPC Decoder Design for Multigigabit-per-Second Applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 1, pp. 116 –124, Jan 2010.

[23] F. Angarita, J. Valls, V. Almenar, and V. Torres, "Reduced-Complexity Min-Sum Algorithm for Decoding LDPC Codes With Low Error-Floor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 7, pp. 2150 –2158, July 2014.

[24] T. Mohsenin, D. Truong, and B. Baas, "A Low-Complexity Message-Passing Algorithm for Reduced Routing Congestion in LDPC Decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 5, pp. 1048 –1061, May 2010.

[25] A. Cevrero, Y. Leblebici, P. lenne, and A. Burg, "A 5.35 mm2 10GBASE-T Ethernet LDPC decoder chip in 90 nm CMOS," in *IEEE Asian Solid State Circuits Conference (A-SSCC)*, 2010, Nov 2010, pp. 1 –4.

[26] D. Bao, X. Chen, Y. Huang, C. Wu, Y. Chen, and X.-Y. Zeng, "A single-routing layered LDPC decoder for 10Gbase-T Ethernet in 130nm CMOS," in *17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012, Jan 2012, pp. 565 –566.

[27] F. Angarita, V. Torres, A. Perez-Pascual, and J. Valls, "High-throughput FPGA-based emulator for structured LDPC codes," in *19th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2012, Dec 2012, pp. 404 –407.

[28] Q. Diao, J. Li, S. Lin and I. F. Blake, "New Classes of Partial Geometries and Their Associated LDPC Codes," in *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 2947 –2965, June 2016.

[29] Q. Huang, J. Kang, L. Zhang, S. Lin, and K. Abdel-Ghaffar, "Two reliability-based iterative majority-logic decoding algorithms for LDPC codes," *IEEE Trans. Commun.*, vol. 57, no. 12, pp. 3597 –3606, Dec. 2009.

[30] H. Chen, K. Zhang, X. Ma, and B. Bai, "Comparisons between reliability-based iterative min-sum and majority-logic decodng algorithms for LDPC codes," *IEEE Trans. Commun.*, vol. 59, no. 7, pp. 1766 –1771, Jul. 2011.

[31] A. Darabiha, A. Carusone, and F. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 1, pp. 74 –78, Jan. 2008.

[32] *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Broadband Wireless Access Systems*, IEEE Std. 802.16-2009, May 29, 2009.

[33] Fabián Enrique Angarita Preciado, "Diseño de decodificadores de altas prestaciones para códigos LDPC," Tesis doctoral, Universidad Politécnica de Valencia, 2013.

[34] Cédric Marchand, Laura Conde-Canencia and Emmanuel Boutillon, "Architecture and Finite Precision Optimization for Layered LDPC Decoders," *Journal of Signal Processing Systems* , 2011, pp. 185 –197, July 2011.

[35] [Zhang2010] An Efficient 10GBASE-T Ethernet LDPC Decoder Design With Low Error Floors, *IEEE Journal of Solid-State Circuits,* vol. 45, no. 4, April 2010.

[36] [Schlafer2013] P. Schlafer, N. Wehn, M. Alles, and T. Lehnigk-Emden, "A new dimension of parallelism in ultra high throughput LDPC decoding," in *IEEE Workshop on Signal Processing Systems (SiPS). IEEE*, pp. 153 –158, 2013.

[37] [Balatsoukas2015] A. Balatsoukas-Stimming, M. Meidlinger, R. Ghanaatian, G. Matz, and A. Burg, "A fully-unrolled LDPC decoder based on quantized message passing," in *IEEE Workshop on Signal Processing Systems (SiPS),* pp. 1 –6, Oct. 2015.