Design of ultra high throughput rate NB-LDPC decoder

Hassan Harb

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

Université Bretagne Sud

THESE DE DOCTORAT DE

L'UNIVERSITE BRETAGNE SUD
COMMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par
Hassan Harb

Conception du décodeur NB-LDPC à débit ultra-élevé

Thèse présentée et soutenue à Lorient, le 08-11-2018
Unité de recherche : Lab-STICC UMR 6285
Thèse N° : 504

Rapporteurs avant soutenance :

ANDREAS Burg	Assistant Professeur, Ecole Polytechnique Fédérale de Lausanne
ZHENGYA Zhang	Assistant Professeur, University of Michigan

Composition du Jury :

Président :	Olivier Berder	Professeur, IUT Lannion
Examinateurs :	Catherine Douillard	Professeur, Dept. Electronique, IMT Atlantique
Dir. de thèse :	Emmanuel Boutillon	Professeur, Lab-STICC Université de Bretagne Sud
Co-dir. de thèse :	Laura Conde-Canencia	Assistant Professeur, Lab-STICC Université de Bretagne Sud

Invité

Bertrand Le Gal	Assistant Professeur, IMS Lab Bordeaux

Titre : Conception du décodeur NB-LDPC à débit ultra-élevé

Mots clés : NB-LDPC, H-CN, VN, FTSES.

Résumé : Les codes correcteurs d'erreurs Non-Binaires Low Density Parity Check (NB-LDPC) sont connus pour avoir de meilleure performance que les codes LDPC binaires. Toutefois, la complexité de décodage des codes non-binaires est bien supérieure à celle des codes binaires. L'objectif de cette thése est de proposer de nouveaux algorithmes et de nouvelles architectures matérielles de code NB-LDPC pour le décodage des NBLDPC. La première contribution de cette thése consiste à réduire la complexité du noeud de parité en triant en amont ses messages d'entrées. Ce tri initial permet de rendre certains états très improbables et le matériel requis pour les traiter peut tout simplement être supprimé. Cette suppression se traduit directement par une reduction de la complexité du décodeur NB-LDPC, et ce, sans affecter significativement les performances de décodage.

Un modéle d'architecture, appelée "architecture hybride" qui combine deux algorithmes de l'état de l'art ("l'Extended Min Sum" et le "Syndrome Based") a été proposé afin d'exploiter au maximum le pré-tri. La these propose aussi de nouvelles méthodes pour traiter les noeuds de variable dans le context d'une architecture pré-tri. Différents exemples d'implémentations sont donnés pour des codes NB-LDPC sur GF(64) et GF(256). En particulier, une architecture très efficace de décodeur pour un code de rendement 5/6 sur GF(64) est présentée. Enfin, une problématique récurrente dans les architectures NB-LDPC, qui est la recherche des P minimums parmi une liste de taille N_s, est abordée. La thése propose une architecture originale appelée first-then-second minimum pour une implantation efficace de cette tâche.

Title : Design of ultra high throughput rate NB-LDPC decoder

Keywords : NB-LDPC, H-CN, VN, FTSES.

Abstract The Non-Binary Low Density Parity Check (NB-LDPC) codes constitutes an interesting category of error correction codes, and are well known to outperform their binary counterparts. However, their non-binary nature makes their decoding process of higher complexity. This PhD thesis aims at proposing new decoding algorithms for NB-LDPC codes that will be shaping the resultant hardware architectures expected to be of low complexity and high throughput rate. The first contribution of this thesis is to reduce the complexity of the Check Node (CN) by minimizing the number of messages being processed. This is done thanks to a pre-sorting process that sorts the messages intending to enter the CN based on their reliability values, where the less likely messages will be omitted and consequently their dedicated hardware part will be simply removed. This reliability-based sorting enabling the processing of only the highly reliable messages induces a high reduction of the hardware complexity of the NB-LDPC decoder. Clearly, this hardware reduction must come at no significant performance degradation. A new Hybrid architectural CN model (H-CN) combining two state-of-the-art algorithms - Forward-Backward CN (FB-CN) and Syndrome Based CN (SB-CN) - has been proposed. This hybrid model permits to effectively exploit the advantages of pre-sorting.

This thesis proposes also new methods to perform the Variable Node (VN) processing in the context of pre-sorting-based architecture. Different examples of implementation of NB-LDPC codes defined over GF(64) and GF(256) are presented. For decoder to run faster, it must become parallel. From this perspective, we have proposed a new efficient parallel decoder architecture for a 5/6 rate NB-LDPC code defined over GF(64). This architecture is characterized by its fully parallel CN architecture receiving all the input messages in only one clock cycle. The proposed new methodology of parallel implementation of NB-LDPC decoders constitutes a new vein in the hardware conception of ultra-high throughput rate decoders. Finally, since the NB LDPC decoders requires the implementation of a sorting function to extract P minimum values among a list of size N_s, a chapter is dedicated to this problematic where an original architecture called First-Then-Second-Extrema-Selection (FTSES) has been proposed.

Acknowledgement

It is my honor to present this work for my parents who encouraged and supported me.

I am thankful for who trusted and chose me to be the one who works with them, I mean Dr. Ali Al Ghouwayel, Prof. Emmanuel Boutillon, Prof. Ali Alaedine and Dr. Laura Conde-Canencia. Fruitful discussions of experiments with them are gratefully acknowledged. With them, I knew what devotion, dedication, determination and discipline mean. Their company taught me how to perform knowledge in a real technical work.

Sincerely, I highly appreciate the dedicated and effective effort and ideas that were proposed by Dr. Cédric Marchand to make this work successful.

Of course I cannot but thank Dr. Bertarnd Le Gal for the unforgettable knowledge and care he provided me during my accommodation in Bordeaux, where I learned a new tool that was extremely useful to complete this work and it will be helpful in my future.

The authors would like to acknowledge the valuable comments and suggestions of the reviewers Prof. Zhang Zhengya and Prof. Andreas Burg, which have improved the quality of this work.

The authors would like to acknowledge the jury members Prof. Olivier Berder and Prof. Catherine Douillard who paid attention for our work and gave us their precious time to have fruitful and objective discussion with them.

Hassan HARB.

Contents

List of Figures

List of Tables

Acronyms

A	Area
AE	Area Efficiency
AER	Area Efficiency Ratio
APP	A Posteriori Probability
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BP	Belief Propagation
BPSK	Binary Phase-Shift Keying
$\mathbb{C}$	Set of complex numbers
C	Critical Path
CO	Comparator Only
C	Comparator
CS	Comparator Swap
CL	Cycle Latency
CG	Control Generator
CC	Clock Cycle
CU	Control Unit
CAM	Content Addressable Memory
CN	Check Node
CNP	Check Node Processor
CR	code rate equal to $\frac{K}{N} = 1 - \frac{d_v}{d_c}$
DBV	Discard Binary Vector
DU	Decorrelation Unit

DPU	Decorrelation Processor Unit
DMU	Decision Making Unit
DMUR	DMU Reordering
DeBl	Decorrelation + Devision Block
DMUS	DMU Storage
DAVINCI	Design And Versatile Implementation of Non-binary wireless Communications based on Innovative LDPC codes
DVB	Digital Video Broadcast
EF-CN	Extended Forward Check Node
EE	Energy Efficiency
ESU	Elementary Sorting Unit
EMB	Encoded Modulated Bits
ES	Extrinsic Storage
ECN	Elementary Check Node
EMS	Extended Min-Sum
FB-CN	Forward Backward Check Node
FTSES	First then Second Extrema Selection
FER	Frame Error Rate
FMU	First Minimum Unit
FPHD	Fully Parallel Hybrid Decoder
FFT	Fast Fourier Transform
FIFO	First-In First-Out
Gel/s	Gega elements per second
GFRB	GF Routing Block
GF	Galois Field
H-CN	Hybrid Check Node
H	Parity Check Matrix
HEP	High Energy Efficiency
HD	Hard Decision
I	Intrinsic messages
IL-MwBRB	Improved Layered Multiple-symbol-reliability weighted Bit-Reliability Based

IW	Input Wrapper
IS	Intrinsic Storage
IFFT	Inverse Fast Fourier Transform
LDPC	Low Density Parity Check
LLR	Log Likelihood Ratio
Lab-STICC	Laboratoire des Sciences et Techniques de l'Information de la Communication et de la Connaissance
LDPC	Low-Density Parity-Check
LLR	Log-Likelihood Ratio
LUT	Look Up Table
MS	Min Sum
NB	Non Binary
NB-LDPC	Non-Binary LDPC
NR	Normalization + Reordering
O.S	Occupied Slices
OG	Outputs Generator
OMS	Offset Min Sum
OW	Output Wrapper
P-FB	FB-CN with Presorting
P	Power
P_{clk}	Clock Period
PCG	Possible Candidates Generator
PS	Parallel Structure
PE	Power Efficiency
PER	Power Efficiency Ratio
PTB	Parity Test Block
PCM	Parity Check Matrix
QC-LDPC	Quasi Cyclic LDPC
$\mathbb{R}$	Set of real numbers
RE	Redundant Elimination
RTL	Register Transfer Level
RS	Redundant Suppression
RAM	Random Access Memory

ROM	Read Only Memory
SB-CN	Syndrome Based Check Node
S-FB	FB-CN without Presorting
SPC	Skip Processing Controller
SN	Syndrome Node
SA	Saving Amount
SRE	Sorter + Redundant Elimination
SU	Sorting Unit
SMU	Second Minimum Unit
SCRB	Stopping Criteria Router Block
SM	Switching + Multiplication
T-MM	Trellis Min Max
TS	Tree Structure
TEC	Throughput Error Computation
VN	Variable Node
VNP	Variable Node Processor
VHDL	VHSIC Hardware Description Language
$\mathbb{N}$	Set of natural integers
$\mathbb{Z}$	Set of relative integers
WIFI	Wireless Local Area Network
WIMAX	Worldwide Interoperability for Microwave Access

Parameters

q	Order of GF
m	Number of bits to represent a GF value
n_m	Number of considered GF elements among q GF elements
n_{iter}	Number of maximum iterations
M^+	LLR value of M
$M^{\oplus}$	GF value of M
d_c	Degree of the CN (number of VNs connected to a CN)
d_v	Degree of the VN (number of CNs connected to a VN)
K	number of information symbols
M	number of redundant symbols
$N = K + M$	code length (number of VNs)
$h_{i,j}$	Non-zero value in PCM that connects CN_i with VN_j
$M_{v_j p_i}$	Messages sent from VN_j to CN_i
$M_{p_i v_j}$	Messages sent from CN_i to VN_j
d_v	Degree of the variable node (number of CNs connected to a VN)
d_c	Degree of the check node (number of VNs connected to a CN)

Chapter 1

Introduction

The thesis is a part of a collaborative framework between the Université Bretagne Sud (UBS, France) and the Lebanese Univerisity (LU, Lebanon) and has been supervised by Prof. Emmanuel Boutillon, Prof. Ali Alaeddine, Dr. Ali Al Ghouwayel and Dr. Laura Conde-Canencia. During these years I also collaborated with Dr. Cédric Marchand who provided significant inputs for my work.

In 1948, Shannon showed that reliable communications are possible thanks to error control coding [5]. Since then, numerous error-correcting schemes have been proposed including algebraic and convolutional codes. With the invention of Turbo codes in the early 90s [12], followed by the rediscovery of LDPC [15, 16], iterative decoding algorithms based on trellises or graphs became a main topic of study. Recently, other decoding approaches have been proposed (e.g. with the introduction of Polar codes). Today, error-correcting codes are ubiquitous and adopted in almost every modern digital communication system for wireless communications, sensor networks and deep-space communications, among others. New-generation standards and other emerging applications demand codes with near-optimal error-correcting capabilities. However, the design and implementation of those high-performance error-correcting codes also face many challenges that include low energy consumption, high throughput and low implementation area.

Even if most of the standardized coding schemes are binary, non-binary LDPC codes have been proven to outperform convolutional Turbo codes and binary LDPC codes. In fact, this new family of codes retains the benefits of steep waterfall region for short codewords (typical of Turbo codes) and low error floor (typical of binary LDPC). Another advantage of non-binary LDPC codes is that, compared to binary LDPC, they generally present higher girths, which leads to better decoding performance. Moreover, since non-binary LDPC codes are defined on high-order fields, there is a closer connection between non-binary LDPC and high-order modulation schemes. However,

the main drawback of non-binary LDPC codes is their increased decoding complexity.

The work presented in this report deals with the study of new non-binary LDPC decoding algorithms for high order fields ($q \geq 64$) and their associated architectures. We aim at reducing the hardware complexity and/or increase the area and throughput efficiency. We mainly focus on the Extended Min Sum (EMS) decoding algorithm because of its competitive error-correcting performance. During this PhD we have mainly considered the three following goals: first, the reduction of the decoder cost by eliminating the inner elements not relevant in the output generation. This first goal implies the sorting of the input list, thus, the second goal has been the design of an efficient architecture for this sorting task. The third and final goal was the implementation of a highly parallel decoder for non-binary LDPC codes.

So far, the results obtained through this PhD have been spread in the scientific community through the following publications:

Cédric Marchand, Emmanuel Boutillon, Hassan Harb, Laura Conde-Canencia, Ali Al Ghouwayel, "Extended-Forward Architecture for Simplified Check Node Processing in NB-LDPC Decoders", IEEE International Workshop on Signal Processing Systems (SIPS'2017), Dallas, United States. Oct. 2016.

Hassan Harb, Cédric Marchand, Laura Conde-Canencia, Emmanuel Boutillon, Ali Al Ghouwayel, "Pre-sorted Forward-Backward NB-LDPC Check Node Architecture", IEEE International Workshop on Signal Processing Systems (SIPS'2016), Lorient, France, Oct. 2017.

Titouan Gendron, Hassan Harb, Alban Derrien, Cédric Marchand, Laura Conde-Canencia, Bertand Le Gal and Emmanuel Boutillon, "Demo: Construction of good Non-Binary Low Density Parity Check codes", Demo night at SIPS'2017, Lorient, France, Oct. 2017.

Hassan Harb, Emmanuel Boutillon, Bertrand Le Gal, "Real-time evaluation of NB-LDPC codes thanks to HLS-based hardware emulation", Demo night at DASIP'2018, Porto, Portugal, Oct. 2018.

Cédric Marchand, Emmanuel Boutillon, Hassan Harb, Laura Conde-Canencia and Ali Al Ghouwayel, "Hybrid Check Node Architectures for NB-LDPC Decoders", Accepted in IEEE Transactions on Circuits And Systems-I, August 2018.

Therefore, this manuscript is organized as follows:

Chapter 2: This chapter introduces LDPC codes as well as the main decoding algorithms and their associated architectures. Section 1 presents notation and definitions related to binary and NB-LDPC codes. Section 2 de-

scribes several decoding algorithms such as the Belief-Propagation and the Min-Max. Section 3 and 4 show some of the existing serial CN and VN architectures respectively. Section 4 reviews processing schedules that are considered in most of the literature. Finally, Section 6 highlights some of the state-of-the-art high-throughput decoding architectures.

Chapter 3: This chapter presents my different contributions to improve the existing NB-LDPC decoder architectures. These contributions are detailed at the different blocks of the decoder. Section 1 shows the modifications on the CN processor block and Section 2 presents the modifications on the VN processor block.

Chapter 4: This chapter is dedicated to parallel pipelined architectures providing higher throughput and better hardware efficiency than the serial ones. Section 1 shows a new technique to implement the LLR generator that is included in most of the NB-LDPC decoder algorithms. Then, section 2 presents the proposed parallel pipelined sorter algorithm along with an example of its extended approach.

Chapter 5: The proposed high-throughput fully-parallel pipelined NB-LDPC decoder architecture is shown in this chapter. Section 1 introduces the considered NB-LDPC code along with the decoding algorithm and the simulation results. Section 2 shows the global decoder architecture where the parallelism of the exchanged data is presented along with the memorization system and the timing diagram. Section 3 presents in details the architecture of each block. Section 4 shows the timing diagram of the global processing of the decoder where the frame interleaving is demonstrated. The chapter continues with section 5 where the synthesis analysis is shown. Finally, section 6 shows the global architecture of the hardware emulation.

Chapter 6: This chapter concludes the work and presents our perspectives.

Finally, it is worth mentioning that not all the work done in the 3 years period of my PhD has been included in this document. I have decided to focus only on hardware implementation. Thus, concerning NB-LDPC matrix construction, in a few words, I have contributed to the generation of the Lab-STICC NB-LDPC database[(1)] by writing a gecode program[(2)] (a constraint programming Frame work) that optimizes both the girth of the matrix and the affectation of GF coefficients on non-nul positions (see [56]). Recently, I also proposed a new NB-LDPC structure that allows, thanks to a trick, to use a NB-LDPC decoder of a certain rate to decode NB-LDPC codes of higher rate, opening thus the path toward hardware flexibility.

[(1)] http://www-labsticc.univ-ubs.fr/nb_ldpc/MatricesDir/toto.html
[(2)] www.gecode.org

Chapter 2

NB-LDPC codes: Principles, Decoding Algorithms and Architectures

This chapter first introduces NB-LDPC codes. Then, two state-of-the-art check node algorithms are recalled and two different schedules of decoding process (Layered and Flooding) are described. Finally, some of the NB-LDPC state-of-the-art decoding architectures are presented. (Section 2.1 and most of Section 2.2 are derived from [7]). For mathematical background about the GF definition and construction, the reader is referred to Appendix A.

2.1 Non-Binary LDPC codes defined on a Galois field

An LDPC code is a linear block code defined by a sparse Parity Check Matrix (PCM), denoted by H, of dimensions $M \times N$ designed over GF($q = 2$). This code is binary since its symbols belong to GF(2) = $\{0, 1\}$. The number of rows, M, corresponds to the number of parity check constraints of the code. The number of columns, N, corresponds to the length of the codewords. A codeword consists of K information symbols and $M = N - K$ redundancy symbols added by the encoder. The parity check constraints of H must be respected by the codewords in the construction. Thus, a message C of length N is a codeword if and only if $C.H^T$=0, where H^T is the transposed matrix of H.
Let us consider the following example of a PCM with $M = 4$ and $N = 6$:

$$H = \begin{bmatrix} h_{0,0} & h_{0,1} & h_{0,2} & 0 & 0 & 0 \\ 0 & h_{1,1} & 0 & h_{1,3} & h_{1,4} & 0 \\ h_{2,0} & 0 & 0 & h_{2,3} & 0 & h_{2,5} \\ 0 & 0 & h_{3,2} & 0 & h_{3,4} & h_{3,5} \end{bmatrix}$$

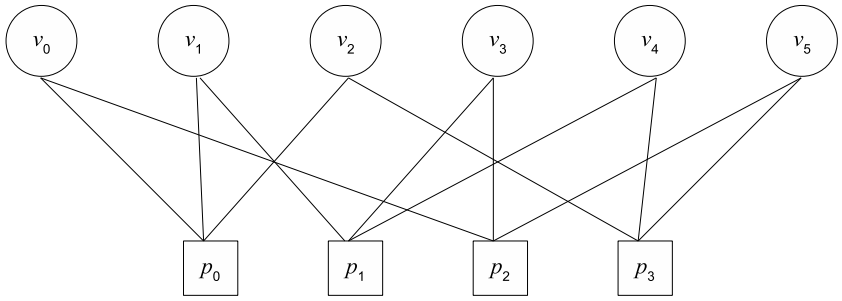

Figure 2.1: A graphic representation of an LDPC code with a bipartite graph.

A codeword $C = [c_0, c_1, c_2, c_3, c_4, c_5]$ satisfies the four following equations:

$$h_{0,0}.c_0 + h_{0,1}.c_1 + h_{0,2}.c_2 = 0 \tag{2.1}$$

$$h_{1,1}.c_1 + h_{1,3}.c_3 + h_{1,4}.c_4 = 0 \tag{2.2}$$

$$h_{2,0}.c_0 + h_{2,3}.c_3 + h_{2,5}.c_5 = 0 \tag{2.3}$$

$$h_{3,2}.c_2 + h_{3,4}.c_4 + h_{3,5}.c_5 = 0 \tag{2.4}$$

An LDPC code can also be represented by a bipartite graph (or Tanner graph) [21]. This kind of graph provides a complete description of the structure of the code and also helps to describe the decoding algorithms as will be explained in Section 2.2. A bipartite graph composed of two sets of nodes such that two nodes of the same set are connected only through one node of the other set. In the case of an LDPC code, we talk about the set of parity Check Nodes (CN) and the set of Variable Nodes (VN). A CN represents a row in the PCM (or equivalently a parity constraint) and a VN represents a column (or equivalently a symbol of the codeword). Consequently, the bipartite graph associated to an LDPC code represented by a PCM H of dimensions $M \times N$ is composed of M CNs and N VNs. A CN p_i is related to a VN v_j if the element of the i^{th} row and j^{th} column of the PCM is non-zero (or equivalently, if the j^{th} symbol of the codeword participates in the i^{th} parity constraint). Thus, the example of the matrix H mentioned before can be represented by a Tanner graph as shown in Fig. 2.1.

The number of non-zero symbols in each column of PCM and the number of non-zero symbols in each row are respectively denoted by d_v and d_c. An LDPC code is called regular if d_v is constant for all the columns in H and $d_c = \frac{N}{M}.d_v$ is constant for all rows. Otherwise, the code is called irregular. Although irregular codes have better performance because of their highly randomized structure, regular codes are usually structured codes which makes them hardware friendly from a decoding perspective. It is possible to locate the regularity of a code using its bipartite graph. The code is regular if the number of outgoing edges of each VN and the number of outgoing edges

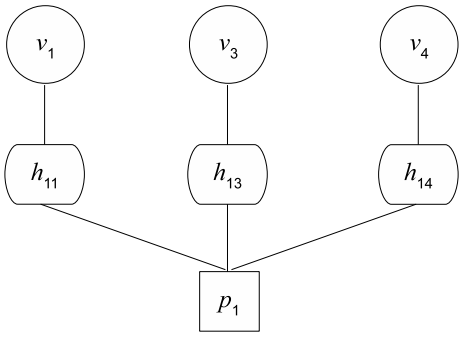

Figure 2.2: A graphic representation for a CN in case of NB-LDPC code [7].

of each CN are constant. Therefore, d_v and d_c are called the connectivity degrees of the VNs and CNs respectively. In the case of a regular LDPC code defined by a full rank matrix, i.e, no row of the matrix is linear combination of other rows, the code rate R of the code can be expressed as a function of d_v and d_c:

$$R = \frac{K}{N} = \frac{N - M}{N} = 1 - \frac{d_v}{d_c}. \tag{2.5}$$

In this manuscript, we consider the case of LDPC codes defined on Galois fields GF($q = 2^m$), $m > 1$, and known as NB-LDPC. Thus, the elements of the PCM matrix belong to Galois field GF($q = 2^m$), $m > 1$ and the matrix products of the parity equations use the internal composition laws of the Galois field. Therefore, a new class of nodes called the permutation nodes are added to the bipartite graph of Fig. 2.1 to model the multiplication of the symbols of the codeword by the non-zero elements of the PCM. Fig. 2.2 illustrates the partial bipartite graph of equation (2.2) by adding the permutation nodes that correspond to the elements $h_{1,1}$, $h_{1,3}$ and $h_{1,4}$.

Binary LDPC codes have asymptotic performance approaching the Shannon limit [12, 13]. However, for small or medium size codewords, the performance of the binary LDPC codes degrades considerably. It is shown in [19] that this loss can be compensated by using NB-LDPC codes of high cardinality. In addition, the high cardinality of the codes ensures better resistance to packet errors [20]. However, the performance gain introduced with high Galois fields significantly increases the complexity of the decoding algorithms and their practical implementations.

Next section describes some of the state-of-the-art NB-LDPC decoding algorithms: *Belief Propagation* (BP) [19], Log-BP [22], Min-Sum [22], Extended Min-Sum (EMS) [24, 25] and Min-Max [27] algorithms.

2.2 Iterative decoding algorithms for NB-LDPC codes

BP decoding algorithms are based on the bipartite graph defined by the NB-LDPC code. They are also called message-passing algorithms because, at each iteration, messages are transmitted from CNs to VNs and vice versa. We distinguish two types of messages:

- ▷ Intrinsic or *a priori* messages are computed from the channel observations. They are called intrinsic because the information they contain only comes from the channel. At the initialization stage, these messages are directly sent to all the CNs.
- ▷ Extrinsic messages are computed from messages coming from other branches of the graph. Outgoing extrinsic messages from a VN are computed from an intrinsic message and extrinsic messages from the connected CNs. Outgoing extrinsic messages from a CN are computed from incoming extrinsic messages (from the connected VNs) and with the local parity constraint.

The decoder should be able to converge on a valid codeword after a finite number of iterations. In practice, the decoding algorithm can be stopped according to two criteria. The simplest is to set the number of iterations independently of the convergence of the decoder. The second criterion, which permits to reduce the latency of the decoder, consists in stopping the decoding as soon as it converges to a valid codeword (an estimated codeword $\hat{C}$ is valid if it satisfies the syndrome $\hat{C}.H^T = 0$). However, to avoid an infinite execution in case the decoder fails to converge to a valid codeword, a maximum number of iterations is fixed.

In the BP algorithm, the exchanged messages are *a posteriori* probabilities calculated on the symbols of the codeword. However, the BP algorithm [19] suffers from a prohibitive computational complexity, dominated by $O(q^2)$, which mainly comes from the calculations carried out during the update of the parity constraints.

Barnault *et al.* proposed in [21] the FFT-BP algorithm in which the updates of the parity constraints are made in the frequency domain. This transforms the convolution products into simple multiplications. Thus, additional operations of Fourier transform, direct and inverse, are added between the VNs and the CNs to ensure the transition from the probability domain to the frequency domain, and vice versa. Although the complexity of the FFT-BP algorithm is considerably reduced to the order of $O(q\log(q))$, a large number of multiplications remains necessary to perform the update of the nodes in the graph.

The log-BP algorithm [22] performs the four deccoding steps in the logarithmic domain to allow a hardware layout less sensitive to quantization errors, and therefore better suited to fixed-point arithmetic. However, the update of the CNs always requires a

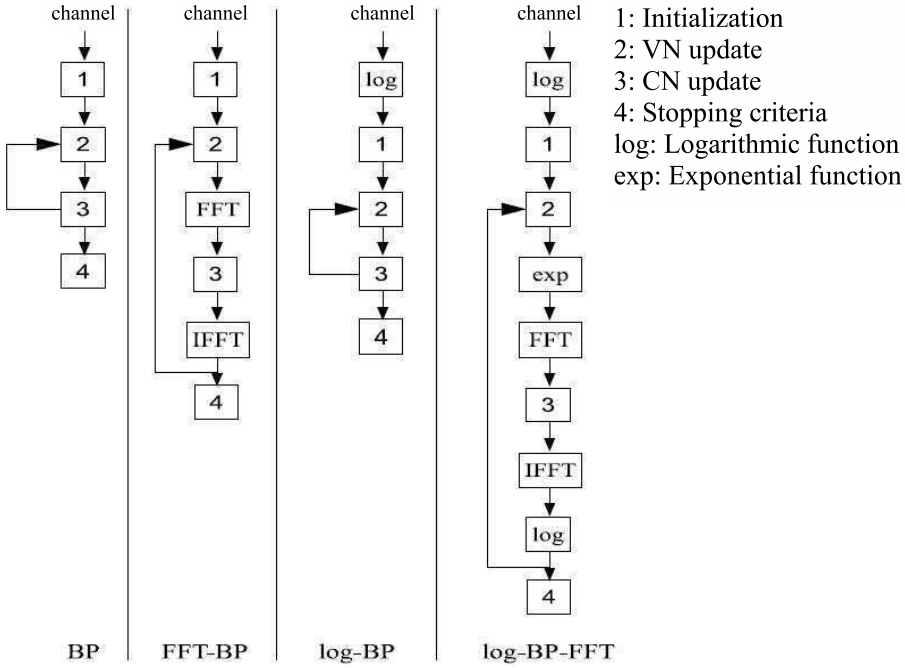

Figure 2.3: The main algorithms for optimal NB-LDPC decoding [7].

large amount of calculation and the complexity of the decoder remains dominated by $O(q^2)$. A direct combination of the FFT-BP and log-BP algorithms is not advantageous because the calculation of the Fourier transform is very complex in the logarithmic domain.

To simultaneously benefit from the advantages of the FFT-BP and log-BP algorithms, Song et al. proposed in [20] the log-BP-FFT algorithm. In this algorithm, the VNs are processed in the logarithmic domain. The extrinsic messages of the VNs undergo a double transformation to pass from the logarithmic domain to the probabilities domain and from the probabilities domain towards the frequency domain in which the CNs will be processed. The extrinsic messages of the CNs in turn undergo a double transformation to return back to the logarithmic domain of the VNs. However, the log-BP-FFT algorithm requires look-up tables to ensure the conversion between the probabilities domain and the logarithmic domain. These tables have the disadvantage of consuming a lot of memory resources, a consumption that increases with the degree of parallelism of the decoder. Fig. 2.3 illustrates the steps of the different decoding algorithms mentioned above.

The BP [19], FFT-BP [21], log-BP [22] and log-BP-FFT [20] algorithms are optimal decoding algorithms because they do not use any mathematical approximation to reduce the complexity of the decoding. The BP algorithm and its variants guarantee optimal decoding performance but they are not of great interest for a hardware imple-

mentation. Therefore, other algorithms based on approximations of the BP algorithm are proposed in order to ensure a reasonable performance/complexity tradeoff. We cite mainly the algorithm Min-Sum [22] and its variant EMS (Extended Min-Sum) [24,25]. A detailed comparison of the optimal and suboptimal algorithms cited above can be found in [26]. Besides the EMS algorithm, there is also the Min-Max algorithm [27] which can be considered as an approximation of the Min-Sum algorithm, and which consequently provides poorer performance.

In what follows, we detail the algorithms BP, log-BP, Min-Sum, EMS and Min-Max. We adopt the following mathematical convention: Let $V = [v_0, v_1, \ldots, v_{n-1}]$ be vector composed of n elements. If i is a positive integer or zero, the notation $V(i)$ indicates the element of position i in V. If $\beta \in \mathrm{GF}(2^m)$, the notation $V[\beta]$ indicates the element associated with the symbol β in V.

2.2.1 BP algorithm

Let $c = [c_0, c_1, \ldots, c_{N-1}]$, $c_i \in \mathrm{GF}(q)$, be the transmitted codeword. The decoding algorithm should converge toward a valid codeword $\hat{c} = [\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_{N-1}]$ from $y = [y_0, y_1, \ldots, y_{N-1}]$, the noisy version of c. The decoding is successful if $\hat{c} = c$.

In the BP algorithm, the intrinsic information of the VN v_i is a q-ary vector of *a posteriori* probabilities defined as:

$$I_i = [p(v_i = \beta_0|y_i), p(v_i = \beta_1|y_i), \ldots, p(v_i = \beta_{q-1}|y_i)] \tag{2.6}$$

where $p(a|b)$ is the conditional probability of a given b.

Let $i = 0, 1, \ldots, M-1$ and $j = 0, 1, \ldots, N-1$. If the element h_{ij} of the PCM H is not zero then $M_{v_j p_i}$ denotes the message sent by the VN v_j to the CN p_i and $M_{p_i v_j}$ the message sent by the CN p_i to the VN v_j. The steps of the BP algorithm are:

(a) Initialization: Each VN v_i transmits its intrinsic information to the CNs connected to it.

(b) Permutation: Before entering CN p_i, the message $M_{v_j p_i}$ is multiplied by the non-zero element h_{ij} of the PCM. The resultant message $\tilde{M}_{v_j p_i}$ is computed as:

$$\tilde{M}_{v_j p_i}[\beta] = M_{v_j p_i}[\beta.h_{ij}] \qquad \beta \in \mathrm{GF}(q) \tag{2.7}$$

(c) CN update: the update of the CN p_i is given by:

$$M_{p_i v_j}[\beta] = \sum_{\substack{\sum_{s \neq j} \theta_s = \beta \\ h_{is} \neq 0}} \prod_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{v_s p_i}[\theta_s] \tag{2.8}$$

where β and θ_s are $\mathrm{GF}(q)$ symbols. The update of p_i is done by calculating the probability of all symbol combinations that satisfy the parity equation.

(d) Inverse permutation: Before entering VN v_j, the message $M_{p_i v_j}$ is divided by the non-zero element h_{ij} of the PCM. The resultant $\tilde{M}_{p_i v_j}$ is obtained as:

$$\tilde{M}_{p_i v_j}[\beta] = M_{p_i v_j}[\beta.h_{ij}^{-1}] \qquad \beta \in \mathrm{GF}(q) \tag{2.9}$$

(e) VNs update: A VN v_j receives d_v messages $\tilde{M}_{p_i v_j}$ and generates d_v messages $M_{v_j p_i}$. Outgoing messages from v_j are computed by (2.10). Each outgoing message is a function of all incoming messages to v_j except the one from p_i.

$$M_{v_j p_i}[\beta] = \mu_{v_j p_i} \times I_j[\beta] \times \prod_{\substack{s \neq i \\ h_{sj} \neq 0}} \tilde{M}_{p_s v_j}(\beta) \qquad \beta \in \mathrm{GF}(q) \tag{2.10}$$

where $\mu_{v_j p_i}$ is a normalization factor such that $\sum_{\beta \in \mathrm{GF}(q=2^m)} M_{v_j p_i}[\beta] = 1$.

(f) Estimation of the codeword: at every iteration, the *a priori* probability vector (APP_j) is computed as follows:

$$\mathrm{APP}_j[\beta] = \mu_{v_j}.I_j[\beta].\prod_{h_{s,j} \neq 0} \tilde{M}_{p_s v_j}(\beta) \qquad \beta \in \mathrm{GF}(q) \tag{2.11}$$

where μ_{v_j} is a normalization factor such that $\sum_{\beta \in GF(q=2^m)} \mathrm{APP}_j[\beta_i] = 1$. The decision is made based on selecting the symbol of highest probability in APP_j as:

$$\hat{c}_j = \underset{\beta \in \mathrm{GF}(q)}{\mathrm{argmax}}\{\mathrm{APP}_j[\beta]\} \qquad j = 0, 1, \ldots, N-1 \tag{2.12}$$

If the set of symbols of $\hat{c}_j$ forms a codeword then the decoding is considered as finished.

2.2.2 Log-BP algorithm

The reliability of a symbol can be measured by the Log-Likelihood Ratio (LLR) as defined in the following equation:

$$\mathrm{LLR}(\beta) = \ln\frac{p(v_j = \beta|y_j)}{p(v_j = \beta_0|y_j)} \qquad \beta \text{ and } \beta_0 \in \mathrm{GF}(q) \tag{2.13}$$

On the one hand, replacing the probabilities with LLRs in (2.8, 2.10, and 2.11) transforms the multiplication operations into additions and on the other hand reduces the quantization errors. Thus, in the log-BP algorithm, the intrinsic information of a VN v_j is defined by:

$$I_j = [0, \ln\frac{p(v_j = \beta_1|y_j)}{p(v_j = \beta_0|y_j)}, \ldots, \ln\frac{p(v_j = \beta_{q-1}|y_j)}{p(v_j = \beta_0|y_j)}] \tag{2.14}$$

The messages circulating on the bipartite graph are composed by LLRs. The log-BP algorithm keeps the same steps of the BP algorithm while modifying the update equations. The update of a VN v_j is described as:

$$M_{v_j p_i}[\beta] = I_j[\beta] + \sum_{\substack{s \neq i \\ h_{sj} \neq 0}} \tilde{M}_{p_s v_j}(\beta) \qquad \beta \in \mathrm{GF}(q) \tag{2.15}$$

The update of CN p_i is computed as:

$$M_{p_i v_j}[\beta] = \ln\Big(\sum_{\substack{\sum_{s \neq j} \theta_s = \beta \\ h_{is} \neq 0}} \exp\Big(\sum_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{v_s p_i}[\theta_s] \Big) \Big) \qquad \beta \in \mathrm{GF}(q) \tag{2.16}$$

Finally, the update of the *a priori* information can be written as:

$$\mathrm{APP}_j[\beta] = I_j[\beta] + \sum_{h_{s,j} \neq 0} \tilde{M}_{p_s v_j}(\beta) \qquad \beta \in \mathrm{GF}(q) \tag{2.17}$$

2.2.3 Min-Sum algorithm

The Min-Sum algorithm is proposed in [24] to reduce the complexity of the log-BP algorithm by making an approximation of (2.16). Indeed, in the Min-Sum algorithm, the update of CN p_i can be written as:

$$M_{p_i v_j} \approx \max_{\substack{\sum_{s \neq j} \theta_s = \beta \\ h_{is} \neq 0}} \Big\{ \sum_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{v_s p_i}[\theta_s] \Big\} \tag{2.18}$$

Thus, the Min-Sum algorithm simplifies the decoder by eliminating the lookup tables needed to implement the exponential and logarithmic functions, and by minimizing the number of the arithmetic operations.

2.2.4 EMS algorithm and its variants

The EMS caracteristics can be summarized as:

Truncation of the exchanged messages: To further simplify the Min-Sum decoder, the authors in [24] introduced the idea of truncating the messages that circulate on the bipartite graph from q to the n_m most reliable symbols. However, the value of n_m must be carefully chosen to avoid performance loss.

Extra memories for the GF symbols: In the log-BP algorithm, the messages are vectors composed of q unsorted reliability values. In addition, it is not necessary

to explicitly indicate the value of the symbol associated with each of the reliabilities since it can be easily deduced by its position in the message. Due to truncation, messages from the EMS algorithm must be sorted and symbol values must be explicitly mentioned. The messages that circulate in the bipartite graph are represented as $M = [(\mathrm{LLR}(\theta_k), \theta_k)]_{0 \leq k < n_m - 1}$, such that θ_k is a variable in GF(q) and $\mathrm{LLR}(\theta_{k'}) \geq \mathrm{LLR}(\theta_{k''})$ if $k' < k''$. In the following, $M^{\oplus}$ represents the partial message that contains the set of GF symbols of message M and M^{+} represents the vector that contains the set of LLRs of the message M. The most reliable symbol in M is $M^{\oplus}(0)$ and the less reliable one is $M^{\oplus}(n_m - 1)$.

Compensating candidates: The truncation of messages leads to performance degradation that can be compensated by using a constant reliability value noted γ for symbols not retained during truncation. The value of γ is calculated as follows:

$$\gamma = M^{+}(n_m - 1) + O \tag{2.19}$$

in which O is a scalar that can be determined by simulation to minimize the Bit Error Rate (BER) or theoretically as described in [25].

The steps of EMS algorithm can be summarized as follows:

(a) Initialization: each VN v_i sends the most n_m reliable intrinsic information symbols to its set of connected CNs.

(b) VNs update: A VN v_i receives d_v messages $\tilde{M}_{p_i v_j}$ and a compensation scalar γ_i associated to each $\tilde{M}_{p_i v_j}$. The value of γ_i is determined by (2.19). The sorted message $M_{v_j p_i}$ contains the n_m most reliable symbols by combining the intrinsic information with the incoming messages except p_i itself. The reliability of a symbol $M^{\oplus}_{v_j p_i}(k)_{k=0,1,\ldots,n_m-1}$ is obtained by:

$$M^{+}_{v_j p_i}(k) = I_j[M^{\oplus}_{v_j p_i}(k)] + \sum_{\substack{s \neq i \\ h_{s,i} \neq 0}} W_s(k) \tag{2.20}$$

such that

$$W_s(k) = \begin{cases} \tilde{M}_{p_s v_j}[M^{\oplus}_{v_j p_i}(k)] & \text{if} \quad M^{\oplus}_{v_j p_i}(k) \in \tilde{M}_{p_s v_j} \\ \gamma_s & \text{else} \end{cases}$$

(c) Permutation: each symbol of $M_{v_j p_i}$ will be multiplied by the element $h_{ij} \neq 0$ of the PCM.

$$\tilde{M}^{\oplus}_{v_j p_i}(k) = h_{ij} \cdot M^{\oplus}_{v_j p_i}(k) \qquad k = 0, 1, \ldots, n_m - 1 \tag{2.21}$$

(d) CNs update: the reliability of a symbol of an outgoing message is calculated as in 2.18. $M_{p_i v_j}$ outgoing messages contain the most reliable n_m symbols.

(e) Inverse permutation: each symbol of $M_{p_i v_j}$ will be divided by the element $h_{ij} \neq 0$ of PCM.

$$\tilde{M}^{\oplus}_{p_i v_j}(k) = h_{ij}^{-1} \cdot M^{\oplus}_{p_i v_j}(k) \qquad k = 0, 1, \ldots, n_m - 1 \tag{2.22}$$

(f) Estimation of the codeword: at every iteration, each VN v_j updates a vector of *prior* LLRs APP_j as:

$$APP_j[\beta] = I_j[\beta] + \sum_{h_{s,j} \neq 0} W_s(\beta) \qquad \beta \in \mathrm{GF}(q) \tag{2.23}$$

such that

$$W_s[\beta] = \begin{cases} \tilde{M}^{+}_{p_s v_j}[\beta], & \text{if } \beta \in \tilde{M}^{\oplus}_{p_s v_j} \\ \gamma_s, & \text{else} \end{cases}$$

Finally the decision is taken based on (2.12).

2.2.5 Min-Max algorithm

The LLR value as defined in sections 1.3.1 and 1.3.3 may assume negative values. However, it would be easier to deal only with positive values. Therefore, the author of [27] proposed the following definition of the LLRs:

$$LLR(\beta) = -\ln \frac{p(x = \beta | y)}{\max\limits_{\theta \in \mathrm{GF}(2^m)} \{p(x = \theta | y)\}} \qquad \beta \in \mathrm{GF}(2^m) \tag{2.24}$$

where $y = (y_0, y_1, \ldots, y_{m-1})$ is the channel observation and $x = (x_0, x_1, \ldots, x_{m-1})$ is the transmitted symbol.

In this definition, the normalization is done by the probability of the most reliable symbol. It follows that the LLR of this symbol is always zero and the LLRs of the other symbols are positive.

Also in [27], the author proposed the Min-Max algorithm which allows to simplify the processing at the check nodes level by replacing the sum in (2.18) by the operator max:

$$M_{p_i v_j}[\beta] \approx \min_{\substack{\sum \\ s \neq j \\ h_{is} \neq 0}} \{ \max_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{v_s p_i}[\theta_s] \} \tag{2.25}$$

The messages of the Min-Max algorithm can be truncated like in the EMS algorithm.

2.3 FB and SB CNs algorithms

This section reviews the two state-of-the-art implementations of the EMS algorithm: the Forward-Backward (FB) [22] and the Syndrome-Based (SB) [23]. Then, the new innovative technique called presorting is introduced to show that sorting the CN input messages can lead to significant savings in terms of computational complexity and hardware implementation.

2.3.1 Forward-Backward CN processing

The FB-CN algorithm exploits the commutative and associative properties of the addition in GF(q) and factorizes (2.18) using a set of 2-input 1-output ECNs. For the sake of simplicity, the inputs of the CN are denoted by $\{U_i\}_{i=0,\ldots,d_c-1}$ and the outputs are denoted by $\{V_i\}_{i=0,\ldots,d_c-1}$. The CN processing is split into three layers: forward, backward and merge, each one containing $d_c - 2$ ECNs [25]. As Fig. 2.4 shows, an ECN processes a single output C as a function of two inputs A and B. Fig. 2.5 shows the resulting structure for a FB-CN with $d_c = 6$ inputs using $(d_c - 2) \times 3 = 12$ ECNs, each ECN being represented by a block $\boxplus$. Intermediate results of the ECNs are reused in the later stages, avoiding re-computations and thus reducing the amount of processing. Several reported hardware implementations of NB-LDPC decoders use this efficient FB architecture [31] [59].

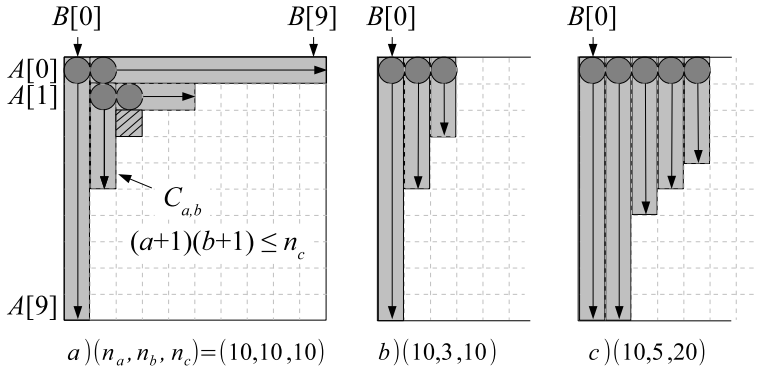

Figure 2.4: S-bubble ECN and generalized S-bubble ECN.

The ECN processing [31] can be described in three steps.

1. **Addition:** for each couple of indexes $(a, b) \in \{0, 1, \ldots, n_m - 1\}^2$, the output tuple

$$C_{a,b} = (c^+(x), x) = (A^+[a] + B^+[b], A^{\oplus}[a] \oplus B^{\oplus}[b]) \quad x \in \mathrm{GF}(q) \tag{2.26}$$

is computed.

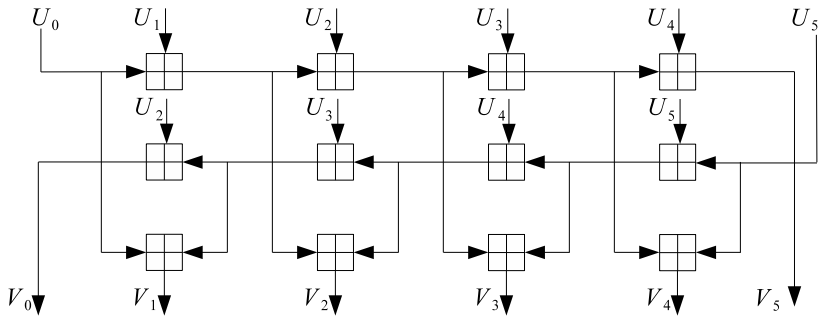

Figure 2.5: FB-CN processing with $d_c = 6$.

Note that since $A^+[0] = 0$ and $B^+[0] = 0$, the first output value of C is $C(0) = (0, A^{\oplus}[0] \oplus B^{\oplus}[0])$. For the sake of clarity, the three ECN steps are represented by $\boxplus$ as:

$$C = A \boxplus B \tag{2.27}$$

In [31], it is shown that (2.26) needs to be evaluated only for indexes (a, b) that verify $(a+1)(b+1) \leq n_{op}$. In fact, since vectors A^+ and B^+ are sorted in increasing order in terms of LLR, any couple (a', b') verifying $(0 \leq a' \leq a)$ and $(0 \leq b' \leq b)$ gives $C^+_{a',b'} \leq C^+_{a,b}$. Consequently, there are at least $(a+1)(b+1)$ couples (a', b') verifying $C^+_{a',b'} \leq C^+_{a,b}$. In other words, if $(a+1)(b+1) > n_{op}$, then $C_{a,b}$ does not belong to the set of the n_{op} smallest values and thus, does not need to be evaluated.

As proposed in [48], the notion of *potential bubbles* is proposed by specifying the index variation ranges n_a and n_b of the two entries A and B, i.e. $0 \leq a < n_a$ and $0 \leq b < n_b$ and the one of the output C as n_c, i.e. $0 \leq c < n_c$. Note that n_a and n_b should be smaller than or equal to n_c. In Fig. 2.4(a), the subset of *potential bubbles* is represented in grey for $n_m = n_{op} = 10$. Fig. 2.4(b) and Fig. 2.4(c) show the potential bubbles when $(n_a, n_b, n_c) = (10, 3, 10)$ and $(n_a, n_b, n_c) = (10, 5, 20)$, respectively.

2. **Sorting**: the couples $(c^+(x), x)$ are sorted in increasing order of $c^+(x)$. The output vector C contains the first n_m ordered couples corresponding to the first n_m smallest values of $c^+(x)$.

3. **Redundancy elimination**: if two couples (a, b) and (a', b') correspond to the same GF value, i.e., $A^{\oplus}[a] \oplus B^{\oplus}[b] = A^{\oplus}[a'] \oplus B^{\oplus}[b']$, the one with higher LLR is suppressed [(1)] during this Redundancy Elimination (RE) step. In order to generate at least n_m valid outputs with a high probability, a number $n_{op} > n_m$

[(1)] according to (2.24), high LLR corresponds to low reliability

outputs is generated by each ECN (typically, $n_{op} = n_m + 2$) [24].

A serial hardware implementation of the Bubble CN architecture was presented in [31]. Suboptimal versions considering only the subset of the most probable potential bubbles (the first two rows and two columns) were presented in [31], [59] and [43].

2.3.2 Syndrome-based CN processing

The SB-CN algorithm [23] relies on the definition of a deviation path and its associated syndrome. In the sequel, $n_{m_{in}}$ (resp. $n_{m,out}$) refers to the size of the input (resp. output) vector of a CN.

A **deviation path**, denoted by δ, is defined as a d_c-tuple of integer values, i.e. $\delta = (\delta(0), \delta(2), \ldots, \delta(d_c - 1))$, with $\delta(i) \in \{0, 1, \ldots, n_{m_{in}} - 1\}$, $i = 0, 1 \ldots, d_c - 1$. A syndrome associated to a deviation path δ is denoted by $S(\delta)$ and defined as the 3-tuple $(S^+(\delta), S^{\oplus}(\delta), S^D[\delta])$ with:

$$S^+(\delta) = \sum_{i=0}^{d_c-1} U_i^+[\delta(i)], \tag{2.28}$$

$$S^{\oplus}(\delta) = \bigoplus_{i=0}^{d_c-1} U_i^{\oplus}[\delta(i)], \tag{2.29}$$

$$S^D[\delta][i] = \begin{cases} 0, & \text{if } \delta(i) = 0 \\ 1, & \text{otherwise} \end{cases}, \tag{2.30}$$

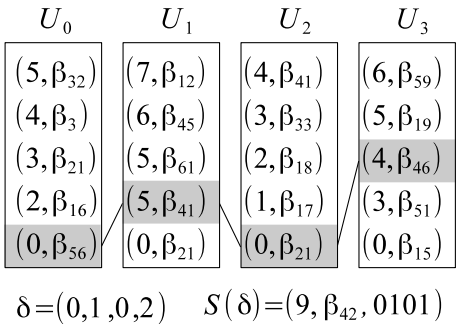

Figure 2.6: Example of a deviation path.

where $S^+(\delta)$ is an LLR value, $S^{\oplus}$ is an element of GF(q) and $S^D[\delta]$ is a binary vector of size d_c called Discard Binary Vector (DBV). Fig. 2.6 shows an example of a CN for $q = 64$, $d_c = 4$ and input messages $U_i, i = 0, \ldots, d_c - 1$ of size $n_{m_{in}} = 5$. In this figure, the deviation path $\delta = (0, 1, 0, 2)$ is represented by a grey shade in each input vector.

It is also represented with straight lines linking $U_1[0]$, $U_2[1]$, $U_3[0]$ and $U_4[2]$. Assuming that the elements of GF(64) are represented by the power of a primitive element β, of GF(64) constructed using the primitive polynomial $P[\beta] = \beta_6 + \beta + 1$, the syndrome associated to δ is $S(\delta) = (0{+}5{+}0{+}4, \beta_{56} \oplus \beta_{41} \oplus \beta_{21} \oplus \beta_{46}, 0101) = (9, \beta_{42}, 0101)$.

Let Δ_0 be the set of all possible deviation paths that can contribute to an output value, i.e., $\Delta_0 \subset \{0, \ldots, n_{m_{in}} - 1\}^{d_c}$. Using the syndrome associated to a deviation path, (2.18) can be reformulated as

$$v_i^+(x) = \min_{\delta \in \Delta_0, S^{\oplus}(\delta) \oplus U_i^{\oplus}[\delta(i)] = x} \left\{ S^+(\delta) - U_i^+[\delta(i)] \right\} \tag{2.31}$$

The DBV is used to reduce the complexity of (2.31) by avoiding redundant computation. In fact, if $S^D[\delta](i) = 0$, then $\delta(i) = 0$ and $U_i^+[\delta(i)] = 0$. It is thus possible to simplify (2.31) as

$$v_i^+(x) = \min_{\delta \in \Delta_0, S^D[\delta][i] = 0, S^{\oplus}(\delta) \oplus U_i^{\oplus}[0] = x} \left\{ S^+(\delta) \right\} \tag{2.32}$$

Finally, (2.32) is further reduced by replacing $\delta \in \Delta_0$ by $\delta \in \Delta$ where Δ is a subset of Δ_0 with a reduced cardinality $|\Delta| = n_s$ as described in [23].

The SB-CN algorithm proposed in [23] is summarized in Algo. 1 and its associated architecture is presented in Fig. 2.7. Step 1 is performed by the Syndrome unit, Step 2 by the Sorting unit and, finally, Step 3 by d_c Decorrelation Units (DU) and d_c RE units. The DUs are represented in parallel to show the inherent parallelism of the SB-CN. The RE units discard couples with a GF value already generated (last test of step 3 in Algo. 1). Note that in [23] the sorting process is done only partially.

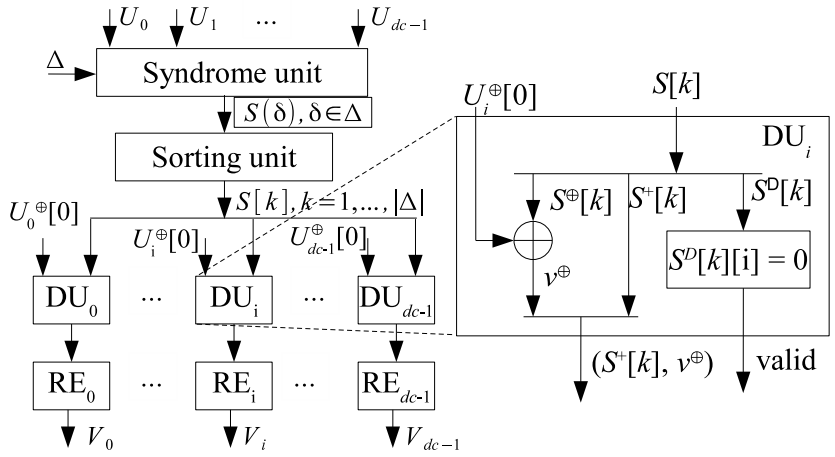

Figure 2.7: Syndrome-based CN processing (left part) and details of the DU unit (right part).

Offline processing:
Select a subset $\Delta \subset \Delta_0$ of cardinality $|\Delta| = n_s$ (which is a trade-off between performance and complexity).
Initialization:
for $i \leftarrow 0$ **to** $d_c - 1$ **do**
  $j_i \leftarrow 0$
end
Processing:
Step 1 (syndrome computation): $\forall\, \delta \in \Delta$, compute $S(\delta)$
Step 2 (sorting process): sort the syndromes in the increasing order of $S^+(\delta)$ to obtain an ordered list $\{S[k]\}_{k=1,2,\ldots,|\Delta|}$ of syndromes;
Step 3 (decorrelation and RE):
for $k \leftarrow 1$ **to** $|\Delta|$ **do**
  for $i \leftarrow 0$ **to** $d_c - 1$ **do**
    if $S^D[k][i] = 0$ **and** $j_i < n_{m,out}$ **then**
      $v_i^{\oplus} \leftarrow S^{\oplus}[k] \oplus U_i^{\oplus}[0]$
      if $v_i^{\oplus} \notin \{V_i[l]^{\oplus}\}_{l=0\ldots j_i-1}$ **then**
        $V_i[j_i] \leftarrow (S^+[k], v_i^{\oplus})$
        $j_i \leftarrow j_i + 1$;
  end
end

Algorithm 1: The SB-CN algorithm.

Fig. 2.7 also shows a detailed scheme with the operations in a DU. S^D is the d_c-wide bit vector that indicates for which output edges the syndrome should be discarded during the decorrelation process. A simple reading of bit i in the binary vector S^D validates or not the syndrome for the output edge i.

2.3.3 Presorting

The new innovative technique called presorting is shown in this section. First a redefinition of the LLR value is presented then the presorting description is shown.
Let us say that the element that represents $\{U_i\}_{i=0,\ldots,d_c-1}$ is e_i in which $e_i \in \mathrm{GF}(q)$. Each input e_i can take q values. Similarly to (2.24), each element of the probability distribution E associated to e can be expressed in the logarithmic domain as the LLR denoted by $e^+(x)$:

$$e^+(x) = -\ln\left(\frac{P(e=x)}{P(e=\bar{x})}\right) \tag{2.33}$$

where $\bar{x}$ is the hard decision on e obtained by taking the most probable GF symbol, i.e. $\bar{x} = \arg\max_{x\in\mathrm{GF}(q)} P(e=x)$.
By definition of the LLR, we have: $e^+(\bar{x}) = 0$ and $\forall x \in \mathrm{GF}(q)$, $e^+(x) \geq 0$. The distribution (or message) E associated to e is thus $E = \{e^+(x)\}_{x\in\mathrm{GF}(q)}$.

Input The d_c input message $\{U_i\}_{i=0,1,\ldots,d_c-1}$.
Step 1: Extract vector $U^1 = (U_0^+[1], U_1^+[1], \ldots, U_{d_c-1}^+[1])$
Sort U^1 in ascending order to generate U'^1.
return permutation $\pi = (\pi(0), \ldots, \pi(d_c-1))$ associated to the sorting process: $U'^1(i) = U^1(\pi(i))$, $i = 0, 1 \ldots d_c - 1$.
Step 2: Permute input vectors using the permutation π:
for $i = 0, 1, \ldots, d_c - 1$, $U'_i = U_{\pi(i)}$
Step 3: Perform the CN process with input vectors $\{U'_i\}_{i=0,1,\ldots,d_c-1}$ to generate output vectors $\{V'_i\}_{i=0,1,\ldots,d_c-1}$.
Step 4: Permute output vector using the inverse permutation π^{-1}: for $i = 0, 1, \ldots, d_c - 1$, $V_{\pi(i)} = V'_i$

Algorithm 2: Pre-sorting principle

The idea of the input pre-sorting is to polarize the statistics of d_c variable-to-check messages by sorting them according to the reliability of the hard decision input, i.e., the probability $P(e_i = U_i^{\oplus}[0])$, $i = 0, 1, \ldots, d_c-1$. The reason for this approach is that many bubbles in the ECN are very unlikely to contribute to the output, suppressing them does not affect performance but can lead to architectural simplifications.
Considering Eq. (2.33) and knowing that $\sum_{x \in \mathrm{GF}} P(e_i = x) = 1$, the probability $P(e_i = U_i^{\oplus}[0])$, $i = 0, 1, \ldots, d_c - 1$ can be expressed as:

$$P(e_i = U_i^{\oplus}[0]) = \frac{1}{\sum_{j=0}^{q-1} e^{-U_i^+[j]}} \tag{2.34}$$

Note that in Eq. 2.34, the values of $U_i^+[j]$ for $j \geq n_m$ are equal to $U_i^+[n_m - 1] + O$, where O is a constant offset value, as detailed in 2.19 [25]. Since $U_i^+[0] = 0$ and, for $j > 2$, $U_i^+[1] \leq U_i^+[j]$, then $P(e_i = U_i^{\oplus}[0])$ can be approximated by:

$$P(e_i = U_i^{\oplus}[0]) \approx \frac{1}{1 + e^{-U_i^+[1]}} \tag{2.35}$$

In other words, the higher the value of $U_i^+[1]$, the higher $P(e_i = U_i^{\oplus}[0])$. From this, we can state that the pre-sorting step is performed according to vector $U^1 = (U_0^+[1], U_1^+[1], \ldots, U_{d_c-1}^+[1])$ as described by the Algorithm 2. As shown in the example of Fig. 2.8 for $n_m = 5$ and $d_c = 4$, the non used entries are in dashed area, so only a reduced number of values in the sorted vectors $\{U'_i\}_{i=0,1,\ldots,d_c-1}$ are considered as inputs to the EMS CN block. This observation motivated the original approach described in the following subsection.
The SB-CN with presorting is presented in [47] where the number of considered deviation paths was reduced after the suppression of the paths that are unlikely to contribute to an output.

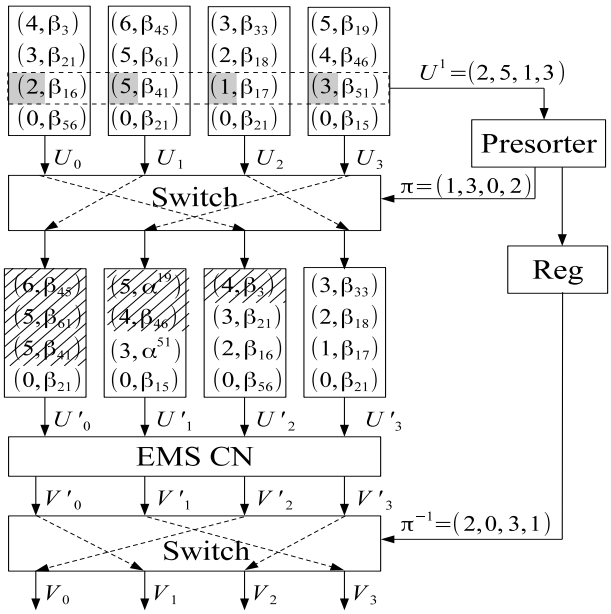

Figure 2.8: Pre-sorting principle.

2.4 Description of an existing VN architecture

The VN of $d_v = 2$ proposed in [7] Chapter 2 Section 2.2.1 operates in three modes: generation of the intrinsic couple candidates, VN update and decision making. In the following, we show the VN update and the decision making modes. First, an example is shown then the architecture is presented.

2.4.1 An example of the VN functionality

In this section an example of the VN functionality is shown, the example in update mode is presented first then the example of the decision mode is shown.

2.4.1.1 An example in update mode

Fig. 2.9 shows a VN v connected to two CNs p_0 and p_1. Let us say that $n_{m_{out}} = 8$ is the length of the extrinsic messages M_{p_0v} and $n_{m_{in}} = 4$ is the length of M_{vp_1}. Each element belongs to M_{p_0v} and M_{vp_1} is a couple of (LLR, GF). The intrinsic GF vector I is required in the update mode. Therefore, Let $M^{+}_{p_0v} = \{0, 2, 7, 19, 20, 20, 20, 20\}$, $M^{\oplus}_{p_0v} = \{\beta_1, 0, \beta_4, \beta_5, \beta_0, \beta_2, \beta_3, \beta_6\}$, $I^{+} = \{0, 7, 15\}$ and $I^{\oplus} = \{\beta_5, 0, \beta_0\}$ be the input vectors of v.

The update mode is divided into two phases, M_{p_0v} is processed in phase 1 and I is processed in phase 2. Concerning phase 1, the intrinsic LLR value of each element in $M^{\oplus}_{p_0v}$ is computed. Let $IM_{p_0v} = \{63, 7, 18, 0, 15, 33, 69, 45\}$ be the vector of intrinsic LLR value of $M^{\oplus}_{p_0v}$ (the computation of IM_{p_0v} is performed by the eLLR block in the architecture). Then, the two vectors $M^{+}_{p_0v}$ and IM_{p_0v} are added to form the updated

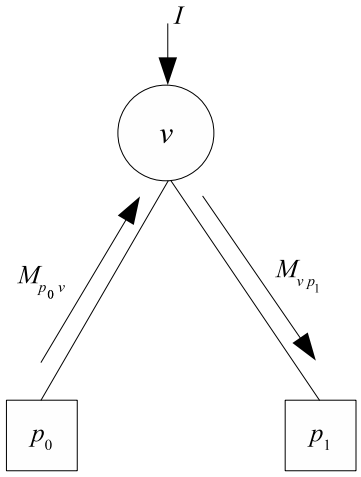

Figure 2.9: A VN v connected to two CNs p_0 and p_1.

extrinsic LLR values $M^{+}_{VN} = M^{+}_{p_0 v} + IM_{p_0 v} = \{63, 9, 25, 19, 35, 53, 89, 65\}$. Therefore, M^{+}_{VN} and $M^{\oplus}_{VN} = M^{\oplus}_{p_0 v} = \{\beta_1, 0, \beta_4, \beta_5, \beta_0, \beta_2, \beta_3, \beta_6\}$ are inputs of a sorter block to extract the most $n_{m_{out}}$ reliable GF symbols in terms of LLR value, let $M^{s+}_{p_0 v} = \{9, 19, 25, 35\}$ is the most reliable updated extrinsic LLR values stored in the sorter and $M^{s\oplus}_{p_0 v} = \{0, \beta_5, \beta_4, \beta_0\}$ its associated GF values.
After that, phase 2 is performed. The last extrinsic LLR value $M^{+}_{p_0 v}[7] = 20$ is considered as offset of I^{+}, so the second input vector of the sorter is $M^{+}_{VN} = I^{+}+20+\mathrm{O}$ along with $M^{\oplus}_{VN} = I^{\oplus}$, O is an offset value. Each time a redundant GF symbol occurs, its associated LLR value is replaced by the Sat (maximum LLR) value. Thus, since $I^{\oplus}[0] = \beta_5$ and $I^{\oplus}[1] = 0$ are already processed and generated as valid GF symbols in phase 1, the intrinsic message vectors fed to the sorter are $M^{+}_{VN} = \{Sat, Sat, 35 + \mathrm{O}\}$ and $M^{\oplus}_{VN} = \{\beta_5, 0, \beta_0\}$. Finally, after the normalization, the output of the VN in update mode is $M^{+}_{vp_1} = \{0, 10, 16, 26\}$ and $M^{\oplus}_{vp_1} = \{0, \beta_5, \beta_4, \beta_0\}$.

2.4.1.2 An example in decision mode

The first $n_s = 3$ elements in M_{vp_1} are saved in a Content-Addressable Memory (CAM), so let $M^{+}_{\mathrm{CAM}} = \{0, 10, 16\}$ and $M^{\oplus}_{\mathrm{CAM}} = \{0, \beta_5, \beta_4\}$ be the saved data in the CAM. Let us say that p_1 replied to v by $M^{+}_{p_1 v} = \{0, 19, 41, 47, 48, 48, 48, 48\}$ and $M^{\oplus}_{p_1 v} = \{\beta_0, \beta_6, \beta_2, \beta_3, 0, \beta_1, \beta_4, \beta_5\}$. Again, there are two phases in decision mode, phase 1 to process $M_{p_1 v}$ and phase 2 to process M_{CAM} (the message contained in the CAM). The two vectors are updated as:

$$M^{n+}_{p_1 v}[i] = \begin{cases} M^{+}_{p_1 v}[i] + M^{+}_{\mathrm{CAM}}[j] & \text{if } M^{\oplus}_{p_1 v}[i] = M^{\oplus}_{\mathrm{CAM}}[j] \\ M^{+}_{p_1 v}[i] + M^{+}_{\mathrm{CAM}}[2] + \mathrm{O} & \text{Otherwise} \end{cases}$$

then

$$M^{n+}_{\mathrm{CAM}}[k] = M^{+}_{\mathrm{CAM}}[k] + M^{+}_{p_1 v}[7] + \mathrm{O}$$

Where $i = 0, \ldots, 7$, $j = 0, 1, 2$, $k = 0, 1, 2$ and O = 1. Therefore, $M_{p_1 v}^{n+} = \{18, 36, 58, 64, 48, 65, 64, 58\}$, $M_{p_1 v}^{n\oplus} = \{\beta_0, \beta_6, \beta_2, \beta_3, 0, \beta_1, \beta_4, \beta_5\}$, $M_{\text{CAM}}^{n+} = \{49, 59, 65\}$ and $M_{\text{CAM}}^{n\oplus} = \{0, \beta_5, \beta_4\}$. Finally, β_0 is the GF symbol corresponding to the minimum LLR value and hence it is the output of the decision block.

2.4.2 VN architecture in update mode

The VN architecture in update mode is shown in Fig. 2.10 where $n_{m_{out}} = n_{m_{in}} = n_m$.

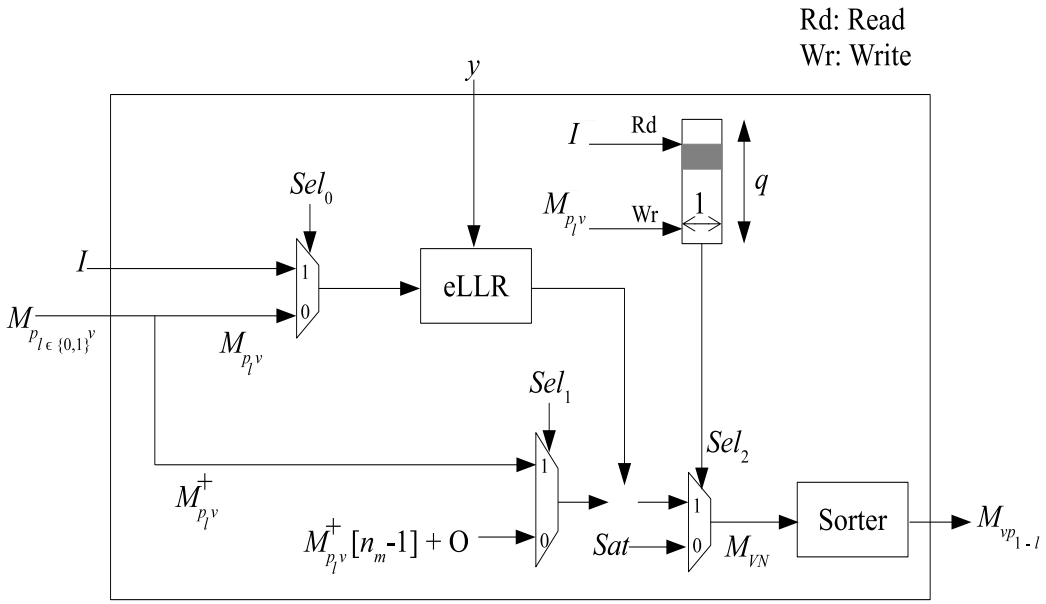

Figure 2.10: Architecture of the VN [7] in update mode.

Updating a VN takes place serially in two phases. As a first step, the VN updates the $M_{p_{l\in\{0,1\}}v}$ extrinsic message from the CN Processor (CNP). Therefore, the signal Sel_0 takes the value 0 and the signals Sel_1 and Sel_2 take the value 1. The update of the message $M_{p_l v}$ is given by:

$$\begin{cases} M_{\text{VN}}^{\oplus}[i] = M_{p_l v}^{\oplus}[i], \\ M_{\text{VN}}^{+}[i] = M_{p_l v}^{+}[i] + I^{+}[M_{p_l v}^{\oplus}[i]], \end{cases} \qquad i = 0, 1, \ldots, n_m - 1 \tag{2.36}$$

The intrinsic LLRs $I^{+}[M_{p_l v}^{\oplus}[i]]$ used to update the LLRs of the $M_{p_l v}$ messages are progressively generated by the eLLR block. The Flag block is a q-bits register (each bit corresponds to a symbol of the Galois field). The bits of this register are updated by:

$$Flag[M_{p_l v}^{\oplus}[i]] = 1, \qquad i = 0, 1, \ldots, n_m - 1 \tag{2.37}$$

In a second step, the VN Processor (VNP) updates the LLRs of the n_β symbols of the intrinsic message GF values $I^{\oplus}$. Therefore, the signal Sel_0 takes the value 1 and

the signal Sel_1 takes the value 0. If a symbol of the message $I^{\oplus}$ belongs to the vector messages $M_{pl}v$ (that means if $Flag[I^{\oplus}[i]] = 1$, $i = 0, 1, \ldots, n_\beta - 1$) then the signal Sel_2 takes the value 0 and the LLR associated with this symbol is saturated (fed with the maximum LLR value) in order to avoid duplicate symbols in the outgoing messages from the VNP. The update of the messages $I^{\oplus}$ is done by:

$$M_{\mathrm{VN}}^{\oplus}(n_m + i) = I^{\oplus}[i]$$

$$M_{\mathrm{VN}}^{+}(n_m + i) = \begin{cases} I^{+}[i] + M_{plv}^{\oplus}(n_m - 1) + \mathrm{O} & \text{if } Flag[I^{\oplus}[i]] = 0 \\ Sat & \text{otherwise} \end{cases} \tag{2.38}$$

$$i = 0, 1, \ldots, n_\beta - 1$$

The message M_{VN} of size $(n_m + n_\beta)$ is progressively introduced in the Sorter block to be sorted in increasing order. The message $M_{vp_{1-l}}$ is obtained by selecting the first n_m outgoing elements of the Sorter block. The latency of the VNP in this mode is $L_{VNP,1} = n_m + n_\beta + 2$ clock cycles as shown in Fig. 2.11.

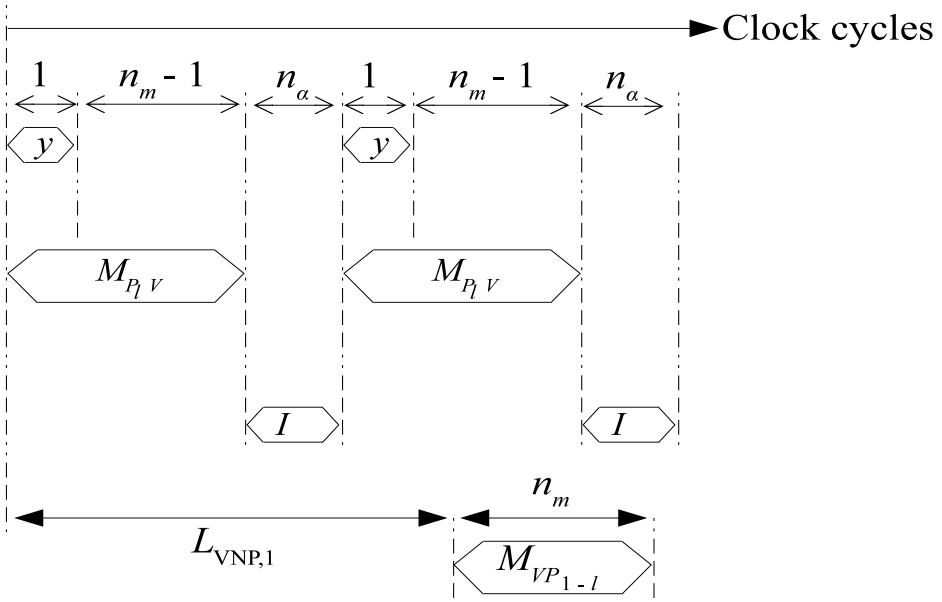

Figure 2.11: Timing diagram of VN in update mode [7].

2.4.2.1 Architecture of the Sorter block

The different types of comparators used in this manuscript are shown in Fig. 2.12: Comparator Only (CO) (Fig. 2.12.a), Comparator (C) (Fig. 2.12.b), Comparator-Swaps (CS) (Fig. 2.12.c) and 2-to-1 multiplexers (MUX2-1). The CO generates only a comparison signal defined as: $c_{pq} = 1$ if $x_p < x_q$, otherwise $c_{pq} = 0$. The comparator C selects the minimum value (m) as follows:

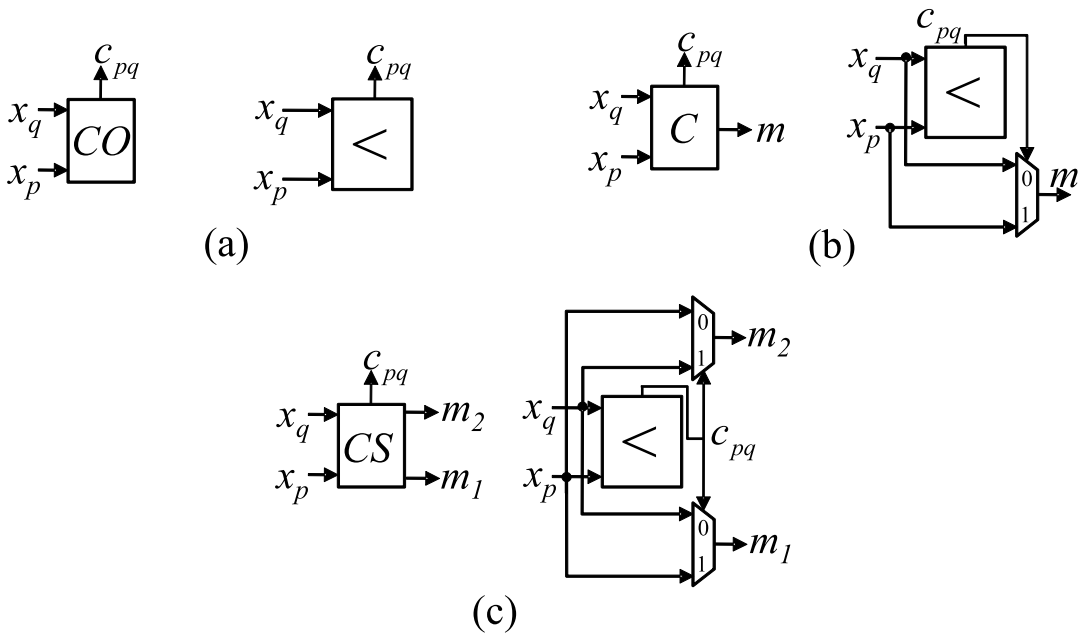

Figure 2.12: (a) Comparator Only (CO), (b) Comparator (C), (c) Comparator Swap (CS) and (d) ESU (4-SU) Architecture.

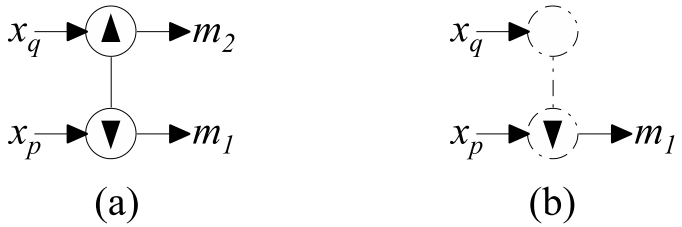

Figure 2.13: (a) CS, (b) C.

- if $x_p < x_q \Rightarrow c_{pq} = 1, m = x_p$
- if $x_p \geq x_q \Rightarrow c_{pq} = 0, m = x_q$.

The CS is composed of one comparator and two MUX2-1s. This CS sorts the input values where the lower and upper outputs represent the first minimum (m_1) and second minimum (m_2) values respectively as described below:

- if $x_p < x_q \Rightarrow c_{pq} = 1, m_1 = x_p$ and $m_2 = x_q$
- if $x_p \geq x_q \Rightarrow c_{pq} = 0, m_1 = x_q$ and $m_2 = x_p$.

For sake of simplicity in representing some architectures, we use the symbols shown in Fig. 2.13.a and Fig. 2.13.b as CS and C respectively.

The architecture of the sorter is shown in Fig. 2.14. This architecture consists of n_m (number of outputs) stages regardless of the size of the input list. In addition, the latency of this architecture is equal to the number of inputs. So to sort a list containing $q = 64$ elements, the latency is $L_{sorter} = 64$ clock cycles.

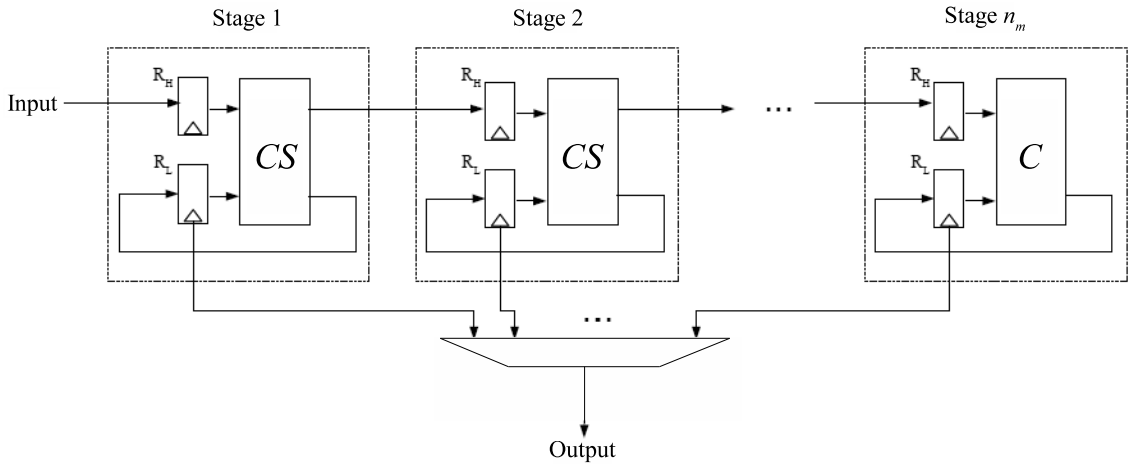

Figure 2.14: Architecture of the Sorter block [7].

Each stage of the sorter consists of two registers (R_H and R_L) and a CO. R_H (register high) stores the last element provided at the input and R_L (register low) stores the minimum element in terms of LLR value. Each stage sends the symbol of higher LLR to the next one. Thus, the n_m R_L registers, from stage 1 to stage n_m, form a list sorted in ascending order. On arrival of the last element of the inputs, the result of R_L of stage 1 is fed to the output. In the next cycle, the sorter selects the couple from R_L of stage 2, and so on.

2.4.3 VN architecture in the decision-making mode

In the decision mode, only the n_s messages from M_{vp_1} are stored in the CAM. Therefore, the sets Λ_1, Λ_2 and Λ_3 are defined as:

$\Lambda_1 = \{M^{\oplus}_{vp_1}[1]\}_{i=0,1,\ldots,n_s-1} \cap M^{\oplus}_{p_1v}$

$\Lambda_2 = \{M^{\oplus}_{vp_1}[1]\}_{i=0,1,\ldots,n_s-1} \setminus \Lambda_1$

$\Lambda_3 = M^{\oplus}_{p_1v} \setminus \Lambda_1$

The computation of the a *priori* information APP is given by:

$$APP[x]_{x\in\Lambda_1\cup\Lambda_2\cup\Lambda_3} = \begin{cases} M^+_{vp_1}[x] + M^+_{p_1v}[x], & \text{if } x \in \Lambda_1 \\ M^+_{vp_1}[x] + M^+_{p_1v}(n_m - 1) + \mathrm{O}, & \text{if } x \in \Lambda_2 \\ M^+_{p_1v}[x] + M^+_{vp_1}(n_s - 1) + \mathrm{O}, & \text{otherwise} \end{cases} \tag{2.39}$$

Then, the decision is made by:

$$\hat{c} = \underset{x\in\Lambda_1\cap\Lambda_2\cap\Lambda_3}{\operatorname{argmin}} \{APP[x]\} \tag{2.40}$$

Therefore, the architecture of the decision making is illustrated in Fig. 2.15 where n_b is the number of bits to represent an LLR value.

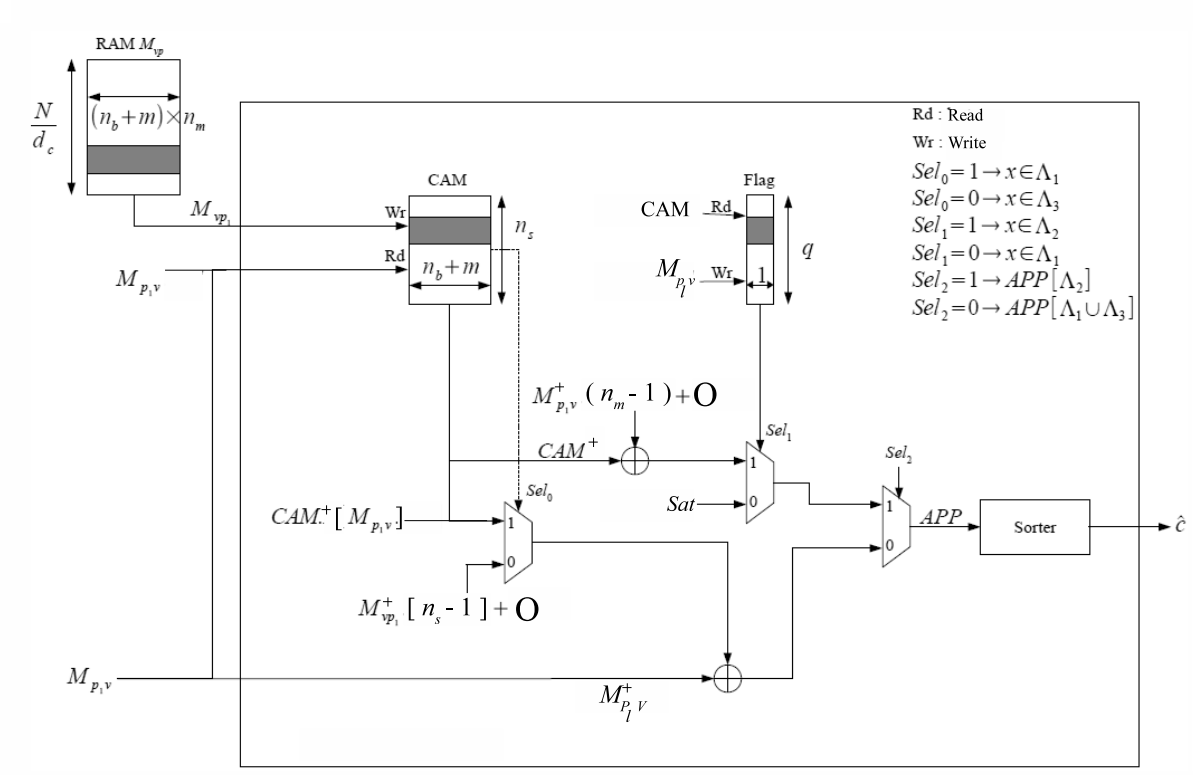

Figure 2.15: VN architecture in decision-making mode (only active blocks are shown) [7].

The VNP receives the first n_s M_{vp_1} messages and stores them in the CAM. Then, it computes the APP message in two phases:

- ▷ *Phase* 1: The VNP computes the APP values associated with the set $\Lambda_1 \cup \Lambda_3$ (the symbols of the message M_{p_1v}). The signal Sel_2 takes value 0. Whenever

a symbol of Λ_1 exists in CAM, the signal Sel_0 takes value 1 otherwise it takes value 0. The register Flag stores the symbols of the set $\Lambda_1 \cup \Lambda_3$ ($Flag[M^{\oplus}_{p_1 v}[i]] = 1, i = 0, 1, \ldots, n_m - 1$).

▷ *Phase* 2: The VNP computes the APP values of the set Λ_2 (the symbols stored in the CAM except those in the set Λ_1). The signal Sel_2 takes value 1. The symbols of CAM that belong to the set Λ_1 are identified by the Flag register. In the presence of these symbols the signal Sel_1 takes value 0 and its associated APP value is saturated to not be processed by the Sorter block.

During both phases, the APP values are progressively introduced in the Sorter block. The decision is made by selecting the symbol with the smallest APP value. The latency of the VNP during decision-making mode is $L_{VNP,2} = n_m + n_s + 2$ as shown in Fig. 2.16.

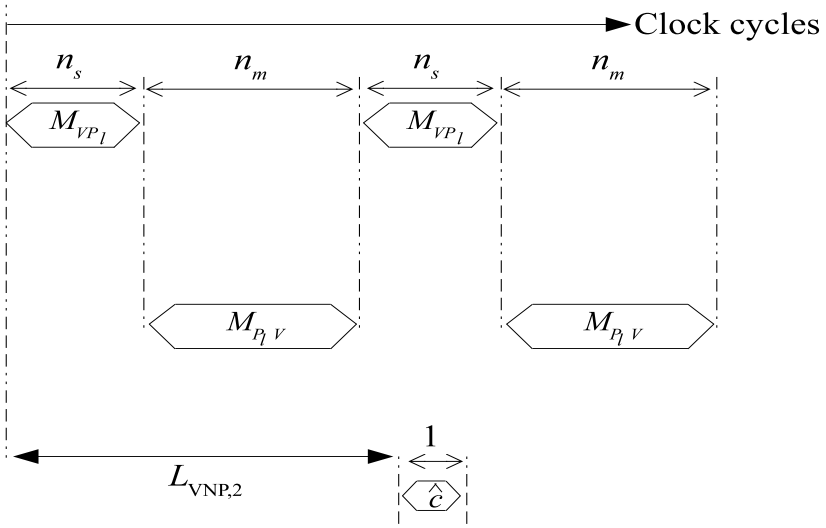

Figure 2.16: Timing diagram of VN in decision mode [7].

2.5 Layered vs. Flooding decoder scheduling

This section reviews the principles of two different decoder schedulings over the Tanner graph: the layered and the flooding scheduling.

To complete one iteration, the four CNs and their connected VNs must be updated as shown in Fig. 2.17.a). The two schedules of processing are performed as follows:

Flooding: Let us take v_2 that is connected to p_0 and p_3 as shown in Fig. 2.17.b), v_2 feeds the two CNs p_0 and p_3 by the same vector messages, i.e, $M_{v_2,p_0} =$

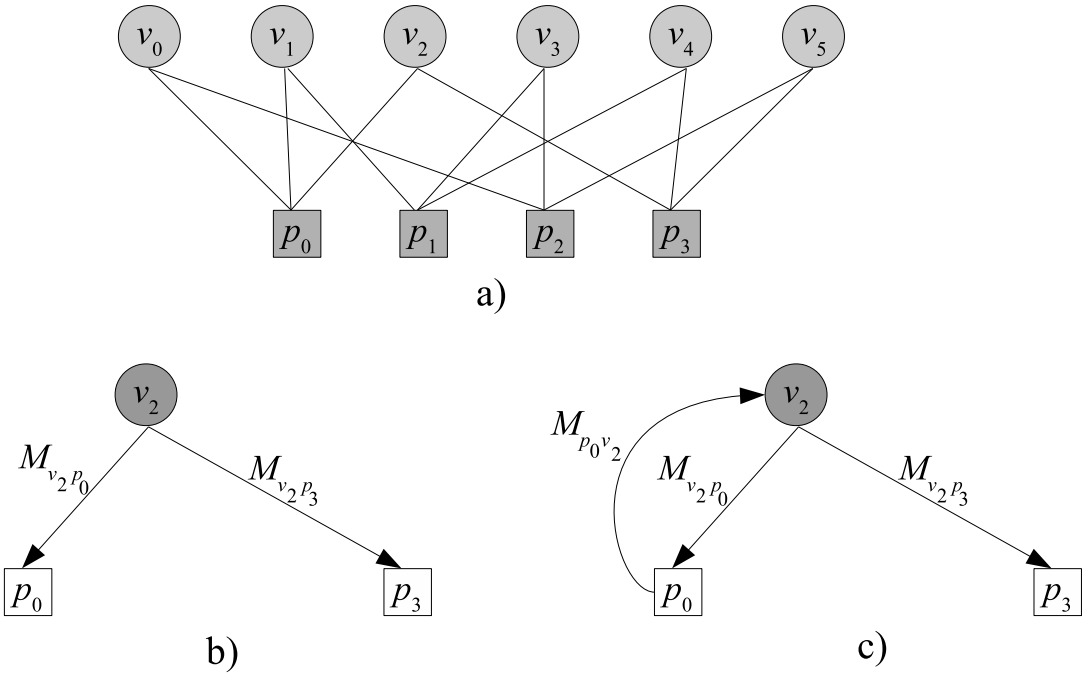

Figure 2.17: Stages of processing.

M_{v_2,p_3}. Thus, the CNs are not benefiting from the updated VN messages in the same iteration.

Layered: In this approach, v_2 feeds CN p_0 by M_{v_2,p_0} as shown in Fig. 2.17.c), then v_2 receives M_{p_0,v_2} to generate the updated vector messages M_{v_2,p_3} which is sent to CN p_3. Thus, the CNs benefit from the updated VN messages in the same iteration.

Each type of scheduling has its advantage: the layered one provides better performance and the flooding one presents lower latency. So the choice between them should be done on this basis. In our work, the proposed parallel pipelined decoder, we consider the flooding schedule of processing.

2.6 State-of-the-art NB-LDPC decoder architectures

In this section, some NB-LDPC decoder architectures are presented to show the last advances on this area.

2.6.1 A fully parallel NB-LDPC decoder with fine-grained dynamic clock gating

The decoder architecture presented in [59] implements the EMS algorithm over a GF(64) (160, 80) regular-(2, 4) NB-LDPC code with $n_m = 16$. In this approach,

$N = 160$ VNs and $M = 80$ CNs are implemented and the exchanged inputs and outputs are interleaved to reduce the latency per one iteration process. In addition, the bubble ECN proposed in [39] is modified in order to increase the frequency. Furthermore, the satisfaction of the PCM equations is checked for two reasons: First, to apply the Clock Gating on the blocks that are unnecessary to be processing anymore. Applying the clock gating approach on a block means making it not functioning anymore which leads to reduce the power consumption. Second, to stop the decoding of the current frame and start processing the next one before the ending of the maximum number of iterations which further increases the throughput by considering the average number of iterations.
However, even though there are N VNs and M CNs processed in parallel, the functionality of each block is still serial which affects negatively the throughput and the hardware efficiency. The number of clock cycles needed per iteration is equal to 47 which results in a global throughput rate of 1.22 Gbits/s.

2.6.2 Trellis-Based extended Min-Sum algorithm Decoder

The authors in [29] adopted the T-EMS algorithm for their NB-LDPC decoder implementation. Even if, in this architecture, complexity is independent of the check node degree, it significantly increases with q. In fact, only GF(4) practical implementations are considered in [29], leading to a throughput of 2.4 Gbits/s for the serial decoder architecture and up to 3.6 Gbits/s for the parallel decoder architecture.

2.6.3 A 21.66 Gbps Non-Binary LDPC decoder for high-speed communications

The authors in [58] introduce a new algorithm called *Improved Layered Multiple-symbol-reliability weighted Bit-Reliability Based* (IL-MwBRB). In fact, their proposed algorithm is derived from the MwBRB algorithm [32]. The pipelining and parallelism in the architecture are considered to increase the throughput to 21.66 Gbits/s. Unlike the mentioned decoding algorithms (FFT-BP, EMS, T-EMS and Min-Max algorithms), IL-MwBRB algorithm processes the reliability messages at the bit level which leads to performance degradation.

2.7 Conclusion

In this chapter, we started by recalling the definition of LDPC codes defined over Galois field GF($q = 2^m$). Then, we have discussed the main algorithms used in the decoding of NB-LDPC codes. The BP algorithm is optimal but its hardware implementation is not feasible. Although the FFT-BP algorithm converts the convolution operation to a multiplication, it is still heavy to perform in hardware. The Log-BP algorithm replaces the multiplication by a simple addition operation but a lot of memories are required to store the q symbols. The sub-optimal algorithm, EMS, and its

variants at the CN level have been described. The key idea in the EMS algorithm is to truncate $n_m << q$ symbols to be exchanged between the CNs and the VNs, thus both hardware cost and memory consumption are reduced.
Afterward, the FB, SB and presorting algorithms are shown. The impact of the presorting on the FB-CN and the new hybrid CN are shown in next chapter.
Then, the layered and flooding schedules of the decoding process have been reviewed. Finally, we presented some of the state-of-the-art NB-LDPC decoder architectures, where the throughput rate of each decoder architecture has been discussed.

Chapter 3

Efficient architectures for NB-LDPC decoding

This Chapter presents the main contributions of this thesis which are the new architectures for the CNP and the VNP. Section 1 describes the improvements done on the existing CNP to obtain the new parallel CNP architecture and section 2 shows the new VNP architecture. These architectures will constitute the global decoder described in Chapter 5. In this chapter, only an example of high code rate (CR=$\frac{5}{6}$) will be considered. The proposed algorithm/architecture can easily be extended to other code rates: an example of extension from $d_c = 12$ to $d_c = 16$ is also given.

3.1 New Check Node Architectures

In this section we focus on the CNP and we describe the advantages of the new innovative presorting technique in terms of hardware cost on the FB-CN and the two recent proposed architectures called Extended Forward CN (EF-CN) and Hybrid CN (H-CN). Finally, a block called Skip Processing Controller (SPC) is presented. The idea is to define a criteria and when it is satisfied, the CN processing is skipped. Thus, the role of the SPC block is to test this criteria and indicate to the CNP whether the processing is to be skipped. This permits to reduce the decoding latency per iteration and hence to increase the global throughput or reduce the global power consumption.

3.1.1 FB-CN with presorting

As shown in chapter 2, pre-sorting allows a significant reduction of the number of syndromes that need to be computed in the pre-sorted SB architecture. In this chapter, we apply the presorting technique to the FB-CN architecture. The basic principle is similar to the SB architecture: bubbles that are unlikely to be used are simply discarded, which leads to hardware complexity reduction.
A statistical study of the behavior of the bubbles in the input vectors $\{U'_i\}_{i=0,1,...,d_c-1}$ allow us to predict for each ECN the bubbles that can be omitted without affecting

the global performance of the FB-CN processing. This study is performed through the observation of each ECN processing during the Monte-Carlo simulation of a (576, 480) GF(64)-LDPC code at SNR = 3.5 dB over more than ten thousand decoded frames. The effect of the bubble suppression is translated into hardware complexity reduction. In what follow, the FB-CN with presorting.

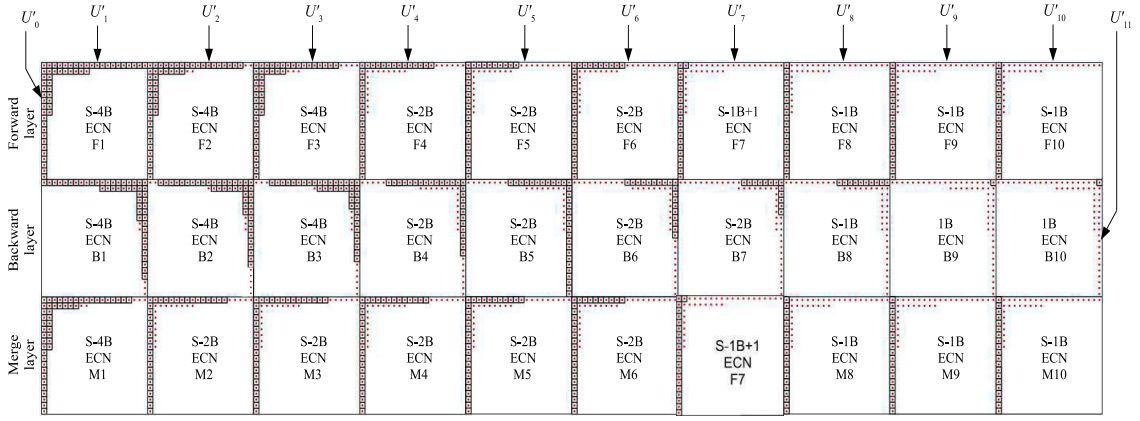

Figure 3.1: Matrix representation of a S-Bubble Check FB-CN with $d_c = 12$ and $n_m = 20$. The $b = 1680$ red circles represent the bubbles in the original FB-CN algorithm. The squares represent the remaining $b^o = 648$ bubbles after the pruning process in the S-FB algorithm.

To be specific, Fig. 3.1 represents a S-Bubble Check FB-CN with $d_c = 12$, composed of three layers of $d_c - 2 = 10$ ECNs each. The points (inside or outside each small black square) represent the positions of the processed bubbles with the S-Bubble architecture [43], which are a total of $b = 1680$. The black squares represent the positions of the bubbles that contribute to an output after applying the pre-sorting technique. They represent a number of $b^o = 648$ bubbles, i.e, 40% of the initial number of bubbles. Consider for example ECN B10 in the Backward layer, only one bubble is used in practice for implementation: a S-Bubble architecture at this ECN clearly implies a waste of resources. The idea is then to implement for each ECN the most simplified architecture that guarantees a correct ECN processing as detailed in the following section. Please also note that the pre-sorting technique requires extra hardware blocks compared to the classical *unsorted* CN architecture: a d_c-input vector sorter and two permutation networks (or switches). We will also show in section 3.1.3 that the area cost of this extra hardware is compensated with the ECN simplifications, leading to an optimised global CN implementation.

3.1.2 Proposed FB-CN Architecture

This section first describes the elementary blocks constituting the proposed FB-CN architecture: sorter, switch and simplified ECNs. Then, the global CN architectures for different d_c values are presented.

3.1.2.1 Sorter

Several sorting algorithms have been proposed in the literature based on serial [44] and parallel approaches [67]. The selection of the most suitable sorter architecture is based on two main criteria: hardware complexity and speed performance. The architecture of the sorter we implemented is a semi-parallel architecture based on the algorithm proposed in [45]. The architecture is composed of $d_c/2$ stages where each stage contains $d_c - 1$ comparator-swap blocks. As shown in Fig. 3.2, since the processing time of the FB-CN processor will be greater than the sorting time, we have implemented only one stage that will be running $d_c/2$ times in order to sort an input vector of size d_c and to generate the permutation order vector π where each L_i, $i = 0, \ldots, d_c - 1$, is a LLR value. Thus, MUX 1 selects for only one clock cycle (first clock cycle) $\{(U_0^+[1], 0), \ldots, (U_{d_c-1}^+[1], d_c-1)\}$ then selects $\{(L_0, \pi(0)), \ldots, (L_{d_c-1}, \pi(d_c-1))\}$ for the rest $d_c/2 - 1$ clock cycles. The latency of this sorter architecture is $d_c/2$ cycles and it constitutes a good trade-off between complexity and performance.

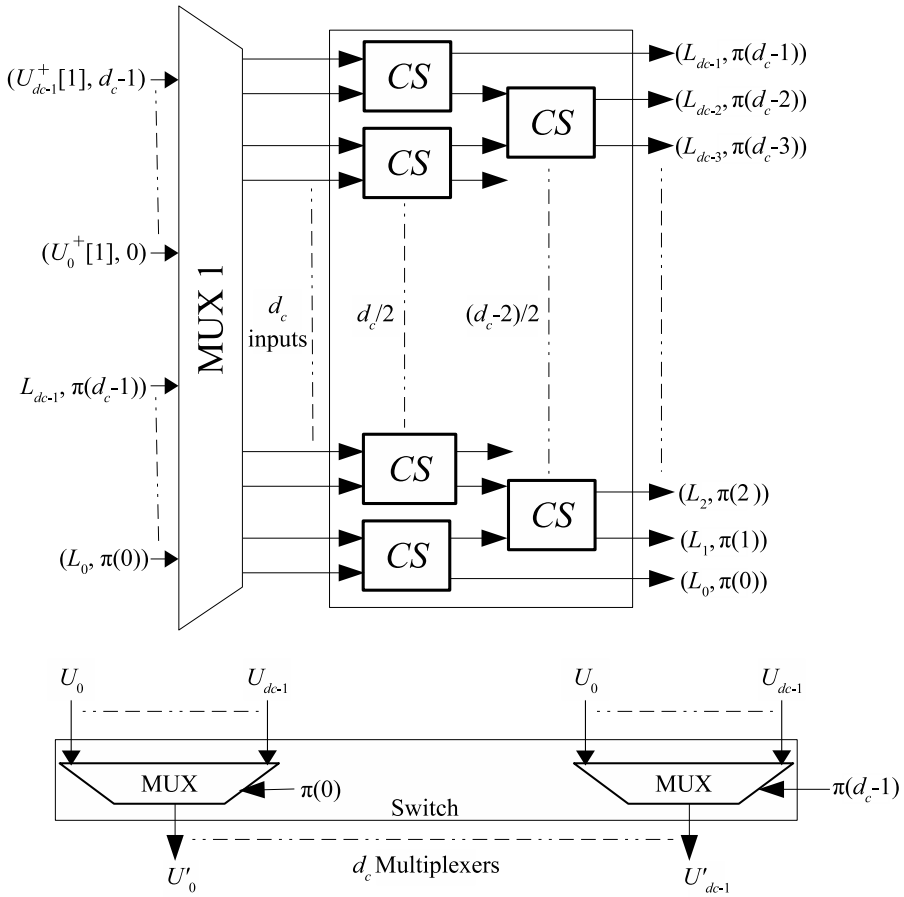

Figure 3.2: Architecture of the Sorter and Switch blocks. The Sorter architecture follows [45].

3.1.2.2 Switch

The Switch block receives the d_c inputs $\{U_i\}_{i=0,1,\ldots,d_c-1}$ and permutes them based on the permutation vector π received from the Sorter. This Switch is composed of d_c multiplexers of size d_c-to-1, as shown in Fig. 3.2.

3.1.2.3 Simplified ECNs

As previously mentioned, the hardware resources of each ECN can be reduced without affecting performance. Five different structures of ECN can be considered in Fig. 3.1:

S-4B: This ECN architecture, known as S-bubble ECN, is described in [43] where four bubbles are compared per clock cycle. It is composed of 4 FIFO blocks, a minimum detector C of 4 input values, two arithmetic adders (the two adders related to A[1] and B[1]) and four modulo-2 adders implemented using XOR gates as shown in Fig. 3.3. The candidates in each FIFO are sorted, the MIN block detects the minimum among four candidates and then its corresponding index will be increased by 1. The Flag is to check whether the current reliable symbol is redundant or not and the multiplexer is to either select the current reliable symbol or the saturated data in case of redundancy.

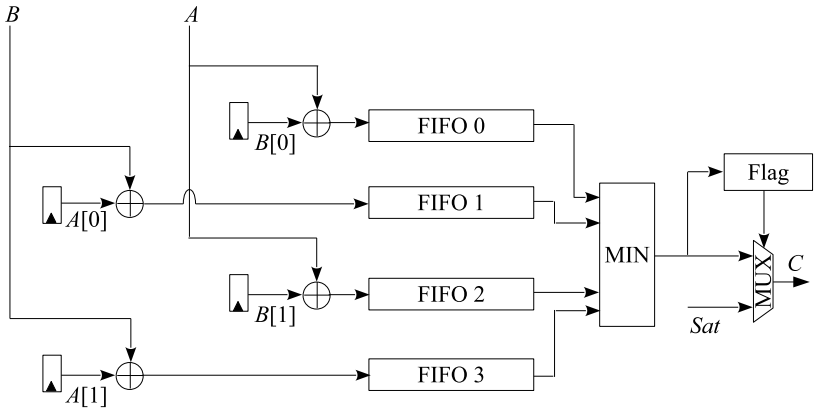

Figure 3.3: S-4B architecture.

S-2B: It is based on the S-bubble ECN but composed of only the first row and first column of the matrix shown in Fig. 2.4. Thus, only two bubbles are compared per clock cycle, two FIFO blocks are needed with only one comparator C and two modulo-2 adders (see Fig. 3.4). There are 10 S-2B ECNs used in case of $d_c = 12$ as shown in Fig. 3.1, these ECNs are F3, F4, F5, F6, B4, B5, B6, B7, M2 and M3.

S-1B: This vector ECN generates the output C as: $C^{+}_{a,0} = A^{+}[a]$, $C^{\oplus}_{a,0} = A^{\oplus}[a] \oplus B^{\oplus}[0]$ $(a = 0, \ldots, n_m - 1)$. Note that vectors A and B can be exchanged depending

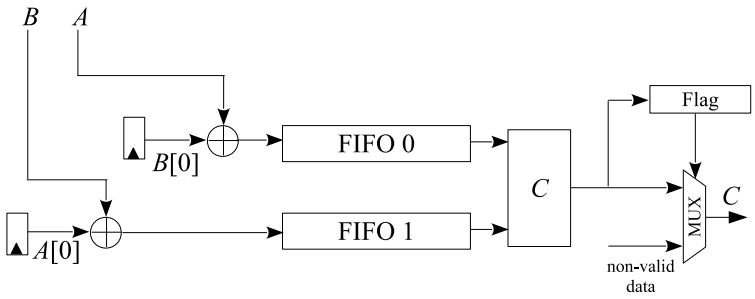

Figure 3.4: S-2B architecture.

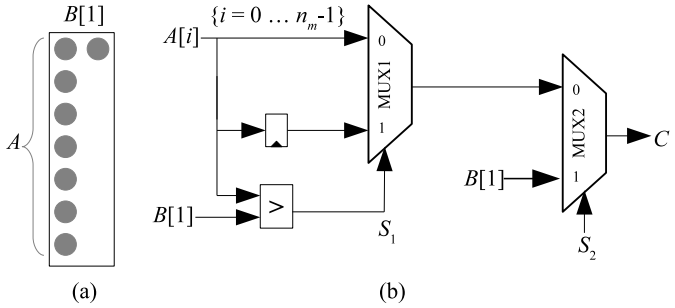

Figure 3.5: S-1B+1 ECN and its architecture.

on the distribution of the bubbles. The only required component of S-1B ECN is the XOR gate.

S-1B+1: The bubbles considered in this ECN processing are shown in Fig. 3.5.a and the architecture in Fig. 3.5.b. It is composed of a comparator, two 2-to-1 multiplexers and a single register. The control signal S_1 is initially 0 and then set to 1 for all the following cycles if and only if $A^+[i] > B^+[1]$, $i = 1, 2, \ldots n_m - 1$ and $C^{\oplus}_{i,0} \neq C^{\oplus}_{0,1}$, where $C^{\oplus}_{i,0} = A^{\oplus}[i] \bigoplus B^{\oplus}[0]$ and $C^{\oplus}_{0,1} = A^{\oplus}[0] \bigoplus B^{\oplus}[1]$. The control signal S_2 is also initialized to 0 and keeps this value while $A^+[i] < B^+[1]$. It will be turned to 1 for only one cycle when $A^+[i] > B^+[1]$ and $C^{\oplus}_{0,1}$ is different to all the symbols $C^{\oplus}_{j,0}$, $j < i$, already output. The only required component for S-1B+1 ECN is the XOR gate.

1B: This ECN considers a single bubble where the output is the most reliable element $C^{\oplus}_{0,0} = A^{\oplus}[0] \bigoplus B^{\oplus}[0]$.

3.1.2.4 ECN simplifications for global CN with different d_c values

The statistical analysis and architectural ECN simplifications were performed for $d_c = 6, 8, 12$ and 20, i.e. coding rates 2/3, 3/4, 5/6 and 9/10, respectively. For $d_c = 12$ (CR=$\frac{5}{6}$), Fig. 3.1 depicts the architecture retained for each ECN. Table 3.1 presents the number of each kind of ECN being implemented in the S-FB CN for several d_c

values. From these results we can predict significant potential area gains specially for high d_c values: for example, for $d_c = 20$ the S-Bubble architecture will be replaced 10 times by the 1B architecture.

Table 3.1: Number of ECN schemes for different d_c values.

	S-FB				
d_c	S-4B	S-2B	S-1B	S-1B+1	1B
6	5	7	-	-	-
8	9	5	2	1	1
12	6	10	8	4	2
20	12	7	24	1	10

Therefore, the higher the value of d_c, the higher the number of the suppressed bubbles and hence the higher the gain obtained with the presorting.

3.1.3 Implementation and simulation results

To quantify the interest of the pre-sorting technique in FB-CN architectures we have implemented the different architectural designs on a FPGA device. We also show simulations results of the new approach where no performance loss is obtained.

3.1.3.1 Implementation results

We considered the Xilinx VIRTEX 6, xc6vlx240t-2ff1156 FPGA device to obtain synthesis results. The five ECN architectures were synthesized to obtain the results presented in Table 3.4. The LLR and GF values are quantified on 6 bits. The 1B and S-1B ECNs have negligible complexity and a maximum frequency of 714 MHz. Also, the S-1B+1 and S-2B ECNs have reduced complexity and operate at higher frequencies compared to S-4B.
Table 3.3 summarizes the overall complexity of the FB-CN for different d_c values. Please note that "S-FB" stands for the S-Bubble CN implementation and that "P-FB" stands for the presorting approach proposed in this paper. The proposed architecture leads to a global CN complexity reduction of 5% for $d_c = 6$, 43% for $d_c = 12$ and 54% for $d_c = 20$, compared to the state-of-the-art S-FB architecture.
Table 3.3 also shows the synthesis results of the Sorter and Switch blocks. These extra blocks (1 Sorter and 2 Switches) of the pre-sorting step constitute about 30% (46%) of the total area for $d_c = 12$ (resp. $d_c = 20$), but the ECN architectural simplifications compensate for this and a global gain of 43% (resp. 54%) is obtained.
Even if the implementation of the variable node is out of the scope of this study, let us note that significant area reduction is expected as the number of sorted values to compute for CN input messages is reduced. For $d_c = 12$, the $n_m = 20$ values (for each message) is reduced to a maximum of 10 for U'_1 and a minimum of 1 for U'_{12}, as shown in Fig. 3.1.

Table 3.2: Post synthesis results for different ECN schemes on a Xilinx Virtex 6 FPGA.

ECN	Number of occupied slices	Frequency (MHz)	Latency (cycles)
1B	7	714	1
S-1B	17	714	1
S-1B+1	35	349	1
S-2B	82	334	2
S-4B	138	269	2

Table 3.3: Post-synthesis results for the FB-CNs with (P-FB) and without (S-FB) pre-sorting on a Xilinx FPGA device.

FB-CN		Nb. of occupied slices				
d_c	Case	Sorter	Switch	CN	Total	Gain
6	S-FB	0	0	1,617	1,617	5%
	P-FB	50	93	1,268	1,532	
8	S-FB	0	0	2,481	2,481	17%
	P-FB	77	142	1,701	2,061	
12	S-FB	0	0	4,666	4,666	43%
	P-FB	160	283	1,858	2,653	
20	S-FB	0	0	6,519	6,519	54%
	P-FB	386	495	1,232	2,955	

3.1.3.2 Simulation results

Finally we present bit-true Monte-Carlo simulation results over the Additive White Gaussian Noise (AWGN) channel with a Binary Phase Shift Keying (BPSK) modulation scheme. Extrinsic and intrinsic LLR messages are quantified on 6 bits and the a-posteriori LLRs on 8 bits. The two scenarios presented in Fig. 3.1 are considered: 1) the S-FB which corresponds to the state-of-the-art S-Bubble approach without pre-sorting, 2) the P-FB which corresponds to the approach with pre-sorting and ECN simplifications.
Fig. 3.6 shows the simulation results for a (576, 480), $d_c = 12$ GF(64)-LDPC code. The pre-sorting technique shows negligible performance degradation while implementation results in Table 3.3 shows 43 % complexity reduction with $d_c = 12$. Thus, if slight degradation is accepted, further simplification is possible.

3.1.4 Extended Forward and hybrid CN

The FB and the SB CN architectures as well as the application of the presorting technique to both of them have been presented. In this section, we consider the hybridization of these approaches in a unique CN architecture. The first proposed

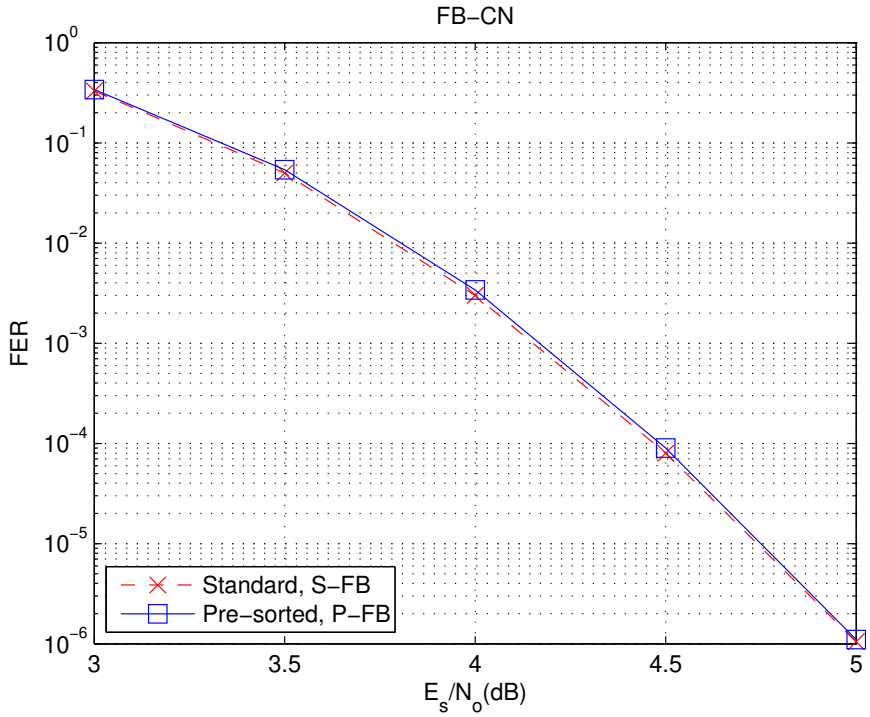

Figure 3.6: Simulation results of NB-LDPC decoding algorithms for (576, 480) code over GF(64) and $d_c = 12$ under AWGN channel.

hybrid architecture uses an Extended Forward (EF) processing to dynamically generate the set of syndromes. The objective is to take advantage of the simplicity of the SB architecture while keeping the complexity linear with d_c. The second original architecture introduces presorting in the EF to further reduce the complexity. Finally, a new level of hybridization is performed to take the most advantage of the presorting technique in the EF architecture.

3.1.4.1 Syndrome computation using the EF processing

A syndrome set S_b can also be computed by performing a forward iteration on all the inputs of the CN using a serial concatenation of ECNs (2.27) as

$$S_b = \boxplus_{i=0}^{d_c-1} U_i. \tag{3.1}$$

Applying the SB CN approach with ECNs of parameters (n_m, n_m, n_m) provides n_m syndromes sorted in increasing order of their associated LLR values. The syndrome set can be computed in a serial scheme as shown in Fig. 3.7 or it can also be computed with $\lceil \log_2(d_c) \rceil$ layers of ECNs using a tree structure.
Thanks to the use of ECNs, the computed syndrome set is sorted and can be directly applied to the decorrelation process. Note that in [23], a sorting process is required after the syndrome computation. However, the ECNs used in the EF architecture

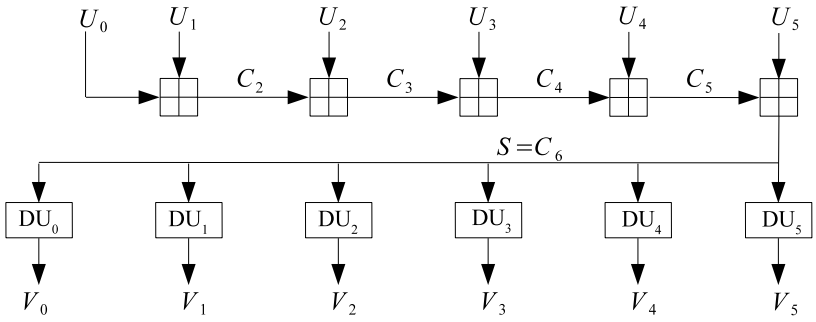

Figure 3.7: EF CN Architecture.

require a small additional patch compared to those in the FB architecture. The role of this patch is to construct the DBV (2.30), denoted here by $c^D_{a,b}$, during the ECN processing. The ECN addition (2.26) is then modified as

$$\begin{aligned} C_{a,b} = (c^+_{a,b}, c^{\oplus}_{a,b}, c^D_{a,b}) = (&A^+[a] + B^+[b], \\ &A^{\oplus}[a] \oplus B^{\oplus}[b], \\ &A^D[a] \| B^D[b]), \end{aligned} \tag{3.2}$$

where $\|$ represents the concatenation operation of two binary vectors.
The CN inputs are initialized with a DBV value of one bit as follows: $U^D_i[0] = 0$ and $U^D_i[j] = 1, \forall j > 0$. Thanks to the DBV computation, the output of the EF processing is similar to the output of the SB processing just before the decorrelation. In particular, the notion of deviation path can be also applied to the EF processing, with the only difference that the set of deviation paths Δ_{EF} is input dependent, while Δ is predefined offline in the SB architecture [23].
A first drawback of the EF is that the number of computed syndromes is typically $3 \times n_{m,out}$ to compensate the discarded redundant syndromes.

Even with this approach, the first simulation results of the EF algorithm showed significant performance degradation compared to the FB algorithm [50]. The reason of this performance degradation is the RE process performed by each ECN: since an ECN performs RE, no more than one ECN output could be associated to a given GF value.

However, since the ECN outputs in the EF algorithm are partial syndromes, RE may discard useful partial syndromes that would construct valid complete syndromes at the end of the EF processing.

In Fig. 3.8, an example of CN with $d_c = 4$, $n_{m,in} = 2$ and $n_{m,out} = 3$ is presented to illustrate the problem. The two deviation paths $\delta_1 = (1, 0, 0, 0)$ and $\delta_2 = (0, 1, 0, 0)$ lead to the same GF value, i.e., $\beta_4 + \beta_0 + \beta_4 + 0 = \beta_0 + \beta_4 + \beta_4 + 0 = \beta_0$. The output $C_1 = U_1 \boxplus U_2$ of the first ECN is equal to $C_1 = \{(0, 0, 00), (1, \beta_{24}, 10), (2, \beta_{24}, 01), (3, 0, 11)\}$

before the RE and equal to $C_1 = \{(0, 0, 00), (1, \beta_{24}, 10)\}$ after RE. Note that the seed of the partial syndrome δ_2 is eliminated. The final output in this example will be $S = C_3 = \{(0, \beta_4, 0000), (1, \beta_0, 1000), (7, \beta_{17}, 0010)\}$ and after the decorrelation unit, $V_0 = \{(0, \beta_{24}), (7, \beta_{47})\}$, instead of $V_0 = \{(0, \beta_{24}), (2, 0), (7, \beta_{47})\}$.

The key idea to avoid this problem is to allow redundant GF values in the syndrome set. Thus, removing the RE process from the ECN processing avoids performance degradation. Moreover, as a beneficial side effect, it also reduces the complexity of the ECN without impacting latency. In fact, the effect of the RE operation was to introduce idle cycles in the pipeline each time a symbol was deleted. The introduction of idle cycle is now avoided.

Let us define a modified ECN operation with symbol $\boxplus'$ where the ECN addition is performed as in (3.2) and no RE is performed. The syndrome set of size n_m can then be computed as

$$S'_b = \boxplus'^{d_c-1}_{i=0} U_i \tag{3.3}$$

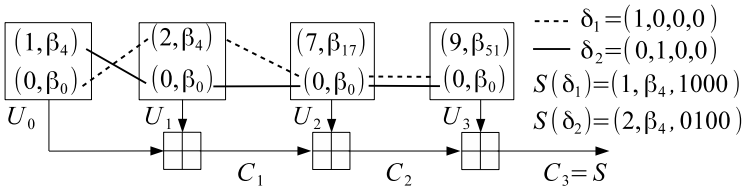

Figure 3.8: Example to illustrate the redundant syndromes.

The RE process will then take place after the decorrelation operation performed by the DUs. As previously mentioned, the set of paths Δ in the SB CN is pre-determined offline, while it is determined dynamically on the fly in the EF CN according to the current LLR values being processed. This leads to a significant reduction of the total number n_s of syndromes to be computed [49].

3.1.4.2 EF CN with presorting

As shown in previous sections, presorting leads to significant hardware savings by reducing the number of candidate GF symbols to be processed within the CN. In this section, we show that this presorting technique, when applied to the message vectors entering the EF CN, leads to a high complexity reduction of the CN architecture. This architectural reduction is obtained by reducing the number of bubbles to be considered at each ECN. For this, we perform a statistical study based on Monte-Carlo simulation that traces the paths of the GF symbols that contribute to the output of the CN, in their way across the different ECNs. This statistical study [48] identifies in each ECN how often a given bubble contributes to an output. This information allows pruning

the bubbles that never or rarely contribute. More formally, this study is conducted through the following two steps:

1. **Monte Carlo simulation** giving the trace of the different bubbles (each time a bubble b is used in an output message, its score $\gamma(b)$ is incremented).

2. **ECN pruning** that aims at discarding the less important bubbles, thus simplifying the ECN architectures.

How to prune low-score bubbles for best efficiency is still an open question. However, we propose here a method that prunes bubbles based on the statistics of their scores at each ECN. Let I_b be a sorted set of indexes of the potential bubbles of a given ECN verifying $\forall (b, b') \in I_b^2,\ b < b' \Rightarrow \gamma(b) \leq \gamma(b')$. Let τ be a real between 0 and 1 and let Γ be the cumulative score of all bubbles, i.e., $\Gamma = \sum_{b \in I_b} \gamma(b)$. The pruning process suppresses the first p bubbles associated to the first p indexes of I_b, with p defined as

$$p = \arg\max_{p' \in I_b} \{ \sum_{b=0}^{p'} \gamma(b) \leq \tau\Gamma \}. \tag{3.4}$$

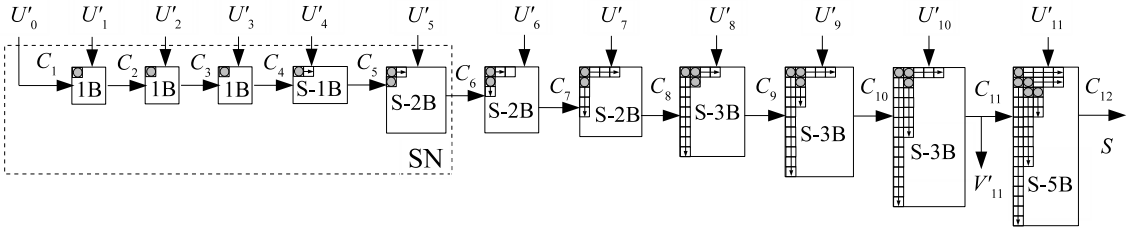

Figure 3.9: Architecture of the proposed PS EF CN with $d_c = 12$, $n_b \leq 4$ ($n_{m,in} = 5$), $n_c^{12} = n_s = 20$, where n_c^{12} is the number of output bubbles of S-5B.

After this pruning process, the structure of some ECNs is greatly simplified. The choice of the values of τ is a trade-off between hardware complexity and performance. As an example, Fig. 3.9 represents the remaining bubbles after the pruning process for a $d_c = 12$ GF(64) (144, 120) NB-LDPC code with $n_{m,out}$ set to 16. The pruning process has been performed for a SNR of 5 dB and a value of τ equal to 0.01, leading to different simplified ECN architectures similar to the case of S-FB with one extra type of ECN which is S-xB described as:

- **S-xB:** with $x > 1$, also known as S-bubble ECN. As described in [43], this architecture compares x bubbles per clock cycle.

In Fig. 3.9, we represent each bubble in an ECN by a filled circle and the direction for the next bubble by an arrow. The number of squares in each ECN represents the depth of the FIFO in its architecture. Note that the complexity of the ECNs increases from left to right. In fact, only trivial ECN blocks, i.e. 1B, S-1B, S-2B architectures, are required on the left part while a S-5B ECN is required on the right part. It

is possible to regroup several ECNs in a single component called Syndrome Node (SN). As detailed hereafter, this SN computes sorted partial syndromes in only one clock cycle, leading to significant latency reduction besides the hardware reduction compared to the EF.

3.1.4.3 The Syndrome Node

In Fig. 3.9 the first 3 ECNs are of type 1Bs and can be processed together in a single clock cycle by simply adding the most reliable GF values of all inputs: $C_4 = \{(0, C_4^{\oplus}[0], C_4^{D}[0])\}$, where $C_4^{\oplus}[0] = \bigoplus_{i=1}^{4} U_i'^{\oplus}[0]$ and $C_4^{D}[0] = 0000$. Also, the $n_c^6 = 3$ values of C_6 can also be computed in one clock cycle. In fact, thanks to presorting, $U_6'^{+}[1] \leq U_5'^{+}[1]$ and the first three partial syndromes are

$$
\begin{aligned}
C_6[0] &= (0, C_4^{\oplus}[0] \oplus U_5'^{\oplus}[0] \oplus U_6'^{\oplus}[0], 000000) \\
C_6[1] &= (U_6'[1]^{+}, C_4^{\oplus}[0] \oplus U_5'^{\oplus}[0] \oplus U_6'^{\oplus}[1], 000001) \\
C_6[2] &= (U_5'[1]^{+}, C_4^{\oplus}[0] \oplus U_5'^{\oplus}[1] \oplus U_6'^{\oplus}[0], 000010),
\end{aligned}
$$

In summary, we can consider all these computations to belong to a unique block, i.e. the SN, that involves several ECNs (to be specific, 5 ECNs in the example of Fig. 3.9) but that generates its outputs in a single clock cycle.

3.1.4.4 Hybridization between FB and EF CN architectures

Combining the EF architecture and the FB approach leads to a reduction of the total number of needed syndromes to guarantee a given number of valid output syndromes. Fig. 3.10 shows the average number of syndromes that should be computed for a given output V_i' to obtain, with a probability of 90%, $n_{op} = 18$ valid syndromes. This number is denoted by $n_s^{0.9}(i)$ and varies for each output V_i'. Note that when the presorting technique is considered, $n_s^{0.9}(i)$ increases with i. To decode without performance degradation, the number of computed syndromes is bounded by the number of syndromes required by the last output, i.e. $n_s = n_s^{0.9}(12) = 46$ in the example of Fig. 3.10.
Fig. 3.9 shows that V_{11}' can be directly obtained from C_{11} without DU, as C_{11} contains the contribution of all the inputs except U_{12}', i.e. $V_{11}' = C_{11}$. This result can be seen as the application of the FB algorithm on the output V_{11}' since the forward process of the FB algorithm is included in the EF systematically generating V_{11}'. Consequently, the number of required syndromes can be reduced from $n_s(12) = 46$ down to $n_s(11) = 36$. This reduces the overall complexity and latency of the EF CN architecture without performance degradation. Note that this constitutes a first example of a hybrid architecture where one output is generated with the FB approach and the other $d_c - 1 = 11$ outputs with the EF CN. This kind of approach can be generalized, as described by the following.

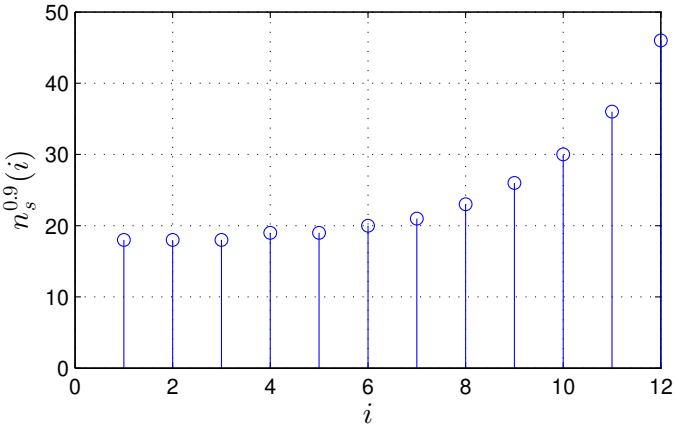

Figure 3.10: Maximum number of syndromes needed to be generated, for each output V'_i, $n_{op} = 18$ valid syndromes. The output number is denoted by i. The code rate is $R = 5/6$ and $\mathrm{E}_b/\mathrm{N}_0 = 4.5$ dB.

3.1.4.5 General notations for hybrid architectures

Let HB(ρ_{SN}, ρ_{EF}, ρ_{FB}) be an hybrid architecture that combines the SN, EF and FB schemes. The first ρ_{SN} inputs are processed by a SN block, the next ρ_{EF} inputs are processed by an EF block and the remaining n_{FB} inputs are processed by a FB block. Obviously, $\rho_{SN} + \rho_{EF} + \rho_{FB} = d_c$.

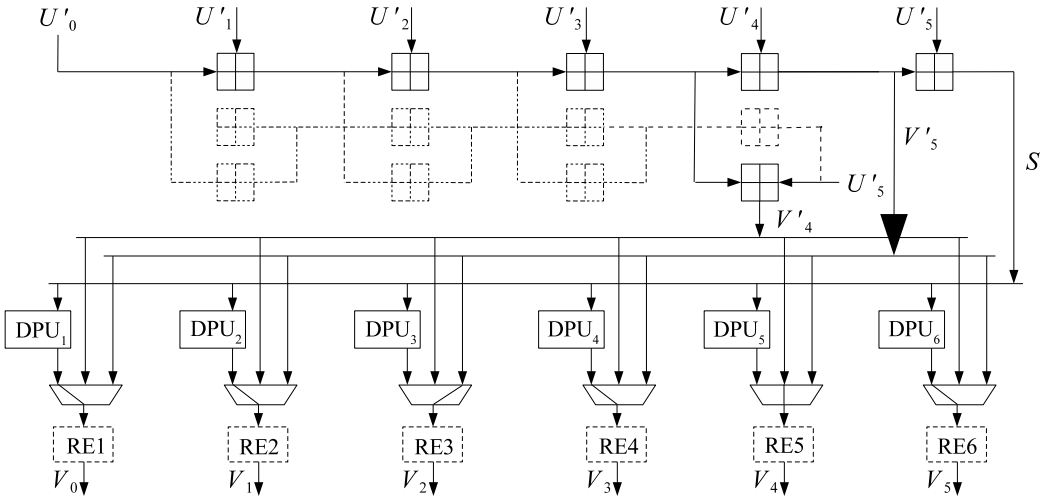

Figure 3.11: HB(0, 4, 2) architecture for a CN with $d_c = 6$. The last two outputs V'_3 and V'_4 are generated by a classical FB architecture.

Fig. 3.11 shows the HB(0, 4, 2) architecture for a CN of degree 6. As shown, V'_4 and V'_5 are computed using the FB algorithm in order to further reduce the number of required syndromes. There are several possible HB architectures between the EF

(i.e., HB(0, 6, 0)) and the classical FB-CN (i.e., HB(0, 0, 6)). Note that V'_5 (resp. V'_4) should bypass the decorrelation units and should be directly connected to $V_{\pi(5)}$ (resp. $V_{\pi(4)}$). Fig. 3.11 shows the case where $\pi(5) = 2$, i.e. the third multiplexer connects V_2 to V'_5, and $\pi(4) = 4$, i.e. the fifth multiplexer connects V_4 to V'_4. Finally, V_0, V_1, V_3 and V_5 are each one connected to the output of the corresponding DPU. Fig. 3.12 shows the HB(6, 4, 2) architecture for a CN of degree 12. V'_{10} and V'_{11} are computed using the FB algorithm and a SN is used to process the 6 first input U'_0 to U'_5.

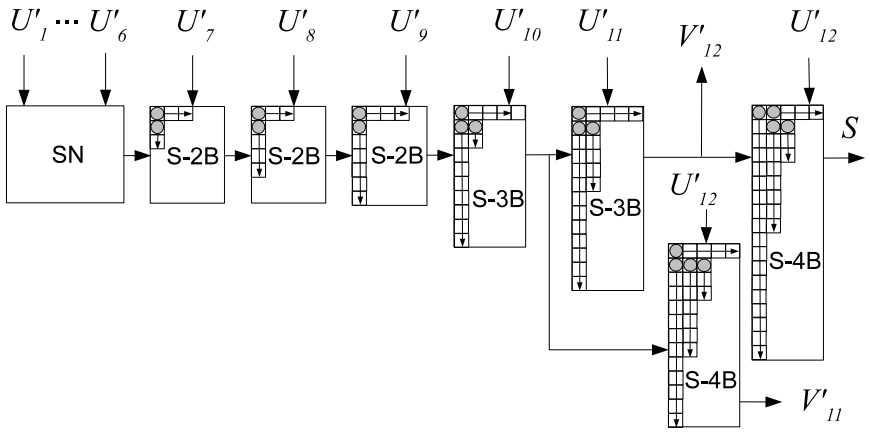

Figure 3.12: HB(6, 4, 2) architecture with $d_c = 12$, $n_{m,out} = 16$, $n_{m,in} = 5$ and $n_s = 20$.

3.1.4.6 Choice of parameters $(\rho_{SN}, \rho_{EF}, \rho_{FB})$

The determination of the CN architecture parameters, i.e. $(\rho_{SN}, \rho_{EF}, \rho_{FB})$ for the macro level, the internal structure of the EF and the FB blocks (the parameters of each ECN) for the micro level, is a complex problem. It can be formulated as an optimization problem: how to minimize the hardware complexity without introducing significant performance degradation. In this section, we have first limited the value of ρ_{FB} to 1 and 2. Then, for the two hypothesis $(0, d_c - \rho_{FB}, \rho_{FB})_{\rho_{FB}=1,2}$, we have applied the method described in Section 3.1.4.2 to determine the parameters of each ECN of the EF and FB blocks. Note that after the automatic raw pruning process described in Section 3.1.4.2 , the parameters are further tuned by hand using a "try and see (i.e. estimate performance by simulation)" method. Once the pruning process is completed, the value of ρ_{SN} is fixed in order to optimize the hardware efficiency of the CN architecture. In fact, at a given point, CN with parameters $(\rho_{SN}+1, \rho_{EF}-1, \rho_{FB})$ will have a higher hardware complexity than CN with parameters $(\rho_{SN}, \rho_{EF}, \rho_{FB})$ but with a lower decoding latency.

3.1.4.7 Suppression of final output RE

In some decoder implementations [31] [52] with $d_v = 2$, the VNs connected to a CN are updated right after the CN update. For example, in Fig. 3.11 a variable node unit may be connected directly to each output V_0 to V_5 right after the RE units RE_1 to RE_6. In these implementations, RE is performed in the VN. In this case, the RE block can be removed from the hybrid architecture for complexity reduction. The suppression of the d_c RE blocks is specially interesting for high d_c values. In a HB architecture with final RE, the RE reduces the number of output messages from n_s (in case that all message are valid) to $n_{m,out}$. By removing the RE, the number of output messages becomes $n_s > n_{m,mout}$. The impact on complexity is limited since the $n_{m,out}$ elements are not stored but computed on the fly serially by the VN. However, it may impact slightly the VN consumption since the VN will have to deal with n_s elements instead of $n_{m,mout}$ elements. The suppression of RE does not affect the algorithm output, and thus, does not affect performance.

3.1.5 Performance and complexity analysis

We consider GF(64)-LDPC and GF(256)-LDPC codes to obtain performance and post-synthesis results for the different proposed decoding architectures.

3.1.5.1 Performance

We ran bit-true Monte-Carlo simulations over the AWGN channel with BPSK modulation scheme. The different parameters were set as follows: extrinsic and intrinsic LLR messages quantified on 6 bits, the *a posteriori* LLRs on 7 bits and the maximum number of decoding iterations to 10. The matrices used in our simulations are available in [41].
Fig. 3.13 shows the obtained Frame Error Rate (FER) for a GF(64) code of size (864,720) bits, code rate $R = 5/6$, $d_c = 12$ and $d_v = 2$ over the AWGN channel. We consider the FB decoder in [43] as a reference, i.e. S-bubble algorithm with 4 bubbles, $n_m = 16$ and $n_{op} = 18$. We simulated the HB(6, 6, 0) or EF, the HB(6, 5, 1) and the HB(6, 4, 2) architectures with the same number of computed syndromes $n_s = 20$. Fig. 3.12 shows the HB(6, 4, 2) architecture, for which no performance degradation is observed. We observe less than 0.05 dB of performance loss for the HB(6, 5, 1) and around 0.2 dB for the HB(6, 6, 0) configuration. We then conclude from these simulation results that the hybrid architectures can achieve the same performance as the FB architecture and outperform the EF architecture while needing 3 or 4 less syndromes compared to the original EF approach.
Fig. 3.14 shows performance results for a GF(256)-LDPC code of size (1152, 960) bits, code rate $R = 5/6$, $d_c = 12$ and $d_v = 2$. We consider as a reference the FB decoder with a S-bubble architecture [43], 6 bubbles, $n_m = 40$ and $n_{op} = 45$. The HB(5, 5, 2) architecture presents the same performance as the FB and the HB(5, 6, 1) shows

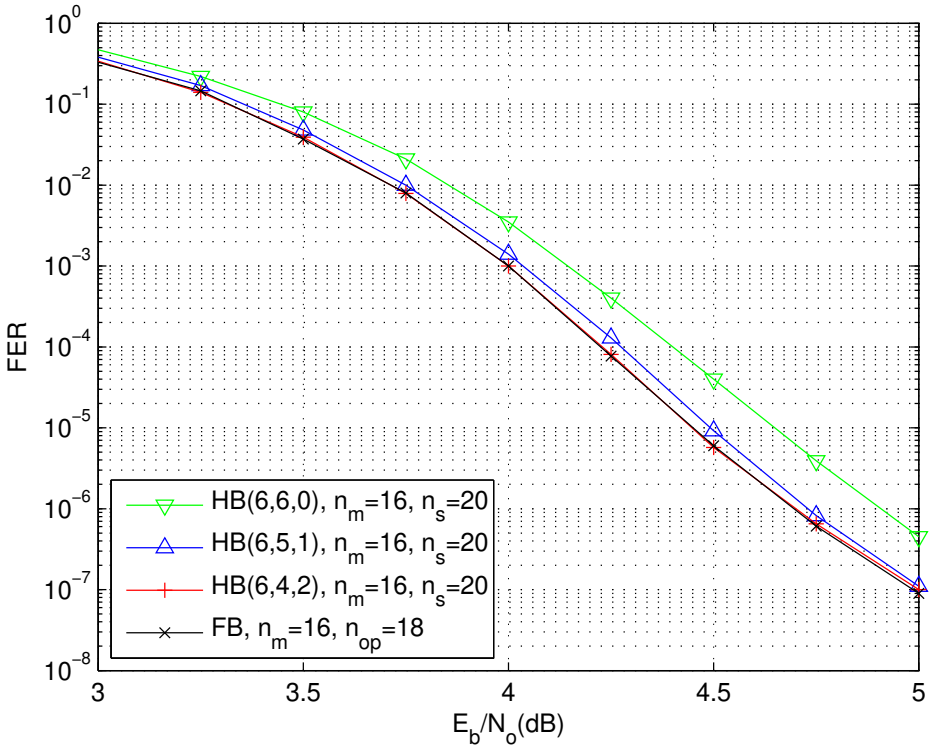

Figure 3.13: FER performance for a (144, 120) NB-LDPC code over GF(64).

a performance loss smaller than 0.05 dB. The HB(7,5,0) architecture (or equivalently the EF architecture) presents around 0.1 dB of performance loss compared to the FB. We can then conclude that this new family of hybrid architectures allows for significant complexity reduction in CN implementations without any performance loss compared to more complex state-of-the-art solutions.
Finally, Fig. 3.15 shows the simulation results of one of the rare GF(64) implementation for high rate in the literature where CR= 7/8, $d_c = 16$ and $d_v = 2$. We consider as a reference the FB decoder with a S-bubble architecture [43], 4 bubbles, $n_m = 16$ and $n_{op} = 18$. The performance of the Trellis Min-Max (T-MM) algorithm [51] is also presented for comparison (same code rate and length are considered). The architecture used is the same as the one presented in Fig. 3.12 except that the SN includes four more 1B ECNs, (i.e. the HB(10, 4, 2) CN architecture). Once again, at lower hardware complexity, the hybrid architectures show similar performance as the original FB architecture and outperform the T-MM based one.

3.1.5.2 Implementation results

For complexity and power analysis, we considered the implementation of the architectures on 28 nm FD-SOI technologies targeting a clock frequency of 800 MHz. The different kinds of ECNs presented in Fig. 3.9, were synthesized individually to provide

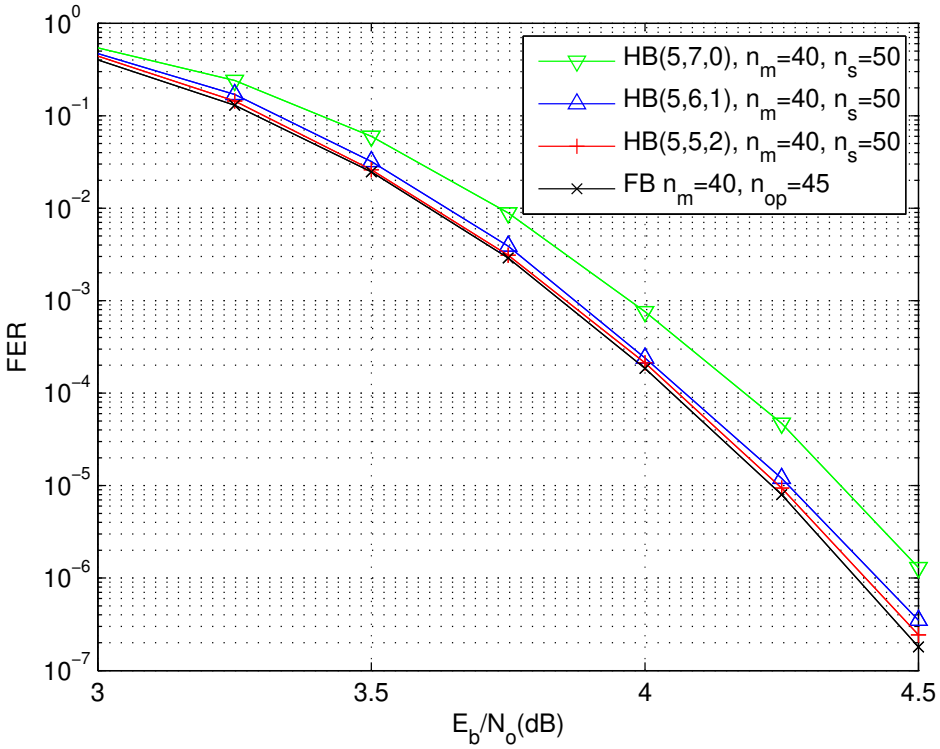

Figure 3.14: FER performance for a (144, 120) NB-LDPC code over GF(256)

.

the results in Table 3.4. Additionally, the synthesis results of the SN, the S-bubble with RE (used in the FB-CN), the sorter, the switch, the DU (Fig. 2.7) and the RE units synthesis results are also provided. The sorter is implemented using a serial architecture as in [48], and the switch is a cross bar switch. The minimum clock period (P_{clk}) is given in nanoseconds with a clock uncertainty of 0.05 ns and a setup time of 0.02 ns. The Cycle Latency (CL) represents the number of clock cycles between the first input and the first output. A reduction factor of 57, 34 and 7 is observed betweeen the 1B and the S-4B RE architectures in terms of area, power and clock period, respectively. These results show that significant gain can be obtained even if it implies the overcost of the presorter, the switch and the DPU units.
Table 3.5 summarizes the implementation results for all the CN architectures presented in this chapter, for a GF(64) and a GF(256)-LDPC codes with $d_c = 12$. In this Table we present the synthesis results with and without RE considering that the RE can be suppressed in implementations when $d_v = 2$ [31]. The Check Latency of CN, $CL(CN)$, is the clock cycles latency between the first input and the first output of a CN, taking into account the latency of the ECNs, the Pre-Sorter, the switch, the DPU, the RE and the GF multiplication and division. For the FB architecture, $CL(CN)$ is given as: CL(CN) = CL(mult) + $(d_c - 2)$ × CL(S − 4BRE) + CL(div) = 22. For the HB(6,4,2) architecture (Fig. 3.12), CL(CN) = CL(Sorter) + CL(Switch) + CL(SN) +

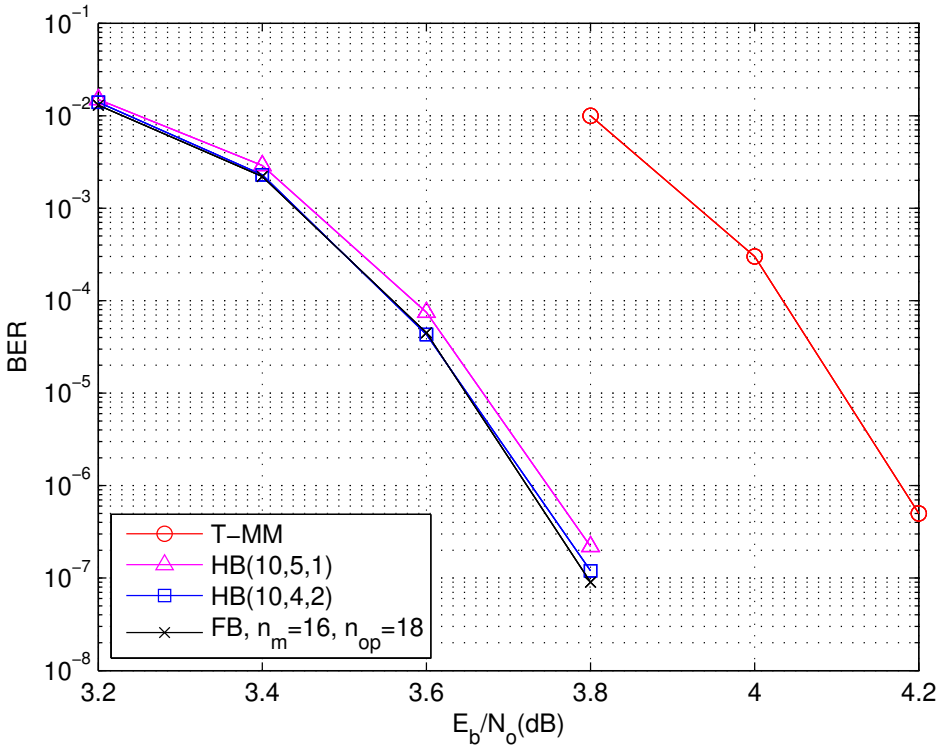

Figure 3.15: BER performance for a (1536, 1344) NB-LDPC code over GF(64).

$3 \times \text{CL(S} - \text{2B)} + 2 \times \text{CL(S} - \text{3B)} + \text{CL(S} - \text{4B)} + \text{CL(DPU)} = 14$, considering that the multiplication is performed in the same cycle as the switch and the division is performed in the same cycle as the DPU.
For GF(256) results, the FB architecture is with $n_m = 40$ and S-6B ECNs, the EF and HB architectures consider ECNs with a maximum $n_{m,in}$ value of 6.

3.1.5.3 Area and energy efficiency comparison

To compare the efficiency of the different CN architectures, we consider the number of computed CNs per second as follows: $T_{CN} = F_{clk}/(\text{CL(layer)})$ where CL(layer) is the periodicity of a CN computation in a layered decoder and F_{clk} is the clock frequency of the design. In our design, $\text{CL(layer)} = \text{CL(CN)} + \text{CL(VN)} + n_{m,out} + n_{m,int}$ where CL(VN) is 7, $n_{m,int} = 4$ for GF(64) and $n_{m,int} = 6$ for GF(256). Table 3.5 illustrates the implementation results of the different CN architectures with and without PS and RE. As shown, the PS highly reduces the complexity of the EF and FB architectures. RE induces additional area and power consumption. It is clear that the HB architectures are less area and power consuming as compared to the FB and EF ones. In order to give a more accurate assessment, we have evaluated two new metrics: Area Efficiency (AE) and Energy Efficiency (EE) defined in the following. AE is defined as the number of computed CN per second per mm^2: $\text{AE} = T_{CN}/\text{Area}$.

Table 3.4: Post-synthesis results for different ECN architectures and CN sub-units on 28 nm FD-SOI technology.

		Area (μm^2)	**Power** (mW)	$\boldsymbol{P_{clk}}$ (ns)	**CL** (cycles)
ECNs	1B	77	0.081	0.15	1
	S-1B	170	0.16	0.25	1
	S-2B	2570	1.66	0.79	1
	S-3B	3227	2.15	0.88	1
	S-4B	4022	2.57	1.03	1
	S-6B	5413	3.43	1.11	1
	S-4B RE	4428	2.76	1.03	2
	S-6B RE	5818	3.64	1.11	2
CN Sub-units	6-input SN	354	0.34	0.31	1
	PreSorter 12	1196	0.96	0.84	6
	PreSorter 16	1600	1.07	0.84	8
	Switch	2724	1.95	0.28	1
	DPU	187	0.177	0.22	1
	RE	606	0.407	0.71	1
	mult 64	107	0.070	0.34	1
	mult 256	178	1.082	0.43	1

EE is defined as the number of computed CNs per mJ per second: EE $= T_{CN}/(\text{Power})$. The clock frequency F_{clk} is set at 800 MHz. Table 3.6 compares both AE and EE for the most relevant architectures of each type, i.e., FB, EF and HB. The HB(6,5,1) and HB(6,4,2) in GF(64) improve the AE compared with the FB architecture by a factor of 6.8 and 6.2, respectively. When comparing the EE, the improvement factors are of 6.4 and 5.5.
To compare the HB to the SB, we refer to [54]. Table 3.7 presents the obtained results where the areas are evaluated in a 65 nm CMOS technology. In fact, the area of the HB architecture (marked with an asterisk) was obtained from 28 nm technology with a normalization factor of 4. The throughput T_e is expressed in terms of giga elements (output symbols) per second (Gel/s) and the area efficiency in terms of Gel/s mm^2. In our case, $T_e = d_c \times F_{clk} \times n_{m,out}/\text{CL(layer)}$.
Focusing on AE as a function of d_c for HB and SB, one can note that AE of HB increase with d_c while AE of SB decrease with d_c. The d_c threshold for which HB becomes more efficient than SB can be estimated at $d_c = 10$ deduced as a linear interpolation of obtained result.

3.1.5.4 Throughput

The throughput can be greatly improved by processing p CNs in parallel in a layered decoder [50] [52]. Because we consider a code with $d_v = 2$, the VNs are cascaded after the CN as in [31] [50]. The throughput of a layered decoder is given by $T = (N \times F_{clk} \times$

Table 3.5: Post-synthesis results for CN architectures on 28 nm FD-SOI technology.

GF	**CN**	**Area** (mm^2)	**Power** (mW)	P_{clk} (ns)	**CL(CN)** (cycles)
64	FB	0.140	94	1.02	22
	PS FB	0.037	26	1.12	20
	EF RE	0.125	83	1.22	13
	PS EF	0.0328	26	1.12	19
	PS EF RE	0.037	28	1.12	19
	HB(6,6,0)	0.0227	14.9	1.03	15
	HB(6,6,0) RE	0.0269	17.1	1.03	15
	HB(0,10,2)	0.0257	16.5	1.17	19
	HB(0,10,2) RE	0.0292	18.4	1.17	19
	HB(6,5,1)	0.0228	15.2	1.04	15
	HB(6,5,1) RE	0.0271	17.4	0.99	15
	HB(6,4,2)	0.0259	19	1.00	15
	HB(6,4,2) RE	0.0306	19.4	0.99	15
	HB(10,4,2)	0.0307	30	0.99	17
	HB(10,4,2) RE	0.0363	35	0.95	17
256	FB	0.328	210	1.14	22
	PS EF	0.074	54	1.06	19
	PS EF RE	0.0909	65	1.17	19
	HB(5,6,1)	0.0753	45	1.2	16
	HB(5,6,1) RE	0.0871	48	1.15	16
	HB(5,5,2)	0.0803	45.4	1.20	16
	HB(5,5,2) RE	0.094	52	1.20	16

Table 3.6: Area and energy efficiency for different architectures.

GF	**CN**	**CL(layer)** (cycles)	T_{CN} ($\frac{\text{Mcn}}{\text{s}}$)	**AE** ($\frac{\text{Mcn}}{\text{mm}^2\,\text{s}}$)	**EE** ($\frac{\text{Mcn}}{\text{mJ s}}$)
64	FB	47	20.8	148	0.18
	PS FB	45	19.8	535	0.68
	PS EF	46	19.0	579	1.45
	HB(6,5,1)	42	23.1	1013	1.25
	HB(6,4,2)	42	23.8	919	1.00
	HB(10,4,2)	44	22.7	739	0.61
256	FB	74	11.8	36	0.051
	HB(5,6,1)	69	12.1	160	0.257
	HB(5,5,2)	69	12.1	151	0.255

Table 3.7: HB and SB comparison

CN	Tech. (nm)	d_c	F_{clk} (MHz)	Area (mm^2)	T_e ($\frac{Gel}{s}$)	AE ($\frac{Gel}{s\,mm^2}$)
SB	65	4	400	0.067	5.2	77.5
SB	65	12	280	0.189	2.9	15.3
HB	28	12	1000	0.1036*	4.57	37.2
HB	28	16	1000	0.1228*	5.82	40.0

$p)/(M \times \mathrm{CL(layer)} \times it_{avr})$ where it_{avr} is the average number of iterations and F_{clk} is fixed at 800 MHz. The layered GF(64) (1536, 1344) code allows different parallelism options depending on the splitting factor [53] ($p = 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96$) in two layers and the average number of iterations at SNR = 4.0 is 1.5. With a parallelism of 12, the throughput can reach 4.4 Gbps for GF(64) and 3.3 Gbps for GF(256) at SNR equal to 4.0 db.

3.1.6 CN Skip Processing Controller (SPC)

In this section, the SPC block before the CNP is shown. The aim of the SPC is to reduce the average number of processed CN per iteration. Consequently, the power consumption and/or the throughput will be improved. In the following we explain how the SPC block operates.

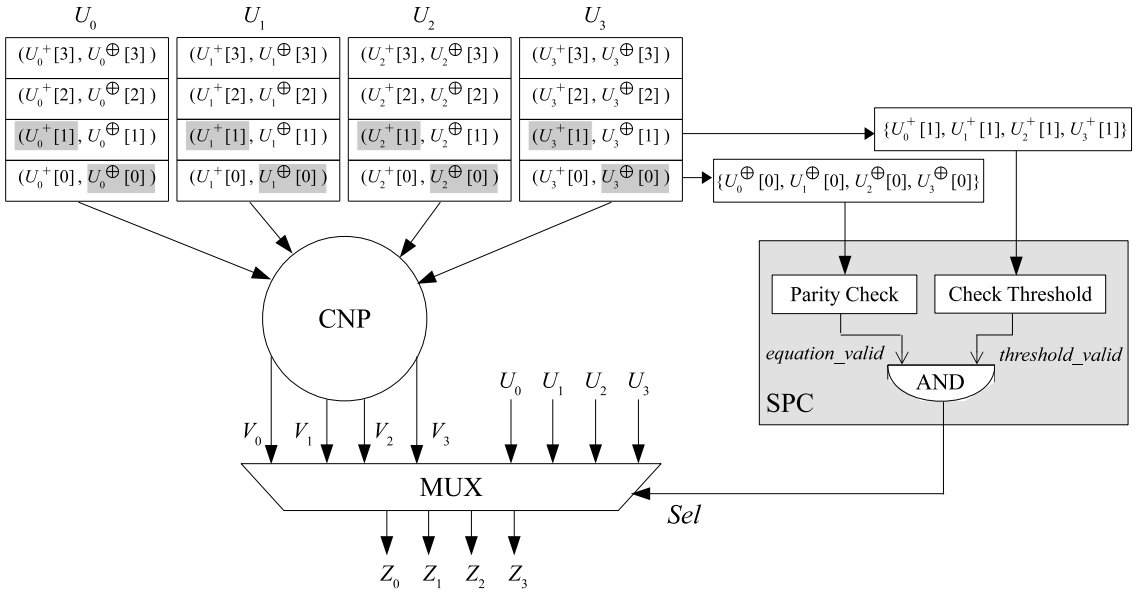

Figure 3.16: CN with SPC.

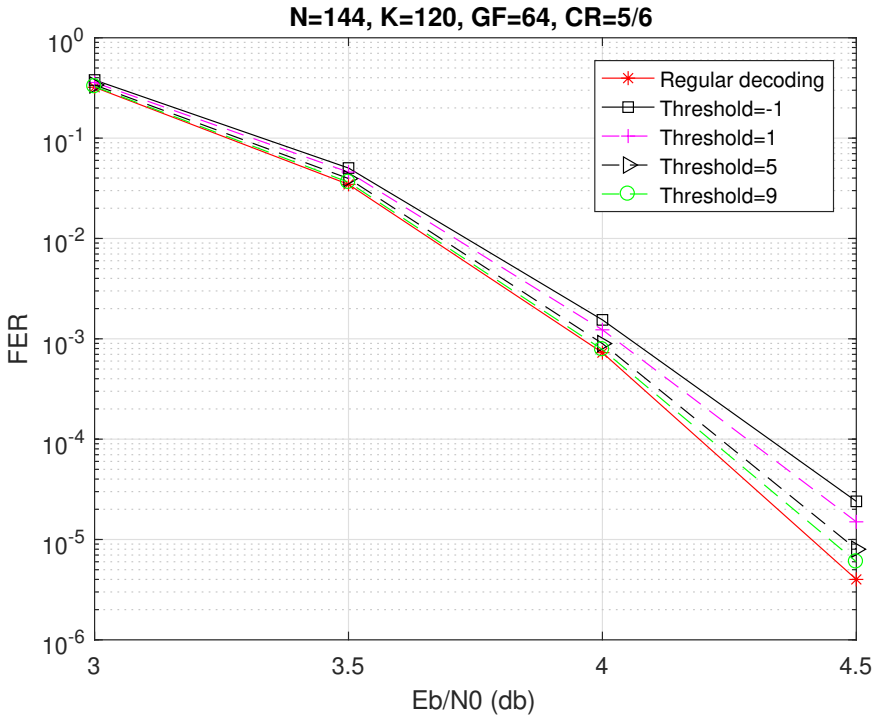

Figure 3.17: Simulation results in case of CR=5/6.

Fig. 3.16 shows the principal of the proposed block in case of $d_c = 4$ and $n_m = 4$. The idea is to skip the CN processing when a predefined criteria is satisfied. Thus, a SPC is inserted at the CN input. The SPC performs two operations: 1) Parity check: $\sum_{i=0}^{d_c-1} h_{ji}.U_i^{\oplus}[0] = 0$ $(j = 0, \ldots, M-1)$; 2) Check threshold: if $U_i^{+}[1] > \gamma$ where $i = 0, \ldots, d_c - 1$ and γ is a predefined threshold. If both conditions (1) and (2) are satisfied, the SPC gives an indication to skip the CN processing and hence the outputs of MUX are $Z_i = U_i$, otherwise, the CN update will be performed and hence $Z_i = V_i$ $(i = 0, \ldots, d_c - 1)$.
Fig. 3.17 shows the simulation results for different threshold values for a (144, 120) NB-LDPC code defined over GF(64). In the worst case where $\gamma = -1$ (i.e, only condition (1) must be satisfied) there is a performance loss of about 0.2 db compared to the regular decoding process, while the degradation is 0.06 db in case of $\gamma = 9$.

Fig. 3.18 shows the percentage of skipping CN processing. For $\gamma = -1$, the saving reached about 50% for Eb/N0 = 4.5 db while it is equal to 5% for Eb/N0 = 3 db. In case of threshold = 9, the saving varies between 1% and 7% (Eb/N0 is between 3 db and 4.5 db).

Fig. 3.19 and Fig. 3.20 show the simulation and saving results respectively for a (210,189) NB-LDPC.

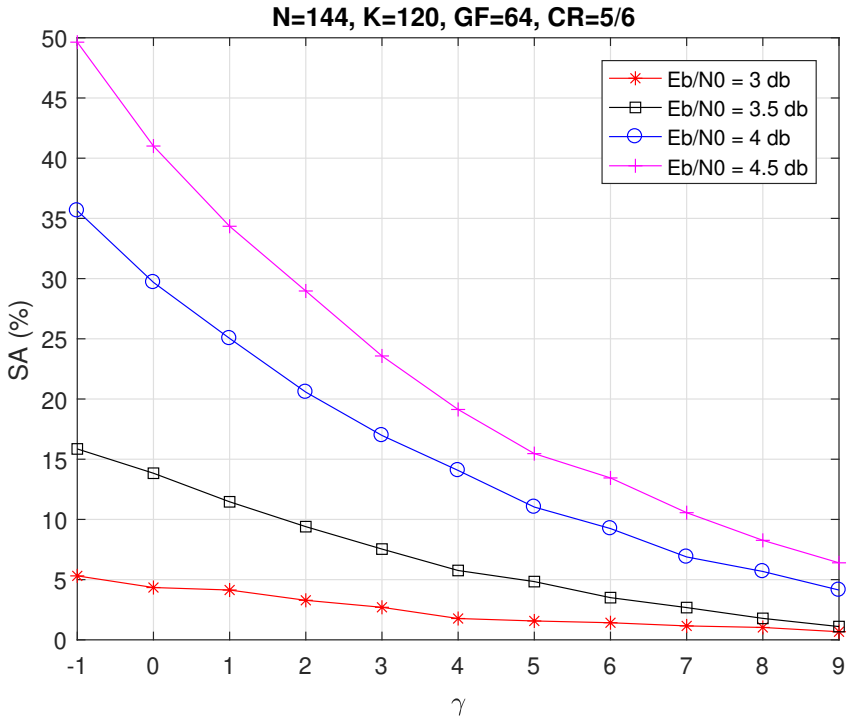

Figure 3.18: Saving in % of not making a CN for CR=5/6.

The Saving Amount (SA) is computed as follows:

$$N_c = N_f \times M \times N'_a - N_s.$$

$$\text{SA} = 100 \times \left(\frac{N_f \times M \times N_a - N_c}{N_f \times M \times N_a} \right) = 100 \times \left(\frac{N_s - N_f \times M \times (N'_a - N_a)}{N_f \times M \times N_a} \right). \quad (3.5)$$

In which, N_f is the total number of simulated frames, N'_a is the average number of iterations of the decoding process with SPC, N_s is the number of skipped CNs and hence N_c is the total number of executed CNs in case of CN with SPC. Thus, SA can be negative in case that the term $N_f \times M \times (N'_a - N_a) > N_{SCN}$, i.e, if the total number of executed CNs in case of SPC is greater than the number of executed CNs in case of regular decoding process ($N_c > N_f \times M \times N_a$). This case does not occur if a fixed number of iterations is considered, which in the worst case SA will be equal to 0 %.

3.2 New VNP architecture

In this section we present the modifications we propose to improve the VNP architecture presented in [6] and described in section 2.4. We first present the VNP

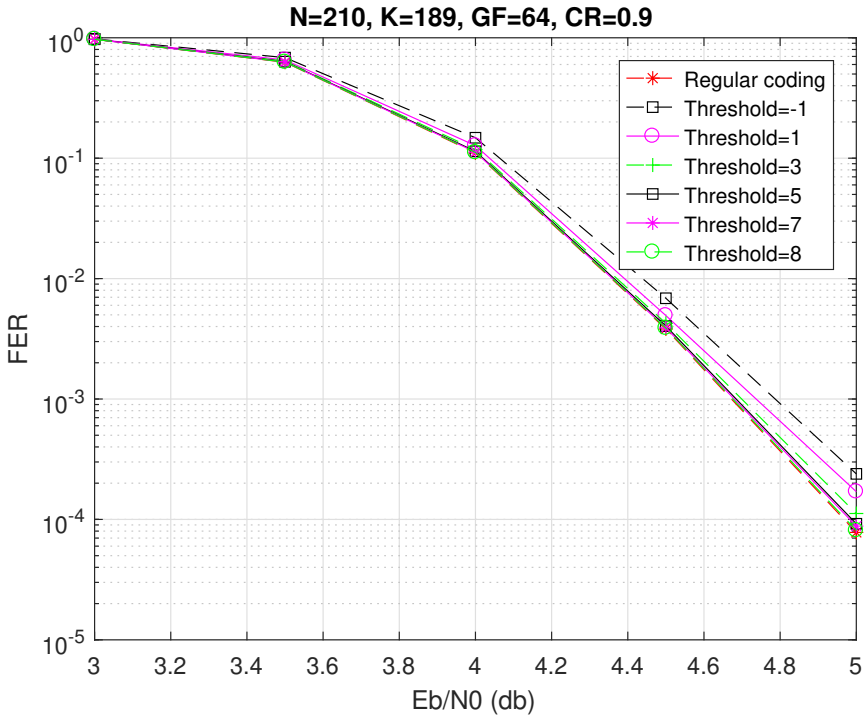

Figure 3.19: Simulation results in case of CR=9/10.

architecture in both update and decision modes, then a complexity analysis is given.

3.2.1 Proposed VN architecture

The proposed VN architecture performs the three operations mentioned in section 2.4: generation of the intrinsic couple candidates, VN update mode and decision-making mode. The simplification is done on both VN update and decision-making modes.

3.2.1.1 VNP in update mode

Fig. 3.21 depicts the architecture of the new VNP architecture. In the first phase, the signal *Sel* is set to 0 to update the message M_{pl^v} described in equation (3.6). In the second phase, *Sel* is set to 1 and the n_β intrinsic messages are updated:

$$\begin{aligned} M_{\mathrm{VN}}^{\oplus}(n_m+i) &= I^{\oplus}(i) \\ M_{\mathrm{VN}}^{+}(n_m+i) &= I^{+}(i) + M_{pl^v}^{\oplus}(n_m-1) + \mathrm{O} \qquad i=0,1,\ldots,n_\beta-1 \end{aligned} \tag{3.6}$$

The Sorter+Redundant Elimination (SRE) block sorts the n_m+n_β messages and discards, on the fly, the redundant symbols to generate the most reliable and valid n_m reliable messages. Compared to the architecture depicted in Fig. 2.10, the Flag

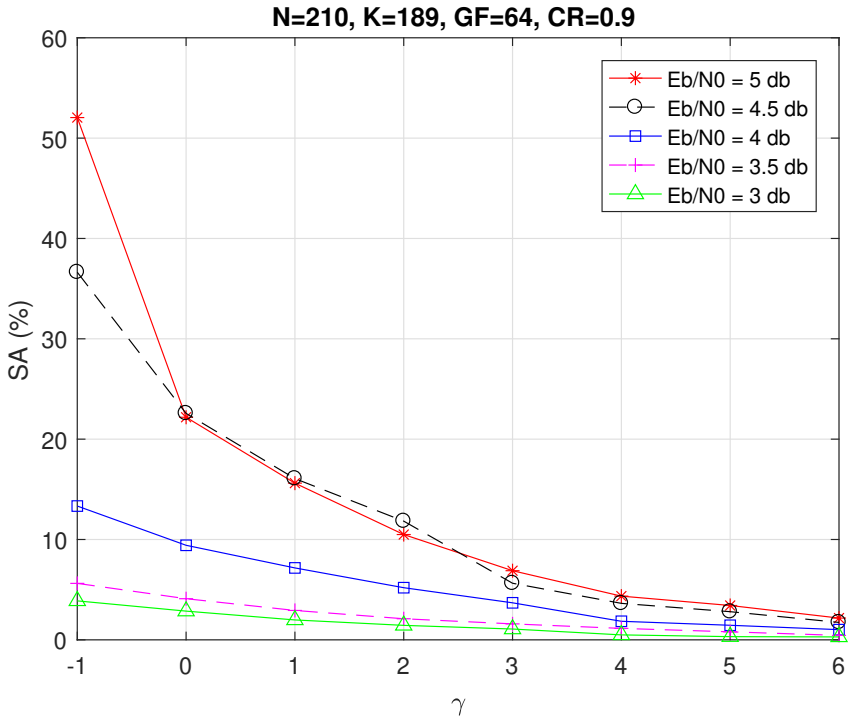

Figure 3.20: Saving in % of not making a CN for CR=3/4.

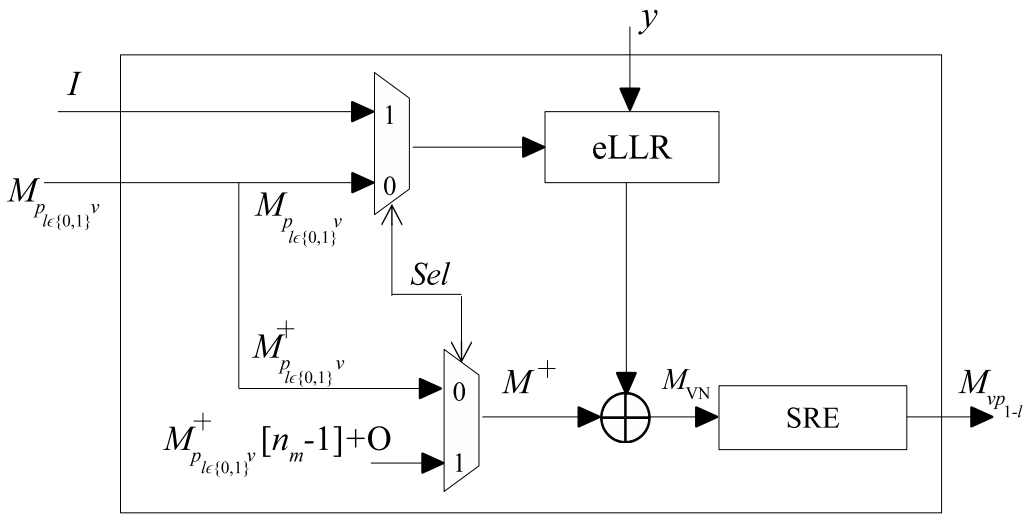

Figure 3.21: Proposed architecture of the VN update mode

of size q with its associated logic control are removed, hence reducing the complexity of the global architecture of the VN.

Architecture of the SRE block

The comparator-swap presented in Fig. 2.14 is modified to also apply the redundancy elimination. The new architecture is presented in Fig. 3.22.

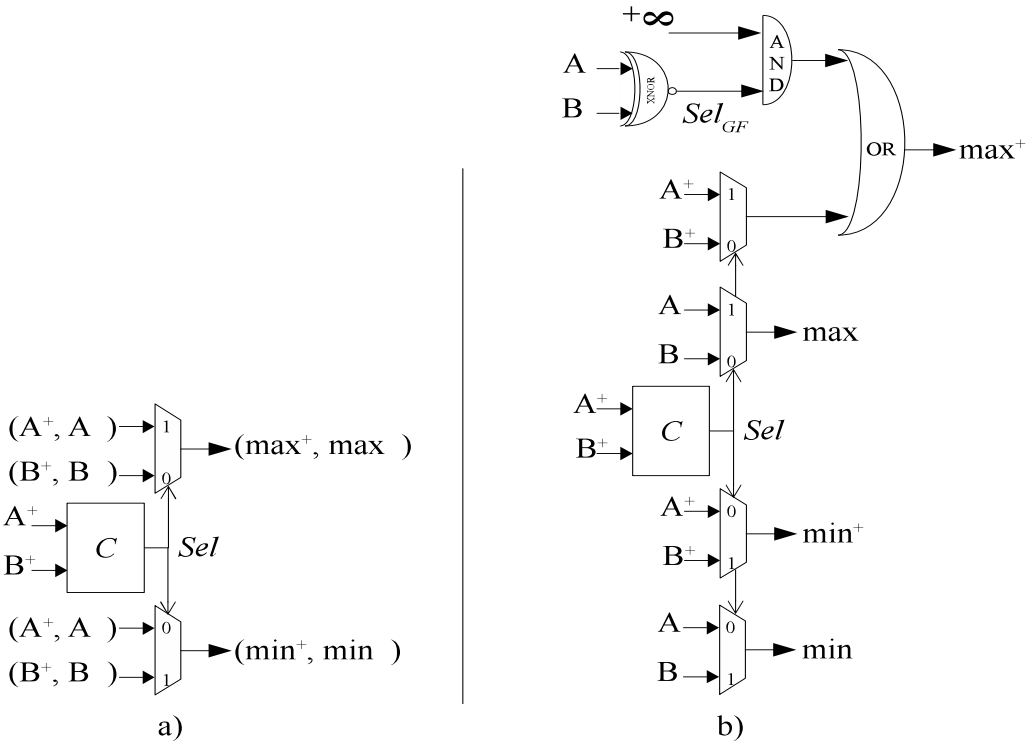

Figure 3.22: Architectures of the classical and the modified comparator-swap

Let us consider two input couples A=(A^+, $A^\oplus$) and B=(B^+, $B^\oplus$). The classical comparator-swap is shown in Fig. 3.22.a), where min=A and max=B when $A^+ < B^+$ ($Sel = 0$) or min=B and max=A when $B^+ < A^+$ ($Sel = 1$). The same approach for the modified one is presented in Fig. 3.22.b), with an additional XNOR gate to compare $A^\oplus$ to $B^\oplus$ and generate a signal $Sel_{GF} = 1$ in case of equality. Thus, if Sel_{GF} is 1, a redundant symbol is detected and should be discarded by setting its LLR to a saturation (maximum LLR) value. We take use of the structure of the incoming messages that are sorted in terms of LLR value. For instance, if $M^\oplus_{p_l v}[i] = M^\oplus_{p_l v}[j]$ ($\{i, j\} \in \{0, 1, \ldots, n_m - 1\}^2, i < j$), and since $M^+_{p_l v}[i] \leq M^+_{p_l v}[j]$ and the same intrinsic LLR value generated by the eLLR block is added on both of them, thus, $M_{\mathrm{VN}}[i] \leq M_{\mathrm{VN}}[j]$. Therefore, $M_{\mathrm{VN}}[i]$ is in position k in the sorter ($k = 0, 1, \ldots, n_m - 1$) then $M_{\mathrm{VN}}[j]$ surely passes by position k and then the suppression will be done.
The suppression part works as follows: if $A^\oplus = B^\oplus$ then $Sel_{GF} = 1$ and $\max^+$ takes the $+\infty$ value (saturation value in practice) which forces its exclusion from the sorted list. Therefore, the latency of VNP in VN mode is still equal to $n_m + n_\beta + 2$ (see Fig. 2.11).

3.2.1.2 Proposed architecture of the decision-making circuit

The decision-making is performed taking into account the three sets Φ_1, Φ_2 and Φ_3:

$\Phi_1 = \{M_{vp_1}^{\oplus}(i)\}_{i=0,1,\ldots,n_s-1} \cap M_{p_1v}^{\oplus}$

$\Phi_2 = M_{p_1v}^{\oplus} - \Phi_1$

$\Phi_3 = \{M_{vp_1}[0]\}$

The computation of the a *priori* information APP is given by:

$$APP[x]_{x\in\Phi_1\cup\Phi_3\cup\{M_{vp_1}[0]\}} = \begin{cases} M_{vp_1}^{+}[x] + M_{p_1v}^{+}[x], & \text{if } x \in \Phi_1 \\ M_{p_1v}^{+}[x] + M_{vp_1}^{+}(n_s-1) + \mathrm{O}, & \text{if } x \in \Phi_2 \\ M_{vp_1}^{+}[0] + M_{p_1v}^{+}(n_m-1) + \mathrm{O} & \text{Otherwise} \end{cases} \tag{3.7}$$

and then the decision is made by:

$$\hat{c} = \underset{x\in\Phi_1\cup\Phi_2\cup\Phi_3}{\operatorname{argmin}} \{APP[x]\} \tag{3.8}$$

.

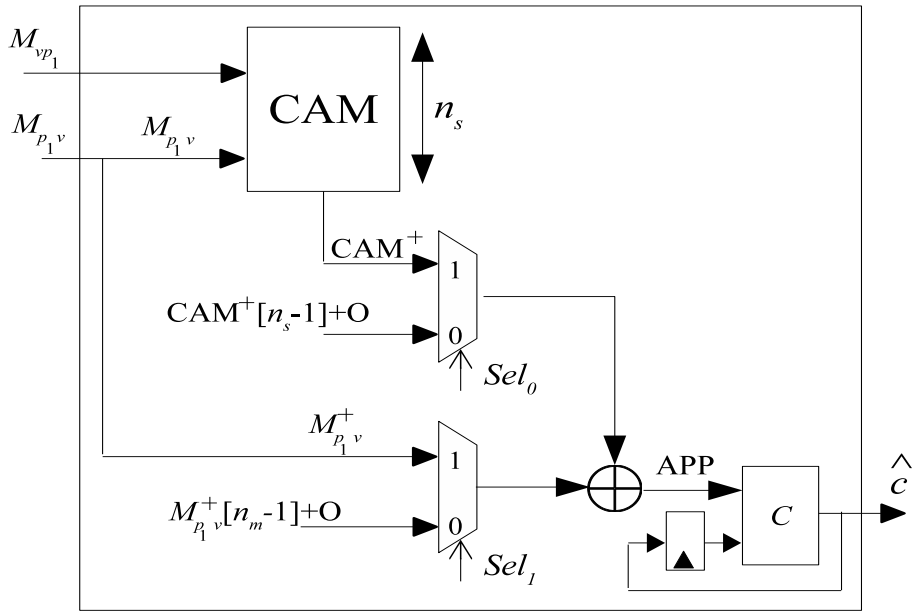

Figure 3.23: Architecture of the proposed VN decision-making mode

As shown in Fig. 3.23, there is no consideration of the redundant elimination since the structure of the incoming symbols helps dismissing the redundant GF values. Let us say that $M_{p_lv}^{\oplus}[i] = M_{p_lv}^{\oplus}[j]$ $(\{i,j\} \in \{0,1,\ldots,n_m-1\}^2, i<j)$ then $\mathrm{APP}^+[i] \leq \mathrm{APP}^+[j]$ since $M_{p_lv}^{+}[i] \leq M_{p_lv}^{+}[j]$, i.e, there is no need to check the equality of $M_{p_lv}^{\oplus}[i]$ and $M_{p_lv}^{\oplus}[j]$ as long as the minimum is being selected considering the two LLR values $\mathrm{APP}^+[i]$ and

$\text{APP}^+[j]$. Therefore, during phase 1, $Sel_1 = 1$, $Sel_0 = 1$ if $M^{\oplus}_{p_l v} \in$ CAM otherwise $Sel_0 = 0$. During phase 2, $Sel_1 = 0$, $Sel_0 = 1$ for only one clock cycle to consider only the most reliable symbol from the CAM $M_{vp_1}[0]$. The other symbols are dismissed since the same LLR value ($M^+_{p_l v}[n_m - 1] + \text{O}$) is added on a sorted list of symbols stored in the CAM. Thus, the latency is equal to $n_m + 3$ whatever the value of n_s (the depth of CAM).

3.2.2 Implementation results

The proposed VNP is better than the one proposed in [7] in terms of both complexity and frequency as shown in Table 3.8 (for $n_m = 12$).

Table 3.8: Complexity analysis of the VNP using Xilinx Virtex6, xc6vlx240t-2ff1156 device

VNP	Nb. of occupied slices	Nb. of slice LUTs	Nb. of slice registers	F (Mhz)
[7]	401	777	631	136
Proposed	212	429	452	211

The proposed VN architecture is superior [7] in which the number of occupied slices is equal to 212 slices and the frequency is equal to 211 Mhz for the proposed architecture while it is equal to 401 slices and 136 Mhz respectively for [7].

3.3 Conclusion

This chapter was dedicated to the proposed architectures of the CNP and VNP that can be included in most of the NB-LDPC decoding algorithms. First, the effect of the presorting on the FB-CN is shown where the number of used bubbles is reduced in each ECN. The implementation results showed that a complexity reduction up to 54 % is obtained for $d_c = 20$ along with similar performance compared to the existing FB-CN without presorting. We then presented a new CN algorithm called extended forward CN, where, all the input vectors are combined by $d_c - 1$ ECNs to generate syndromes that are decorrelated to obtain the appropriate output vectors. To significantly reduce the number of generated syndromes, the FB-CN and the EF-CN are hybridized to form hybrid CN. The proposed CN algorithms are studied with and without presorting. The post-synthesis results on 28 nm ASIC technology showed that the area efficiency is improved by a factor of 6.2 without any performance loss, or by a factor of 6.8 with a performance loss of 0.04 dB compared to FB-CN. Continuing on CN improvements, the skip processing CN approach is introduced. In this approach,

two tests are applied: 1) the parity check test that sums the most reliable GF symbol of each input vector U_i, $i = 0, \ldots, d_c - 1$; 2) the LLR test that compares each $U_i^+[1]$ to a pre-defined threshold. If these two conditions are satisfied, the CN processing is skipped, otherwise the CN update is performed. Skipping some of the CNs leads to power reduction and/or throughput increase. The statistical study showed that the percentage of skipping CNs can reach 50% for low predefined threshold and high Eb/N0 respecting the acceptance of the performance loss.
Finally, we proposed a simplified VN architecture. In case of the update mode, we proposed to merge the redundant suppression with the sorter block, while in the decision-making mode, we exploited the incoming sorted messages in terms of LLR value to remove some components and reduce the latency by considering only the first candidate from CAM instead of n_s candidates. The implementation results on virtex 6 FPGA device showed that the number of occupied slices is reduced by a factor of 1.9 and the frequency is increased by a factor of 1.55.

Chapter 4

Parallel pipelined architectures: LLR generator and extrema selection algorithms

As shown in the previous chapter, NB-LDPC architecture based on EMS algorithm requires sorting architectures. In order to increase the decoding throughput of a decoder, the solution is to shift from serial sorting structures to fully parallel sorting structures. This chapter is dedicated to this problem. The study was motivated by the objective of designing a very high speed NB-LDPC decoder but the obtained results can also be applied in many other applications. To cite few of them, sorting is required in data mining, databases [79], [80], [81], digital signal processing [82], [83], network processing, communication switching systems [84], [85], scientific computing [86], searching, scheduling [86], pattern recognition, robotics [87], [87], pattern recognition, robotics [88], image and video processing [89], [90], [91], and high-energy physics (HEP) [92].
First section is dedicated to the parallel generation of Non-Binary LLR from binary LLR (component used in chapter 5). Then, the general problem of finding two extrema among N_s values is studied and a new architecture named *First then Second Extrema Selection* is proposed. Finally, this architecture is extended to the problem of finding M_s extrema in a list of size N_s, a problem which is recurrent in the EMS based NB-LDPC algorithm, as well as many other algorithms.

4.1 Parallel pipelined LLR generator

Designing an efficient LLR generator with low complexity and high throughput rate is required when the decoder is operating at high throughput rate. This section proposes an efficient fully parallel pipelined architecture of the LLR computation for NB-LDPC codes designed over GF(64) for BPSK (equivalently QPSK) channel.

4.1.1 Definition of the LLRs

Let $X = (x_0, x_1, ..., x_{m-1})$ be a GF($q = 2^m$) symbol composed by $m = \log_2(q)$ binary symbols and let $Y = (y_0, y_1, ..., y_{m-1})$ be the log likelihood vector associated to the binary vector X as:

$$y_i = \log\left(\frac{\mathrm{P}(x_i \neq 0/a_i)}{\mathrm{P}(x_i = 1/a_i)}\right) \tag{4.1}$$

where a_i is the observation of the channel. For example, in case of a BPSK (or QPSK) modulation channel, x_i is associated to the symbol $\mathrm{B}(x_i) = (-1)^{x_i}$ and the received sample is the AWGN channel $a_i = \mathrm{B}(x_i) + w_i$ where w_i is a realization of a white Gaussian noise of variance σ^2. Developing 4.1, $y_i = \dfrac{2a_i}{\sigma^2}$. Thus, let $\bar{X} = (\bar{x}_0, \bar{x}_1, ..., \bar{x}_{m-1})$ be the Hard Decision (HD) on Y (for $i = 0, 1, \ldots, m-1$, if sign(y_i) > 0, then $\bar{x}_i = 0$, $\bar{x}_i = 1$ otherwise). For an AWGN channel with a noise of variance σ^2, $\ln(P(A|X))$ is given by:

$$\ln(P(A|X)) = \ln(\prod_{i=0}^{m-1} P(a_i|x_i))$$

$$= m\ln(\frac{1}{\sqrt{2\pi}\sigma}) - \sum_{i=0}^{m-1}\left(\frac{(a_i - B(x_i))^2}{2\sigma^2}\right) \tag{4.2}$$

where $A = \{a_0, \ldots, a_{m-1}\}$. Thus, equation (4.2) is maximized in case of $X = \bar{X}$. Therefore, considering the hypothesis that the GF(q) symbols are equiprobable, the reliability X^- of a symbol X is defined as:

$$X^- = \ln\left(\frac{P(A|X)}{P(A|\bar{X})}\right). \tag{4.3}$$

which, using 4.2 can be developed as:

$$X^- = \sum_{i=0}^{m-1}\left(-\frac{(a_i - B(x_i))^2}{2\sigma^2} + \frac{(a_i - B(\bar{x}_i))^2}{2\sigma^2}\right)$$

$$= -\frac{1}{2\sigma^2}\sum_{i=0}^{m-1}\left(2a_i(B(\bar{x}_i) - B(x_i))\right). \tag{4.4}$$

By definition of $\bar{X}$, X^- is a negative number. In order to deal with positive numbers, the quantity $X^+ = -X^-$ will be considered in what follows. Using equation (4.4), X^+ can be written as:

$$X^+ = \frac{2}{\sigma^2}\sum_{i=0}^{m-1}|a_i|\Delta(x_i, a_i) \tag{4.5}$$

$$= \sum_{i=0}^{m-1}|y_i|\Delta(x_i, y_i) \tag{4.6}$$

where X^+ is the LLR value of X and $\Delta(x_i, y_i) = 0$ if $(-1)^{x_i} = \text{sign}(y_i)$, $\Delta(x_i, y_i) = 1$ otherwise. Note that, by definition, $\bar{X}^+ = 0$.

4.1.2 Proposed architecture

This section describes the new proposed parallel architecture that generates the first n_m most reliable LLR values with their associated GF symbols in a parallel and pipelined fashion. The key idea is to perform the LLR calculation starting from a sorted list of positive binary LLRs rather than a random list. This LLR calculation is performed in four steps:

(1) Parallel sorting of the absolute value of the LLR y_i, $i = 0, \ldots, m-1$, where a permutation set $\Pi = \{\pi(0), \ldots, \pi(m-1)\}$ is generated.

(2) Design of a predefined set containing the potential candidates that contribute in the generation of the list of the most n_m reliable intrinsic symbols. The design of this set is based on the fact that the binary LLRs are sorted in increasing order thanks to the permutation Π.

(3) Parallel sorting of the predefined set to extract the n_m most reliable intrinsic symbols.

(4) Inverse permutation of the GF values in order to generate the original GF symbols that correspond to the binary LLRs before permutation.

Note that step 2 can be performed offline once the size n_m is defined. The sorted elements in step 3 constitutes only the list of desired symbols.

4.1.2.1 Parallel sorting of the channel observations

The first step is then to sort the received LLR y_i, $i = 0...m-1$, using the odd-even sorting algorithm [55] of size m. Fig. 4.1 shows the fully parallel pipelined sorter that receives the absolute value of the LLR $|y_i|$ and sorts them generating the couples $s_i = (s_i^+, \pi(i))$, $i = 0, \ldots, m-1$, where $\pi(i) \in \{0, \ldots, m-1\}$ and $s_i^+ = |y_{\pi(i)}|$ (i.e, $0 \leq s_0^+ \leq s_1^+ \leq \cdots \leq s_{m-1}^+$).
Therefore, the potential candidates are generated in next step based on s_i, $i = 0, \ldots, m-1$, where n_π potential candidates are generated in the next step among 64 possible combinations of the elements of the set $S = \{s_0, s_1, \ldots, s_5\}$. The value of n_π is determined according to the number of desired symbols n_m that implies the generation of all the possible potential candidates that could contribute to the list of the first n_m most reliable symbols. Then using a parallel sorter, the n_m intrinsic candidates are extracted among the n_π input candidates. Each combination means LLR addition of two elements or more from s_i, $i = 0, \ldots, 5$.

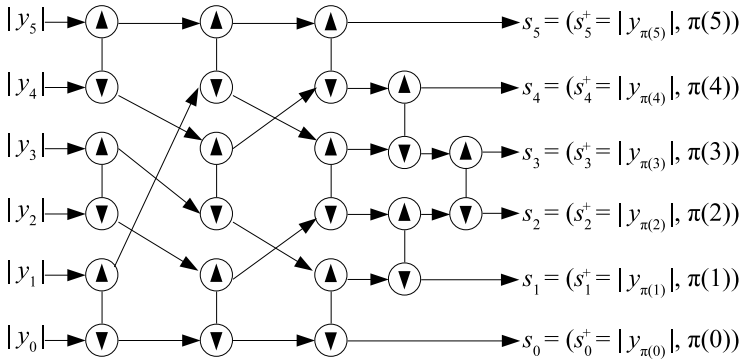

Figure 4.1: Sorter architecture of the observed bits.

4.1.2.2 Design of the pre-defined set of potential candidates

The second step is to generate the set that should contain all the potential candidates needed to select the first n_m most reliable symbols among $q = 2^m$. This set is denoted by $\Phi_\pi = \{J_0, J_1, \ldots, J_{n_\pi - 1}\}$, where n_π represents the cardinality of this set, and $J_i = (J_i^+, J_i^\oplus)$ denotes the i^{th} candidate with LLR value J_i^+ and GF symbol $J_i^\oplus$. Let us consider the design of the set Φ_π over GF($q = 64$). Note that the same approach can be applied to any GF order. For sake of simplicity, we start the generation from a hard decision $J_0 = (J_0^+, J_0^\oplus) = (0, (0, 0, 0, 0, 0, 0))$, where $J_0^\oplus$ is considered as the zero GF symbol. Once the generation of the list is completed, all the n_m generated symbols will be Xored with the real hard decided GF symbol $\bar{X}$. Thus, the n_π potential candidates are generated one by one as follows:

1) The first element being determined to be J_0, the second element has only one candidate symbol that is $(s_0^+, (1, 0, 0, 0, 0, 0))$, where the bit in position 0 is inverted according to equation (4.6). Note that the factor $\frac{2}{\sigma^2}$ is omitted, and it will be compensated in the quantization operation to be performed prior to the generation of the fixed point representation.
2) It is clear that the second minimum element has also only one candidate that is $(s_1^+, (0, 1, 0, 0, 0, 0))$.
3) The fourth element is $\min(s_0^+ + s_1^+, s_2^+)$. Thus, the two possible candidates are $(s_2^+, (0, 0, 1, 0, 0, 0))$ and $(s_0^+ + s_1^+, (1, 1, 0, 0, 0, 0))$.
4) The candidates of the fifth element depends on the fourth element being determined. This means, if $s_0^+ + s_1^+ = \min(s_0^+ + s_1^+, s_2^+)$ then the fifth element will be s_2^+, otherwise, $\min(s_0^+ + s_1^+, s_3^+)$. Therefore, the new possible candidate that could be selected as the fifth element is $(s_3^+, (0, 0, 0, 1, 0, 0))$.

This process continues till reaching the n_m^{th} element with all its possible candidates. Table 4.1 shows all the elements of the set Φ_π that constitute all the possible candidates needed to extract $n_m = 12$ symbols. In this case Φ_π is of cardinality $n_\pi = 16$.

Table 4.1: The elements of $\Phi_{n_\pi=16}$.

$J_0 = (J_0^+, J_0^\oplus) = (0, (0,0,0,0,0,0))$
$J_1 = (J_1^+, J_1^\oplus) = (s_0^+, (1,0,0,0,0,0))$
$J_2 = (J_2^+, J_2^\oplus) = (s_1^+, (0,1,0,0,0,0))$
$J_3 = (J_3^+, J_3^\oplus) = (s_2^+, (0,0,1,0,0,0))$
$J_4 = (J_4^+, J_4^\oplus) = (s_3^+, (0,0,0,1,0,0))$
$J_5 = (J_5^+, J_5^\oplus) = (s_4^+, (0,0,0,0,1,0))$
$J_6 = (J_6^+, J_6^\oplus) = (s_5^+, (0,0,0,0,0,1))$
$J_7 = (J_7^+, J_7^\oplus) = (s_0^+ + s_1^+, (1,1,0,0,0,0))$
$J_8 = (J_8^+, J_8^\oplus) = (s_0^+ + s_2^+, (1,0,1,0,0,0))$
$J_9 = (J_9^+, J_9^\oplus) = (s_1^+ + s_2^+, (0,1,1,0,0,0))$
$J_{10} = (J_{10}^+, J_{10}^\oplus) = (s_0^+ + s_1^+ + s_2^+, (1,1,1,0,0,0))$
$J_{11} = (J_{11}^+, J_{11}^\oplus) = (s_0^+ + s_3^+, (1,0,0,1,0,0))$
$J_{12} = (J_{12}^+, J_{12}^\oplus) = (s_1^+ + s_3^+, (0,1,0,1,0,0))$
$J_{13} = (J_{13}^+, J_{13}^\oplus) = (s_2^+ + s_3^+, (0,0,1,1,0,0))$
$J_{14} = (J_{14}^+, J_{14}^\oplus) = (s_0^+ + s_1^+ + s_3+, (1,1,0,1,0,0))$
$J_{15} = (J_{15}^+, J_{15}^\oplus) = (s_0^+ + s_4^+, (1,0,0,0,1,0))$

4.1.2.3 Sorting of the potential candidates

The last step is to sort the pre-defined potential candidates J_k, $k = 0, \ldots, 15$, to generate the list of n_m sorted LLR J_i^s, $i = 0, \ldots, n_m - 1$, where we are considering $n_m = 12$. The first three outputs are $J_0^s = J_0$, $J_1^s = J_1$ and $J_2^s = J_2$. For the remaining 9 outputs, we propose the sorter architecture shown in Fig. 4.2. The sorter receives the 13 elements J_k, $k = 3, \ldots, 15$, and extract the first 9 symbols having the smallest LLR values. Note that there are some sorted couples in the set of 13 input candidates that are exploited to reduce the number of comparators and multiplexers. Therefore, the 13 elements are split up into 4 sets based on the LLR value as: $\{J_3^+ \leq J_4^+ \leq J_5^+ \leq J_6^+\}$, $\{J_7^+ \leq J_8^+ \leq J_9^+ \leq J_{10}^+\}$, $\{J_{11}^+ \leq J_{12}^+ \leq J_{13}^+\}$, $\{J_{14}^+\}$ and $\{J_{15}^+\}$.

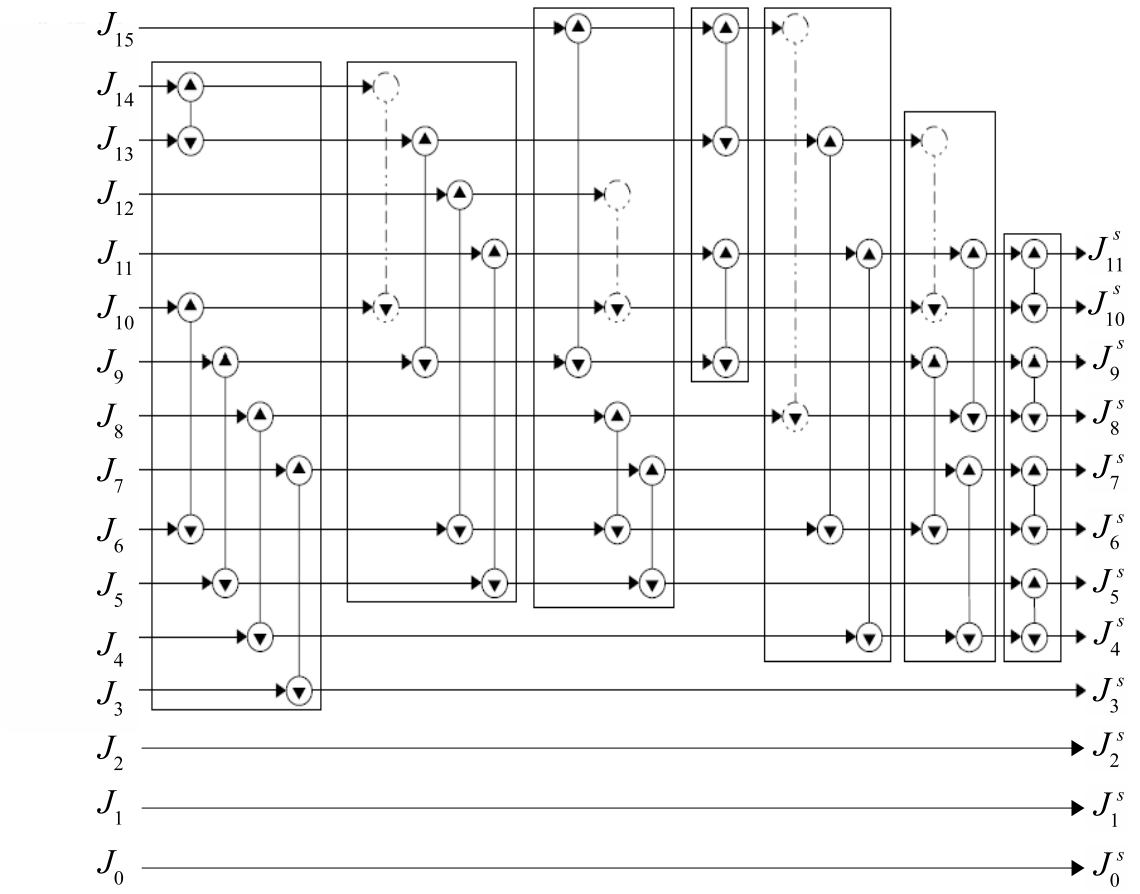

Figure 4.2: Sorter Architecture generating the most reliable n_m intrinsic LLRs, $n_m = 12$.

The sorter architecture is based on odd-even [55] algorithm. It is composed of 7 sets of CSs operating concurrently in each set. It contains a total of 22 CSs and 4 Cs with a critical path of $T = 7 \times T_{CS}$, where T_{CS} is the execution time of one CS.

4.1.2.4 Inverse permutation of the GF values of J^s

Each GF value $J_k^{s\oplus}$, $k = 0, \ldots, 11$, is generated based on the sorted binary LLRs s_i, and should be permuted to constitute the real GF symbols that corresponds to the unsorted LLRs y_i. This inverse permutation is performed according to Π^{-1} (the inverse of Π) and the permuted GF symbols are Xored with $\bar{X}$, as previously explained to obtain the intrinsic candidates $I_k^{\oplus}$ as follows: $I_k^{\oplus}[i] = J_k^{s\oplus}[\Pi^{-1}(i)] \bigoplus \bar{x}_i$, $I_k^{+} = J_k^{s+}$, $k = 0, \ldots, 15$, $i = 0, 1, \ldots, 5$.

4.1.3 Complexity analysis

In this section, the implementation results on virtex 6, xc6vlx240t -2 ff1156 FPGA device of $n_m = \{4, 6, 8, 10, 12\}$ are shown. Note that from an architecture designed

for $n_m = 12$, it is straightforward to design an architecture for $n_m < 12$ (one needs just to prune unused hardware).

Table 4.2: Synthesis results on virtex 6, xc6vlx240t -2 ff1156 FPGA device.

Algorithm	n_m	O. S	F (MHz)	Periodicity (Clock Cycles)		Efficiency (Mhz/O. S)			Throughput (Msymbols/s)		
				Proposed	[37]	Proposed	[37]	Factor gain	Proposed	[37]	Factor gain
Proposed	12	516	402	1	8	0.779	0.191	4	4824	315	15.3
	10	480	408		6	0.85	0.255	3.33	4080	350	11.65
	8	451	406		4	0.9	0.383	2.34	3248	420	7.7
	6	295	413		2	1.4	0.766	1.8	2478	630	3.9
	4	167	556		1	3.3	1.53	2.15	2224	840	2.65
[37]	All cases	137	210								

Table 4.2 presents the hardware cost of the proposed architecture compared to the systolic one. The complexity of each design is defined as the number of Occupied Slices (O. S).. The complexity of the systolic architecture is constant since the 6 stages (in case of GF(64)) are performed for any value of n_m. However, in terms of hardware efficiency, there is a factor gain ranging from 2.15 up to 4, while in terms of throughput, the gain factor varies from 2.65 up to 15.4.

$$\text{Efficiency (Mhz/O. S)} = \frac{\text{F}}{\text{O. S} \times \text{Periodicity}}, \tag{4.7}$$

$$\text{Throughput (Msymbols/s)} = \frac{\text{F} \times n_m}{\text{Periodicity}}, \tag{4.8}$$

The Efficiency and the throughput are computed based on equations 4.7 and 4.8 respectively, where the Periodicity is the latency between two set of inputs.

4.1.4 Example for $n_m = 4$

In this section we show the architecture of the parallel pipelined LLR generator for $n_m = 4$. In this specific case, only the set $\{s_0, s_1, s_2\}$ is required. Fig. 4.3 shows the architecture of the sorter observed bits. Comparing this architecture with Fig. 4.1, we notice that three comparator swaps are replaced by three comparators.
The possible candidates are generated as:

$J_0 = (J_0^+, J_0^\oplus) = (0, \bar{Z})$,

$J_1 = (J_1^+, J_1^\oplus)$ where $J_1^+ = s_0^+$ and $J_1^\oplus$ is equal to $\bar{Z}$ except that $\bar{z}_{\pi(0)}$ is replaced by its opposite $z_{\pi(0)}$,

$J_2 = (J_2^+, J_2^\oplus)$ where $J_2^+ = s_1^+$ and $J_2^\oplus$ is equal to $\bar{Z}$ except that $\bar{z}_{\pi(1)}$ is replaced by its opposite $z_{\pi(1)}$,

$J_3 = (J_3^+, J_3^\oplus)$ where $J_3^+ = s_2^+$ and $J_3^\oplus$ is equal to $\bar{Z}$ except that $\bar{z}_{\pi(2)}$ is replaced by its opposite $z_{\pi(2)}$,

$J_4 = (J_4^+, J_4^\oplus)$ where $J_4^+ = s_0^+ + s_1^+$ and $J_4^\oplus$ is equal to $\bar{Z}$ except that $\bar{z}_{\pi(0)}$ and $\bar{z}_{\pi(1)}$

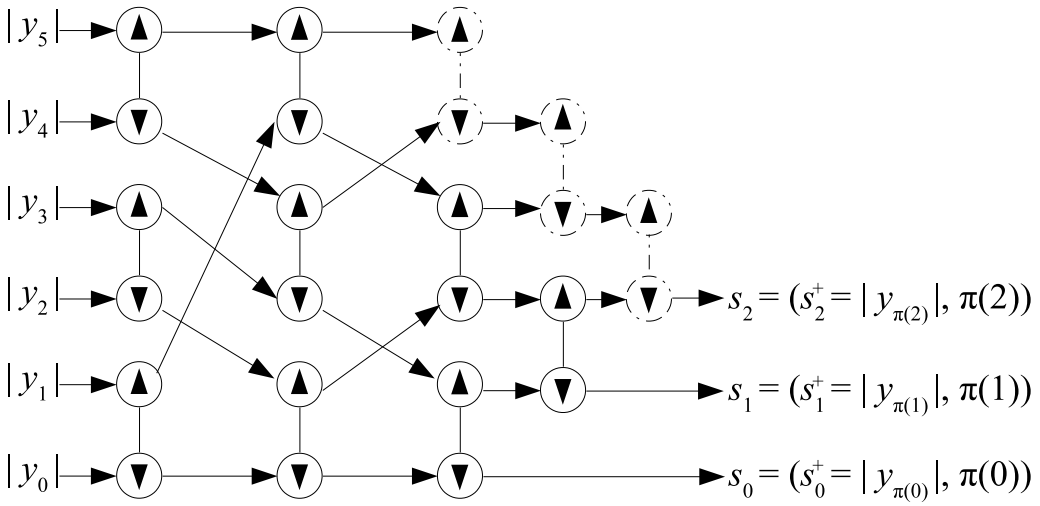

Figure 4.3: Sorter architecture of the observed bits.

are replaced by their opposite $z_{\pi(0)}$ and $z_{\pi(1)}$ respectively.
Then, the outputs are generated as: $I_0 = J_0$, $I_1 = J_1$, $I_2 = J_2$ and $I_3 = \min(J_3, J_4)$ where the min function returns the couple that is having the lowest LLR value.

Fig. 4.4 shows the architecture that generates the possible candidates $\{J_0, J_1, J_2, J_3, J_4\}$ and the intrinsic outputs $\{I_0, I_1, I_2, I_3\}$. The Control Generator (CG) generates the control signals c_{ij}, $i = 0, 1, 2$ and $j = 0, \ldots, 5$, as:

$$c_{ij} = \begin{cases} 1 & \text{If } \pi(i) = j \\ 0 & \text{Otherwise} \end{cases} \tag{4.9}$$

this process is to recognize the positions of the first three minimum observed bits. For instance, $c_{04} = 1$, $c_{12} = 1$ and $c_{25} = 1$ means that $|y_4|$ is the first minimum, $|y_2|$ is the second minimum and $|y_5|$ is the third minimum respectively.

Then, the Possible Candidates Generator (PCG) block is to generate the set $\{J_1, J_2, J_3, J_4\}$ as:

$$J_i^{\oplus}[j] = \begin{cases} \bar{x}_j & \text{If } c_{kj} = 0 \\ x_j & \text{Otherwise} \end{cases} \tag{4.10}$$

and

$$J_i^{+}[j] = s_k^{+} \tag{4.11}$$

where $i = 1, 2, 3$, $j = 0, \ldots, 5$ and $k = i - 1$. While J_4 is computed as: $J_4^+ = s_0^+ + s_1^+$, then the control signals $c_i = c_{0i}$ or c_{1i}, $i = 0, \ldots, 5$, to indicate which two bits have the first and second minimum LLR values and hence the GF value $J_4^{\oplus}$ is generated by the set of 2-to-1 MUXs.

Finally, the Outputs Generator (OG) block generates the four intrinsic candidates $\{I_0, I_1, I_2, I_3\}$ using one comparator.

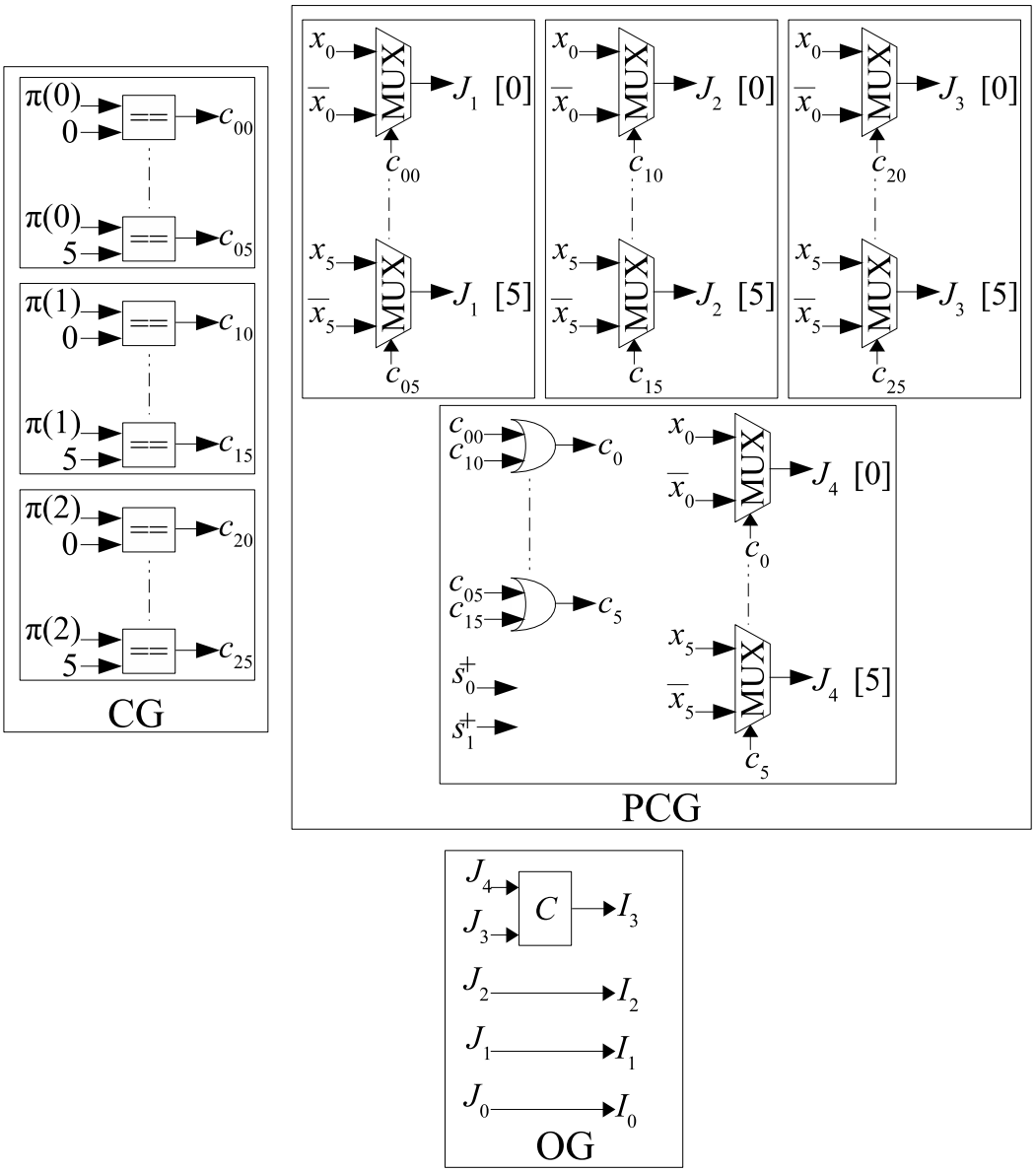

Figure 4.4: Architecture of the intrinsic outputs.

This architecture is used in the proposed parallel pipelined decoder shown in chapter 5.

4.2 Parallel pipelined architecture for extrema selection algorithm

In this section, we consider the the design of a parallel architecture of a sorter extracting the first two extrema among N_s input values. The proposed algorithm is called *First-Then Second Extrema Selection* (FTSES). A detailed analysis of the elementary sorting units, computational complexity, structural modularity and pipelining techniques are addressed. Finally, one of the possible extension cases of the proposed two extrema sorter is presented to show that it can be generalized to detect $M_s > 2$ minimum/maximum values among N_s values and still outperforms its odd-even counterpart sorter algorithm in some cases.

4.2.1 Problem Statement and Proposed Algorithm

Given a set of N_s numbers, $S = \{x_0, x_1, \ldots, x_{N_s-1}\}$, the first minimum is defined as $m_1 = \min\{S\}$, and $m_2 = \min\{S \backslash x_p = m_1\}$, where the $\backslash$ operator indicates exclusion of only one element representing the first minimum value, with $p \in \{0, 1, \ldots, N_s - 1\}$. This section first presents the Elementary Sorting Unit (ESU) receiving four inputs (4-SU) along with its basic blocks, and then the architecture of the global N_s-input Sorting Unit (N_s-SU) for $N_s = 2^k$.

4.2.1.1 Algorithm

The basic blocks shown in Fig. 2.12 are organized in a modular structure to obtain the ESU architecture shown in Fig. 4.5.

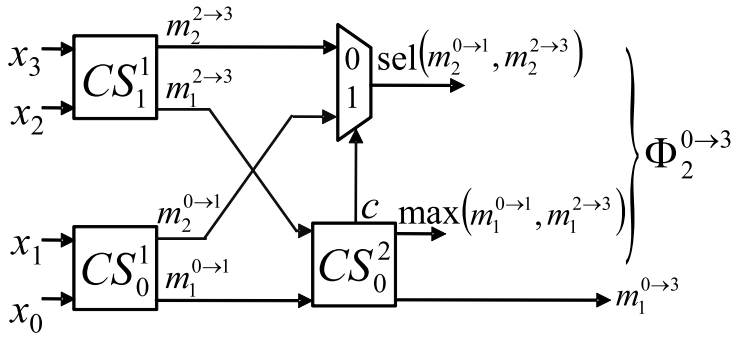

Figure 4.5: ESU (4-SU) Architecture.

In the following, each CS will be appended by an upper index indicating the stage it belongs to, and a lower index indicating its position within the stage. Therefore, the notations of the first and second minimums will be updated accordingly by appending an upper index indicating the range of indices of the input elements from which they are extracted. Fig. 4.5 shows the internal structure of a 4-SU composed of two stages.

The first stage is composed of two CSs: $CS_j^1, j = 0, 1$, operating concurrently to generate their sorted values $(m_1^{2j\to 2j+1}, m_2^{2j\to 2j+1})_{j=0,1}$, the first and second minimums of the j^{th} CS receiving the $2j^{th}$ and $(2j+1)^{th}$ inputs.

The second stage is composed of one CS denoted by CS_0^2 and one MUX2-1. The CS_0^2 receives the first minimums from the previous stage $(m_1^{2j\to 2j+1})_{j=0,1}$ and sorts them producing the first minimum, $(m_1^{4j\to 4j+3})_{j=0}$, among the four inputs. The second minimum, $(m_2^{4j\to 4j+3})_{j=0}$, will be the minimum value of the set $\Phi_2^{0\to 3} = \{\max(m_1^{0\to 1}, m_1^{2\to 3}), \text{sel}(m_2^{0\to 1}, m_2^{2\to 3})\}$, where the function $\text{sel}(m_2^{0\to 1}, m_2^{2\to 3})$ denotes a selection function that selects $m_2^{0\to 1}$ if $(m_1^{0\to 1} < m_1^{2\to 3})$ i.e., $c = 1$, $m_2^{2\to 3}$ otherwise $(c = 0)$. The selected value $(m_2^{2j\to 2j+1})$ comes from the CS_j^1 that contains the first local minimum being determined at the second stage, thus the second minimum will be either this selected value or $\max(m_1^{0\to 1}, m_1^{2\to 3})$. This ensures that the second local minimum is always contained in the set $\Phi_2^{0\to 3}$. The ESU function is described in details in Algorithm 3.

In general for any stage i, the first two minimums generated locally by a given CS_j^i can be denoted by $(m_1^{j2^i\to(j+1)2^i-1}, m_2^{j2^i\to(j+1)2^i-1})$, where j denotes the index of the CS within the i^{th} stage. The ranges of i and j, i.e., the total number of stages and the number of CSs per stage will be discussed later in section III.

Read the four inputs: $x_p, p = 0, 1, 2, 3$
Run the first stage, $i = 1$:
for $j = 0$ **to** 1 **do**
  Execute CS_j^1:
  if $x_{2j} < x_{2j+1}$ **then**
    $c = 1, m_1^{2j\to 2j+1} = x_{2j}$ and $m_2^{2j\to 2j+1} = x_{2j+1}$
  else
    $c = 0, m_1^{2j\to 2j+1} = x_{2j+1}$ and $m_2^{2j\to 2j+1} = x_{2j}$
  end if
end for
Run the second stage, $i = 2$:
if $m_1^{0\to 1} < m_1^{2\to 3}$ **then**
  $c = 1$; $m_1^{0\to 3} = m_1^{0\to 1}$; $\max(m_1^{0\to 1}, m_1^{2\to 3}) = m_1^{2\to 3}$
  $\text{sel}(m_2^{0\to 1}, m_2^{2\to 3}) = m_2^{0\to 1}$; $\Phi_2^{0\to 3} = \{m_1^{2\to 3}, m_2^{0\to 1}\}$
else
  $c = 0$; $m_1^{0\to 3} = m_1^{2\to 3}$; $\max(m_1^{0\to 1}, m_1^{2\to 3}) = m_1^{0\to 1}$
  $\text{sel}(m_2^{0\to 1}, m_2^{2\to 3}) = m_2^{2\to 3}$; $\Phi_2^{0\to 3} = \{m_1^{0\to 1}, m_2^{2\to 3}\}$
end if

Algorithm 3: ESU Function

4.2.1.2 Architecture of N_s-SU for $N_s = 8$

In a modular fashion, an 8-SU architecture is obtained by implementing two 4-SUs in parallel connected to a third stage composed of one comparator-swap, CS_0^3, and two MUX2-1s as shown in Fig. 4.6. The CS_0^3 receives the two first minimums, $m_1^{0\to3}$ and $m_1^{4\to7}$, and sorts them, hence, their minimum value will constitute the first minimum among the 8 input values entering the 8-SU. The maximum value, $\max(m_1^{0\to3}, m_1^{4\to7})$, will be inserted into the set of second minimum candidates, $\Phi_2^{0\to7}$, whose cardinality is equal to $\log_2(8)$. The key idea in our proposed algorithm is to postpone the determination of the second minimum until the last stage of the sorter in order to reduce the number of comparators to be used in each stage. Thus, instead of determining the local second minimum $m_2^{0\to j-1}$ among j elements, a local set $\Phi_2^{0\to j-1}$ is formed progressively containing $\log_2(j)$ candidate elements at the $\log_2(j)^{th}$ stage, hence delaying the selection of the second minimum until the last stage of the sorting unit. This reduces the complexity of an ESU from (2 CSs, 3 Cs, 1 MUX2-1), needed in the TS approach [71], down to (3 CSs, 1 MUX2-1). However, more outputs are generated at each stage, since a set of second minimum candidates, $\Phi_2^{0\to j-1}$, is propagated to the next stage instead of only one element representing the local second minimum. The set $\Phi_2^{0\to7}$ at the 3^{rd} stage is obtained according to the following equation:

$$\Phi_2^{0\to7} = \begin{cases} \Phi_2^{0\to3} \cup \{m_1^{4\to7}\} & \text{if } c = 1 \\ \Phi_2^{4\to7} \cup \{m_1^{0\to3}\} & \text{if } c = 0. \end{cases} \tag{4.12}$$

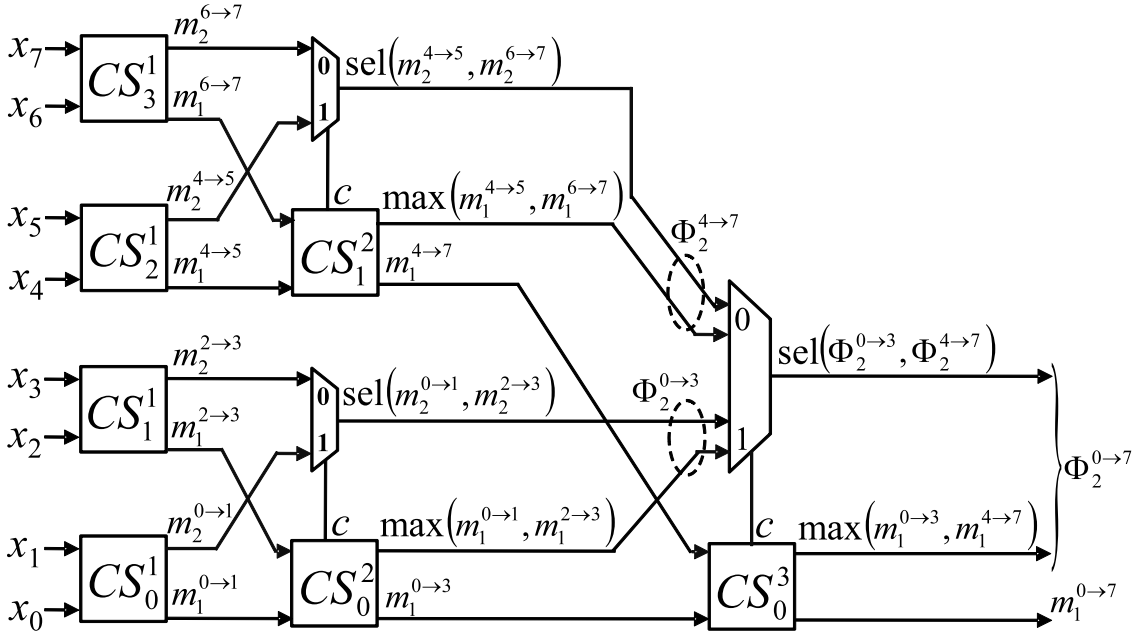

Figure 4.6: 8-SU Architecture

At the 3^{rd} stage, the second minimum $m_2^{0\to7}$ can be simply determined by selecting the minimum of the three elements constituting the set $\Phi_2^{0\to7}$, which can be performed using comparators in a serial or a parallel structure. The implementation of the unit that generates the second minimum will be discussed in more details in next section.

4.2.2 Proposed N_s-SU Architecture: Complexity and Performance Analysis

This section describes the architecture of the global N_s-SU, $N_s = 2^k$ for any integer k, and discusses the complexity in terms of hardware resources, CSs, Cs and MUX2-1s, as well as the performance in terms of execution time.

4.2.2.1 Global N_s-SU Architecture

A N_s-SU architecture is thus composed of two main units: FMU and SMU. The FMU can be designed in a hierarchical fashion using, at the last stage of the binary tree structure, two $N_s/2$-SUs connecting their first minimums $m_1^{0\to\frac{N_s}{2}-1}$ and $m_1^{\frac{N_s}{2}\to N_s-1}$ to the $CS_0^{\log_2 N_s}$ that sorts them generating the global first minimum $m_1^{0\to N_s-1}$ as shown in Fig. 4.7. The sets $\Phi_2^{0\to\frac{N_s}{2}-1}$ and $\Phi_2^{\frac{N_s}{2}\to N_s-1}$, each being of cardinality $\log_2(\frac{N_s}{2})$, are connected to a multiplexer controlled by the signal c generated by the $CS_0^{\log_2 N_s}$ component to select one of the sets accordingly. The upper output of the $CS_0^{\log_2 N_s}$ along with the selected Φ_2 forms the set $\Phi_2^{0\to N_s-1}$ fed to the SMU to select the minimum value contained in this set that will constitute the second global minimum $m_2^{0\to N_s-1}$.

Each of the $N_s/2$-SU is implemented hierarchically starting from the reference level of hierarchy ($L = 1$) contained in 4-SU, and successively repeated $\log_2(\frac{N_s}{2}) - 1$ times. Thus, the global architecture is obtained using $\log_2(N_s)-1$ hierarchy levels containing $\log_2 N_s$ stages of CS that form the FMU, plus the SMU.

4.2.2.2 Complexity Analysis

The first stage of the FMU ($i = 1$) contains $N_s/2 = 2^{k-1}$ CSs and no multiplexers other than those contained in the CSs, where each CS contains two MUX2-1s. The second stage ($i = 2$) contains 2^{k-2} CSs and 2^{k-2} MUX2-1s. More generally, the i^{th} stage of FMU, for $i = 1, 2, \ldots k$ contains 2^{k-i} CSs and $(i-1)2^{k-i}$ MUX2-1s, or equivalently 2^{k-i} Cs and $(i+1)2^{k-i}$ MUX2-1s. In total, the number of Cs contained in the FMU is given by $\sum_{i=1}^{k} 2^{k-i}$=N_s-1, and the number of MUX2-1s is given by $\sum_{i=1}^{k}(i+1)2^{k-i}$. It can be shown that the total number of MUX2-1s reduces to $\sum_{i=1}^{k} i2^{k-i} + \sum_{i=1}^{k} 2^{k-i} = N_s(\frac{2(N_s-1)-k}{N_s}) + N_s(\frac{N_s-1}{N_s}) = 3N_s - k - 3$.

The SMU can be implemented using k-1 Cs implemented in a Tree Structure (TS) composed of $\lceil \log_2 k \rceil$ stages as shown in Fig. 4.8.a, where $\lceil x \rceil$ denotes the nearest integer greater than or equal to x. This TS architecture of the SMU will be denoted

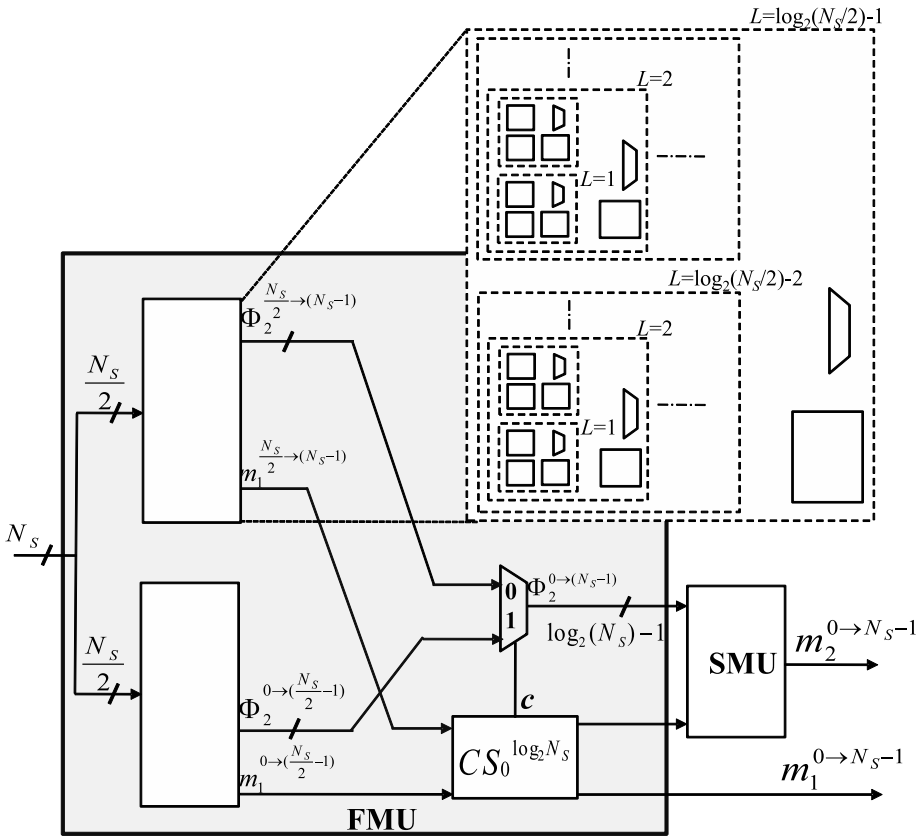

Figure 4.7: N_s-SU Architecture, $N_s = 2^k$

by SMU-TS. Note that each comparator implemented in the SMU-TS refers to the comparator C shown in Fig. 1.a, which contains only one MUX2-1. Alternatively, the SMU can be implemented in only one stage using $k(k-1)/2$ simple COs implemented in parallel and comparing concurrently the elements of $\Phi_2^{0\to N_s-1} = \{e_0, \ldots, e_{k-1}\}$ by pairs. For instance, each element e_p is compared to all elements e_q, where $p, q \in \{0, 1, \ldots, k-1\}$, $p \neq q$, and $e_p, e_q \in \Phi_2^{0\to N_s-1}$. Each CO receives a pair of two elements and generates a signal c_{pq}, with $c_{pq} = 1$ if $e_p < e_q$, otherwise $c_{pq} = 0$.
The control signals c_{pq} are fed to a control logic encoder to generate the selection signals $s_j = (\wedge_{i=0}^{j-1} \overline{c}_{ij}) \wedge (\wedge_{i=j+1}^{k-1} c_{ji})$, $\wedge$ is the logic-AND operation and $i, j \in \{0, 1, \ldots, k-1\}$. The second minimum is thus selected as: $m_2^{0\to N_s-1} = \vee_{i=0}^{k-1}(s_i \wedge e_i)$; $\vee$ is the logic-OR operation.

This Parallel Structure (PS), denoted by SMU-PS, is shown in Fig. 4.8.b. The overall complexity of the proposed architecture using the different SMU structures is shown in Table 4.3.

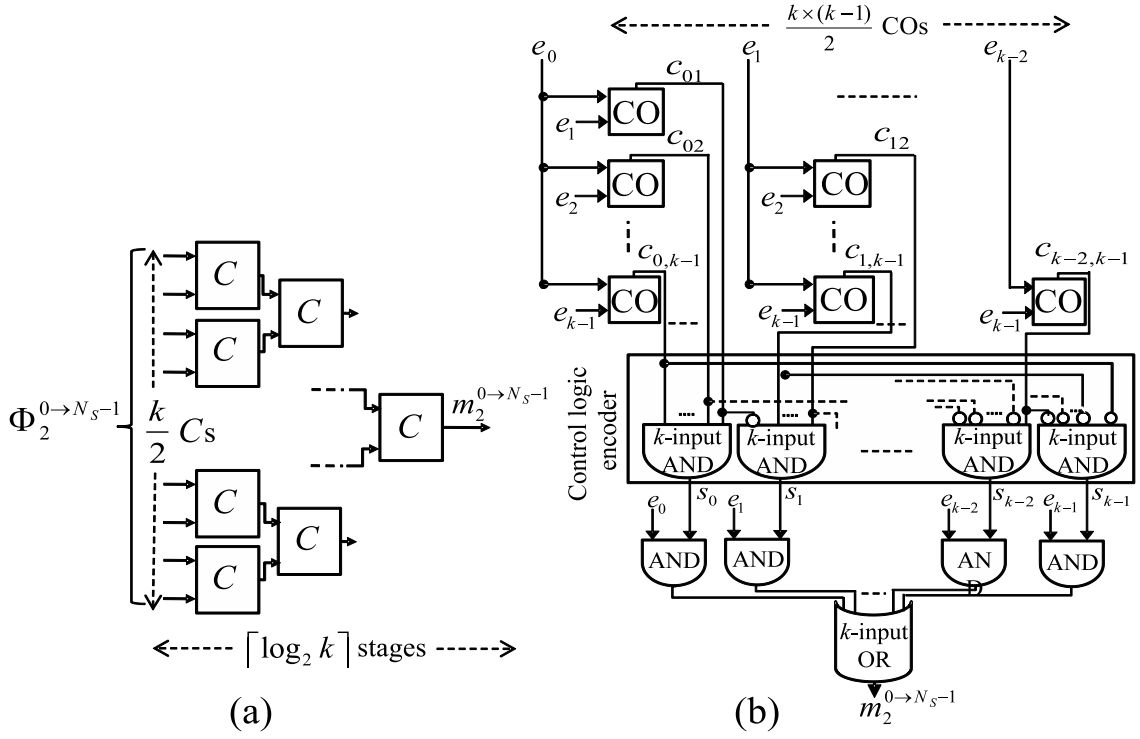

Figure 4.8: SMU Architectures for N_s-SU, $N_s = 2^k$: (a) SMU-TS (b) SMU-PS

4.2.2.3 Timing Analysis

The critical path T of the proposed sorting unit architecture can be easily computed as $T = T_{FMU} + T_{SMU}$, where T_{FMU} and T_{SMU} denote the propagation time of the FMU and SMU respectively. Each of the $k = \lceil \log_2(N_s) \rceil$ stages of FMU has a critical path equal to $T_C + T_{MUX2-1}$, where T_C denotes the time needed to perform one comparison, thus $T_{FMU} = k(T_C + T_{MUX2-1})$. The time T_{SMU} depends on the selected architecture as shown in Fig. 4.8. For the SMU-TS, $T_{SMU} = T_{SMU-TS} = \lceil \log_2 k \rceil (T_C + T_{MUX2-1})$. For the parallel structure, $T_{SMU} = T_{SMU-PS} = T_C + \lceil \log_2 k \rceil T_{AND} + T_{AND} + \lceil \log_2 k \rceil T_{OR}$, where T_{AND} and T_{OR} denote the propagation time of the logic AND and OR gates respectively. By approximating $T_{MUX2-1} \approx T_{AND} + T_{OR}$, and $T_{MUXk-1} \approx \lceil \log_2 k \rceil T_{MUX2-1}$, T_{SMU-PS} can be approximated as $T_{SMU-PS} \approx T_C + \lceil \log_2 k \rceil T_{MUX2-1}$.

4.2.2.4 Discussion

Theoretically, and in comparison with the TS-based architecture [71], our proposed architecture offers an important saving in the number of comparators (Table 4.3), where a reduction factor close to 2 is obtained. The number of multiplexers is remained the same. In terms of critical path, the proposed architecture imposes an additional delay due to the SMU. However, using the SMU-PS, the critical path of

the proposed architecture is reduced by a factor $\log_2(k)T_C$ and becomes smaller than the critical path of the TS-based architecture as shown in the rightmost column of Table 4.3. We omitted the theoretical complexity comparison with [72] since they do not have similar structures. However, a technology-based assessment has been performed, where we have considered the ASIC implementation of our algorithm and the algorithms proposed in [71] and [72]. The implementation results will be discussed in next section.

Table 4.3: Computational Complexity Comparison

Complexity/ Critical Path	TS [71]	FTSES Architecture	
		SMU-TS	SMU-PS
# of Comparators (Cs)	$2N_s$-3	$N_s + k - 2$	N_s+$k(k-1)/2 - 1$
# of Multiplexers (MUX2-1)	$3N_s$-4	$3N_s - k - 3 + k - 1 = 3N_s - 4$	
Critical Path	kT_C+ $(2k-1)T_{MUX2-1}$	$\lceil k + \log_2 k \rceil \times$ $(T_C + T_{MUX2-1})$	$(k+1)T_C$+ $\lceil k + \log_2 k \rceil T_{MUX2-1}$

4.2.3 Hardware Implementation

To assist with analyzing implementations, we developed a generic VHDL Register Transfer Level (RTL) code for our proposed algorithm and the algorithms proposed in [71] and [72]. The VHDL models are conveniently parametrized to provide the flexibility of designing and analyzing RTL with a variable number of inputs N_s, as well as different levels of pipeline registers. The designer can easily change (add or remove) each level of pipeline registers to get a design with a different latency, resource requirements, and frequency. This feature helps achieve variable levels of throughput and latency that may be desired.

4.2.3.1 Implementation Results

The different designs are synthesized using a TSMC 28-nm standard cell technology. For all our synthesis results, the parameterizable data width has been set to 6 bits. We reported the results for two different structures: non-pipeline, and 1 stage of pipeline registers placed at the middle stage of each architecture. Table 4.4 shows the post-synthesis results for non-pipelined N_s-SU, where different values of N_s have been considered. As shown, the proposed architecture with SMU-TS consumes less power and permits to get an area reduction ranging from 14 % (resp. 32 %), when N_s=16, up to 31 % (resp. 50 %), when N_s=512, as compared to the architecture proposed in [71] (resp. [72]). However, the proposed architecture suffers from longer critical path imposed by the SMU. When implemented using the parallel structure of SMU (SMU-PS), the proposed architecture becomes less complex with the smallest critical

path for all values of N_s. A slight increase in the power consumption is observed.

Table 4.4: Post-synthesis results of N_s-SU on TSMC 28 nm, Non-Pipelined Architecture (A: Area, C: Critical Path, P: Power)

N_s	[72]			TS [71]			FSTSE, SMU-TS			FSTSE, SMU-PS		
	A (μm^2)	C (ns)	P (μW)	A (μm^2)	C (ns)	P (μW)	A (μm^2)	C (ns)	P (μW)	A (μm^2)	C (ns)	P (μW)
16	5927	3.6	729	4689	3.5	830	4020	4	447	3276	2.8	836
32	13671	4.5	1575	9963	5	1336	7888	5.5	1123	6771	4	867
64	26824	6	2702	20157	6	2297	15265	6.5	1890	13957	5	2173
128	55775	7.5	5742	41207	7	4302	28786	9	3097	28843	6.2	3826
256	114748	8	8807	84275	8.5	7991	57208	10.5	5741	56947	7.6	6843
512	230596	9.4	15892	166292	10	14869	114707	12	10899	113196	8.8	12468

Table 4.5 shows the results for pipelined architectures with 1 pipeline stage. For $N_s \geq 32$, the proposed architecture with SMU-PS has both the smallest critical path and the smallest occupied area, with an area reduction reaching the 50 % as compared to the architecture in [72]. In terms of power consumption, the proposed architecture has a higher consumption for N_s=16, 32 and becomes less power consuming for large values of N_s. If we consider fully pipelined architectures, i.e., a pipeline register is inserted between each two adjacent stages, the critical path of the TS-based architecture [71] would be $T_C + 2T_{MUX2-1}$, while that of the FMU would be $T_C + T_{MUX2-1}$. In this case the SMU should be also pipelined and its critical path will be $\max\{T_C, \lceil \log_2 k \rceil T_{MUX2-1}\}$.

Table 4.5: Post-synthesis results of N_s-SU on TSMC 28 nm, Pipelined Architecture (A: Area, C: Critical Path, P: Power)

N_s	[72]			TS [71]			FSTSE, SMU-PS			Efficiency Ratios vs [72], [71]			
	A (μm^2)	C (ns)	P (μW)	A (μm^2)	C (ns)	P (μW)	A (μm^2)	C (ns)	P (μW)	[72]		[71]	
										AER	PER	AER	PER
16	5525	2.1	1103	4164	2.5	1536	3285	2.7	1144	1.3	0.75	1.17	1.24
32	12345	3.7	1488	9186	3.2	2074	6732	3.2	1950	2.12	0.88	1.36	1.06
64	23783	4	3997	18023	3.7	3665	14105	3.7	3060	1.8	1.41	1.27	1.19
128	54917	5.5	5853	37197	5.2	5082	27533	4.5	5296	2.4	1.35	1.56	1.1
256	108560	6	10901	67647	6.2	8453	55391	6	8214	1.95	1.32	1.26	1.06
512	216230	6.8	22145	153615	7	16821	108413	6.7	14797	2.02	1.51	1.48	1.18

4.2.3.2 Area and power efficiency comparison

In order to compare the efficiency of the different architectures, we evaluated two metrics: Area Efficiency (AE) and Power Efficiency (PE). The AE, indicating the number of symbols sorted per time unit (ns) per area unit (μm^2), is defined as: AE = $1/(N_c \times C[ns] \times A[\mu m^2])$, where N_c represents the number of cycles per sorted symbol. Since there are two symbols sorted in each cycle, N_c is equal to 0.5. Similarly, the PE indicates the number of symbols sorted per seconds per μW, defined as: PE = $1/(N_c \times C[ns] \times P[\mu W])$.

To better illustrate the efficiency comparison, we have evaluated the AE Ratio (AER) (resp. PE Ratio (PER)) defined as the ratio of the AE (resp. PE) of the FTSES architecture to the AE (resp. PE) of each of the architectures [72] and [71].

In Table 4.5, the four rightmost columns show the numerical values of these efficiency ratios for different input lengths. In terms of AE, the proposed architecture outperforms both architectures by a factor ranging from 1.17 up to 2. It is also more power efficient for all values of N_s as compared to [71], and for $N_s \geq 64$ when compared to [72]. Note that these AER and PER results would be more in favor of the proposed architecture if we considered the implementation results of the non-pipelined architectures.

4.2.4 Extension of the proposed sorter

In this section an example of the extension of the proposed sorter is presented, where $N_s = 8$ and $M_s = 4$, i.e, 4 minimum values are detected among 8 values.

Fig. 4.9 shows the architecture of the proposed 8-to-4 sorter algorithm. The set of 8 random values $S = \{x_0, \ldots, x_7\}$ are split up into two sorted sets $S_0 = \{s_0, s_1, s_2, s_3\}$ and $S_1 = \{s_4, s_5, s_6, s_7\}$ by the two 4-to-4 sorter blocks, in which $s_0 \leq s_1 \leq s_2 \leq s_3$ and $s_4 \leq s_5 \leq s_6 \leq s_7$. The sorter of four values is in common between the proposed algorithm and the odd-even algorithm. For sake of simplicity, the control signal C is appended by the index of the CS that is being generated from. The architecture shown in Fig. 4.9 is designed based on the following analysis:

1. The candidate values for m_1:

   - If $m_0 = s_0 \Rightarrow C_0 = 1, m_1 = \min(s_1, s_4)$.
   - If $m_0 = s_4 \Rightarrow C_0 = 0, m_1 = \min(s_0, s_5)$.

   Thus, the four candidates for m_1 are $\{s_0, s_1, s_4, s_5\}$, the selection between s_0 and s_1 is determined by the comparator-swap CS_0 and the selection between s_1 and s_5 indicated by the 2-to-1 MUX controlled by C_0. Then the two appropriate candidates are compared by CS_1.

2. The candidate values for m_2:

   - If $\{m_0, m_1\} = \{s_0, s_1\} \Rightarrow \{C_0, C_1\} = \{1, 1\}, m_2 = \min(s_2, s_4)$.
   - If $\{m_0, m_1\} = \{s_0, s_4\} \Rightarrow \{C_0, C_1\} = \{1, 0\}, m_2 = \min(s_1, s_5)$.
   - If $\{m_0, m_1\} = \{s_4, s_0\} \Rightarrow \{C_0, C_1\} = \{0, 1\}, m_2 = \min(s_5, s_1)$.
   - If $\{m_0, m_1\} = \{s_4, s_5\} \Rightarrow \{C_0, C_1\} = \{0, 0\}, m_2 = \min(s_6, s_0)$.

   Thus, the six candidates for m_2 are $\{s_0, s_1, s_2, s_4, s_5, s_6\}$. The comparator-swaps CS_0 and CS_1, the 2-to-1 MUX and the 4-to-1 MUX permit to extract the two candidates among six candidates to be entered to CS_2 and hence m_2 will be detected.

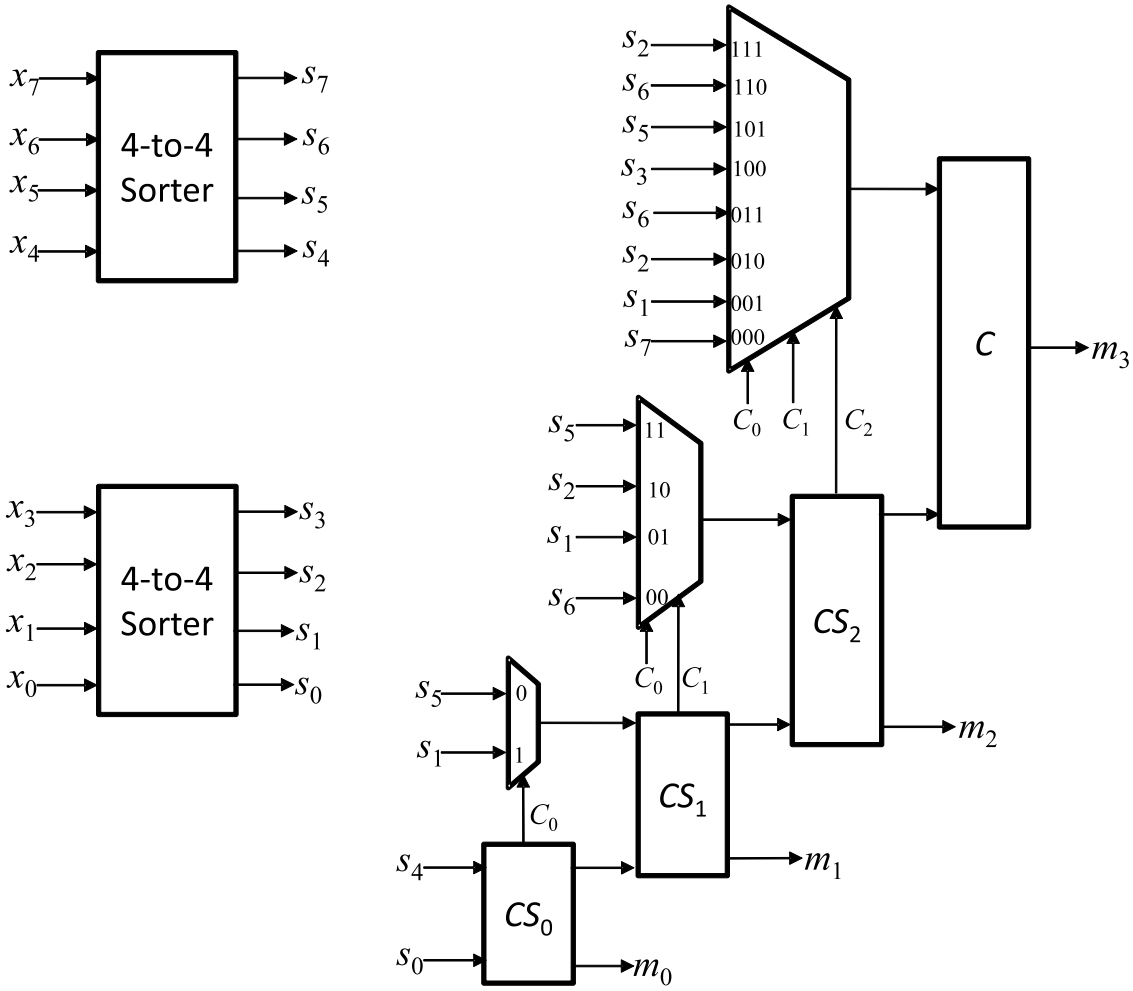

Figure 4.9: Proposed 8-to-4 sorter architecture.

3. The candidate values for m_3:

   - If $\{m_0, m_1, m_2\} = \{s_0, s_4, s_1\} \Rightarrow \{C_0, C_1, C_2\} = \{1, 1, 1\}, m_3 = \min(s_5, s_2)$.
   - If $\{m_0, m_1, m_2\} = \{s_0, s_4, s_5\} \Rightarrow \{C_0, C_1, C_2\} = \{1, 1, 0\}, m_3 = \min(s_1, s_6)$.
   - If $\{m_0, m_1, m_2\} = \{s_0, s_1, s_4\} \Rightarrow \{C_0, C_1, C_2\} = \{1, 0, 1\}, m_3 = \min(s_2, s_5)$.
   - If $\{m_0, m_1, m_2\} = \{s_0, s_1, s_2\} \Rightarrow \{C_0, C_1, C_2\} = \{1, 0, 0\}, m_3 = \min(s_4, s_3)$.
   - If $\{m_0, m_1, m_2\} = \{s_4, s_0, s_5\} \Rightarrow \{C_0, C_1, C_2\} = \{0, 1, 1\}, m_3 = \min(s_1, s_6)$.
   - If $\{m_0, m_1, m_2\} = \{s_4, s_0, s_1\} \Rightarrow \{C_0, C_1, C_2\} = \{0, 1, 0\}, m_3 = \min(s_5, s_2)$.
   - If $\{m_0, m_1, m_2\} = \{s_4, s_5, s_0\} \Rightarrow \{C_0, C_1, C_2\} = \{0, 0, 1\}, m_3 = \min(s_6, s_1)$.
   - If $\{m_0, m_1, m_2\} = \{s_4, s_5, s_6\} \Rightarrow \{C_0, C_1, C_2\} = \{0, 0, 0\}, m_3 = \min(s_0, s_7)$.

   Thus, the eight candidates for m_3 are $\{s_0, s_1, s_2, s_4, s_5, s_6, s_7\}$. The comparator-swaps CS_0, CS_1 and CS_2, the 2-to-1 MUX, the 4-to-1 MUX and the 8-to-1 MUX extract the two candidates among six to be entered to the comparator C and hence m_3 is detected.

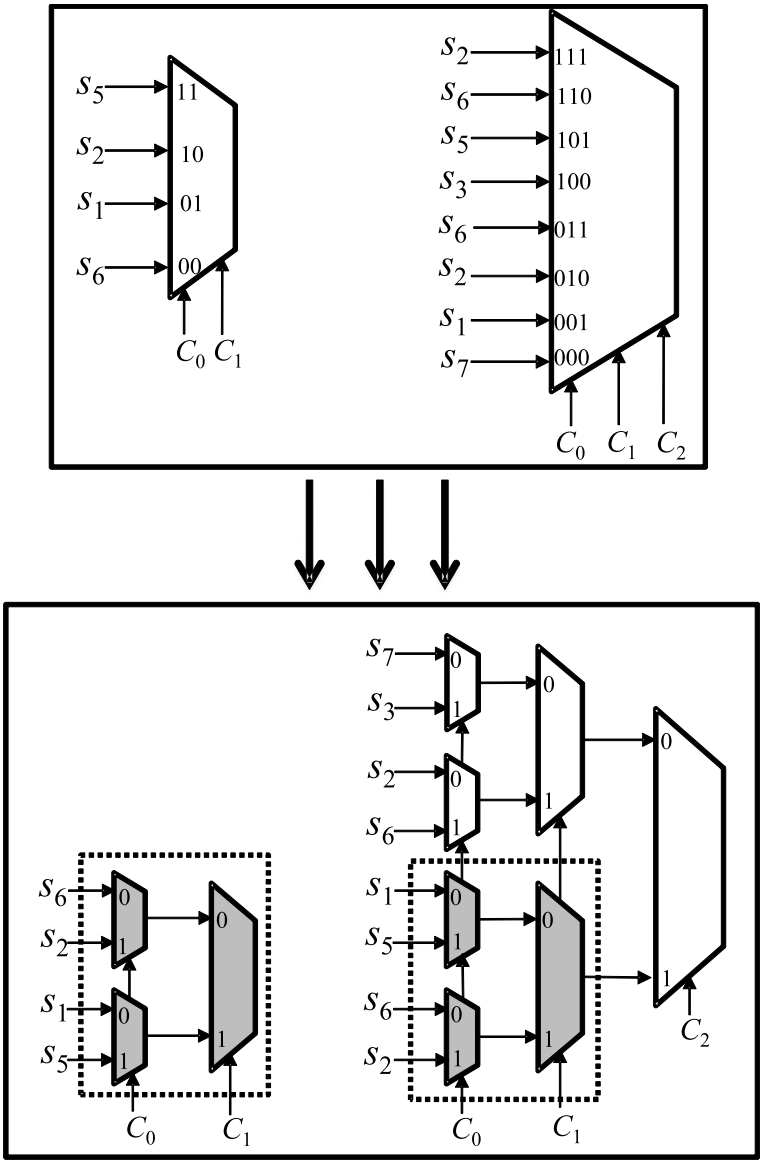

Figure 4.10: Detailed 4-to-1 and 8-to-1 MUXs.

The 8-to-1 MUX of Fig. 4.9 could be simplified where three 2-to-1 MUXs could be removed. Let us detail the 4-to-1 and the 8-to-1 MUXs as shown in Fig. 4.10. As we can notice, the two 2-to-1 MUXs that are surrounded by a dashed rectangle receive the same inputs along with the same control signals. Thus, these two MUXs along with the MUX receiving their outputs can be removed from the 8-to-1 MUX and the corresponding signal can be steered directly from the output of the 4-to-1 MUX implemented in earlier stage. Fig. 4.11 shows the proposed sorter after removing the redundant MUXs.

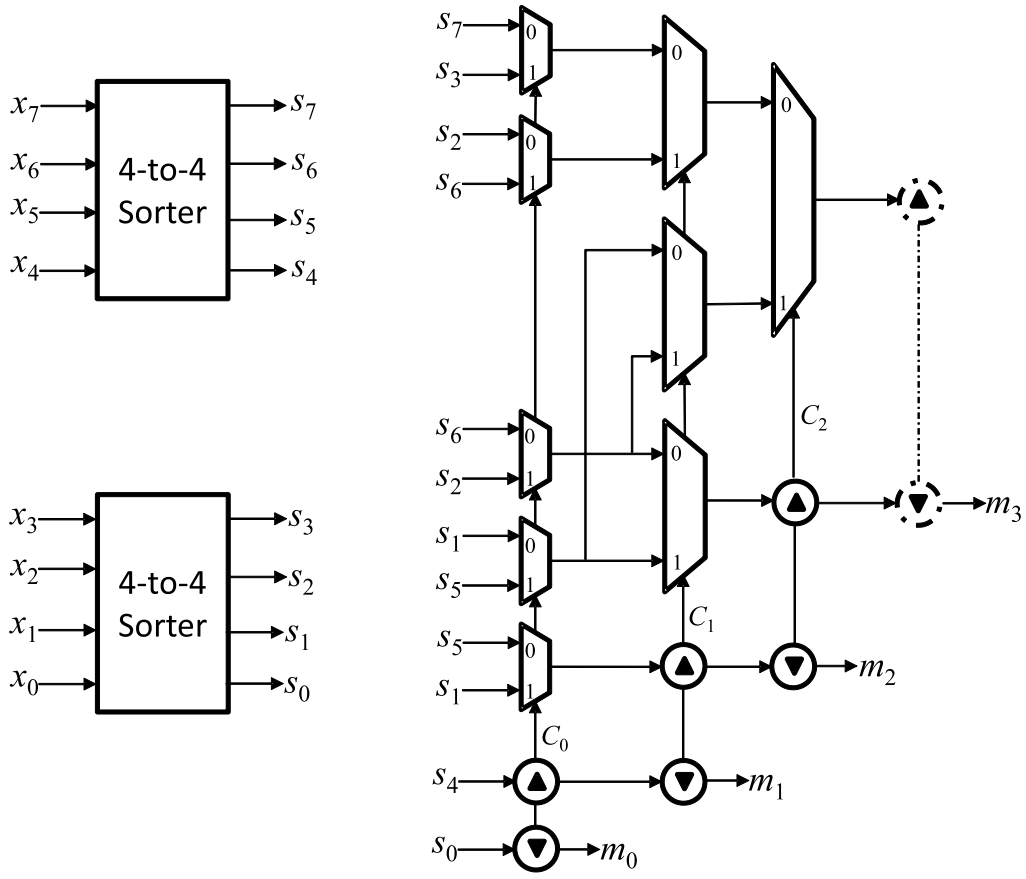

Figure 4.11: Architecture of the simplified proposed 8-to-4 sorter.

Fig. 4.12 presents the 8-to-4 odd-even sorter algorithm [55]. In conclusion, apart the 4-to-4 sorter blocks which are in common between the proposed and the odd-even algorithms, the same number of multiplexers is required for both algorithms (16 2-to-1 MUXs) while the proposed algorithm needs only four comparators while the odd-even algorithm requires eight. Thus, the proposed algorithm is less costly than the odd-even algorithm and the reduction in terms of comparators increases with N_s.

4.3 Conclusion

In this Chapter, we have described new parallel architectures of two core components of the NB-LDPC decoders: LLR generator and sorter.
The proposed LLR generator provides the LLR values of the first n_m most reliable GF symbols. The key idea is to perform the LLR calculation starting from a sorted list of binary LLR values deduced from the received channel observations. A pre-defined set of the candidate elements is defined offline. Then, the outputs are selected based on a parallel and pipelined sorter generating the n_m most reliable GF symbols along

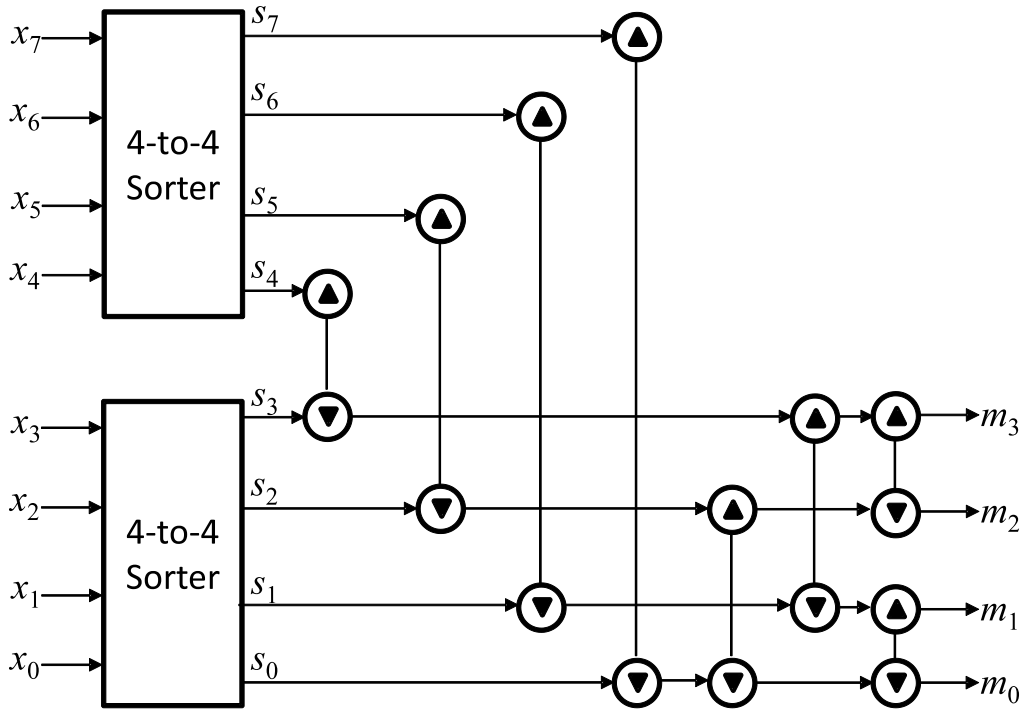

Figure 4.12: Architecture of the odd-even 8-to-4 sorter.

with their associated LLR values. We showed that the proposed LLR generator architecture outperforms the systolic architecture in terms of hardware efficiency and throughput.

Second, we presented the design and the implementation of a low-hardware cost and low latency two-minimum value sorting architecture called FTSES. Theoretical complexity and performance analysis of the proposed sorting algorithm as compared to its best counterpart algorithms in literature have been addressed. Moreover, non-pipelined and pipelined structures have been presented together with synthesis results on TSMC 28 nm standard cell technology. For any size of data stream, $N_s > 32$, the proposed architecture requires the lowest area and offers the highest frequency, where an area efficiency ranging from 1.17 up to 2 is obtained. Furthermore, the theoretical study of the modest example of the extension of the proposed sorter algorithm (8-to-4 sorter) shows that it outperforms the odd-even algorithm in terms of number of comparators while the number of MUXs remains the same. Future work will cover the generalization of the proposed algorithm to extract the $M > 2$ extrema values.

Chapter 5

Proposed parallel and pipelined decoder

In chapter 3, several architectures have been proposed for the serial check node. In this chapter, we revise the hybrid architecture in the context of a fully parallel check node architecture, i.e., a check node where every clock cycle, all the $d_c \times n_m$ input messages are received and processed in parallel. After a fixed latency, all the $d_c \times n_m$ outputs are generated in the same clock cycle. Since the architectural optimization is linked to the code, we first construct a $(N, K) = (144, 120)$ code over GF(64) of code rate $\frac{5}{6}$. Then, from this code, we derive the fully hardware implementation to demonstrate the possibility of defining a high parallel hardware architecture. Note that all the material presented in this chapter is a personal contribution.
The structure of this chapter is as follows. Section 1 defines the code and the parameters used for the implemented Hybrid architecture, then it presents the decoding algorithm, after that it shows the simulation results along with the average number of iterations and throughput versus $\mathrm{E_b/N_0}$. Section 2 introduces the global hardware architecture of the decoder along with the memory structure and the global timing diagram. The chapter continues with section 3 where the hardware description of each component of the decoder is shown. Then, section 4 shows the results on ASIC design and FPGA device of the code. Finally, the hardware emulation is illustrated.

5.1 Code structure and decoding algorithm

This section first introduces the structure of the PCM of the considered NB-LDPC code and its parameters. Then, the decoding algorithm is presented. After that, the simulation results are shown. Finally, the average number of iterations versus $\mathrm{E_b/N_0}$ and the throughput versus $\mathrm{E_b/N_0}$ are presented.

5.1.1 Code Structure

The code considered in this work is a $(144, 120)$ NB-LDPC defined over GF(64) with $d_v = 2$, $d_c = 12$ and a code rate CR=5/6 taken from the LAB-STIC NB-LDPC website [41]. This code is a Quasi-Cyclic LDPC (QC-LDPC) code constructed from the complete 2×12 base matrix $\mathcal{H}$ defined as:

$$\mathcal{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{bmatrix}$$

with an expansion factor equal to 12. Every element $\mathcal{H}(i, j)$, $i = 0, 1$ and $j = 0, 1, \ldots, 11$, during the lifting process of $\mathcal{H}$, is replaced by the 12×12 identity matrix with a right shift rotation of value equal to $\mathcal{H}(i, j)$. Thus, The lifted matrix H is given in Fig. 5.1.

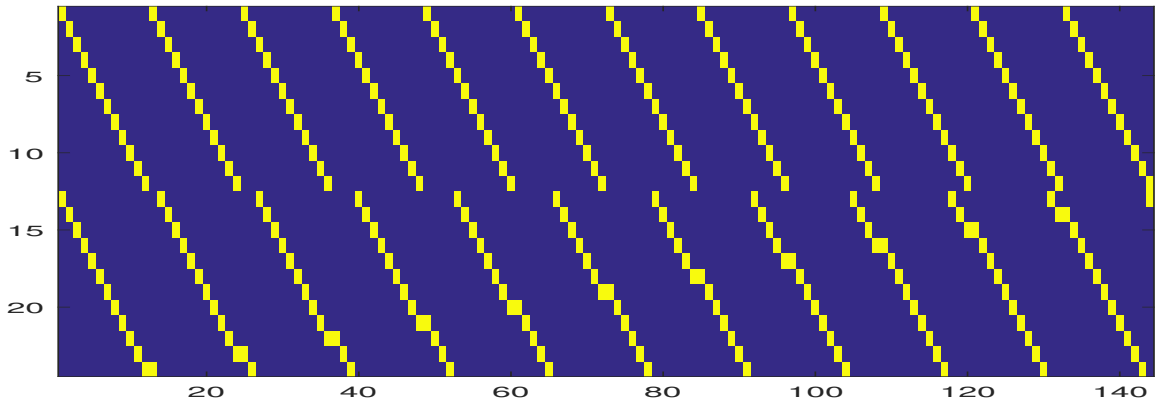

Figure 5.1: The Topology of PCM.

By construction, the girth (i.e., the minimum cycle size in the Tanner Graph associated to the parity check matrix) is equal to 8. The NB-LDPC coefficients are selected passing by two phases as defined in [56]. First, the coefficients of a given check node are selected in order to optimize the property of the binary code associated to the parity check node. In this work, we use the set of coefficients $\{\beta_0, \beta_5, \beta_7, \beta_{25}, \beta_{29}, \beta_{30}, \beta_{38}, \beta_{43}, \beta_{46}, \beta_{50}, \beta_{55}, \beta_{57}\}$. Second, this set of GF coefficients are affected globally to each parity check in order to avoid low weight codewords. The considered primitive polynomial to design the GF values is $P(X) = X^6 + X + 1$.

For sake of simplicity, we consider the representation of PCM shown in Fig. 5.2. The associated PCM is of size $M \times N = 24 \times 144$. Each row represents a CN connected to $d_c = 12$ VNs whose indexes are indicated in each cell of the row. For instance, CN_0 is connected to $\{\mathrm{VN}_0, \mathrm{VN}_{12}, \ldots, \mathrm{VN}_{132}\}$. Those 12 VNs are simply denoted $\{U_0, \ldots, U_{11}\}$ when a CN alone is considered. The 24 CNs are split up into two layers, where the set $\{\mathrm{CN}_0,\ldots,\mathrm{CN}_{11}\}$ constitutes Layer 1 (L_1) and the set $\{\mathrm{CN}_{12},\ldots,\mathrm{CN}_{23}\}$

constitutes Layer 2 (L_2). A layer in a PCM is a set of CNs that do not have common VNs.

	U_0	U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9	U_{10}	U_{11}	
CN_0	0	12	24	36	48	60	72	84	96	108	120	132	
CN_1	1	13	25	37	49	61	73	85	97	109	121	133	
CN_2	2	14	26	38	50	62	74	86	98	110	122	134	
CN_3	3	15	27	39	51	63	75	87	99	111	123	135	
CN_4	4	16	28	40	52	64	76	88	100	112	124	136	
CN_5	5	17	29	41	53	65	77	89	101	113	125	137	L_1
CN_6	6	18	30	42	54	66	78	90	102	114	126	138	
CN_7	7	19	31	43	55	67	79	91	103	115	127	139	
CN_8	8	20	32	44	56	68	80	92	104	116	128	140	
CN_9	9	21	33	45	57	69	81	93	105	117	129	141	
CN_{10}	10	22	34	46	58	70	82	94	106	118	130	142	
CN_{11}	11	23	35	47	59	71	83	95	107	119	131	143	
CN_{12}	0	13	26	39	52	65	78	91	104	117	130	143	
CN_{13}	1	14	27	40	53	66	79	92	105	118	131	132	
CN_{14}	2	15	28	41	54	67	80	93	106	119	120	133	
CN_{15}	3	16	29	42	55	68	81	94	107	108	121	134	
CN_{16}	4	17	30	43	56	69	82	95	96	109	122	135	
CN_{17}	5	18	31	44	57	70	83	84	97	110	123	136	L_2
CN_{18}	6	19	32	45	58	71	72	85	98	111	124	137	
CN_{19}	7	20	33	46	59	60	73	86	99	112	125	138	
CN_{20}	8	21	34	47	48	61	74	87	100	113	126	139	
CN_{21}	9	22	35	36	49	62	75	88	101	114	127	140	
CN_{22}	10	23	24	37	50	63	76	89	102	115	128	141	
CN_{23}	11	12	25	38	51	64	77	90	103	116	129	142	

Figure 5.2: PCM of the (144,120) NB-LDPC code.

The indexes of the non-zero GF coefficients of PCM are shown in Fig. 5.3. These values represent the indexes of the GF symbols used in the GF permutation and

inverse permutation performed at the input and output of the CN respectively. For instance, when making the processing of CN_0, VN_0 is multiplied by the GF value $h_0 = \beta_{43}$ at the input of CN and by $h_0^{-1} = \beta_{43}^{-1} = \beta_{63-43} = \beta_{20}$ before making the VN update, where β_{43}^{-1} is the inverse in GF domain of the GF value β_{43}. Therefore, in the following the non-zero elements $\{h_0, \ldots, h_{11}\}$ vary depending on the CN that is being processed.

5.1.2 Decoding algorithm

In section 3.1.4, the hybrid architecture has been presented to perform the EMS algorithm. For this work, we use a simplified version of the HB(10,0,2) Hybrid architecture, i.e, some bubbles of the HB(10,0,2) have been dismissed to further simplify the hardware and fulfill the constraint of a fully parallel check node decoder. This section presents precisely the implemented algorithm in order of this work to be reproducible. Thus, we will present the exact computation performed by every block along with the precise description of inputs and outputs. Before diving into the decoding steps, let us show how the channel observations are being quantified over 5 bits. Let $y_{s_i} = \{y_{s_i,0}, \ldots, y_{s_i,5}\}$ be the bit representation of y_{s_i}, $i = 0, \ldots, 11$ and $s_i = 0, \ldots, 143$. Thus, each observed bit $y_{s_i,j}$, $j = 0, \ldots, 5$, is quantified as:

$$y_{s_i,j} = \text{sat}((\text{floor}(y_{s_i,j}^{bq} \times Q/\sigma) + 0.5), Q). \tag{5.1}$$

where

$$\text{sat}(a, b) = \begin{cases} b & \text{If} \quad a \geq b \\ -b & \text{If} \quad a \leq -b \\ a & \text{Otherewise} \end{cases} \tag{5.2}$$

$Q = 15$ is the quantization factor and $y_{s_i,j}^{bq}$ is the channel observation before quantization. For instance, when CN_0 is being processed, i.e, the set $\{VN_0, VN_{12}, \ldots, VN_{132}\}$ is considered and hence $s_0 = 0$, $s_1 = 12$, $\ldots$, $s_{11} = 132$ are the indexes associated to VN_0, VN_{12}, $\ldots$, VN_{132} respectively.

Therefore, the decoding steps are:

1. Initialization: Each VN is initialized with its $n_m = 4$ intrinsic candidates I as: $U_i[k] = I_i[k]$, $i = 0, \ldots, 11$ and $k = 0, 1, 2, 3$. The architecture shown in section 4.1.4 is the considered architecture to generate the $n_m = 4$ intrinsic candidates for each observed symbol y_{s_i}.

2. Check Node Variable Node (CN-VN) processing: The CN implemented in the software simulator of the decoder is HB(10, 0, 2) (see section 3.1.4). In the proposed decoder, the CN and the VN units are merged together. The inputs of the CN-VN unit are: 1) the set of vectors $\{U_0, \ldots, U_{11}\}$, where each U_i, $i = 0, \ldots, 11$, carries $n_m = 4$ (LLR, GF) couples; 2) the set of the most reliable GF intrinsic symbols $\{I_0[0], \ldots, I_{11}[0]\}$; 3) the indexes $\{\pi_i[0], \pi_i[1], \pi_i[2]\}$, $i =$

	h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	
CN_0	43	0	7	30	25	46	55	38	50	57	29	5	
CN_1	5	43	0	7	30	25	46	55	38	50	57	29	
CN_2	29	5	43	0	7	30	25	46	55	38	50	57	
CN_3	57	29	5	43	0	7	30	25	46	55	38	50	
CN_4	50	57	29	5	43	0	7	30	25	46	55	38	
CN_5	38	50	57	29	5	43	0	7	30	25	46	55	
CN_6	46	55	38	50	57	29	5	43	0	7	30	25	L_1
CN_7	25	46	55	38	50	57	29	5	43	0	7	30	
CN_8	30	25	46	55	38	50	57	29	5	43	0	7	
CN_9	7	30	25	46	55	38	50	57	29	5	43	0	
CN_{10}	0	7	30	25	46	55	38	50	57	29	5	43	
CN_{11}	43	0	7	30	25	46	55	38	50	57	29	5	
CN_{12}	0	38	7	50	30	57	25	29	46	5	55	43	
CN_{13}	55	0	38	7	50	30	57	25	29	46	5	43	
CN_{14}	43	55	0	38	7	50	30	57	25	29	5	46	
CN_{15}	46	43	55	0	38	7	50	30	57	29	25	5	
CN_{16}	5	46	43	55	0	38	7	50	57	30	29	25	
CN_{17}	25	5	46	43	55	0	38	50	7	57	30	29	
CN_{18}	29	25	5	46	43	55	38	0	50	7	57	30	
CN_{19}	30	29	25	5	46	55	43	38	0	50	7	57	
CN_{20}	57	30	29	25	46	5	55	43	38	0	50	7	
CN_{21}	7	57	30	25	29	46	5	55	43	38	0	50	
CN_{22}	50	7	30	57	25	29	46	5	55	43	38	0	
CN_{23}	0	7	50	30	57	25	29	46	5	55	43	38	L_2

Figure 5.3: The non-zero coefficients of the PCM.

$0, \ldots, 11$, of the 3 minimum values of y_{s_i} required to recompute the n_m intrinsic messages of VN_i; 4) the non-zero elements $\{h_0, \ldots, h_{11}\}$; 5) the inverse of the

non-zero elements $\{h_0^{-1}, \ldots, h_{11}^{-1}\}$; 6) finally, the absolute values of the observed bits $\{\{|y_{s_0,j}|\}, \ldots, \{|y_{s_{11},j}|\}\}$, $j = 0, \ldots, 5$. Therefore, the steps of the CN-VN unit to update the messages are:

- Presorting, switching and GF permutation: the presorting generates the indexes $\Psi = \{\psi[0], \ldots, \psi[11]\}$ based on the presorting principle shown in Fig. 2.8 for $d_c = 12$. Based on Ψ, all the inputs of the CN-VN are switched. The switched data are appended by the symbol $'$ hereafter. The GF permutation is being processed in this phase based on equation (2.21). Thus, the set of vectors $\{U'_0, \ldots, U'_{11}\}$ contains the permuted GF value.
- ECNs computation: The set of vector messages $\{U'_0, \ldots, U'_{11}\}$ are split up into four groups as: 1) $\{U'_0, U'_1, U'_2, U'_3\}$ where each U'_i, $i = 0, \ldots, 3$, contains one couple of LLR equal to 0, i.e, $U'^{+}_i[0] = 0$; 2) $\{U'_4, U'_5, U'_6, U'_7\}$ where each U'_i, $i = 4, \ldots, 7$, contains two couples; 3) $\{U'_8\}$ of three couples; 4) $\{U'_9, U'_{10}, U'_{11}\}$ where each U'_i, $i = 9, 10, 11$, contains four couples. In the following, we show how the vector messages are combined by set of ECNs. First, the set $\{U'_0, \ldots, U'_9\}$ are combined by the Syndrome Node (SN) block as shown in Fig. 5.4. As we can see, the ECNs SN1 to SN9 are very simple, where there are three ECNs of one bubble (SN1, SN2 and SN3), i.e, they require only GF addition (XOR gate), and the number of required bubbles in the rest of the ECNs, SN4, ..., SN9, varies from 2 up to 9. The number of considered bubbles in each SNi, $i = 1, \ldots, 9$, is found by simulation depending on the allowed performance loss.

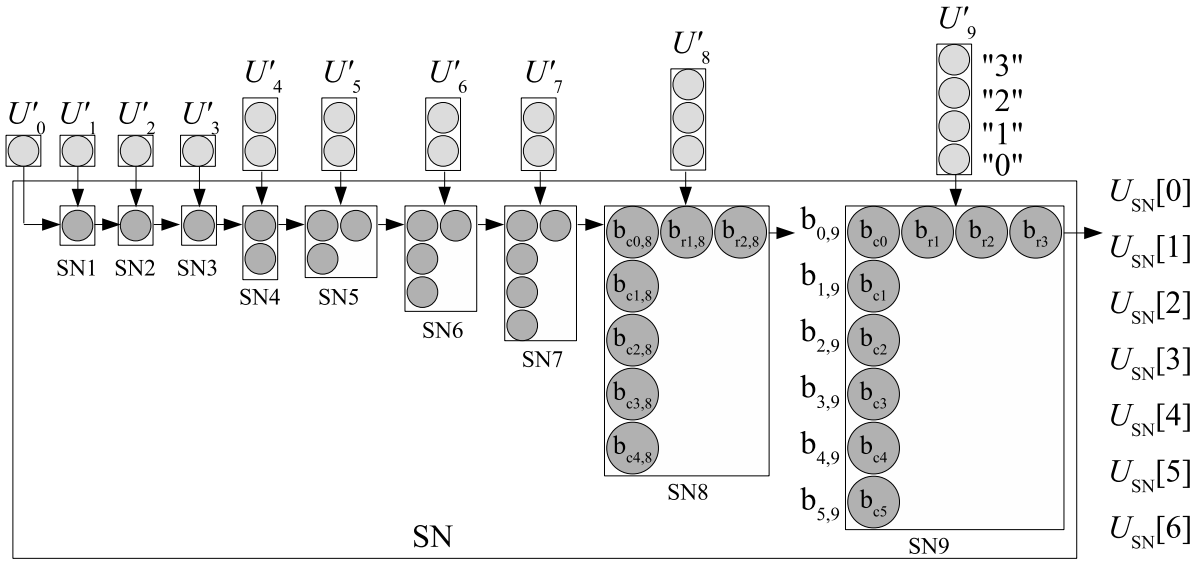

Figure 5.4: SN shape.

The vectors $\{U'_0, \ldots, U'_8\}$ are first combined and then the $\{\mathrm{b}_{0,9}, \ldots, \mathrm{b}_{5,9}\}$, is combined with U'_9 to generate the output set $\{U_{\mathrm{SN}}\}$. Let us start with SN1 to SN8. The bubbles in SN8 are computed wisely considering that $U'^{+}_8[1] \leq \cdots \leq U'^{+}_4[1]$ due to the presorting. Each bubble carries the LLR value, the GF value and the valid syndrome vector b^{vsv} information as:
$\mathrm{b}^{+}_{\mathrm{c0,8}} = 0$, $\mathrm{b}^{\oplus}_{\mathrm{c0,8}} = U'^{\oplus}_0[0] \bigoplus \cdots \bigoplus U'^{\oplus}_8[0]$.
$\mathrm{b}^{+}_{\mathrm{c1,8}} = U'^{+}_7[1]$, $\mathrm{b}^{\oplus}_{\mathrm{c1,8}} = U'^{\oplus}_7[1] \bigoplus U'^{\oplus}_0[0] \bigoplus \cdots \bigoplus U'^{\oplus}_6[0] \bigoplus U'^{\oplus}_8[0]$.
$\mathrm{b}^{+}_{\mathrm{c2,8}} = U'^{+}_6[1]$, $\mathrm{b}^{\oplus}_{\mathrm{c2,8}} = U'^{\oplus}_6[1] \bigoplus U'^{\oplus}_0[0] \bigoplus \cdots \bigoplus U'^{\oplus}_5[0] \bigoplus U'^{\oplus}_7[0] \bigoplus U'^{\oplus}_8[0]$.
$\mathrm{b}^{+}_{\mathrm{c3,8}} = U'^{+}_5[1]$, $\mathrm{b}^{\oplus}_{\mathrm{c3,8}} = U'^{\oplus}_5[1] \bigoplus U'^{\oplus}_0[0] \bigoplus \cdots \bigoplus U'^{\oplus}_4[0] \bigoplus U'^{\oplus}_6[0] \cdots \bigoplus U'^{\oplus}_8[0]$.
$\mathrm{b}^{+}_{\mathrm{c4,8}} = U'^{+}_4[1]$, $\mathrm{b}^{\oplus}_{\mathrm{c4,8}} = U'^{\oplus}_4[1] \bigoplus U'^{\oplus}_0[0] \bigoplus \cdots \bigoplus U'^{\oplus}_3[0] \bigoplus U'^{\oplus}_5[0] \cdots \bigoplus U'^{\oplus}_8[0]$.
$\mathrm{b}^{+}_{\mathrm{r1,8}} = U'^{+}_8[1]$, $\mathrm{b}^{\oplus}_{\mathrm{r1,8}} = U'^{\oplus}_8[1] \bigoplus U'^{\oplus}_0[0] \bigoplus \cdots \bigoplus U'^{\oplus}_7[0]$.
$\mathrm{b}^{+}_{\mathrm{r2,8}} = U'^{+}_8[2]$, $\mathrm{b}^{\oplus}_{\mathrm{r2,8}} = U'^{\oplus}_8[2] \bigoplus U'^{\oplus}_0[0] \bigoplus \cdots \bigoplus U'^{\oplus}_7[0]$.

and

$\mathrm{b}^{\mathrm{vsv}}_{\mathrm{c0,8}} = \{1,1,1,1,1,1,1,1,1\}$.
$\mathrm{b}^{\mathrm{vsv}}_{\mathrm{c1,8}} = \{1,1,1,1,1,1,1,0,1\}$.
$\mathrm{b}^{\mathrm{vsv}}_{\mathrm{c2,8}} = \{1,1,1,1,1,1,0,1,1\}$.
$\mathrm{b}^{\mathrm{vsv}}_{\mathrm{c3,8}} = \{1,1,1,1,1,0,1,1,1\}$.
$\mathrm{b}^{\mathrm{vsv}}_{\mathrm{c4,8}} = \{1,1,1,1,0,1,1,1,1\}$.
$\mathrm{b}^{\mathrm{vsv}}_{\mathrm{r1,8}} = \{1,1,1,1,1,1,1,1,0\}$.
$\mathrm{b}^{\mathrm{vsv}}_{\mathrm{r2,8}} = \{1,1,1,1,1,1,1,1,0\}$.

The VSV indicates whether the bubble is generated from the most reliable GF symbol or not. In more details, let $\mathrm{b} \in \{\mathrm{b}_{\mathrm{c0,8}}, \ldots, \mathrm{b}_{\mathrm{c4,8}}, \mathrm{b}_{\mathrm{r1,8}}, \mathrm{b}_{\mathrm{r2,8}}\}$, if $\mathrm{b}^{\mathrm{vsv}}[i] = 1$, $i = 0, \ldots, 8$, thus $U'_i[0]$ contributes in the computation of $\mathrm{b}^{\oplus}$, otherwise, $U'_i[k]$ is contributing, $k = 1, 2, 3$.

Therefore, the outputs of SN8 are: $\mathrm{b}_{0,9} = \mathrm{b}_{\mathrm{c0,8}}$, $\mathrm{b}_{1,9} = \mathrm{b}_{\mathrm{r1,8}}$ and $\{\mathrm{b}_{2,9}, \mathrm{b}_{3,9}, \mathrm{b}_{4,9}, \mathrm{b}_{5,9}\} = \mathrm{sort}_{5-4}(\{\mathrm{b}_{\mathrm{r2,8}}, \mathrm{b}_{\mathrm{c1,8}}, \mathrm{b}_{\mathrm{c2,8}}, \mathrm{b}_{\mathrm{c3,8}}, \mathrm{b}_{\mathrm{c4,8}}\})$, where in this case the sort_{5-4} function is to detect the four bubbles having the lowest LLR values among five bubbles and hence $\mathrm{b}^{+}_{2,9} \leq \mathrm{b}^{+}_{3,9} \leq \mathrm{b}^{+}_{4,9} \leq \mathrm{b}^{+}_{5,9}$.

Next, SN9 receives the set $\{\mathrm{b}_{0,9}, \ldots, \mathrm{b}_{5,9}\}$ along with U'_9 to generate the outputs $U_{\mathrm{SN}}[0], \ldots, U_{\mathrm{SN}}[6]$, where each $U_{\mathrm{SN}}[i] = (U^{+}_{\mathrm{SN}}[i], U^{\oplus}_{\mathrm{SN}}[i], U^{\mathrm{vsv}}_{\mathrm{SN}}[i])$, $i = 0, \ldots, 6$. The bubbles shown in SN9 along with their VSV information are computed as:

$\mathrm{b}^{+}_{\mathrm{c0}} = 0$, $\mathrm{b}^{\oplus}_{\mathrm{c0}} = \mathrm{b}^{\oplus}_{0,9} \bigoplus U'^{\oplus}_9[0]$, $\mathrm{b}^{\mathrm{vsv}}_{\mathrm{c0}} = (\mathrm{b}^{\mathrm{vsv}}_{0,9}, 1)$.
$\mathrm{b}^{+}_{\mathrm{c1}} = \mathrm{b}^{+}_{1,9}$, $\mathrm{b}^{\oplus}_{\mathrm{c1}} = \mathrm{b}^{\oplus}_{1,9} \bigoplus U'^{\oplus}_9[0]$, $\mathrm{b}^{\mathrm{vsv}}_{\mathrm{c1}} = (\mathrm{b}^{\mathrm{vsv}}_{1,9}, 1)$.
$\mathrm{b}^{+}_{\mathrm{c2}} = \mathrm{b}^{+}_{2,9}$, $\mathrm{b}^{\oplus}_{\mathrm{c2}} = \mathrm{b}^{\oplus}_{2,9} \bigoplus U'^{\oplus}_9[0]$, $\mathrm{b}^{\mathrm{vsv}}_{\mathrm{c2}} = (\mathrm{b}^{\mathrm{vsv}}_{2,9}, 1)$.

$\mathrm{b}_{\mathrm{c3}}^{+} = \mathrm{b}_{3,9}^{+}$, $\mathrm{b}_{\mathrm{c3}}^{\oplus} = \mathrm{b}_{3,9}^{\oplus} \bigoplus U_9'^{\oplus}[0]$, $\mathrm{b}_{\mathrm{c3}}^{\mathrm{vsv}} = (\mathrm{b}_{3,9}^{\mathrm{vsv}}, 1)$.
$\mathrm{b}_{\mathrm{c4}}^{+} = \mathrm{b}_{4,9}^{+}$, $\mathrm{b}_{\mathrm{c4}}^{\oplus} = \mathrm{b}_{4,9}^{\oplus} \bigoplus U_9'^{\oplus}[0]$, $\mathrm{b}_{\mathrm{c4}}^{\mathrm{vsv}} = (\mathrm{b}_{4,9}^{\mathrm{vsv}}, 1)$.
$\mathrm{b}_{\mathrm{c5}}^{+} = \mathrm{b}_{5,9}^{+}$, $\mathrm{b}_{\mathrm{c5}}^{\oplus} = \mathrm{b}_{5,9}^{\oplus} \bigoplus U_9'^{\oplus}[0]$, $\mathrm{b}_{\mathrm{c5}}^{\mathrm{vsv}} = (\mathrm{b}_{5,9}^{\mathrm{vsv}}, 1)$.
$\mathrm{b}_{\mathrm{r1}}^{+} = U_9'^{+}[1]$, $\mathrm{b}_{\mathrm{r1}}^{\oplus} = \mathrm{b}_{0,9}^{\oplus} \bigoplus U_9'^{\oplus}[1]$, $\mathrm{b}_{\mathrm{r1}}^{\mathrm{vsv}} = (\mathrm{b}_{0,9}^{\mathrm{vsv}}, 0)$.
$\mathrm{b}_{\mathrm{r2}}^{+} = U_9'^{+}[2]$, $\mathrm{b}_{\mathrm{r2}}^{\oplus} = \mathrm{b}_{0,9}^{\oplus} \bigoplus U_9'^{\oplus}[2]$, $\mathrm{b}_{\mathrm{r2}}^{\mathrm{vsv}} = (\mathrm{b}_{0,9}^{\mathrm{vsv}}, 0)$.
$\mathrm{b}_{\mathrm{r3}}^{+} = U_9'^{+}[3]$, $\mathrm{b}_{\mathrm{r3}}^{\oplus} = \mathrm{b}_{0,9}^{\oplus} \bigoplus U_9'^{\oplus}[3]$, $\mathrm{b}_{\mathrm{r3}}^{\mathrm{vsv}} = (\mathrm{b}_{0,9}^{\mathrm{vsv}}, 0)$.

Therefore, the outputs of the SN block are generated as: $U_{\mathrm{SN}}[0] = \mathrm{b}_{\mathrm{c0}}$, $U_{\mathrm{SN}}[1] = \mathrm{b}_{\mathrm{r1}}$ and $\{U_{\mathrm{SN}}[2], \ldots, U_{\mathrm{SN}}[6]\} = \mathrm{sort}_{7-5}(\{\mathrm{b}_{\mathrm{c1}}, \mathrm{b}_{\mathrm{c2}}, \mathrm{b}_{\mathrm{c3}}, \mathrm{b}_{\mathrm{c4}}, \mathrm{b}_{\mathrm{c5}}, \mathrm{b}_{\mathrm{r2}}, \mathrm{b}_{\mathrm{r3}}\})$, where in this case the sort_{7-5} function is to detect the five bubbles having the lowest LLR values among seven bubbles and hence $U_{\mathrm{SN}}^{+}[2] \leq \cdots \leq U_{\mathrm{SN}}^{+}[6]$.

In parallel with SN, the two vectors U_{10}' and U_{11}' are combined by ECN1, see Fig. 5.5. The bubbles shown in ECN1 are computed as:

$\mathrm{b}_0^{+} = 0$, $\mathrm{b}_0^{\oplus} = U_{10}'^{\oplus}[0] \bigoplus U_{11}'^{\oplus}[0]$.
$\mathrm{b}_1^{+} = U_{11}'^{+}[1]$, $\mathrm{b}_1^{\oplus} = U_{10}'^{\oplus}[0] \bigoplus U_{11}'^{\oplus}[1]$.
$\mathrm{b}_2^{+} = U_{11}'^{+}[2]$, $\mathrm{b}_2^{\oplus} = U_{10}'^{\oplus}[0] \bigoplus U_{11}'^{\oplus}[2]$.
$\mathrm{b}_3^{+} = U_{11}'^{+}[3]$, $\mathrm{b}_3^{\oplus} = U_{10}'^{\oplus}[0] \bigoplus U_{11}'^{\oplus}[3]$.
$\mathrm{b}_4^{+} = U_{10}'^{+}[1]$, $\mathrm{b}_4^{\oplus} = U_{10}'^{\oplus}[1] \bigoplus U_{11}'^{\oplus}[0]$.
$\mathrm{b}_5^{+} = U_{10}'^{+}[1] + U_{11}'^{+}[1]$, $\mathrm{b}_5^{\oplus} = U_{10}'^{\oplus}[1] \bigoplus U_{11}'^{\oplus}[1]$.
$\mathrm{b}_6^{+} = U_{10}'^{+}[1] + U_{11}'^{+}[2]$, $\mathrm{b}_6^{\oplus} = U_{10}'^{\oplus}[1] \bigoplus U_{11}'^{\oplus}[2]$.
$\mathrm{b}_7^{+} = U_{10}'^{+}[2]$, $\mathrm{b}_7^{\oplus} = U_{10}'^{\oplus}[2] \bigoplus U_{11}'^{\oplus}[0]$.
$\mathrm{b}_8^{+} = U_{10}'^{+}[2] + U_{11}'^{+}[1]$, $\mathrm{b}_8^{\oplus} = U_{10}'^{\oplus}[2] \bigoplus U_{11}'^{\oplus}[1]$.
$\mathrm{b}_9^{+} = U_{10}'^{+}[3]$, $\mathrm{b}_9^{\oplus} = U_{10}'^{\oplus}[3] \bigoplus U_{11}'^{\oplus}[0]$.

Thus, the outputs are generated as: $U_{\mathrm{ECN1}}[0] = \mathrm{b}_0$, $U_{\mathrm{ECN1}}[1] = \mathrm{b}_1$ and $\{U_{\mathrm{ECN1}}[2], \ldots, U_{\mathrm{ECN1}}[6]\} = \mathrm{sort}_{8-5}(\mathrm{b}_2, \ldots, \mathrm{b}_9)$, where in this case the min function is to detect the five bubbles having the lowest LLR values among eight bubbles and hence $U_{\mathrm{ECN1}}^{+}[2] \leq \cdots \leq U_{\mathrm{ECN1}}^{+}[6]$.

After that, as Fig. 5.6 shows, the two vectors U_{SN} and U_{ECN1} are merged to generate the output vector U_{ECN2} of 19 couples. The bubbles shown in Fig. 5.6 are computed as:

$\mathrm{b}_0^{+} = 0$, $\mathrm{b}_0^{\oplus} = U_{\mathrm{SN}}^{\oplus}[0] \bigoplus U_{\mathrm{ECN1}}^{\oplus}[0]$.
$\mathrm{b}_1^{+} = U_{\mathrm{ECN1}}^{+}[1]$, $\mathrm{b}_1^{\oplus} = U_{\mathrm{SN}}^{\oplus}[0] \bigoplus U_{\mathrm{ECN1}}^{\oplus}[1]$.
$\mathrm{b}_2^{+} = U_{\mathrm{ECN1}}^{+}[2]$, $\mathrm{b}_2^{\oplus} = U_{\mathrm{SN}}^{\oplus}[0] \bigoplus U_{\mathrm{ECN1}}^{\oplus}[2]$.
$\mathrm{b}_3^{+} = U_{\mathrm{ECN1}}^{+}[3]$, $\mathrm{b}_3^{\oplus} = U_{\mathrm{SN}}^{\oplus}[0] \bigoplus U_{\mathrm{ECN1}}^{\oplus}[3]$.

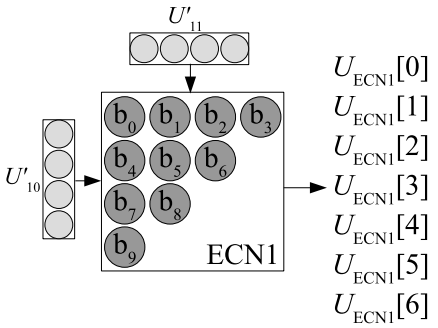

Figure 5.5: ECN1 shape.

$\mathrm{b}_4^+ = U_{\mathrm{ECN1}}^+[4]$, $\mathrm{b}_4^\oplus = U_{\mathrm{SN}}^\oplus[0] \bigoplus U_{\mathrm{ECN1}}^\oplus[4]$.
$\mathrm{b}_5^+ = U_{\mathrm{ECN1}}^+[5]$, $\mathrm{b}_5^\oplus = U_{\mathrm{SN}}^\oplus[0] \bigoplus U_{\mathrm{ECN1}}^\oplus[5]$.
$\mathrm{b}_6^+ = U_{\mathrm{SN}}^+[1]$, $\mathrm{b}_6^\oplus = U_{\mathrm{SN}}^\oplus[1] \bigoplus U_{\mathrm{ECN1}}^\oplus[0]$.
$\mathrm{b}_7^+ = U_{\mathrm{SN}}^+[1] + U_{\mathrm{ECN1}}^+[1]$, $\mathrm{b}_7^\oplus = U_{\mathrm{SN}}^\oplus[1] \bigoplus U_{\mathrm{ECN1}}^\oplus[1]$.
$\mathrm{b}_8^+ = U_{\mathrm{SN}}^+[1] + U_{\mathrm{ECN1}}^+[2]$, $\mathrm{b}_8^\oplus = U_{\mathrm{SN}}^\oplus[1] \bigoplus U_{\mathrm{ECN1}}^\oplus[2]$.
$\mathrm{b}_9^+ = U_{\mathrm{SN}}^+[1] + U_{\mathrm{ECN1}}^+[3]$, $\mathrm{b}_9^\oplus = U_{\mathrm{SN}}^\oplus[1] \bigoplus U_{\mathrm{ECN1}}^\oplus[3]$.
$\mathrm{b}_{10}^+ = U_{\mathrm{SN}}^+[1] + U_{\mathrm{ECN1}}^+[4]$, $\mathrm{b}_{10}^\oplus = U_{\mathrm{SN}}^\oplus[1] \bigoplus U_{\mathrm{ECN1}}^\oplus[4]$.
$\mathrm{b}_{11}^+ = U_{\mathrm{SN}}^+[2]$, $\mathrm{b}_{11}^\oplus = U_{\mathrm{SN}}^\oplus[2] \bigoplus U_{\mathrm{ECN1}}^\oplus[0]$.
$\mathrm{b}_{12}^+ = U_{\mathrm{SN}}^+[2] + U_{\mathrm{ECN1}}^+[1]$, $\mathrm{b}_{12}^\oplus = U_{\mathrm{SN}}^\oplus[2] \bigoplus U_{\mathrm{ECN1}}^\oplus[1]$.
$\mathrm{b}_{13}^+ = U_{\mathrm{SN}}^+[3]$, $\mathrm{b}_{13}^\oplus = U_{\mathrm{SN}}^\oplus[3] \bigoplus U_{\mathrm{ECN1}}^\oplus[0]$.
$\mathrm{b}_{14}^+ = U_{\mathrm{SN}}^+[3] + U_{\mathrm{ECN1}}^+[1]$, $\mathrm{b}_{14}^\oplus = U_{\mathrm{SN}}^\oplus[3] \bigoplus U_{\mathrm{ECN1}}^\oplus[1]$.
$\mathrm{b}_{15}^+ = U_{\mathrm{SN}}^+[4]$, $\mathrm{b}_{15}^\oplus = U_{\mathrm{SN}}^\oplus[4] \bigoplus U_{\mathrm{ECN1}}^\oplus[0]$.
$\mathrm{b}_{16}^+ = U_{\mathrm{SN}}^+[4] + U_{\mathrm{ECN1}}^+[1]$, $\mathrm{b}_{16}^\oplus = U_{\mathrm{SN}}^\oplus[4] \bigoplus U_{\mathrm{ECN1}}^\oplus[1]$.
$\mathrm{b}_{17}^+ = U_{\mathrm{SN}}^+[5]$, $\mathrm{b}_{17}^\oplus = U_{\mathrm{SN}}^\oplus[5] \bigoplus U_{\mathrm{ECN1}}^\oplus[0]$.
$\mathrm{b}_{18}^+ = U_{\mathrm{SN}}^+[5] + U_{\mathrm{ECN1}}^+[1]$, $\mathrm{b}_{18}^\oplus = U_{\mathrm{SN}}^\oplus[5] \bigoplus U_{\mathrm{ECN1}}^\oplus[1]$.
$\mathrm{b}_{19}^+ = U_{\mathrm{SN}}^+[6]$, $\mathrm{b}_{19}^\oplus = U_{\mathrm{SN}}^\oplus[6] \bigoplus U_{\mathrm{ECN1}}^\oplus[0]$.

Note that VSV is not used in ECN2, so this information propagates directly to the output considering that all the bubbles that belong to the same row in Fig. 5.6 have the same VSV information, i.e, $\mathrm{b}_0^{\mathrm{vsv}} = \cdots = \mathrm{b}_5^{\mathrm{vsv}} = U_{\mathrm{SN}}^{\mathrm{vsv}}[0]$, $\mathrm{b}_6^{\mathrm{vsv}} = \cdots = \mathrm{b}_{10}^{\mathrm{vsv}} = U_{\mathrm{SN}}^{\mathrm{vsv}}[1] \ldots$ etc. Thus, the outputs are $U_{\mathrm{ECN2}}[i] = \mathrm{b}_i$, $i = 0, \ldots, 19$, i.e, there is no need to sort the bubbles in terms of their LLR values since the intrinsic LLR values will be added to them during the VN processing and hence the sorted bubbles will become unsorted. The LLR value b_{16}^+ is considered as offset (O) for the VNs processing associated to

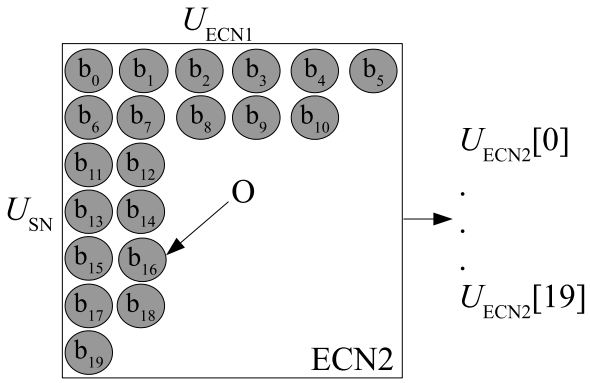

Figure 5.6: ECN2 shape.

the extrinsic messages $\{V'_0, \ldots, V'_9\}$ as will be shown hereafter.

ECN3 and ECN4 operate in parallel with ECN2 to generate the two extrinsic messages V'_{10} and V'_{11} respectively as shown in Fig. 5.7. The two ECNs have the same shape and functionality. Thus, in the following we will show the functionality of ECN3. The required inputs for ECN3 block are $\{U_{\mathrm{SN}}\}$, U'_{11} and the non-zero element h'^{-1}_{11} which is the inverse of h'_{11}. Note that the $U^{\mathrm{vsv}}_{\mathrm{SN}}$ is not needed since the generation of V'_{10} is performed using an ECN and not using a decorrelation operation with U'_{11}, as will be the case for $\{V'_0, \ldots, V'_9\}$. Thus, and only for example, $\mathrm{b}^{+}_{0} = 0$ and $\mathrm{b}^{\oplus}_{0} = (U^{\oplus}_{\mathrm{SN}}[0] \bigoplus U'^{\oplus}_{11}[0]).h'^{-1}_{11}$. The same manner of computing the bubbles shown in ECN1 and ECN2 is applied to the remaining bubbles, i.e, it is only a matter of changing the indexes of U_{SN} and $U'_{11}[0]$.

After that, the bubbles are directly mapped to the outputs as: $V'_{10}[i] = \mathrm{b}_i$ where $i = 0, \ldots, 15$, without the need of a sorter operation. The LLR value b^{+}_{16} is considered as offset for the VN that processes V'_{10}. Thus, in order to save the complexity of detecting the offset value - which is considered as the last valid output in the S-bubble approach - we managed to consider the LLR value of an offline selected bubble as offset.

In this case, two extrinsic messages V'_{10} and V'_{11} are generated independently, while the rest of the extrinsic messages $\{V'_0, \ldots, V'_9\}$ are generated by the decorrelation and inverse permutation processes as will be shown next.

- Decorrelation and inverse permutation: The set of couples $\{U_{\mathrm{ECN2}}[0], \ldots, U_{\mathrm{ECN2}}[19]\}$ carries the combination of all the input vectors $\{U'_0, \ldots, U'_{11}\}$. Thus, to extract the exact extrinsic messages $\{V'_0, \ldots, V'_9\}$, the 19 symbols $\{U_{\mathrm{ECN2}}[0], \ldots, U_{\mathrm{ECN2}}[19]\}$ should be decorrelated as:

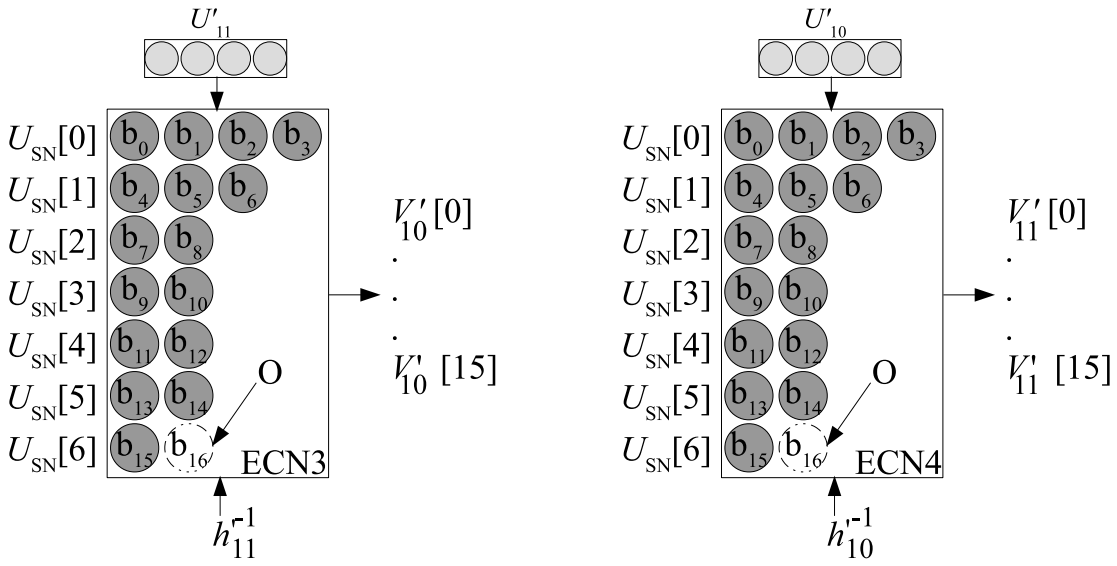

Figure 5.7: ECN3 and ECN4 structures.

$$V_i'[j] = (V_i'^{+}[j], V_i'^{\oplus}[j]) =$$
$$\begin{cases} (U^{+}_{\mathrm{ECN2}}[j], (U^{\oplus}_{\mathrm{ECN2}}[j] \bigoplus U_i'[0]).h_i'^{-1}) & \text{If } U^{\mathrm{VSV}}_{\mathrm{ECN2}}[j][i] = 1 \\ (Sat, 0) & \text{Otherwise} \end{cases} \quad (5.3)$$

where, $i = 0, \ldots, 9$, $j = 0, \ldots, 19$ and Sat is the maximum LLR value. In practice, since we know that only the most reliable symbol is considered from $\{U_0', \ldots, U_3'\}$, we manage to consider all the 20 symbols $\{U_{\mathrm{ECN2}}[0], \ldots, U_{\mathrm{ECN2}}[19]\}$ without checking their associated VSV information.

Now, after generating all the extrinsic messages $\{V_0', \ldots, V_{11}'\}$, the updated messages will be computed. We recall that the length of the extrinsic vectors $\{V_0', \ldots, V_9'\}$ is equal to 19 while the two extrinsic vectors V_{10}' and V_{11}' are of length equal to 16. In addition, the considered offset to process $\{V_0', \ldots, V_9'\}$ is equal to b_{16}^{+} shown in Fig. 5.6 while for V_{10}' and V_{10}' is equal to b_{16}^{+} shown in Fig. 5.7 in ECN3 and ECN4 respectively.

- VNs processing: There are 12 VN blocks in the decoder. The required data for every VN block are: the extrinsic messages V_i', the most reliable intrinsic GF value $I_i'^{\oplus}[0]$, the indexes that are generated by the LLR generator $\{\pi_i[0], \pi_i[1], \pi_i[2]\}$ and the absolute values of the observed bits $\{|y_{s_i,0}|, \ldots, |y_{s_i,5}|\}$, $i = 0, \ldots, 11$. In the following, the range of the index denoting the elements $V_i'[j]$, is $j \in \{0, \ldots, 19\}$, when $i = 0, \ldots, 9$, and $j \in \{0, \ldots, 15\}$, when $i = 10, 11$. The four steps of the VN processing are:
  - Regeneration of the four intrinsic candidates: the most reliable intrinsic GF value $I_i'^{\oplus}[0]$, the indexes $\{\pi_i[0], \pi_i[1], \pi_i[2]\}$ and the ab-

solute value of the observed bits $\{|y_{s_i,0}|,\ldots,|y_{s_i,5}|\}$ are used to regenerate the intrinsic candidates $\{I'_i[0],I'_i[1],I'_i[2],I'_i[3]\}$. These candidates are regenerated considering the architecture shown in Fig. 4.4. Then, the offset value O will be added to each intrinsic LLR value as: $I''^{+}_i[k]=I'^{+}_i[k]+\mathrm{O}$, $k=0,1,2,3$.

- Generation of the intrinsic LLR value of the extrinsic candidates: the intrinsic LLR value of each $V'^{\oplus}_i[j]$ is computed as:
  $I_{V'_i}[j]=\sum_{k=0}^{5}|y_{s_i,k}|.\Delta(V'^{\oplus}_i[j][k],I'^{\oplus}_i[0][k])$.
  Then, $I_{V'_i}[j]$ is added to $V'^{+}_i[j]$ as: $V''^{+}_i[j]=V'^{+}_i[j]+I_{V'_i}[j]$.
- Extraction of the five sorted symbols $\{V'^{s}_i[0],\ldots,V'^{s}_i[4]\}$ having the lowest LLR values among the set $\{V''_i[0],\ldots,V''_i[19],I''_i[0],\ldots,I''_i[3]\}$ where $V'^{s+}_i[0]\leq\cdots\leq V'^{s+}_i[4]$.
- Generation of the four most reliable symbols $\{U'_i[0],\ldots,U'_i[3]\}$ that have no redundant GF values. $V'^{s+}_i[k_1]$ is replaced by Sat value when $V'^{s\oplus}_i[k_0]=V'^{s\oplus}_i[k_1]$ where $k_1=0,\ldots,4$ and $k_0<k_1$. Then, the updated messages $\{U'_i[0],\ldots,U'_i[3]\}$ are detected among the set $\{V'^{s}_i[0],\ldots,V'^{s}_i[4]\}$ where $U'^{+}_i[0]\leq\cdots\leq U'^{+}_i[3]$.

- Normalizing and reordering the updated messages: the updated messages $\{\{U'_0\},\ldots,\{U'_{11}\}\}$ are normalized as: $U'^{+}_i[j]=U'^{+}_i[j]-U'^{+}_i[0]$, $i=0,\ldots,11$ and $j=1,2,3$. Then, it should be reordered to their original order based on Ψ^{-1} as: $U_i=U'_{\psi^{-1}[11-i]}$, $i=0,\ldots,11$.

3. Decision making: The required messages for this process are $\{U'_i[0],U'_i[1],U'_i[2]\}$, $\{V'_i[0],\ldots,V'_i[j]\}$ and the appropriate offset O, $i=0,\ldots,11$. Every $V'_i[j]$ is updated as:

$$V'^{u+}_i[j]=\begin{cases}V'^{+}_i[j]+U'^{+}_i[0] & \text{If } V'^{\oplus}_i[j]=U'^{\oplus}_i[0]\\ V'^{+}_i[j]+U'^{+}_i[1] & \text{If } V'^{\oplus}_i[j]=U'^{\oplus}_i[1]\\ V'^{+}_i[j]+U'^{+}_i[2]+\mathrm{O} & \text{Otherwise}\end{cases}$$

and

$$V'^{u\oplus}_i[j]=V'^{\oplus}_i[j].$$

and the symbol $U'_i[0]$ is updated as: $U'^{u+}_i=U'^{+}_i[0]+\mathrm{O}$ and $U'^{u\oplus}_i=U'^{\oplus}_i[0]$. We recall that the set $\{U'_0,\ldots,U'_9\}$, U'_{10} and U'_{11} have different offset O. Therefore, the decision is made as: $\mathrm{GF}'_i=\min(\{V'^{u}_i[0],\ldots,V'^{u}_i[j],U'^{u}_i\})$, where the min fumction detects the GF value having the lowest LLR value. Finally, the set of $\{\mathrm{GF}'_0,\ldots,\mathrm{GF}'_{11}\}$ is reordered by Ψ^{-1} to generate $\{\mathrm{GF}_0,\ldots,\mathrm{GF}_{11}\}$ as: $\mathrm{GF}_i=\mathrm{GF}'_{\psi^{-1}[11-i]}$, $i=0,\ldots,11$.

4. Stopping criteria: the decoder stops processing if all the $M=24$ equations in the PCM are satisfied, i.e, $\bigoplus_{i=0}^{11}h_i.\mathrm{GF}_i=0$. Remember that the set $\{\mathrm{GF}_0,\ldots,\mathrm{GF}_{11}\}$

and $\{h_0, \ldots, h_{11}\}$ are related to PCM. For instance, let CN_0 be the CN that is being processed, in this case $\{\mathrm{GF}_0, \ldots, \mathrm{GF}_{11}\}$ are the decisions on $\{\mathrm{VN}_0, \ldots, \mathrm{VN}_{132}\}$ and $\{h_0, \ldots, h_{11}\} = \{\beta_{43}, \ldots, \beta_5\}$ (see Fig. 5.2 and Fig. 5.3). On the other hand, the decoder stops processing when n_{it} iterations is reached regardless the satisfaction of the 24 equations.

5.1.3 Simulation results

As we mentioned in section 5.1.2, some bubbles from the HB(10,0,2) are eliminated and hence performance loss is obtained. To compensate this performance degradation, the global number of iterations within the decoder is increased. Fig. 5.8 shows the simulation results of the proposed decoder compared to the FB CN decoder considered as reference in our performance comparison. The FB CN-based decoder is simulated using two different scheduling schemes: flooding and layered. The proposed decoder, in its hardware version, is implemented using only the flooding schedule. This is due to the fact that by its nature, the new parallel decoder allows to start a new CN processing at each clock cycle which leads to reach the second layer of CNs without having completed the processing of the VNs being started in the first layer. Thus, the decoder should enter in an idle time waiting the availability of the required data. To avoid this idle time, we have decided to adopt the flooding schedule. This point will be discussed in more details in next sections. Different number of iterations 8, 15 and 30 are considered for the proposed decoder.
The ECNs implemented in the FB-CN are the S-bubble ECNs for $n_m = 20$ and $n_{op} = 25$. The performance is studied under AWGN channel and the LLR values are quantified over 6 bits.
Comparing the performance of the FB-CN layered schedule decoding with the proposed decoder for different n_{it} values, a performance loss of 0.4 dB is observed when $n_{it} = 8$ and goes down to 0.2 and 0.08 when $n_{it} = 15$ and 30 respectively.
When comparing to the FB CN-based decoder in flooding scheme, performance losses of 0.31 and 0.11 dB are introduced for $n_{it} = 8$ and 15 respectively. On the other hand, a gain of 0.05 dB is obtained when $n_{it} = 30$ is used.

Fig. 5.9 compares the performance of the proposed decoder with the B-LDPC code Offset Min-Sum (OMS) decoding algorithm of $N = 864$ bits, $K = 720$ bits and $\mathrm{CR} = 5/6$. There is more than 0.4 db gain in favor of the proposed decoder. Moreover, code alone is not a fair comparison, i.e, with high order modulation, NB-LDPC codes have some advantages compared to binary code. For instance, no need for iterative demodulation in case of NB-LDPC codes.

Fig. 5.10 shows the average number of iterations versus $\mathrm{E}_b/\mathrm{N}_0$. For the proposed decoder, the average number of iterations has a high discrepancy for low SNR ($\mathrm{E_b/N_0} \leq$ 3 dB) when $n_{it} = 10, 15$ and 30. This discrepancy dramatically decreases when $\mathrm{E_b/N_0} \leq 3.5$ dB and tends to 0 at 4 dB. However, this difference dramatically decreases starting from $\mathrm{E_b/N_0} > 3.5$ db until they meet on $\mathrm{E_b/N_0} = 4$ db. When

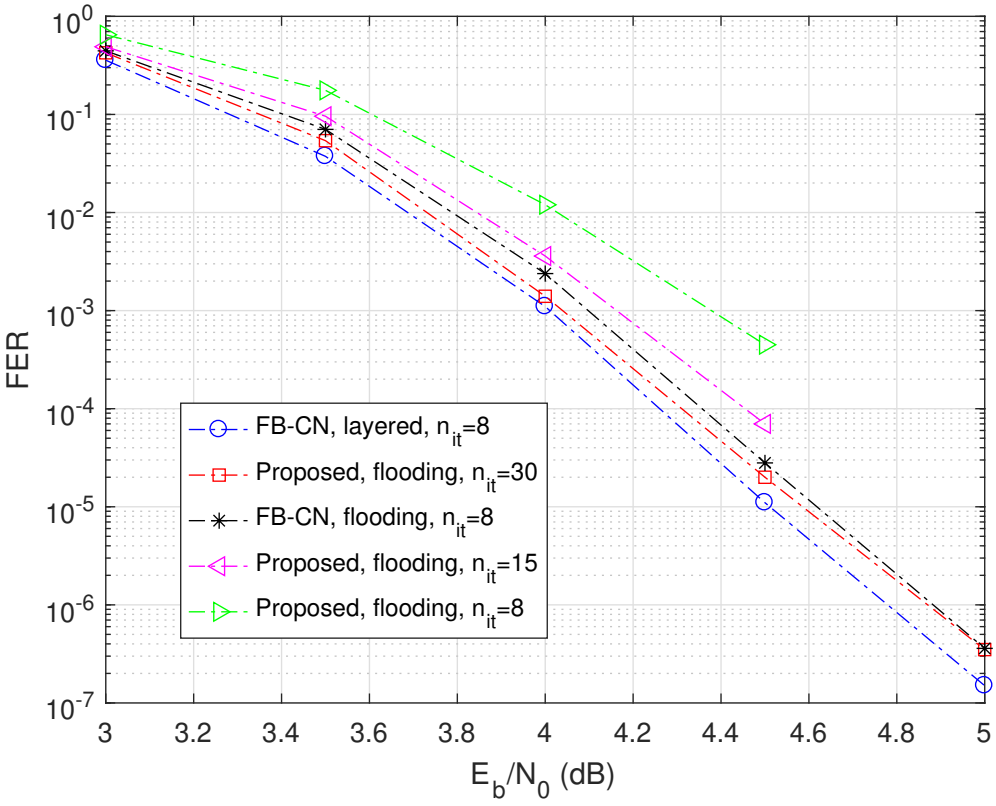

Figure 5.8: FER performance for a (144, 120) NB-LDPC code: Proposed decoder vs FB CN-based decoder.

comparing the proposed decoder $n_{it} = 30$ with the FB-CN flooding schedule and $n_{it} = 8$, the difference is high in favor of FB-CN for 3 db $\leq \mathrm{E_b/N_0} < 4$ db, while the difference is highly reduced for $\mathrm{E_b/N_0} \geq 4$ db until obtaining the same average number if iterations at $\mathrm{E_b/N_0} = 6.5$ db. On the other hand, when comparing the proposed decoder $n_{it} = 30$ with the FB-CN layered schedule and $n_{it} = 8$, the difference is high in favor of FB-CN when 3 db $\leq \mathrm{E_b/N_0} < 5$ db, to reach the same average at $\mathrm{E_b/N_0} = 6.5$ db. Thus, the higher the $\mathrm{E_b/N_0}$ the lower the difference in terms of the average number of iterations a_{it}. The value a_{it} affects the throughput that is computed as:

$$\text{Throughput (Gbits/s)} = \frac{\log_2(q) \times K \times F}{10^3 \times a_{it} \times M} \tag{5.4}$$

where $a_{it} \in \{1, 2, 3, \ldots, n_{it}\}$ is the average number of iterations and F = 650 MHz. We early gave the amount of the frequency to see how the throughput of the proposed decoder varies with $\mathrm{E_b/N_0}$.
Fig. 5.11 shows the throughput versus $\mathrm{E_b/N_0}$ for the proposed decoder in case of $n_{it} = 10, 15$ and 30. We can see the significant difference of the throughput when

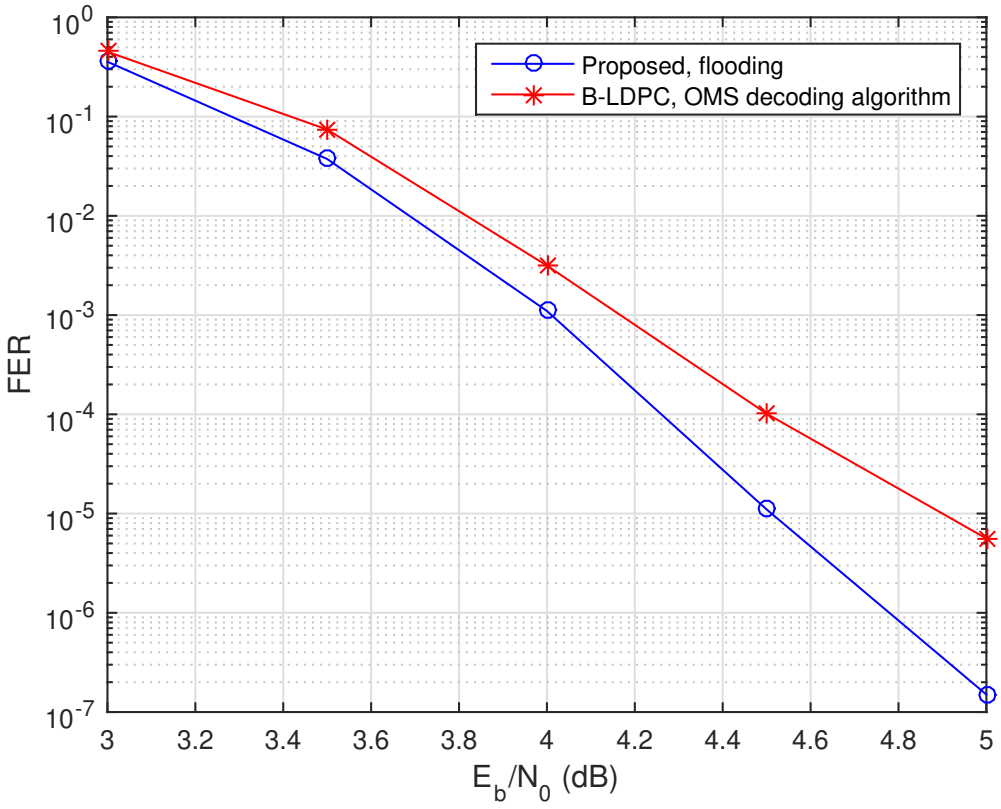

Figure 5.9: FER performance for a (144, 120) NB-LDPC code over GF(64) Proposed decoder vs (864, 720) B-LDPC code over GF(2) OMS decoder.

3 db $\leq \mathrm{E_b/N_0} \leq$ 3.7 db, where it is equal to 0.6 Gbits/s, 0.9 Gbits/s and 1.4 Gbits/s at $\mathrm{E_b/N_0} = 3$ db for $n_{it} = 8, 15$ and 30 respectively. However, when $\mathrm{E_b/N_0} > 3.7$ db, the difference is significantly reduced and tends to zero at $\mathrm{E_b/N_0} = 4$ db.

5.2 Architectural overview

Fig. 5.12 shows the global architecture of the proposed decoder. From left to right, the blocks that constitutes the decoder are:

1. *LLR Generator Block*: The proposed architecture shown in section 4.1.4 is the implemented architecture in this work. Since there are eight observed symbols entering the decoder in parallel, eight LLR Generators are needed. Every LLR generator generates $n_m = 4$ intrinsic couples along with the indexes set. In the proposed decoder, the permutation indexes $\{\pi[0], \pi[1], \pi[2]\}$ are needed to be output to help the regeneration of the intrinsic LLR later on during the decoding process as explained in section 5.1.2. According to the simulation results, setting $n_m = 4$ is a good choice to keep good performance at low hardware

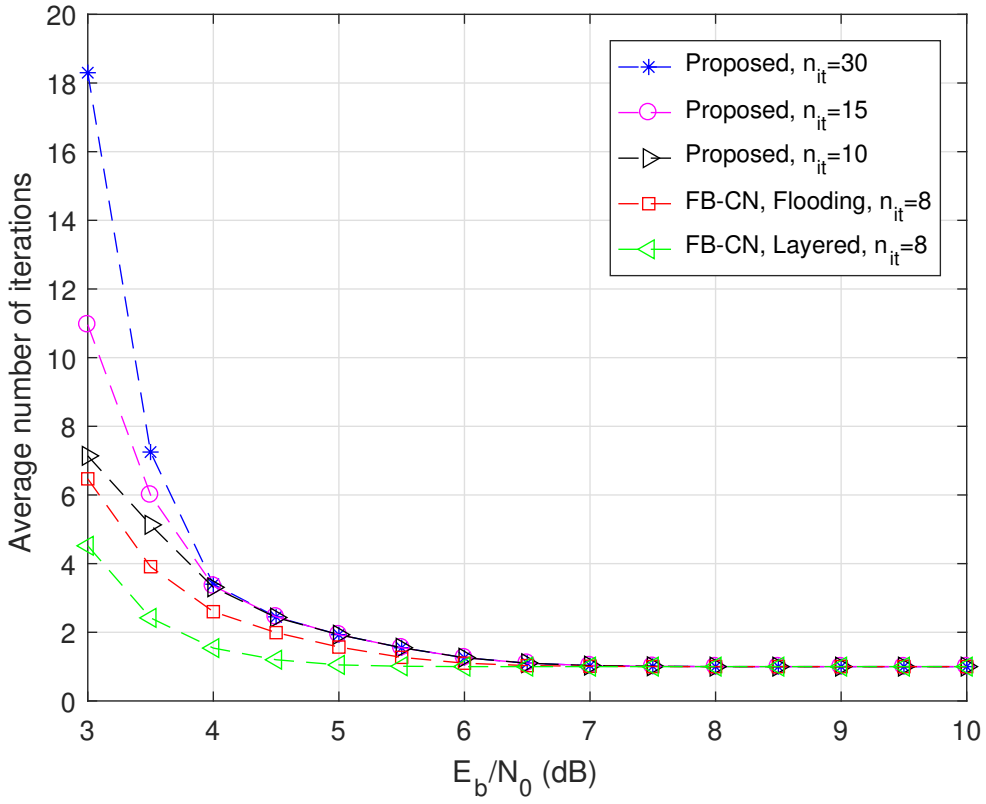

Figure 5.10: Average number of iterations versus E_b/N_0.

complexity. Thus, only four intrinsic LLR values along with their GF symbols and permutation indexes are generated.

2. *Intrinsic Router* block: The observed symbols are being received in order, starting from y_0 up to y_{143}. Thus, the intrinsic router is to send the outputs of the 8 LLR generator blocks to their appropriate locations in the RAM Banks. We purposely managed that eight symbols to be received in parallel $\{y_s, \ldots, y_{s+7}\}$, $s = 0, 8, 16 \ldots, 136$ (details will be shown in section 5.4).

3. *Control Unit* (CU): The CU block controls the read/write operations from/to the RAM Banks. The start signal indicates the arrival of the observed symbols and hence the control signal of the RAM Banks are generated based on a counter in the CU.

4. *RAM Banks*: The RAM Banks store the outputs of the 8 LLR generator blocks and the updated messages by the CN-VN block $U_n[j]$, $i = 0, \ldots, 11$, and $j = 0, \ldots, 3$. In addition, the RAM Banks provide the inputs to the CN-VN block.

5. *CN-VN* block: This block constitutes the core of the decoder that performs the CN and VN processing. It receives $4 \times d_c$ updated messages $\boldsymbol{U} = \{U_0, \ldots, U_{11}\}$

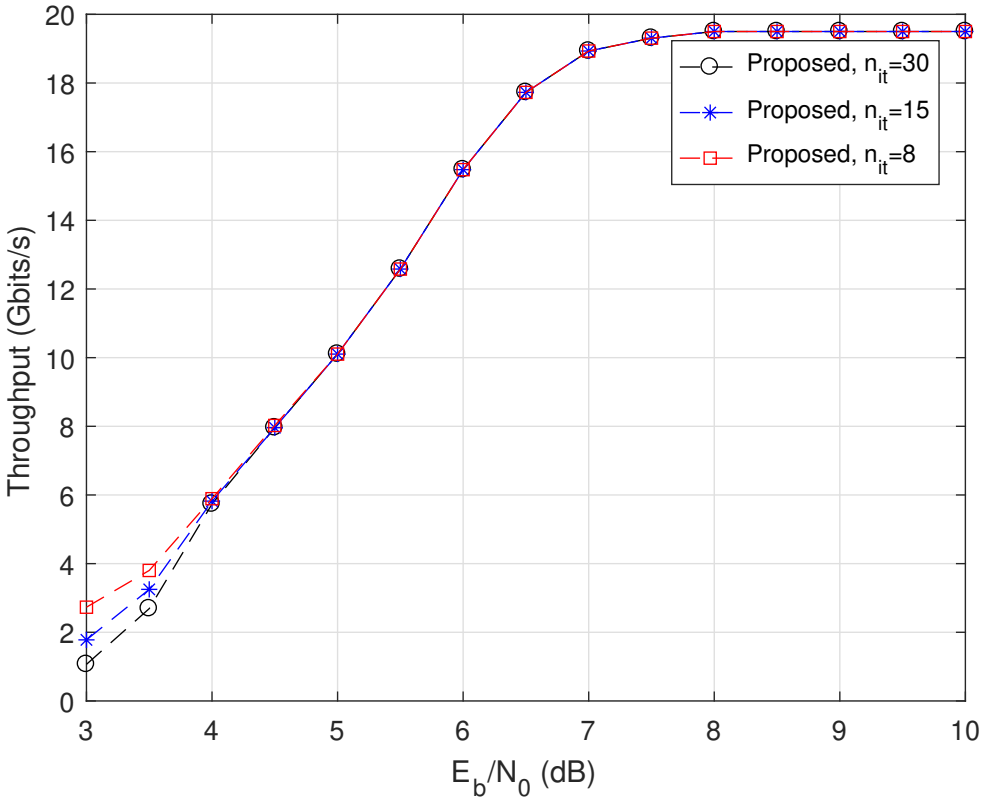

Figure 5.11: Throughput versus E_b/N_0.

where each U_i, $i = 0, \ldots, 11$, is a vector of length $n_m = 4$, and the most reliable GF symbol of each intrinsic vector $I_i^{\oplus}[0]$ along with the permutation indexes $\pi_i[k]$, $i = 0, \ldots, 11$ and $k = 0, 1, 2$. It receives the non-zero elements h_i of the PCM along with their GF inverse h_i^{-1}, $i = 0, \ldots, 11$. The absolute values of the channel bit observations $y_{s_i,j}$ are also needed to regenerate the $n_m = 4$ intrinsic candidates.
For instance, let CN_0 be the current CN to be performed. Thus, the set of VNs $\{\mathrm{VN}_0, \mathrm{VN}_{12}, \mathrm{VN}_{24}, \mathrm{VN}_{36}, \mathrm{VN}_{48}, \mathrm{VN}_{60}, \mathrm{VN}_{72}, \mathrm{VN}_{84}, \mathrm{VN}_{96}, \mathrm{VN}_{108}, \mathrm{VN}_{120}, \mathrm{VN}_{132}\}$ is being updated. Therefore, CN-VN reads from the RAM Banks:

- $\{\{U_0[0], U_0[1], U_0[2], U_0[3]\}, \ldots, \{U_{11}[0], U_{11}[1], U_{11}[2], U_{11}[3]\}\}$, which are associated to VN_0, $\mathrm{VN}_{12}, \ldots, \mathrm{VN}_{132}$.
- $\{I_0^{\oplus}[0], \ldots, I_{11}^{\oplus}[0]\}$, where $I_0^{\oplus}[0], \ldots, I_{11}^{\oplus}[0]$ are the HD associated to VN_0, $\ldots$, VN_{132} respectively.
- The permutation indexes generated by the LLR generators $\{\{\pi_0[0], \pi_0[1], \pi_0[2]\}, \ldots, \{\pi_{11}[0], \pi_{11}[1], \pi_{11}[2]\}\}$ associated to $\{\mathrm{VN}_0, \ldots, \mathrm{VN}_{132}\}$ respectively.

- $\{h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}, h_{11}\} = \{\beta_{43}, \beta_1, \beta_8, \beta_{31}, \beta_{26}, \beta_{47}, \beta_{56}, \beta_{39}, \beta_{51}, \beta_{58}, \beta_{30}, \beta_6\}$ along with their inverse $\{h_0^{-1}, \ldots, h_{11}^{-1}\}$ in the GF domain.
- The absolute values $|y_{s_i,j}|$, for $i = 0, \ldots, 11$, $s_i = 0, 12, \ldots, 132$ and $j = 0, \ldots, 5$.

6. *Decision Making Unit* (DMU): This unit is composed of $d_c = 12$ sub-units dedicated to make decision on all the VNs in order to determine which GF symbol each VN does represent. This block receives V' and U' messages from the CN-VN unit and generates the decided symbols GF$'$.

7. *DMU Reordering* (DMUR) block: due to the presorting at the input of the CN-VN, the outputs of the DMU are not ordered, so the DMUR reorders the outputs according to their original positions.

8. *GF Routing Block* (GFRB): This block is responsible of routing the decided GF symbols to their appropriate positions in the Register holding the 144 GF symbols. These symbols are received in sets of size equal to 12.

9. *Stopping Criteria Router Block* (SCRB): This block selects the GF symbols to be sent to the next block for parity check test. At each clock cycle, 24 GF symbols are read from the 144-Register.

10. *Parity Test Block* (PTB): this block performs the test of all the $M = 24$ parity check equations: $\bigoplus_{i=0}^{11} h_i.\mathrm{GF}_i = 0$. Once the $M = 24$ equations are satisfied, the decoder stops the decoding process of the current frame and starts a new frame. Otherwise, the decoder continues till reaching the maximum allowed number of iterations n_{it}.

The next sections describe in details each component of the decoder.

5.2.1 Memorization system

The RAM Banks shown in Fig. 5.2 consists of: 1) the intrinsic RAM dedicated to store the absolute value of the observed bits and the intrinsic information $\{I[0], \ldots, I[3]\}$ and $\{\pi[0], \pi[1], \pi[2]\}$ generated by the LLR generators; 2) the extrinsic RAMs, where the updated messages $\{U_{n_i}[0], U_{n_i}[1], U_{n_i}[2], U_{n_i}[3]\}$, $i = 0, \ldots, 11$ are saved; 3) The ROM to store the non-zero elements of the PCM and their GF inverse. The structures of the RAM banks are described in the following.

1. Intrinsic RAMs: Fig. 5.13 shows the structure of the intrinsic RAM Banks organized into 12 separate banks denoted by R_i, $i = 0, 1, \ldots, 11$. Each bank R_i is a 12×45 array storing the absolute values of the channel observations y_{s_i} of 12 VN messages each containing an LLR value. Each y_{s_i} is composed of 6 components $y_{s_i,j}$, $j = 0, 1, \ldots, 5$, represented each on 5 bits. In addition, the hard decision

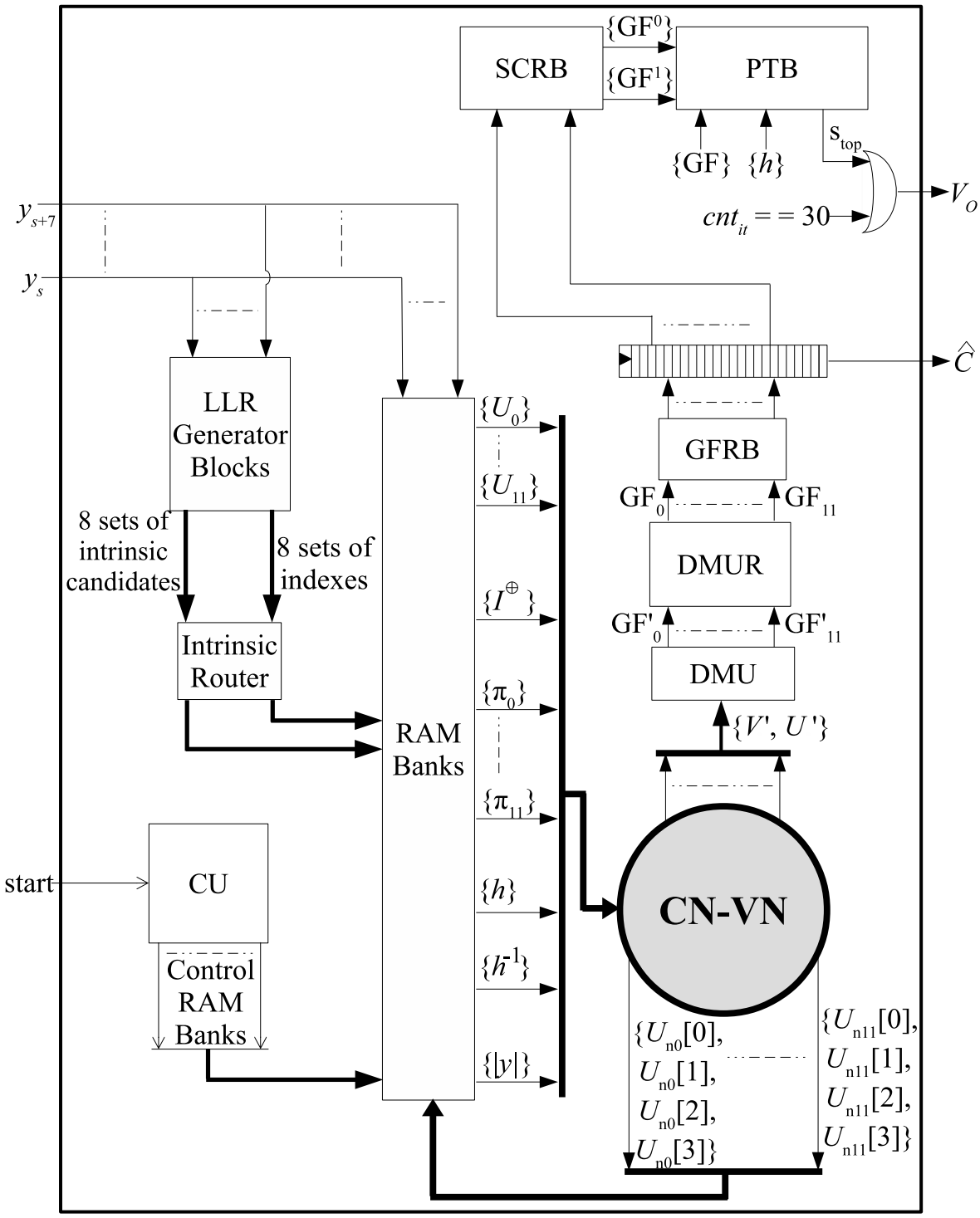

Figure 5.12: Global architecture of the decoder.

$I^{\oplus}[0]$ along with the indexes of the first 3 minimum values $\{\pi[0], \pi[1], \pi[2]\}$ of the sorted y_{s_i} are stored. This information will be exploited later on in the decoding

process. Thus, the required information are concatenated to be stored in each cell as: ($I^{\oplus}[0]$ & $\pi[0]$ & $\pi[1]$ & $\pi[2]$ & $|y_{s_i,0}|$ & $|y_{s_i,1}|$& $|y_{s_i,2}|$& $|y_{s_i,3}|$ & $|y_{s_i,4}|$ & $|y_{s_i,5}|$), where & represents the concatenation operation. The cumulative length of the vector is equal to $6+3+3+3+6\times 5=45$.

R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
0	12	24	36	48	60	72	84	96	108	120	132
1	13	25	37	49	61	73	85	97	109	121	133
2	14	26	38	50	62	74	86	98	110	122	134
3	15	27	39	51	63	75	87	99	111	123	135
4	16	28	40	52	64	76	88	100	112	124	136
5	17	29	41	53	65	77	89	101	113	125	137
6	18	30	42	54	66	78	90	102	114	126	138
7	19	31	43	55	67	79	91	103	115	127	139
8	20	32	44	56	68	80	92	104	116	128	140
9	21	33	45	57	69	81	93	105	117	129	141
10	22	34	46	58	70	82	94	106	118	130	142
11	23	35	47	59	71	83	95	107	119	131	143

Figure 5.13: 12 intrinsic RAMs.

2. Extrinsic RAMs: The 24 extrinsic RAMs denoted by R_iL_j are shown in Fig. 5.14, where R for RAM and L for Layer, $i=0...11$ and $j=1,2$. These RAMs are represented as columns in Fig. 5.14. The extrinsic RAMs store the updated messages $\{U_{n_i}[0],U_{n_i}[1],U_{n_i}[2],U_{n_i}[3]\}$. When performing the CN-VN processing, the data of the VNs associated to L_1 are read from L_2 and the results are stored in L_1, and vice versa. For instance, let CN_0 be the CN that is being processed by CN-VN, i.e, the VNs that are being read are $\{VN_0,\ldots,VN_{132}\}$. The intrinsic information are read from the RAMs in Fig. 5.13 as $\{R_0[0]\ ,\ldots,\ R_{11}[0]\}$. The input messages are read from the RAMs L_2 shown in Fig. 5.14, where VN_0 is associated to the first element in R_0L_2, VN_{12} is associated to the last element in R_1L_2, VN_{24} is associated to the 11^{th} element in R_2L_2 and so on. Then, each output $\{U_{n_i}[0],U_{n_i}[1],U_{n_i}[2],U_{n_i}[3]\}$, $i=0,\ldots,11$, is saved in its appropriate position in R_iL_1, where $\{U_{n_0}[0],U_{n_0}[1],U_{n_0}[2],U_{n_0}[3]\},\ldots,\{U_{n_{11}}[0],U_{n_{11}}[1],U_{n_{11}}[2],U_{n_{11}}[3]\}$ are the updated messages associated to $VN_0,\ldots,VN_{132}$.

   The depth of each cell of the extrinsic RAMs is equal to 42 bits. There are 4 GF values each of 6 bits and 3 LLR values each of 6 bits, considering that the most reliable GF value of LLR equal to 0 ($U^{+}_{n_i}[0]=0$). Thus, every cell

R_0L_1	R_1L_1	R_2L_1	R_3L_1	R_4L_1	R_5L_1	R_6L_1	R_7L_1	R_8L_1	R_9L_1	$R_{10}L_1$	$R_{11}L_1$
0	12	24	36	48	60	72	84	96	108	120	132
1	13	25	37	49	61	73	85	97	109	121	133
2	14	26	38	50	62	74	86	98	110	122	134
3	15	27	39	51	63	75	87	99	111	123	135
4	16	28	40	52	64	76	88	100	112	124	136
5	17	29	41	53	65	77	89	101	113	125	137
6	18	30	42	54	66	78	90	102	114	126	138
7	19	31	43	55	67	79	91	103	115	127	139
8	20	32	44	56	68	80	92	104	116	128	140
9	21	33	45	57	69	81	93	105	117	129	141
10	22	34	46	58	70	82	94	106	118	130	142
11	23	35	47	59	71	83	95	107	119	131	143

R_0L_2	R_1L_2	R_2L_2	R_3L_2	R_4L_2	R_5L_2	R_6L_2	R_7L_2	R_8L_2	R_9L_2	$R_{10}L_2$	$R_{11}L_2$
0	13	26	39	52	65	78	91	104	117	130	143
1	14	27	40	53	66	79	92	105	118	131	132
2	15	28	41	54	67	80	93	106	119	120	133
3	16	29	42	55	68	81	94	107	108	121	134
4	17	30	43	56	69	82	95	96	109	122	135
5	18	31	44	57	70	83	84	97	110	123	136
6	19	32	45	58	71	72	85	98	111	124	137
7	20	33	46	59	60	73	86	99	112	125	138
8	21	34	47	48	61	74	87	100	113	126	139
9	22	35	36	49	62	75	88	101	114	127	140
10	23	24	37	50	63	76	89	102	115	128	141
11	12	25	38	51	64	77	90	103	116	129	142

Figure 5.14: Extrinsic RAMs.

receives ($U_{n_i}^{\oplus}[0]$ & $U_{n_i}^{\oplus}[1]$ & $U_{n_i}^{\oplus}[2]$ & $U_{n_i}^{\oplus}[3]$ & $U_{n_i}^{+}[1]$ & $U_{n_i}^{+}[2]$ & $U_{n_i}^{+}[3]$). The extrinsic RAMs are initialized by the intrinsic messages at the beginning of the decoding process.

3. ROM: The non-zero elements of the PCM and its inverse are stored in ROM block. Fig. 5.15 shows the 24 cells (rows) of the ROM block. Each cell is of size equal to $6 \times 24 = 144$ bits since every non-zero GF value h_i consists of 6 bits and so its inverse h_i^{-1}, $i = 0, \ldots, 11$. For instance, ROM[0] is related to the non-zero elements of CN_0, so it contains the GF values $\{\beta_{43}, \beta_0, \beta_7, \beta_{30}, \beta_{25}, \beta_{46}, \beta_{55}, \beta_{38}, \beta_{50}, \beta_{57}, \beta_{29}, \beta_5, \beta_{43}^{-1}, \beta_0^{-1}, \beta_7^{-1}, \beta_{30}^{-1}, \beta_{25}^{-1}, \beta_{46}^{-1}, \beta_{55}^{-1}, \beta_{38}^{-1}, \beta_{50}^{-1}, \beta_{57}^{-1}, \beta_{29}^{-1}, \beta_5^{-1}\}$.

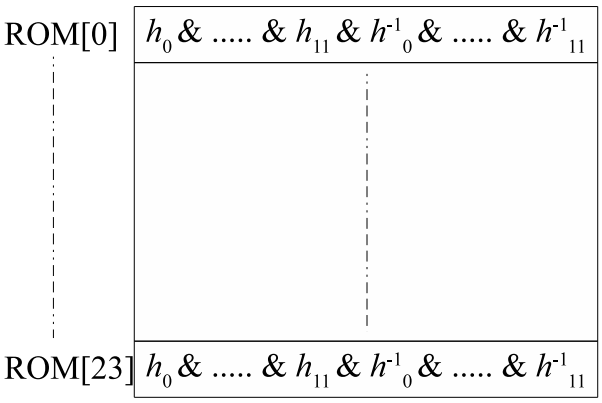

Figure 5.15: ROM block.

5.2.2 Timing diagram of the overall decoder

The timing diagram of the overall decoder is shown in Fig. 5.16. We aim to show the launch phase of the decoder and how the data are being read from the RAMs.
From left to right, the $N = 144$ observed symbols are being received in parallel as 8 symbols per Clock Cycle (CC) $\{y_s, \ldots, y_{s+7}\}$, $s = 0, 8, 16, \ldots, 136$, i.e, the $N = 144$ observed symbols are received after $\frac{144}{8} = 18$ CCs. The reason of receiving eight symbols per CC is explained in section 5.4. However, after 17 CCs the set of VNs that are connected to CN_0 is completed, since VN_{132} is the last connected VN to CN_0. The LLR Generator Blocks start processing immediately after receiving the first set of y_s, it takes 2 CCs latency to start generating outputs and hence the total execution time of this unit is equal to $18 + L_{(LLRG)} = 18 + 2 = 20$ CCs where $L_{(LLRG)}$ is the latency of the LLR generator. The RAM Banks start storing the outputs of the 8 LLR Generator blocks once they start to be output.
After $17 + L_{(LLRG)} = 19$ CCs, the required data of the VNs that are associated to CN_0 are ready to be sent to the CN-VN, and hence the CN-VN keeps reading from the RAM Banks for $M = 24$ CCs. As shown in Fig. 5.16, the VNs that are connected to L1 are updated first where their required data are being read from the RAMs associated to L2, and their outputs start to be saved in the RAMs associated to L1 after 16 CCs. Once all the 12 CNs of L1 have started their update, i.e, after 12 CCs, the first CN of L2 (CN_{12}) starts its update immediately at the next cycle, where their required data are being read from the RAMs associated to L1, and their outputs are being saved in the RAMs associated to L2. The intrinsic information and the non-zero

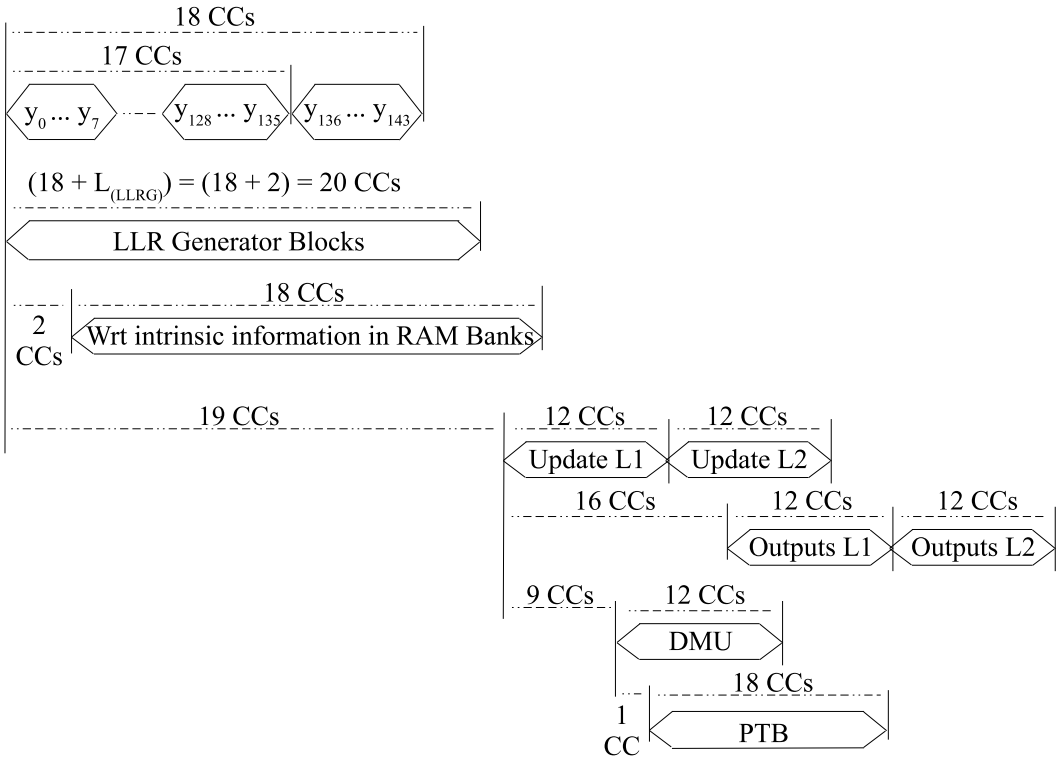

Figure 5.16: Timing diagram of the overall decoder.

elements are being read from the intrinsic RAMs and the ROM block respectively. The 16 CCs latency of the CN-VN block are detailed later.
The DMU block starts making decision after 7 CCs from the beginning of the CN-VN processing (when its required inputs are ready), the decision is being made considering L_1 so it takes 12 CCs to make decision for all the $N = 144$ VNs.
Finally, after 1 CC latency from the DMU block, the PTB block starts checking the validity of the parity check equations. It takes 18 CCs to decide if the decoder can stop the processing of the current frame, i.e, to indicate if the 24 equations are satisfied. In fact, PTB block is composed of two sub-modules to check each equation in PCM. Since the decisions are being made by set of d_c GF decisions per clock cycle, the PTB can check the first 12 CNs (L_1) on the fly using one sub-unit. Then when the 144 decisions are made, the two sub-units operate in parallel to check the validity of the remaining 12 CNs (L_2) and hence the total latency is equal to $12 + \frac{12}{2} = 18$ CCs. Details on this point are explained later
Note that the 19 CCs latency of the launch phase is considered only once at the first iteration of the first codeword. Then the CN-VN processing will be dominating the global throughput of the decoding process. Thus, the number of CCs needed to decode one frame is equal to $24 \times a_{it}$, where the average number of iterations $a_{it} \in \{1, 2, \ldots, n_{it}\}$. Details are shown in section 5.4.

5.3 Decoder components architecture

In this section, we detail the different components of the decoder: the CN-VN, DMU, DMUR and PTB blocks.

5.3.1 The CN-VN block

The CN-VN architecture is shown in Fig. 5.17.

The presorting block receives the set of LLR values $\{U_0^+[1], \ldots, U_{11}^+[1]\}$ to generate the indexes permutation vector $\Psi = \{\psi[0], \ldots, \psi[11]\}$. Depending on Ψ, the Switching + Multiplication (SM) block switches the inputs $\boldsymbol{U} = \{U_0, \ldots, U_{11}\}$, the intrinsic indexes permutation $\Pi = \{\pi_i[0], \pi_i[1], \pi_i[2]\}$, $i = 0, \ldots, 11$, the absolute value of the observed symbols $\{|y_{s_i,0}|, \ldots, |y_{s_i,5}|\}$ associated to the current VNs, the non-zero elements $\{h_0, \ldots, h_{11}\}$ and $\{h_0^{-1}, \ldots, h_{11}^{-1}\}$. In the following, the permuted data is appended by the prime symbol $'$.

Afterward, $\{U'_0, \ldots, U'_9\}$ are combined by the SN block to generate the outputs $U_{\mathrm{SN}}[0]$, $\ldots, U_{\mathrm{SN[6]}}$. The inputs of the SN block are split up into four sets: 1) the set of vectors that have 1 symbol $\{U'_0, U'_1, U'_2, U'_3\} = \{U'_0[0], U'_1[0], U'_2[0], U'_3[0]\}$; 2) the set of vectors that have 2 symbols $\{U'_4, U'_5, U'_6, U'_7\} = \{\{U'_4[0], U'_4[1]\}, \{U'_5[0], U'_5[1], \{U'_6[0], U'_6[1], \{U'_7[0], U'_7[1]\}\}\}\}$; 3) the vector that has 3 symbols $\{U'_8\} = \{U'_8[0], U'_8[1], U'_8[2]\}$ and 4) the vector that has 4 symbols $\{U'_9\} = \{U'_9[0], U'_9[1], U'_9[2], U'_9[3]\}$.

In parallel with SN, U'_{10} and U'_{11} are processed by ECN1 where 4 symbols are considered from each vector. The outputs are $U_{\mathrm{ECN1}}[0], \ldots, U_{\mathrm{ECN1}}[5]$.

Then, ECN2 receives U_{SN} and U_{ECN1} to generate the 20 syndrome couples contained in U_{ECN2}. ECN3 and ECN4 operate in parallel with ECN2. ECN3 processes U_{SN}, U'_{11} and performs the GF inverse permutation using h'^{-1}_{11} to generate V'_{10} of 16 couples. The same does ECN4 for U_{SN}, U'_{10} and h'^{-1}_{10} to generate V'_{11} of 16 couples.

The set $\{h'^{-1}_0, \ldots, h'^{-1}_9\}$ and U_{ECN2} are inputs to the Decorrelation + division block (DeBl). This block is to decorrelate each $U^{\oplus}_{\mathrm{ECN2}}[i]$, $i = 0, \ldots, 19$, from $U'^{\oplus}_j[0]$, $j = 0, \ldots, 9$ by making GF addition between them. Then, it makes the inverse GF permutation by multiplying each decorrelated result by its associated h'^{-1}_j. In more details, $V'^{\oplus}_j[i] = (U^{\oplus}_{\mathrm{ECN2}}[i] \bigoplus U'^{\oplus}_j[0]).h'^{-1}_j$.

The VN update is processed next, where the sets $\{\{\pi'_0[0], \pi'_0[1], \pi'_0[2]\}, \ldots, \{\pi'_{11}[0], \pi'_{11}[1], \pi'_{11}[2]\}\}$, $\{|y'_{s_i,0}|, \ldots, |y'_{s_i,5}|\}$, $i = 0, \ldots, 5$ and $\{V'_0, \ldots, V'_{11}\}$ are entered to their associated VN blocks to generate the unordered updated messages $\{U'_{n_0}, \ldots, U'_{n_{11}}\}$.

Finally, the set of unordered updated messages $\{U'_{n_0}, \ldots, U'_{n_{11}}\}$ is normalized and ordered by the Normalization and Reordering (NR). Each $U'_i[j]$, $j = 1, 2, 3$, is normal-

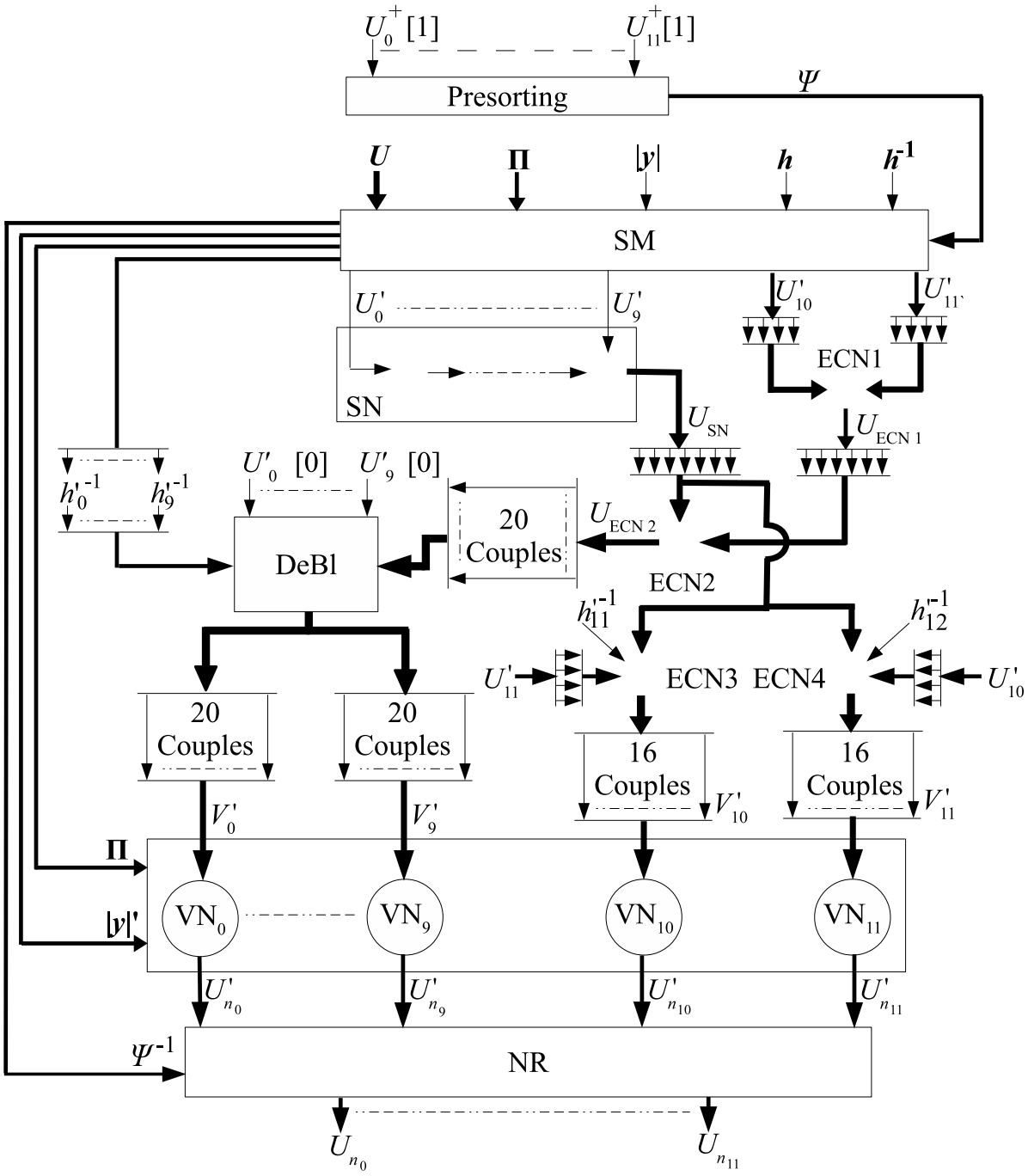

Figure 5.17: Architecture of the CN-VN unit.

ized by subtracting $U_i'[0]$, i.e, $U_i'^{+}[j] = U_i'^{+}[j] - U_i'^{+}[0]$. Then the set $\{U'_{n_0}, \ldots, U'_{n_{11}}\}$ is reordered using a set of MUXs controlled by Ψ^{-1} (the inverse of Ψ) to generate the ordered updated messages $\{U_{n_0}, \ldots, U_{n_{11}}\}$.

Before diving into details for each block in CN-VN, we should highlight the two reasons of making the VNs update before the reordering process. As V'_{10} and V'_{11} are generated independently, they do not have the same size of V'_i, $i = 0, \ldots, 9$, and hence VN_{10} and VN_{11} can be specified to process 16 symbols instead of 20 symbols. Besides that, if the reordering block were before the VNs update, it would have to reorder the set $\{V'_0, \ldots, V'_{11}\}$, i.e, the length of each input set would be 20 symbols instead of 4 symbols (the length of each U'_{n_i}, $i = 0, \ldots, 11$).

5.3.1.1 Presorting architecture

The presorting, based on the odd-even sorter architecture [55], is presented in Fig. 5.18. The index vector $\Psi = \{\psi[0], \ldots, \psi[11]\}$ is obtained based on the sorting of $U_i^+[1]$, $i = 0, \ldots, 11$, and contains the set of switching indexes that permits to obtain the sorted list $\{U_i^{p+}, \ldots, U_i^{p+}\}$, where $U_i^{p+}[1] < U_j^{p+}[1]$, for $i < j$ as will be shown next. The sorter architecture is based on a network of comparator-swaps, where each $\psi[i]$ represents the position of $(i+1)^{th}$ minimum.

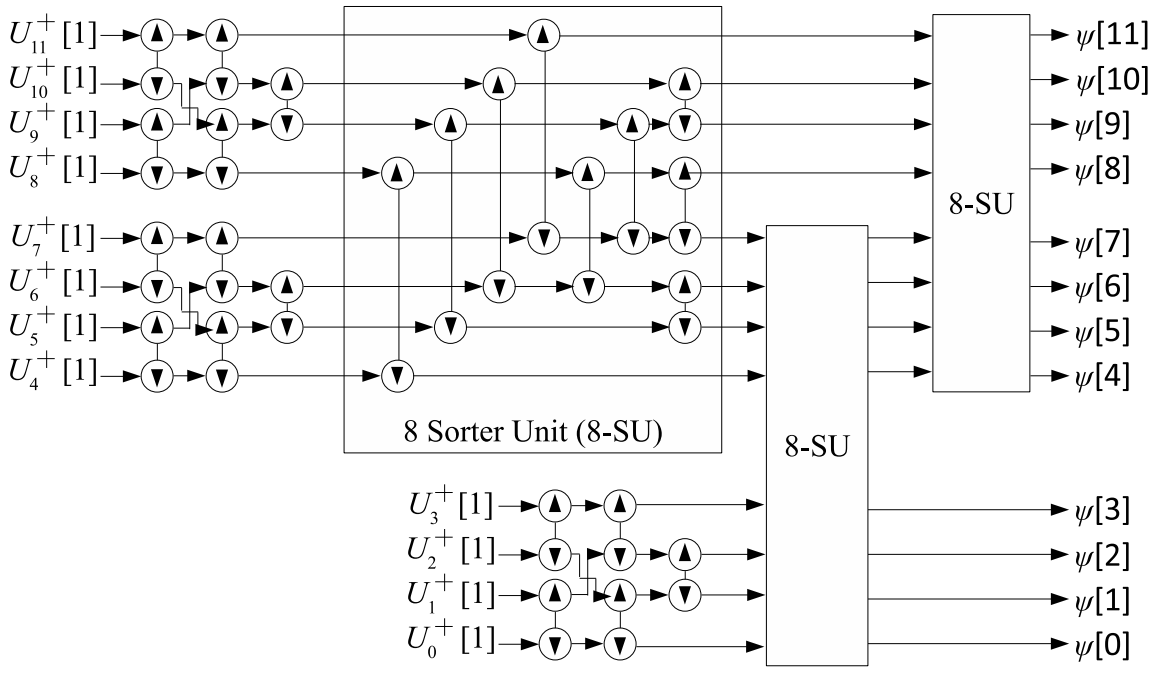

Figure 5.18: Architecture of the presorting block.

5.3.1.2 Switching + Multiplication

The architectures of the switching part and the multiplication part of the SM block are shown in Fig. 5.19 and Fig. 5.20 respectively. Starting with the switching operation, the set $\boldsymbol{U} = \{U_0, \ldots, U_{11}\}$ are switched by 12-to-1 MUXs controlled by

$\{\psi[0],\ldots,\psi[11]\}$. The switched U_i are called U_i^p, $i=0,\ldots,11$. The set $\boldsymbol{U^p} = \{U_0^p,\ldots,U_{11}^p\}$ are split up into four groups depending on the number of the required bubbles from each U_i^p. Recalling SN block in Fig. 5.4, we can say that $U_i^p[0]$ is required for all $i=0,\ldots,11$, while $U_i^p[1]$ is required for $i=4,\ldots,11$ and $U_i^p[2]$ and $U_i^p[3]$ are required for $i=8,\ldots,11$ and $i=9,\ldots,11$ respectively. Thus, the total number of 12-to-1 MUXs to generate the set $\boldsymbol{U^p}$ are 27 MUXs.
The sets $\{\{\pi_0\},\ldots,\{\pi_{11}\}\}$, $\{\{|y_{s_0}|\},\ldots,\{|y_{s_{11}}|\}\}$, $\{h_0,\ldots,h_{11}\}$ and $\{h_0^{-1},\ldots,h_{11}^{-1}\}$ are switched by sets of 12-to-1 MUXs controlled by Ψ. Where each $\{\pi_i\}=\{\pi_i[0],\pi_i[1],\pi_i[2]\}$ and $\{|y_{s_i}|\}=\{|y_{s_i,0}|,|y_{s_i,1}|,|y_{s_i,2}|,|y_{s_i,3}|,|y_{s_i,4}|,|y_{s_i,5}|\}$, $i=0,\ldots,11$.

After switching the input set $\boldsymbol{U}$, 21 bubbles are dismissed (pointed bubbles) among 48 bubbles as shown in Fig. 5.20 in the Multiplication block. The elimination of these bubbles leads to high reduction in terms of complexity of the CN-VN block even though there is an extra hardware for the presorting, switching and reordering blocks.
The GF multiplication is processed after the switching operation to reduce the number of GF multipliers. In more details, performing the GF multiplication before the switching operation requires $d_c \times n_m = 12\times 4 = 48$ GF multipliers, while performing it after the switching operation reduces the number of GF multipliers from 48 down to 27. However, the set $\{h_0,\ldots,h_{11}\}$ has to be switched, that requires 12 MUXs 12-to-1 controlled by Ψ, but in total, the overall solution with less number of GF multipliers is better. Finally, the permuted set $\boldsymbol{U'}=\{U'_0,\ldots,U'_{11}\}$ is generated as: $U'_i = U_i^p.h'_i$, $i=0,\ldots,11$.

5.3.1.3 Syndrome Node (SN)

Fig. 5.4 shows the shape of the ECNs, SN1 , ..., SN9, in the SN block. The number of considered bubbles is significantly reduced thanks to the presorting technique. Since all the shown bubbles in the ECNs obtained by combining $U_i'^{+}[j]$ with $U_i'^{+}[0]=0$, $i=0,\ldots,9$ and $0\geq j\leq 3$, there is no need for LLR additions. Thus, the required output symbols are computed using only GF additions, i.e, simple XOR gates.
Fig. 5.21 recall the notations of the merged $\{\mathrm{SN}_1,\ldots,\mathrm{SN}_8\}$ ECNs. The bubbles that belong to the first column, $\mathrm{b}_{c0,8},\ldots,\mathrm{b}_{c4,8}$, have been computed wisely by exploiting the sorted symbols in terms of LLR value, $U_8'^{+}[1]\leq U_7'^{+}[1]\leq\ldots\leq U_4'^{+}[1]$. In more details, $\mathrm{b}^{+}_{c0,8}=0$, $\mathrm{b}^{+}_{c1,8}=U_7'^{+}[1]$, $\mathrm{b}^{+}_{c2,8}=U_6'^{+}[1]$, $\mathrm{b}^{+}_{c3,8}=U_5'^{+}[1]$, $\mathrm{b}^{+}_{c4,8}=U_4'^{+}[1]$, $\mathrm{b}^{+}_{r1,8}=U_8'^{+}[1]$ and $\mathrm{b}^{+}_{r2,8}=U_8'^{+}[2]$. As for its GF values, $\mathrm{b}^{\oplus}_{c0,8}$, $\mathrm{b}^{\oplus}_{c1,8}$, $\mathrm{b}^{\oplus}_{c2,8}$, $\mathrm{b}^{\oplus}_{c3,8}$, $\mathrm{b}^{\oplus}_{c4,8}$, $\mathrm{b}^{\oplus}_{r1,8}$ and $\mathrm{b}^{\oplus}_{r2,8}$, let us take the following addition as a reference:

$$U_0'^{\oplus}[0]\oplus U_1'^{\oplus}[0]\oplus U_2'^{\oplus}[0]\oplus U_3'^{\oplus}[0]\oplus U_4'^{\oplus}[0]\oplus U_5'^{\oplus}[0]\oplus U_6'^{\oplus}[0]\oplus U_7'^{\oplus}[0]\oplus U_8'^{\oplus}[0]. \quad (5.5)$$

In equation 5.5, the most GF reliable symbols from U_0 to U_8 are added giving the GF value of $\mathrm{b}_{c0,8}$. The GF value of the other bubbles is straightforward to compute,

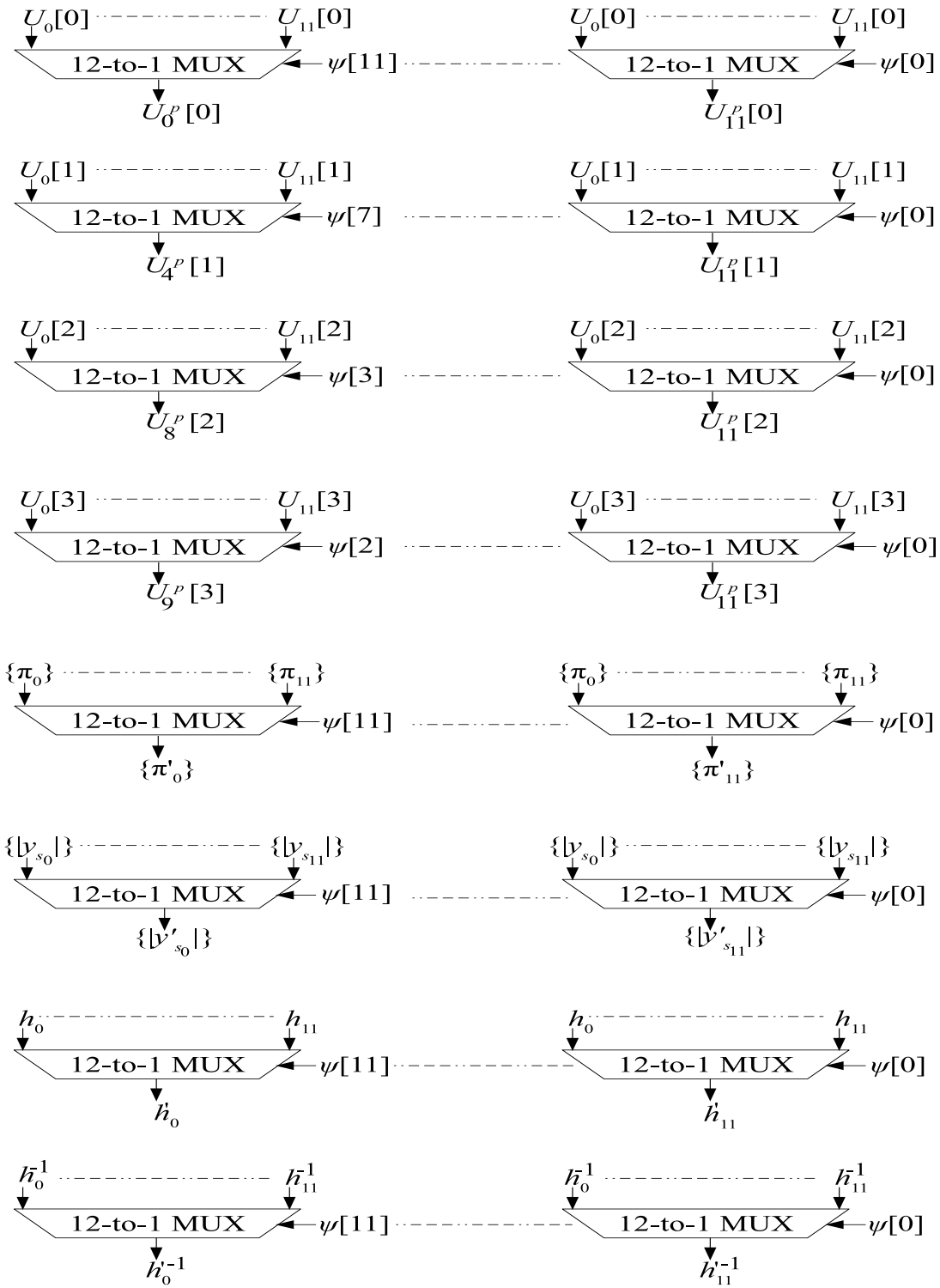

Figure 5.19: The switching part architecture.

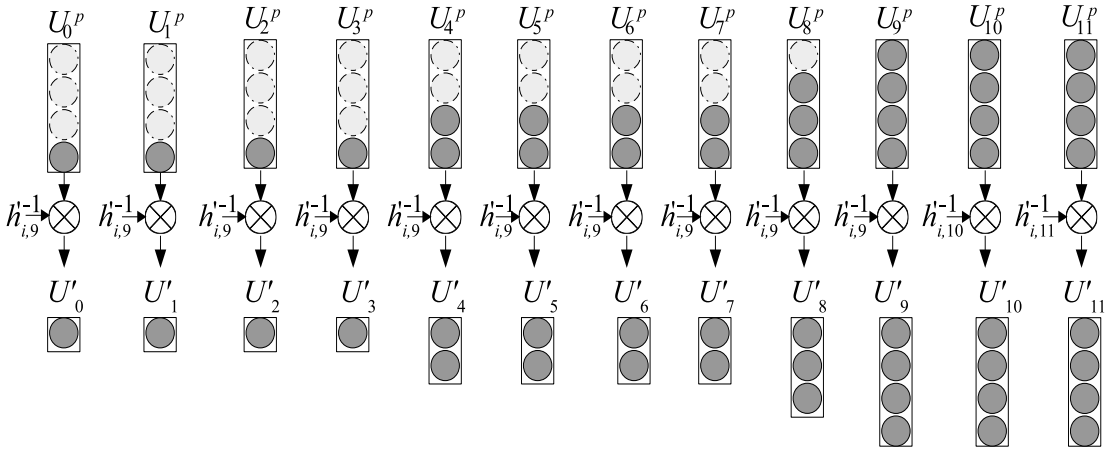

Figure 5.20: The multiplication part architecture.

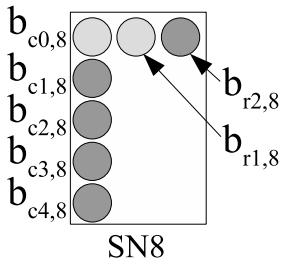

Figure 5.21: The bubbles of SN8.

merely replace $U_i^{'}\oplus[0]$ by $U_i^{'}\oplus[j]$ whenever the LLR of the bubble is equal to $U_i^{'+}[j]$ ($i = 4...8$ and $j = 1, 2$). The GF addition, the mapping of the LLR values and the generation of the VSV information are being processed in the *bubbles generator* block shown in Fig. 5.22.

The architecture of the merged SNi ($i = 1...8$) blocks is shown in Fig. 5.22. It operates as follows: the first output is $b_{0,9} = b_{c0,8}$, the second output is $b_{0,9} = b_{r1,8}$ while the comparators and multiplexers are to select the remaining outputs among the rest of bubbles as:

$$C_{i\mathrm{SN8}} = \begin{cases} 0 & \text{if } b^{+}_{r2,8} \leq b^{+}_{ci,8} \\ 1 & \text{Otherwise} \end{cases} \qquad i = 1, 2, 3, 4$$

$$b_{j+3,9} = \begin{cases} b_{c_{j+1},8} & \text{if } C_{j\mathrm{SN8}} = 0 \\ b_{r2,8} & \text{else if } C_{(j+1)\mathrm{SN8}} = 0 \\ b_{c_{j+2},8} & \text{Otherwise} \end{cases} \qquad j = 0, 1, 2$$

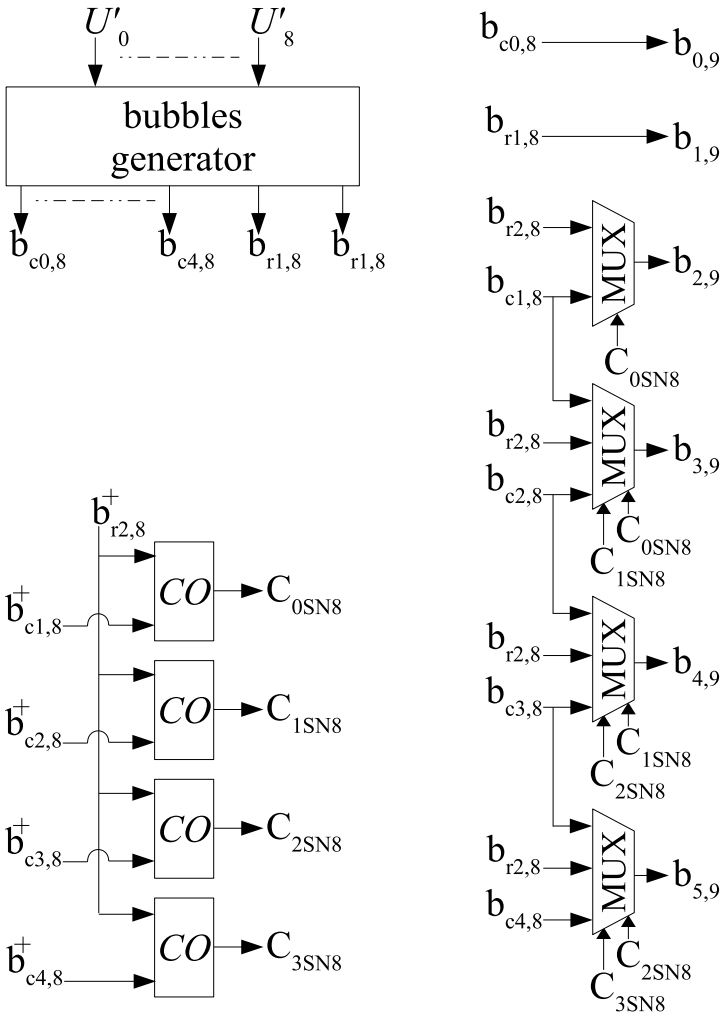

Figure 5.22: Architecture of the merged SN1 to SN8.

The bubbles $\{b_{0,9}, \ldots, b_{5,9}\}$ are fed to SN9. Fig. 5.23 recalls the notations of the bubbles of the SN9 ECN. In which, $b^{+}_{ci} = b^{+}_{i,9}$, $b^{\oplus}_{ci} = b^{\oplus}_{i,9} \oplus U'^{\oplus}_{9}[0]$, $i = 0, \ldots, 5$ and $b^{+}_{rj} = U'^{+}_{9}[j]$, $b^{\oplus}_{rj} = b^{\oplus}_{0,9} \oplus U'^{\oplus}_{9}[j]$, $j = 1, 2, 3$. The VSV vector of these bubbles are generated as explained in section 5.1.2.
Seven output symbols $\{U_{\mathrm{SN}}[0], \ldots, U_{\mathrm{SN}}[6]\}$ are generated among nine bubbles $\{b_{c0}, b_{c1}, b_{c2}, b_{c3}, b_{c4}, b_{c5}, b_{r1}, b_{r2}, b_{r3}\}$.

The architecture of SN9 ECN is shown in Fig. 5.24. It operates similarly to the SN8 ECN, where the processing is split up into two parts. Part 1 generates the bubbles and detects the signals $\{O_0, \ldots, O_4\}$ that have the lowest LLR values among the bubbles $\{b_{c1,9}, b_{c2,9}, b_{c3,9}, b_{c4,9}, b_{c5,9}, b_{r2,9}\}$ using the same approach shown in Fig. 5.22. Then, part 2 detects the outputs $\{U_{\mathrm{SN}}[0], \ldots, U_{\mathrm{SN}}[6]\}$ that have lowest LLR values among the symbols $\{O_0, O_1, O_2, O_3, O_4, b_{c0,9}, b_{r1,9}, b_{r3,9}\}$ using the same approach of

part 1. We note that, $U_{\mathrm{SN}}[0] = \mathrm{b}_{\mathrm{c0,9}}$ and $U_{\mathrm{SN}}[1] = \mathrm{b}_{\mathrm{r1,9}}$.

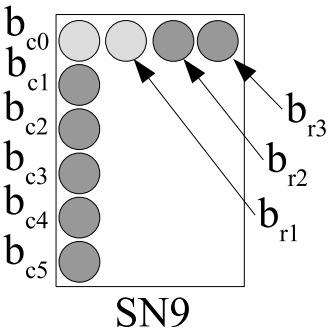

Figure 5.23: The bubbles of SN9.

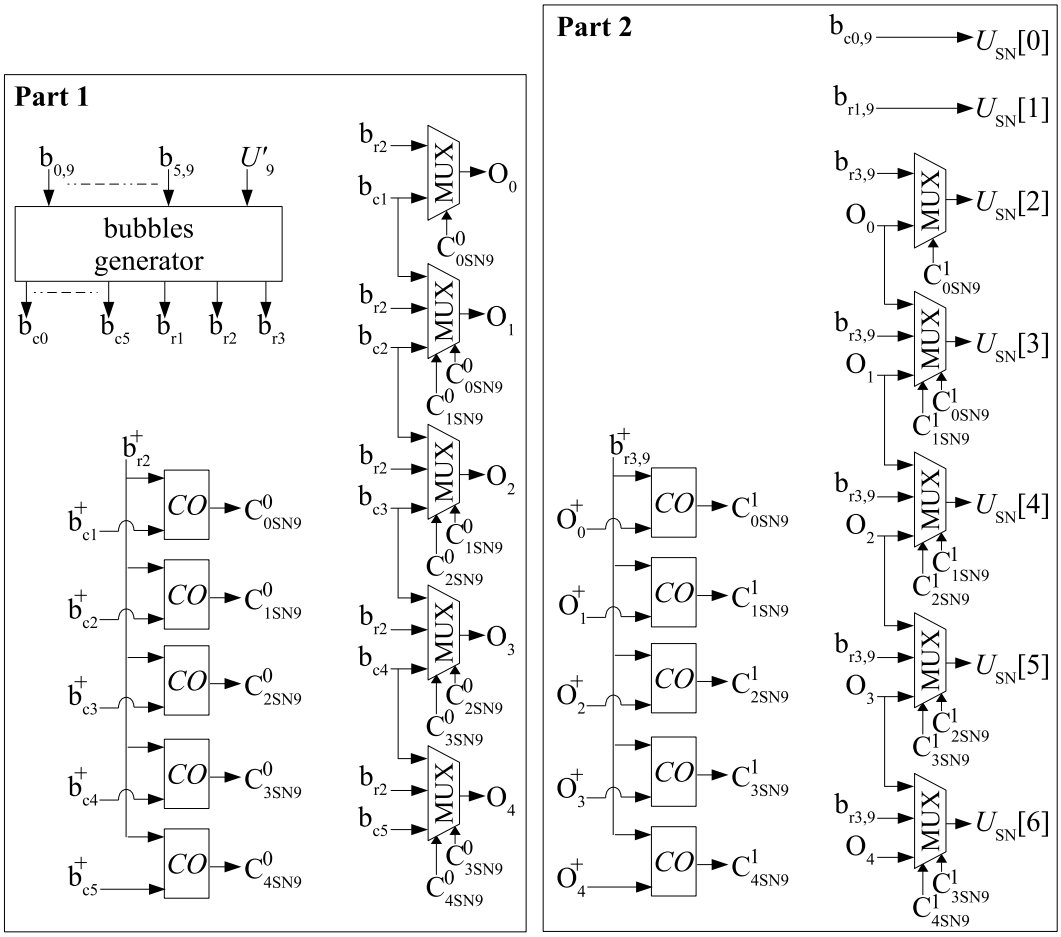

Figure 5.24: Architecture of SN9.

5.3.1.4 The shape and the architecture of ECN1

The shape and the architecture of ECN1 are shown in Fig. 5.25. The bubbles generator block is to generate the required bubbles as explained in section 5.1.2. The output of minimum LLR value is $U_{\mathrm{ECN2}}[0] = \mathrm{b}_0$ and the output of second minimum LLR value is $U_{\mathrm{ECN2}}[1] = \mathrm{b}_1$. As for the rest of outputs, the architecture is designed taking into account there are some presorted bubbles and hence the number of comparators and MUXs are reduced. In more details, $\mathrm{b}_4^+ \leq \mathrm{b}_5^+ \leq \mathrm{b}_7^+$ and $\mathrm{b}_8^+ \geq \mathrm{b}_5^+ \geq \mathrm{b}_7^+$, thus these bubbles are grouped together as shown in Fig. 5.25. Similarly, $\mathrm{b}_2^+ \leq \mathrm{b}_3^+$ and $\mathrm{b}_2^+ \leq \mathrm{b}_6^+$, thus the bubbles $\{\mathrm{b}_2, \mathrm{b}_3, \mathrm{b}_6, \mathrm{b}_9\}$ are grouped together as shown in Fig. 5.25.

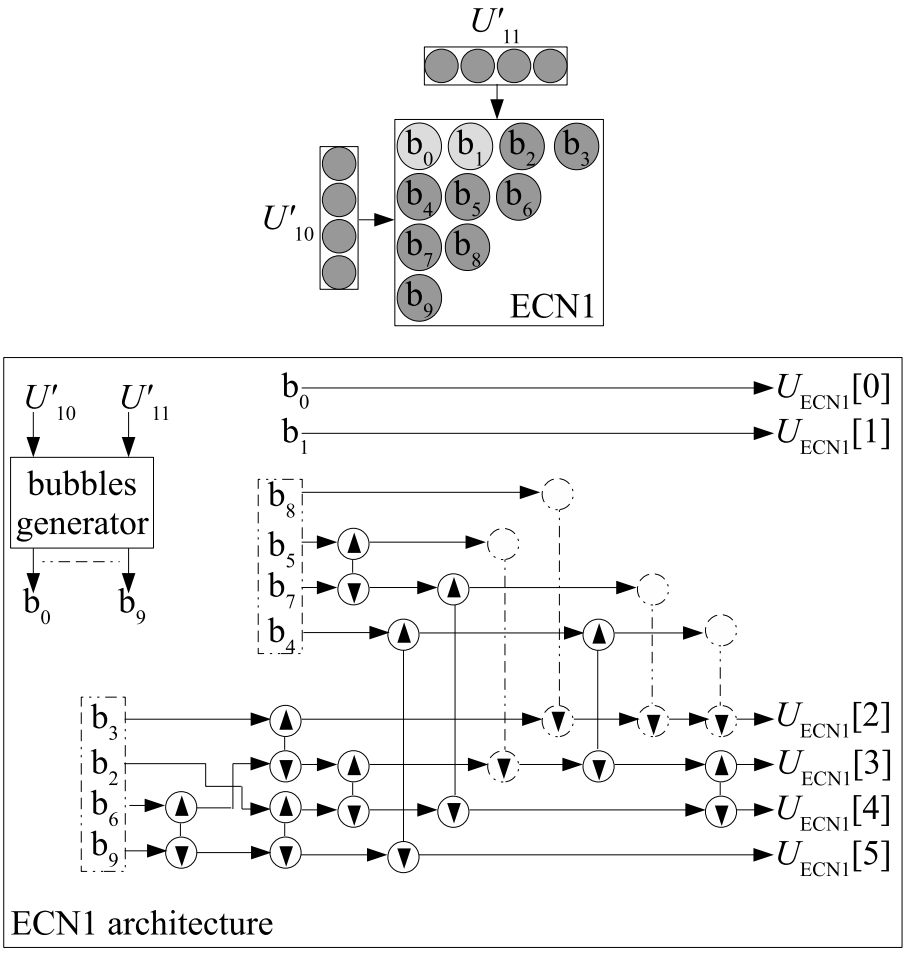

Figure 5.25: The shape and the architecture of ECN1.

5.3.1.5 ECN2, ECN3 and ECN4 architectures

The ECN2, ECN3 and ECN4 are very simple to implement. It is a matter of bubbles generation as shown in Fig. 5.26. The bubbles of each block are generated based on the description in section 5.1.2. The ECN2 bubbles generator consists of GF

and LLR additions while ECN3 and ECN4 consists of GF and LLR addition and GF permutation. The bubbles are directly mapped to the outputs. The b_{16}^{+} value taken from the ECN2 bubbles generator block is the offset of the VNs block that process $\{V'_0, \ldots, V'_9\}$ while the b_{16}^{+} value taken from ECN3 bubbles generator and ECN4 bubbles generator blocks are the two offset values of the VNs that process V'_{10} and V'_{11} respectively.

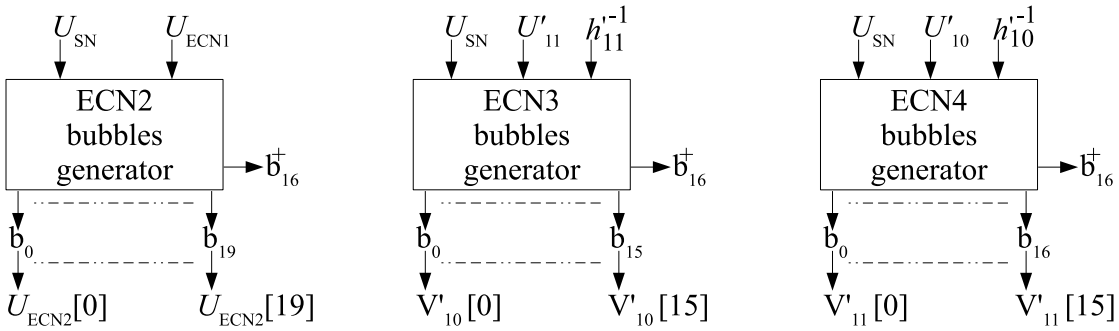

Figure 5.26: bubbles generator of ECN2, ECN3 and ECN4.

It is important to highlight the fact that, in these ECNs, the offset is predefined which permits to avoid the extra hardware that detects the last valid symbol of the outputs that is normally considered as offset.

5.3.1.6 DeBl Architecture

The DeBl architecture is shown in Fig. 5.27, it consists of two blocks operating in parallel to perform the following:

1. Decorrelation + GF permutation: this block is to decorrelate every $U^{\oplus}_{\mathrm{ECN2}}[j]$ from $U'^{\oplus}_{i}[0]$ by making GF addition, then the generated GF value is inversely permuted through its multiplication by h'^{-1}_{i}. In other word, the GF extrinsic messages are computed as: $V'^{\oplus}_{i}[j] = (U^{\oplus}_{\mathrm{ECN2}}[j] \bigoplus U'^{\oplus}_{i}[0]).h'^{-1}_{i}$, $i = 0, \ldots, 9$ and $j = 0, \ldots, 19$.

2. VSV checking: this block is to check if the GF symbol $U^{\oplus}_{\mathrm{ECN2}}[j]$, $j = 0, \ldots, 19$, is computed from the most reliable GF symbol $U'^{\oplus}_{i}[0]$, $i = 4, \ldots, 9$. It operates as:

$$V'^{+}_{i}[j] = \begin{cases} U^{+}_{\mathrm{ECN2}}[j] & \text{If } U^{\mathrm{VSV}}_{\mathrm{ECN2}}[j][i] = 1 \\ Sat & \text{Otherewise} \end{cases} \tag{5.6}$$

   This block is removed for $\{U'_0, U'_1, U'_2, U'_3\}$ since only their most reliable GF symbol is considered to compute $U_{\mathrm{ECN2}}[0], \ldots, U_{\mathrm{ECN2}}[19]$.

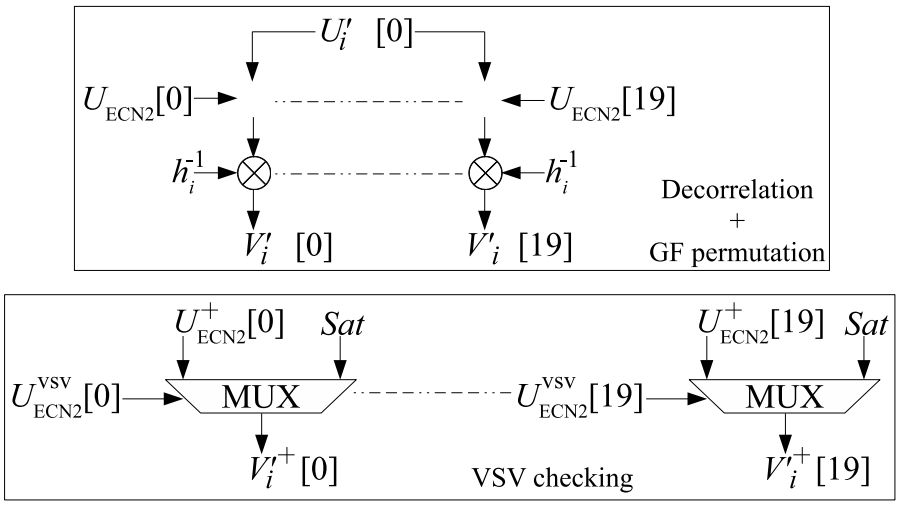

Figure 5.27: DeBl Architecture.

5.3.1.7 VN architecture

The proposed architecture of the VN is shown in Fig. 5.28. Note that the range of the index j is as follows: $j = 19$ for the V'_i, $i = 0, \ldots, 9$, and $j = 15$ for V'_{10} and V'_{11}. The *LLR extrinsic update* block receives $\{|y_{s_i,0}|, \ldots, |y_{s_i,5}|\}$, $\{V'_i[0], \ldots, V'_i[j]\}$ and $I'^{\oplus}_i[0]$ to update the extrinsic LLR value $V'^{+}_i[k]$, $k = 0, \ldots, j$. The eLLR block architecture is shown in Fig. 5.29. Each $V'^{\oplus}_i[k][l]$ is XORed with $I'^{\oplus}_i[0][l]$, $k = 0, \ldots, j$ and $l = 0, \ldots, 5$, to check their equality. In case that the two bits are different, the MUX selects the LLR value $|y_{s_i,l}|$, otherwise, MUX selects the value 0. Then, all the LLR values are added to generate $I_{V'_i}[k]$. After that, $I_{V'_i}[k]$ is added to $V'^{+}_i[k]$ to update it. The *Intrinsic regeneration and offset addition* block is to regenerate the $n_m = 4$ intrinsic candidates $\{I'_i[0], \ldots, I'_i[3]\}$ based on the proposed architecture shown in section 4.1 for $n_m = 4$. Then, the offset O is added to $\{I'^{+}_i[0], \ldots, I'^{+}_i[3]\}$. Regenerating the intrinsic candidates is less complex than storing them in RAMs and then switching them by SM block.

The *Sorter and Redundant Suppression (RS)* block is to detect the four updated messages $\{U'_{n_i}[0], \ldots, U'_{n_i}[3]\}$ having the lowest LLR values among the set of symbols $\{V'_i[0], \ldots, V'_i[j], I'_i[0], \ldots, I'_i[3]\}$ by the j+4-to-5 sorter, and then remove any redundant symbol to obtain the set $\{U'_{n_i}[0], \ldots, U'_{n_i}[3]\}$. The 24-to-5 sorter architecture in case of $j = 19$ is shown in Fig. 5.30. The sub-sorter is a sorter whose some of its inputs are already sorted. Fig. 5.31.a) shows the 4-to-4 sorter, Fig. 5.31.b) shows the 8-to-5 sorter, Fig. 5.31.c) shows the sub 10-to-5 sorter where $x_0 \leq \cdots \leq x_4$ and $x_5 \leq \cdots \leq x_9$, Fig. 5.31.d) shows the sub 8-to-5 sorter where $x_0 \leq \cdots \leq x_3$ and $x_4 \leq \cdots \leq x_7$ and finally Fig. 5.31.e) shows the sub 10-to-5 sorter where $x_0 \leq \cdots \leq x_4$ and $x_5 \leq \cdots \leq x_8$. All the sorter blocks are designed based on the odd-even algorithm [55]. Fig. 5.32 shows the sorter architecture in case of $j = 15$.

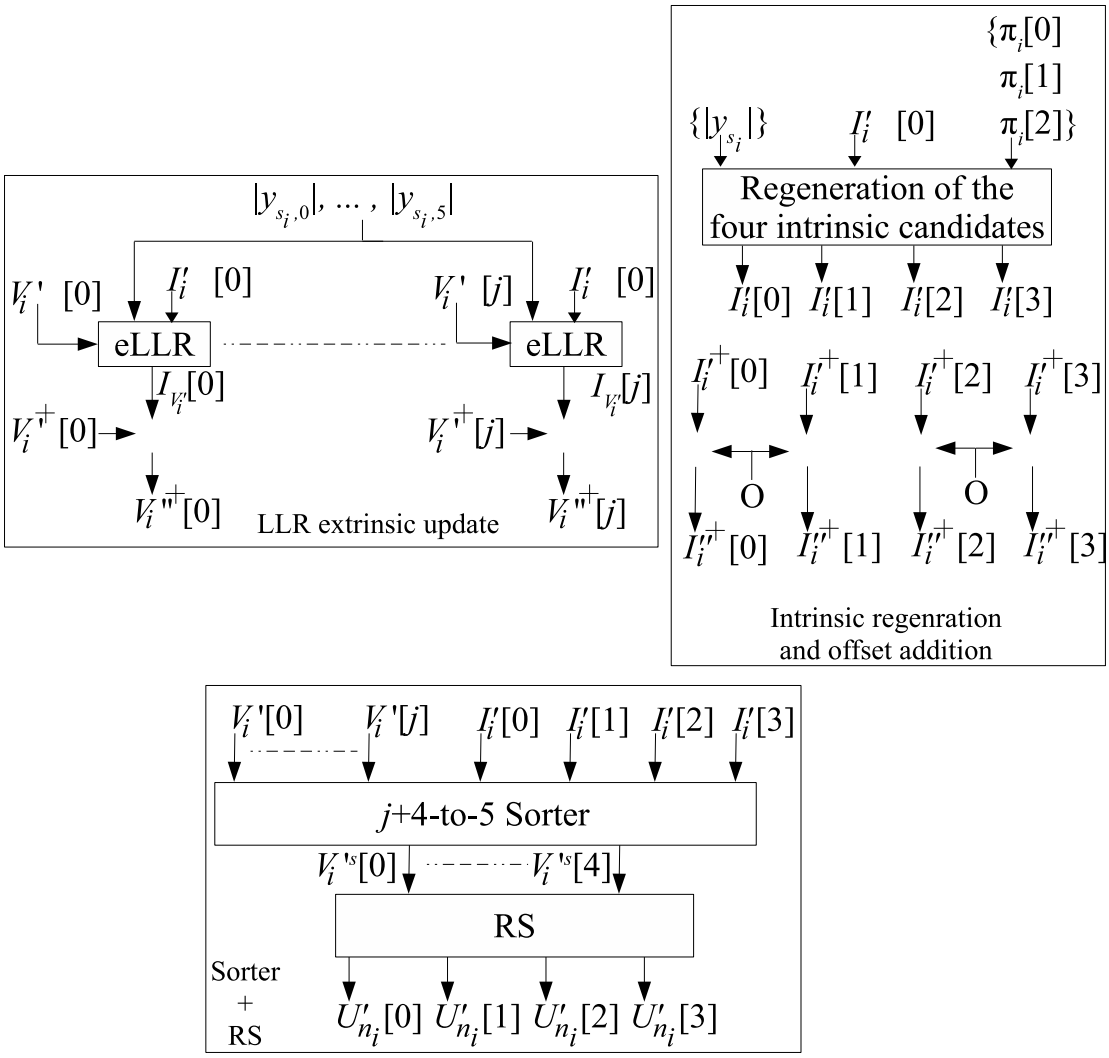

Figure 5.28: VN architecture.

Finally, the RS blocks detects the updated messages $\{U'_{n_i}[0], \ldots, U'_{n_i}[3]\}$ of lowest LLR values that do not have any redundant GF symbols. The architecture of the redundant suppression block is presented in Fig. 5.33. Each $V_i'^{s\oplus}[l_0]$ is compared with $V_i'^{s\oplus}[l_1]$ ($l_0 = 1, \ldots, 4$, $l_1 = 0, \ldots, l_0 - 1$), if $V_i'^{s\oplus}[l_0] = V_i'^{s\oplus}[l_1]$ then $C_{l_0} = 1$, otherwise, $C_{l_0} = 1$. The first output is $U'_{n_i}[0] = V_i'[0]$ and the multiplexers select the rest of outputs based on the control signals $\{C_1, C_2, C_3, C_4\}$ as:

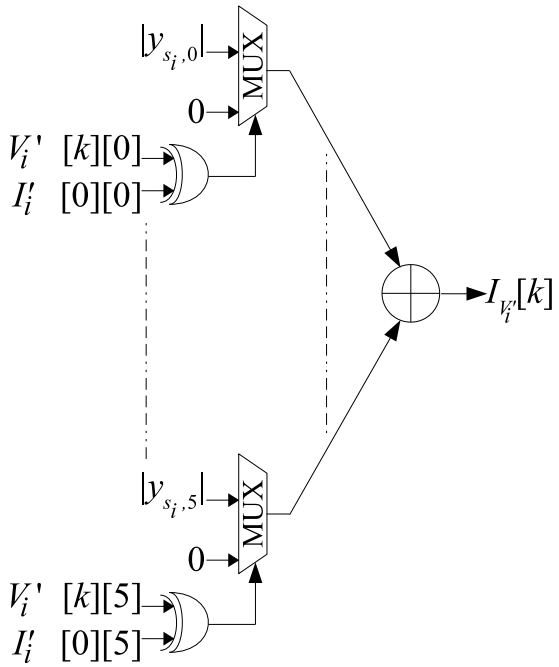

Figure 5.29: eLLR architecture.

$$U'_{n_i}[1] = \begin{cases} V_i'^s[1] & \text{If } C_1 = 0 \\ V_i'^s[2] & \text{Else If } C_2 = 0 \\ V_i'^s[3] & \text{Else If } C_3 = 0 \\ V_i'^s[4] & \text{Else If } C_4 = 0 \\ (Sat, 0) & \text{Otherwise} \end{cases} \tag{5.7}$$

$$U'_{n_i}[2] = \begin{cases} V_i'^s[2] & \text{If } C_1 = 0 \\ V_i'^s[3] & \text{Else If } C_1 = 1 \text{ and } C_2 = 0 \\ V_i'^s[4] & \text{Else If } C_1 = 1 \text{ and } C_2 = 1 \text{ and } C_3 = 0 \\ (Sat, 0) & \text{Otherwise} \end{cases} \tag{5.8}$$

$$U'_{n_i}[3] = \begin{cases} V_i'^s[3] & \text{If } C_1 = 0 \text{ and } C_2 = 0 \\ V_i'^s[4] & \text{Else If } ((C_1 = 1 \text{ and } C_2 = 0) \text{ or } (C_1 = 0 \text{ and } C_2 = 1)) \\ & \qquad \text{and } (C_3 = 0 \text{ and } C_4 = 0) \\ (Sat, 0) & \text{Otherwise} \end{cases} \tag{5.9}$$

5.3.1.8 NR architecture

The NR architecture is shown in Fig. 5.34. First, all the LLR values $U_i'^+[j]$, $i = 0, \ldots, 11$ and $j = 1, 2, 3$, are normalized as: $U_i'^+[j] = U_i'^+[j] - U_i'^+[0]$ (see the Normalization block in Fig. 5.34). Then, $U_i'^+[0]$ is replaced by the LLR value 0. After

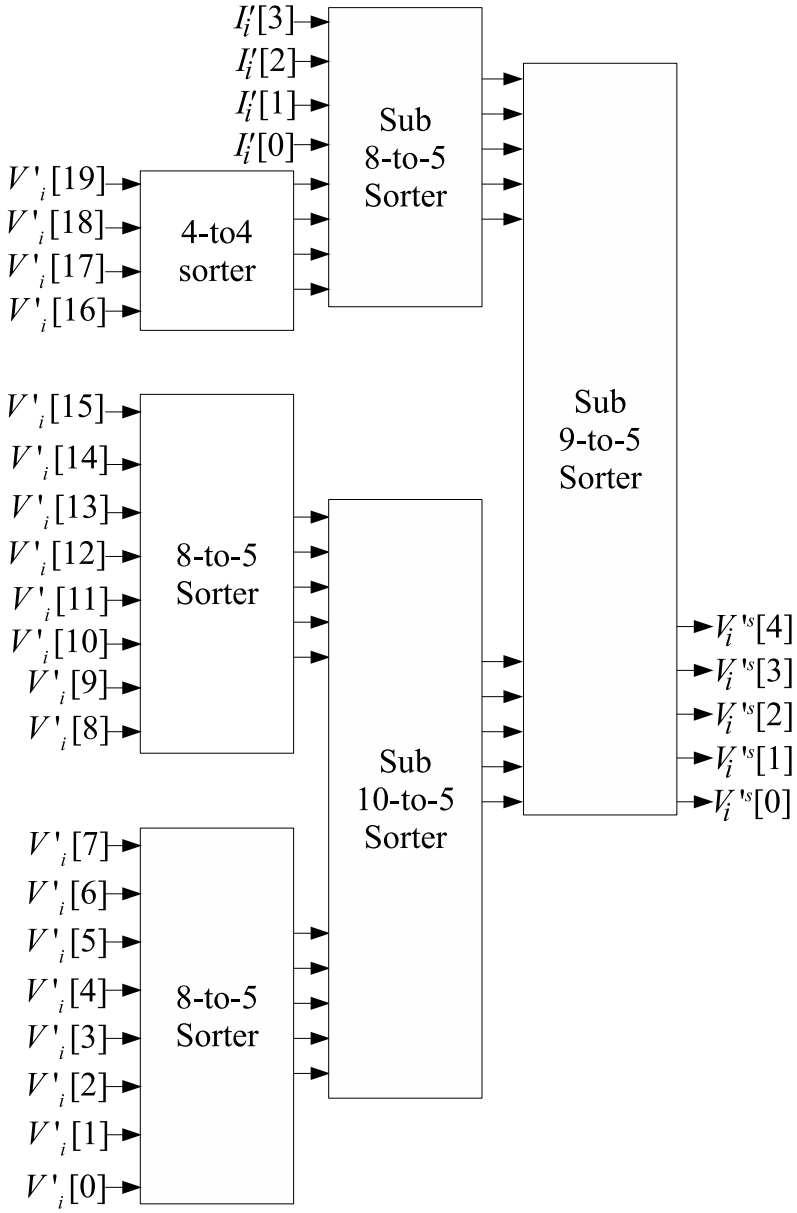

Figure 5.30: 24-to-5 architecture.

that, the vectors $\{U'_0, \ldots, U'_{11}\}$ are reordered as: $U_i = U_{\psi^{-1}[11-i]}$, $i = 0, \ldots, 11$ (see the Reordering block in Fig. 5.34).

5.3.1.9 Timing diagram of the CN-VN unit

Fig. 5.35 shows the timing diagram of the CN-VN unit, where two consecutive CNs, CN_1 (white color) and CN_2 (gray color), are shown to show the pipelining approach. One set of inputs $\{\boldsymbol{U}, \Pi, |\boldsymbol{y}|, \boldsymbol{h}, \boldsymbol{h}^{-1}\}$ per CC is entering the CN-VN unit for each CN_i, $i = 0, \ldots, 23$. The presorting indexes $\Psi = \{\psi[0], \ldots, \psi[11]\}$ are generated after 3 CCs

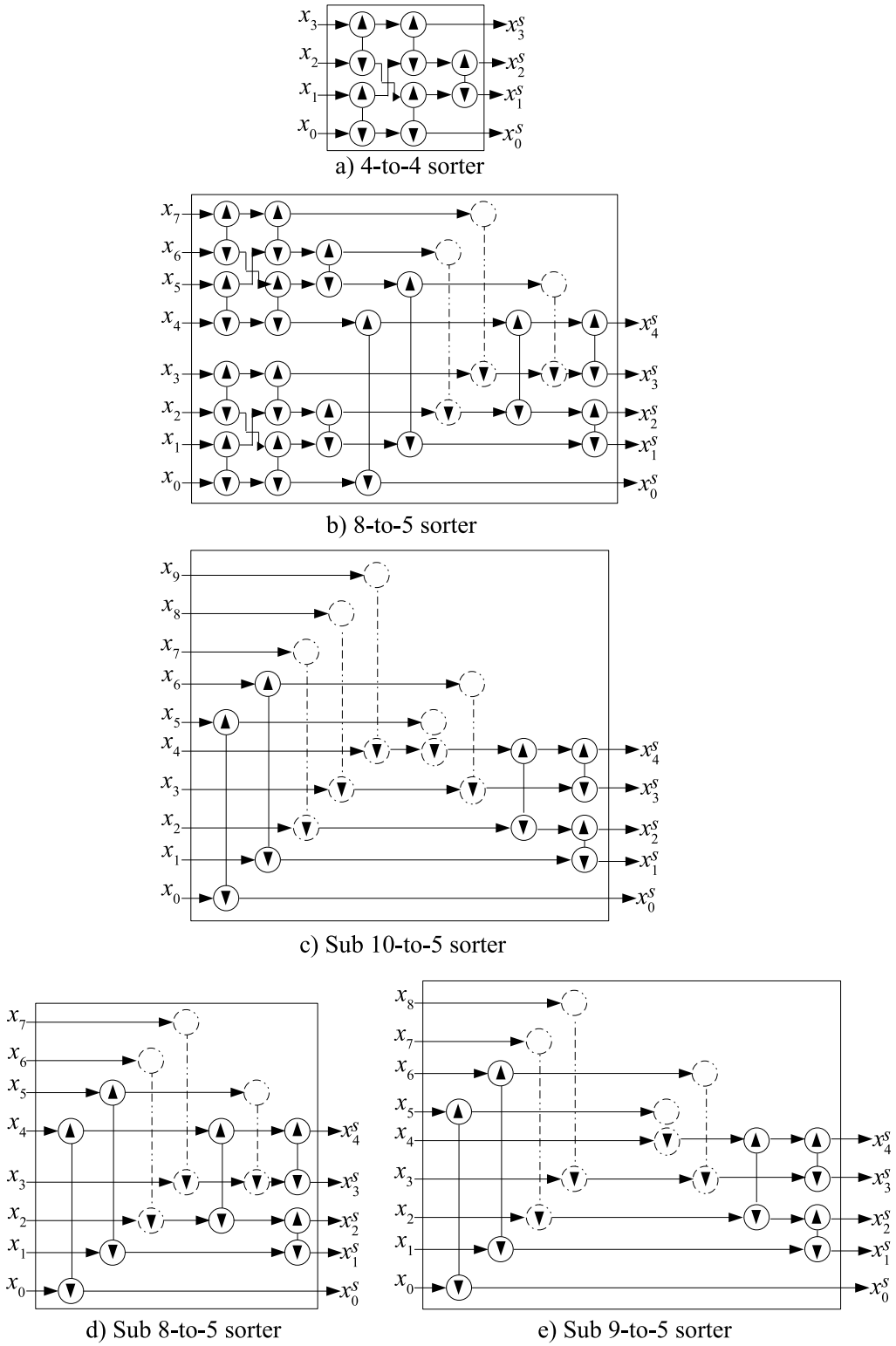

Figure 5.31: Sorters and sub-sorters architectures.

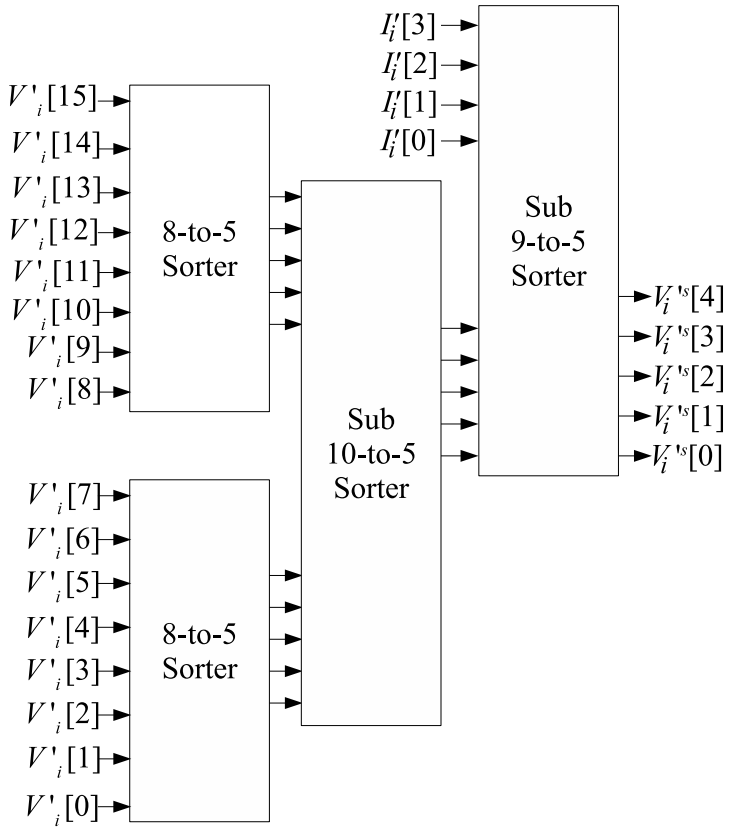

Figure 5.32: 20-to-5 architecture.

latency. Then, the permuted data $\boldsymbol{U}' = \{U'_0, \ldots, U'_{11}\}$, $\Pi' = \{\{\pi'_0\}, \ldots, \{\pi'_{11}\}\}$, $|\boldsymbol{y}'| = \{|y'_0|, \ldots, |y'_{11}|\}$, $\boldsymbol{h}' = \{h'_0, \ldots, h'_{11}\}$, $\boldsymbol{h}'^{-1} = \{h'^{-1}_0, \ldots, h'^{-1}_{11}\}$ and $\Pi'^{-1} = \{\{\pi'^{-1}_0\}, \ldots, \{\pi'^{-1}_{11}\}\}$ are generated after 1 CC latency. After that, the two vectors U_{SN} and U_{ECN1} are generated after 2 CCs latency. Next, The three vectors U_{ECN2}, U_{ECN3} and U_{ECN3} are generated along with $\boldsymbol{V}' = \{V'_0, \ldots, V'_{11}\}$ after 1 CC. Then, the unordered updated messages $\boldsymbol{U}'_{\boldsymbol{n}} = \{U'_0, \ldots, U'_{11}\}$ are generated after 8 CCs. Finally, the updated messages $\boldsymbol{U}'_{\boldsymbol{n}} = \{U'_0, \ldots, U'_{11}\}$ are generated after 1 CC. Thus, the total latency of the CN-VN unit is equal to 16 CCs and it is for one time so the CN-VN unit starts generating the updated messages of every CN_i, $i = 0, \ldots, 23$, consecutively each 1 CC.

The CN-VN unit is pipelined in an optimized way so that the decoder runs at highest possible frequency.

5.3.2 DMU architecture

Fig. 5.36 shows the DMU architecture, where $i = 0, \ldots, 11$ while $j = 19$ in case of the DMUs that process $\{V'_0, \ldots, V'_9\}$ and $j = 15$ in case of the DMUs that process

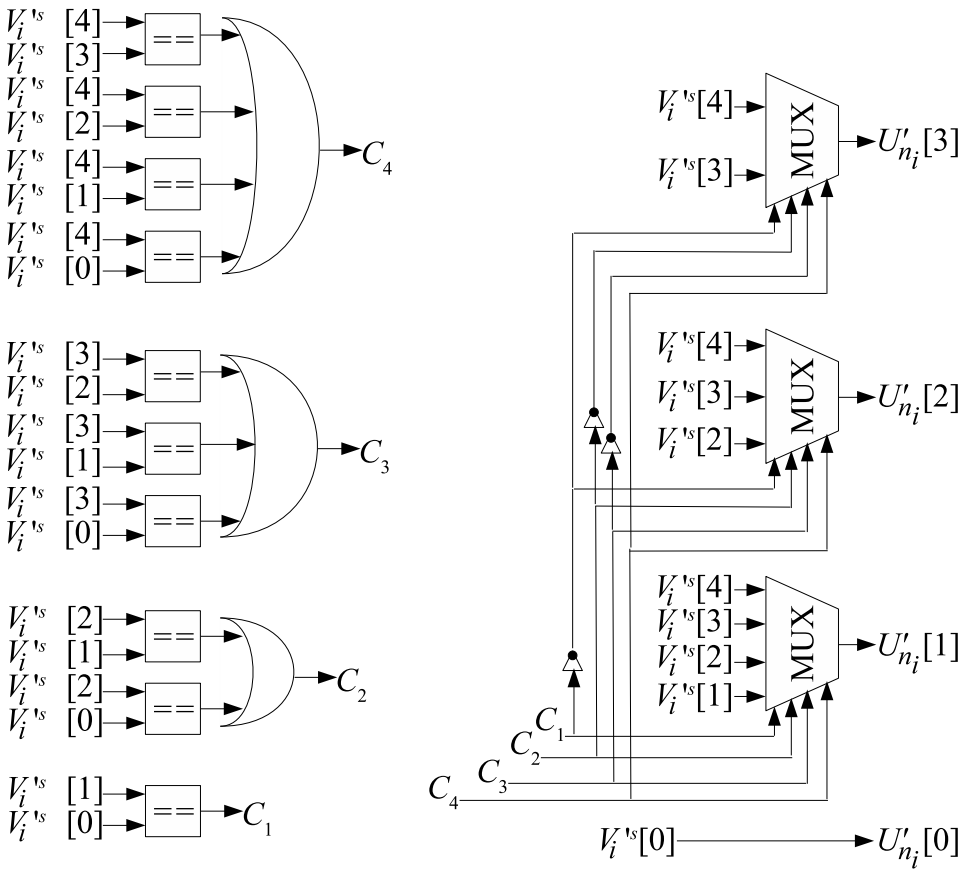

Figure 5.33: Architecture of the redundant suppression block.

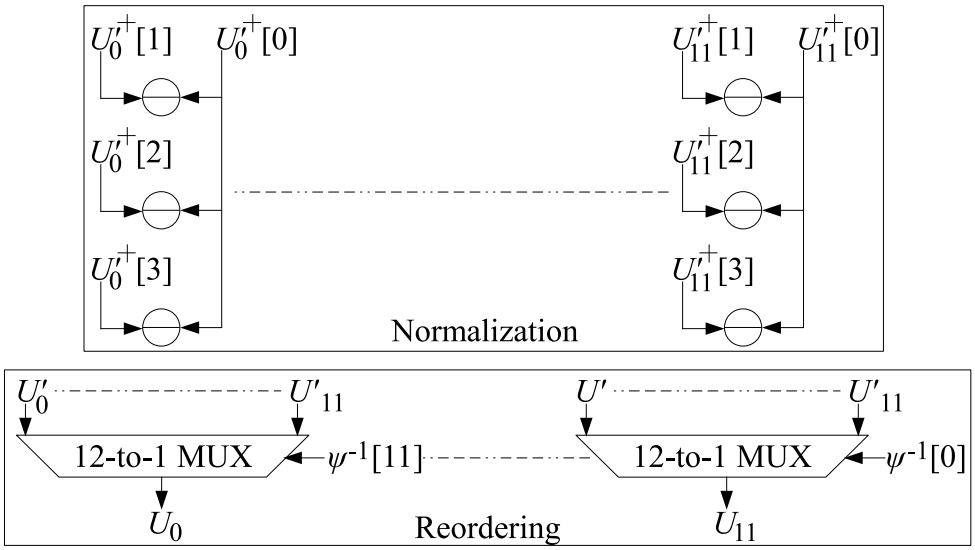

Figure 5.34: NR architecture.

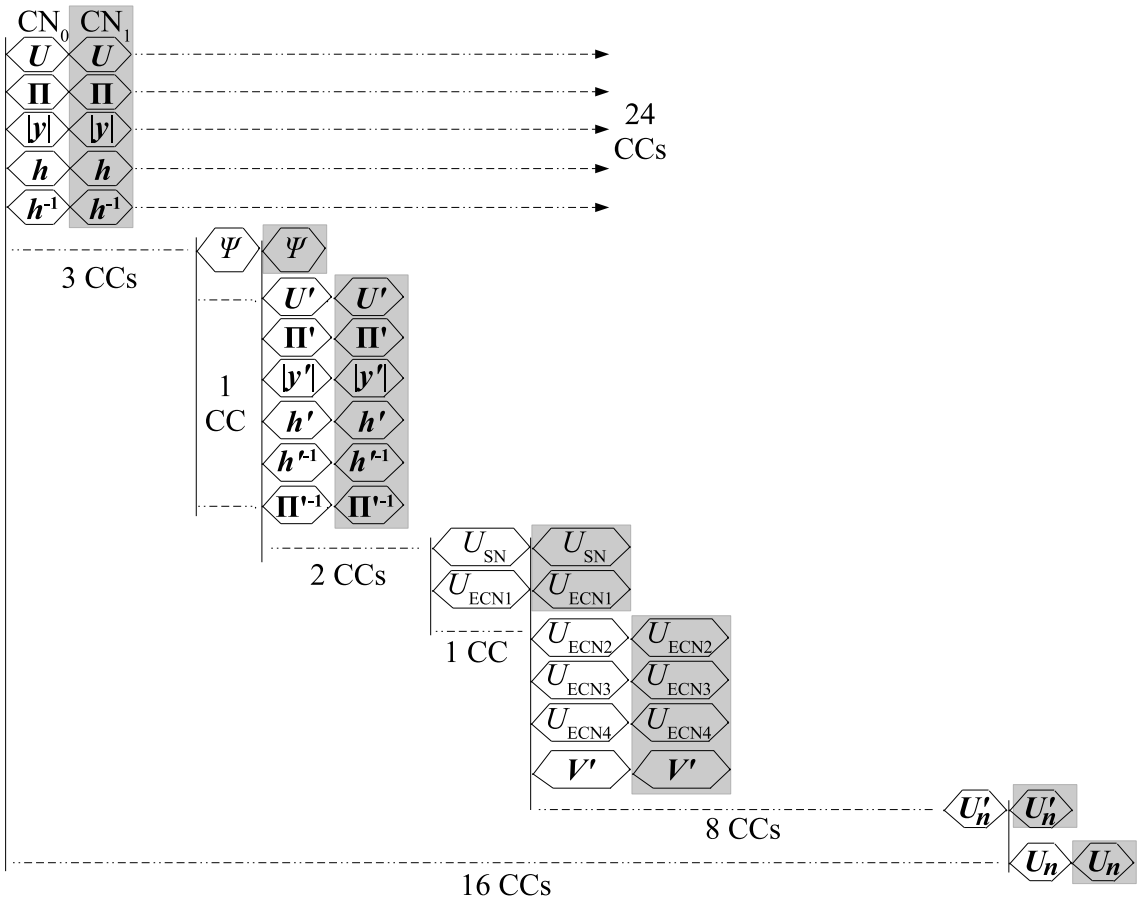

Figure 5.35: Timing diagram of the CN-VN unit.

$\{V'_{10}, V'_{11}\}$. Therefore, the DMU unit consists of three blocks operating consecutively as:

Control generator: generates the control signals $C_{k,0}$ and $C_{k,1}$ are generated, $k = 0, \ldots, j$. In which, $C_{k,l} = 1$ if $V_i'^{\oplus}[k] = U_i'^{\oplus}[l]$, $C_{k,l} = 0$, otherwise, $l = 0, 1$.

Extrinsic LLR update: each $V_i'^{+}[k]$, $k = 0, \ldots, j$, is updated as:

$$V_i'^{+}[k] = \begin{cases} V_i'^{+}[k] + U_i'^{+}[0] & \text{If } C_{k,0} = 1 \\ V_i'^{+}[k] + U_i'^{+}[1] & \text{Else if } C_{k,1} = 1 \\ V_i'^{+}[k] + U_i'^{+}[2] & \text{Otherwise} \end{cases} \tag{5.10}$$

in addition, one updated message $U'_i[0]$ is considered in making the decision where its LLR value is updated as: $U_i'^{+}[0] = U_i'^{+}[0] + \mathrm{O}$.

Finally, the decision GF'_i is taken by detecting the GF symbol of lowest LLR value among $\{V'_i[0], \ldots, V'_i[j], U'_i[0]\}$ using MIN detector block consisting of a tree of com-

parators.

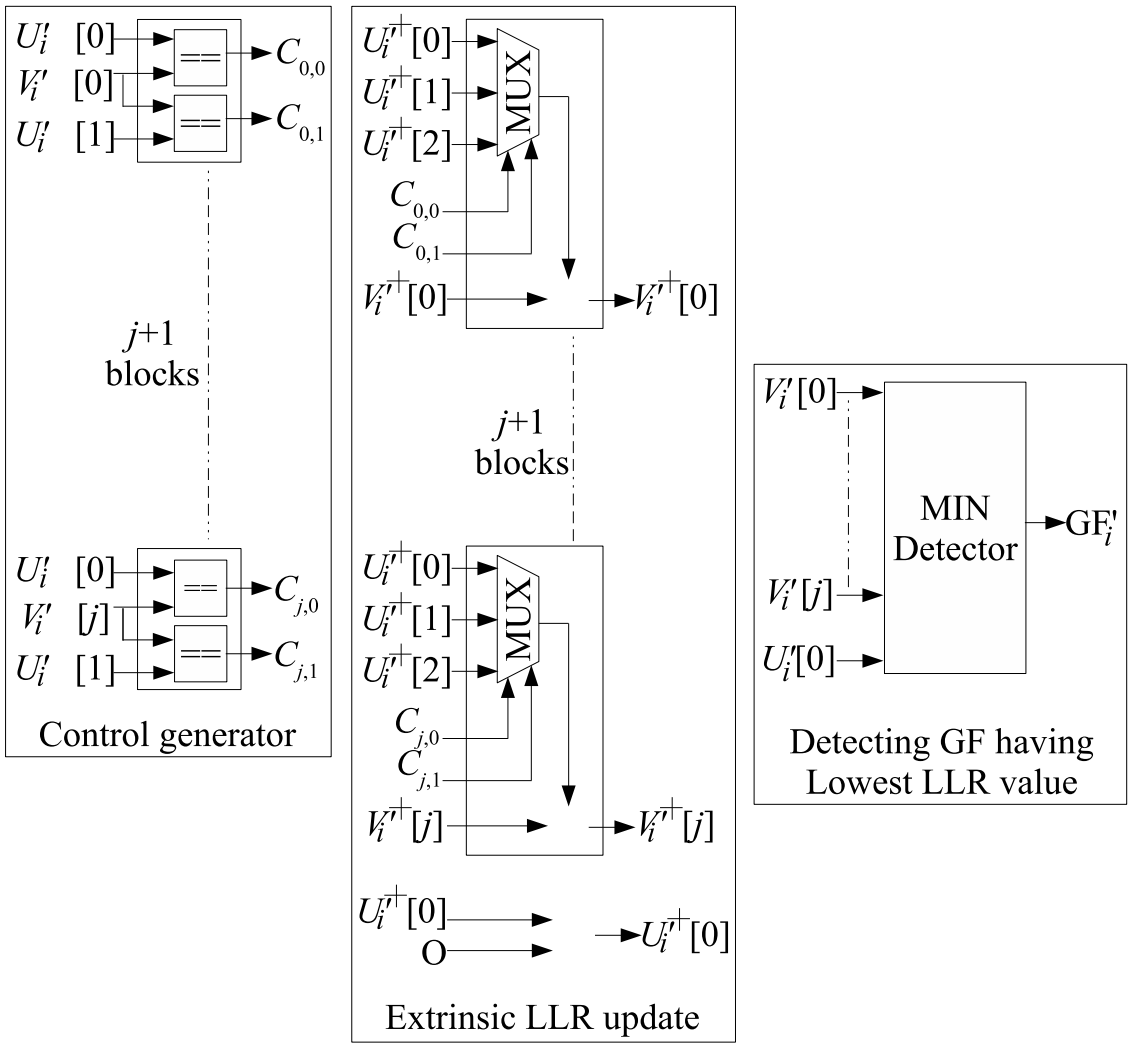

Figure 5.36: DMU Architecture.

After generating the decided GF symbols $\{\mathrm{GF}'_0, \ldots, \mathrm{GF}'_{11}\}$, their reordering to their original order is performed by the DMUR block as shown in Fig. 5.37. The set $\{\mathrm{GF}'_0, \ldots, \mathrm{GF}'_{11}\}$ is reordered as: $\mathrm{GF}'_i = \mathrm{GF}'_{\pi^{-1}[11-i]}$, $i = 0, \ldots, 11$.

Fig. 5.38 shows the timing diagram of the DMU and DMUR units. After 2 CCs latency from receiving $\boldsymbol{V}' = \{V'_0, \ldots, V'_{11}\}$ and $\boldsymbol{U}' = \{V'_0, \ldots, V'_{11}\}$, the list of decided GF symbols $\mathbf{GF} = \{\mathrm{GF}_0, \ldots, \mathrm{GF}_{11}\}$ is obtained. Thus, each clock cycle, one set of $\boldsymbol{V}'$, $\boldsymbol{U}'$ is received for every $\mathrm{CN}_0, \ldots, \mathrm{CN}_{11}$ consecutively to make the decisions and hence 14 CCs is the required execution time to have the GF decisions on all the $N = 144$ VNs.

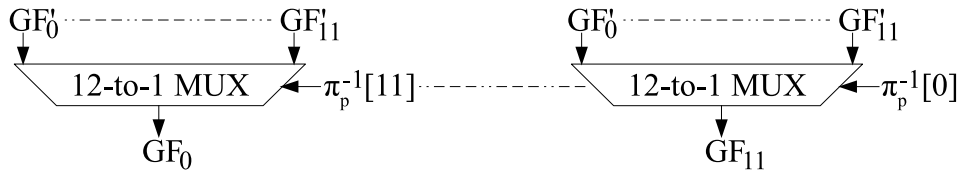

Figure 5.37: DMUR architecture.

Figure 5.38: Timing diagram of the DMU unit.

5.3.3 PTB

Fig. 5.39 shows the architecture of the PTB unit. PTB is to check if the M equations of the PCM are satisfied as:

$$\bigoplus_{i=0}^{11}(h_i.\mathrm{GF}_i) = 0. \tag{5.11}$$

Recalling Fig. 5.38, the decisions on the VNs that are associated to L_1 are being made for each CN_i, $i = 0, \ldots, 11$, consecutively. Thus, PTB unit-1 shown in Fig. 5.39 starts processing L_1 right after that the decisions on the VNs of CN_1 are made. Therefore, PTB unit-1 operates as:

Phase 1: The outputs of DMU are selected first by a set of MUXs controlled by C_{GF}, i.e, $\{\mathrm{GF}_0^M, \ldots, \mathrm{GF}_{11}^M\} = \{\mathrm{GF}_0, \ldots, \mathrm{GF}_{11}\}$. This phase is an idle phase for PTB unit-2.

Phase 2: Once the set of CNs $\{\mathrm{CN}_0, \ldots, \mathrm{CN}_{11}\}$ is processed, the set of MUXs selects the set $\{\mathrm{GF}_0^0, \ldots, \mathrm{GF}_{11}^0\}$ to be processed. The set $\{\mathrm{GF}_0^0, \ldots, \mathrm{GF}_{11}^0\}$ that comes from the SCRB block shown in Fig. 5.12 is related to the set of CNs $\{\mathrm{CN}_{12}, \ldots, \mathrm{CN}_{17}\}$ where one CN_j, $j = 12, \ldots, 17$, is considered each CC.

The non-zero elements $\{h_0, \ldots, h_{11}\}$ are being read from the ROM shown in Fig. 5.15, so they are delayed to be used in PTB unit-1. Each GF_i^M is multiplied by h_i using a set of 12 GF multiplications. Then, the results are added by a tree of 11 GF adders. Note here that the addtition symbol denotes a GF adder, i.e., a simple XOR gate. The result GF^a belongs to GF(64), i.e, it consists of 6 bits. If $\mathrm{GF}^a = 0$, then the parity check is satisfied. The result of the test is stored in its appropriate register where each register is related to one CN. In more details, when checking the validity

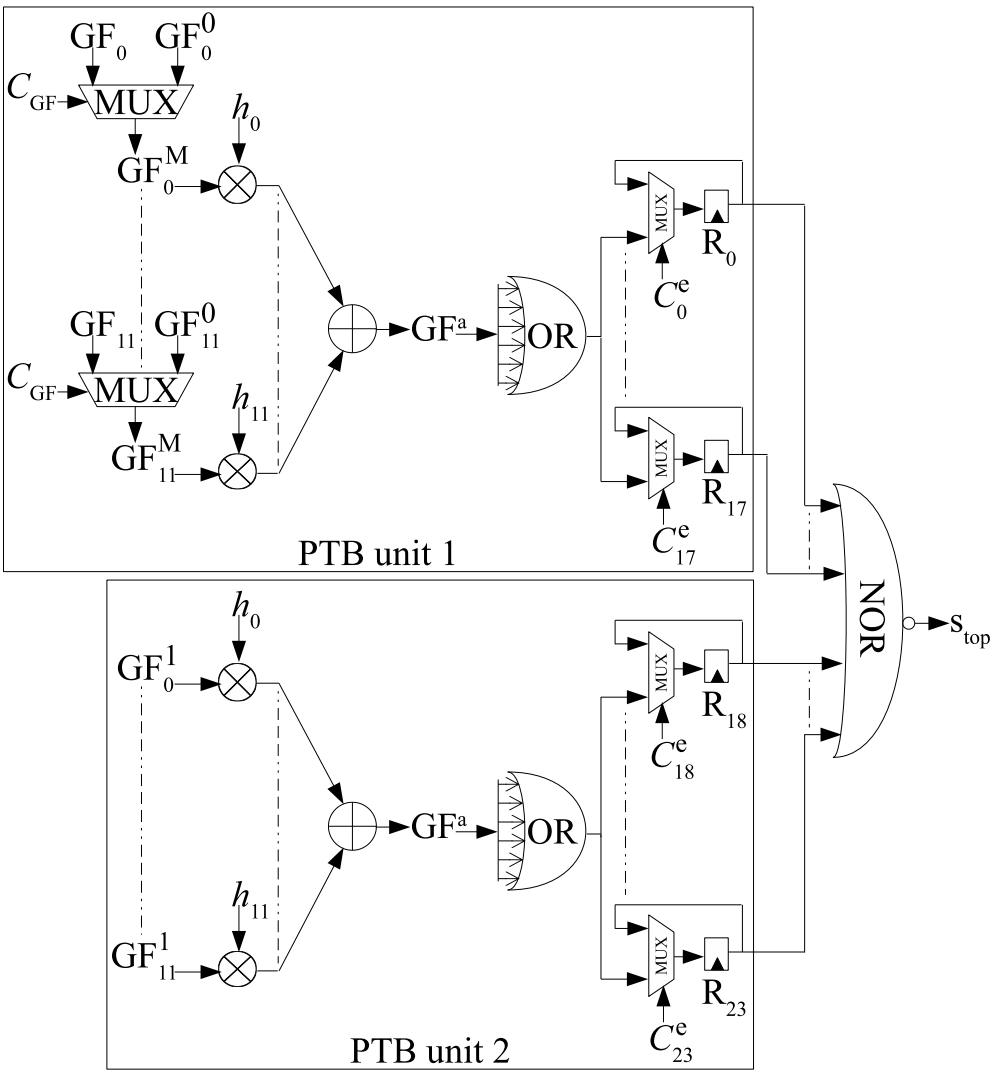

Figure 5.39: SD architecture.

of CN_i, $i = 0, \ldots, 17$, the MUX that is controlled by C_i^{e} selects the result of the OR gate that is associated to CN_i to be saved in R_i.

PTB unit-2 operates in parallel with PTB unit-1 where only the set $\{\mathrm{GF}_0^1, \ldots, \mathrm{GF}_{11}^1\}$ is considered. The set $\{\mathrm{GF}_0^1, \ldots, \mathrm{GF}_{11}^1\}$ that comes from the SCRB block is related to the set of CNs $\{\mathrm{CN}_{18}, \ldots, \mathrm{CN}_{23}\}$ where one CN_j, $j = 18, \ldots, 23$, is considered each CC. The required non-zero elements $\{h_0, \ldots, h_{11}\}$ are read from the ROM block during the processing of PTB unit-2. Thus, the ROM shown in Fig. 5.15 is a dual-port ROM. The functionality of PTB unit-2 is the same as PTB unit-1 where the bit that indicates the validity of CN_j is saved in R_j, $j = 18, \ldots, 23$.

After that all the registers $R_0, \ldots, R_{11}$ are filled, their results are NORed to check if the $M = 24$ equations are satisfied. Thus, the stop signal s_{top} takes the value 1 when all the 24 registers are filled with zero, i.e, all the 24 equations are satisfied, otherwise, s_{top} takes the value 0. Note that this circuit could be simplified by using only a 2-input OR gate that receives: 1) the output of the 6-input OR gate and 2) a feedback from a register placed at its output. This will be one of the optimization that could be done in the next version of the decoder.

Fig. 5.40 and Fig. 5.41 show the timing diagram of the PTB unit during phase 1 and phase 2 respectively. In phase 1, the outputs of the DMU block $\mathbf{GF} = \{GF_0, \ldots, GF_{11}\}$ are directly considered along with their appropriate non-zero elements $\boldsymbol{h} = \{h_0, \ldots, h_{11}\}$. During this phase, the 12 equations associated to $CN_0, \ldots, CN_{11}$ are checked and the results are saved in $R_0, \ldots, R_{11}$. Then, after that all the $N = 144$ GF decisions are made, the equations associated to $CN_{12}, \ldots, CN_{23}$ are checked during phase 2. In this phase, PTB unit-1 and PTB unit-2 operate in parallel to check the equations associated to $CN_{12}, \ldots, CN_{17}$ and $CN_{18}, \ldots, CN_{23}$ respectively. After 18 CCs when the 12 registers are filled, the result s_{top} is generated by the NOR gate.

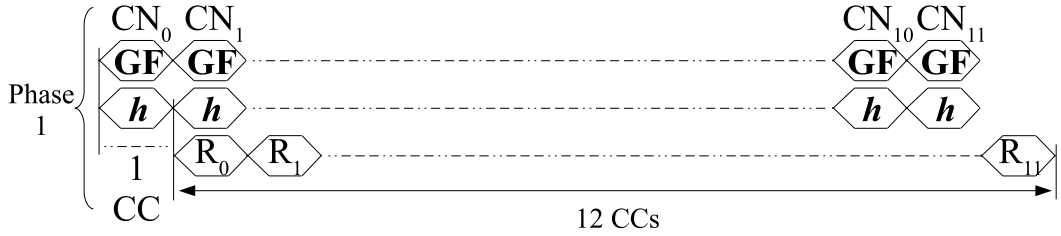

Figure 5.40: Timing diagram of PTB phase 1.

5.4 Timing diagram of the global decoding process

The decoder is designed to be able to overlap the processing of two consecutive frames, which permits to increase the degree of parallelism and thus the global throughput rate. Fig. 5.42 shows the timing diagram when two frames are being processed. The decoder starts processing an iteration of frame 2 immediately after processing an iteration of frame 1 where $M = 24$ CCs is the latency of one iteration. For instance, let us consider that two iterations are required to decode frame 1 while 3 iterations are required to decode frame 2. The decoder starts processing the first iteration of frame 1, then the decoder starts processing the first iteration of frame 2, after that the decoder starts processing the second iteration of frame 1 where the PTB block will indicate that the 24 equations are satisfied so stop processing frame 1. Thus, the observed symbols of frame 3 start entering the decoder in parallel with the processing

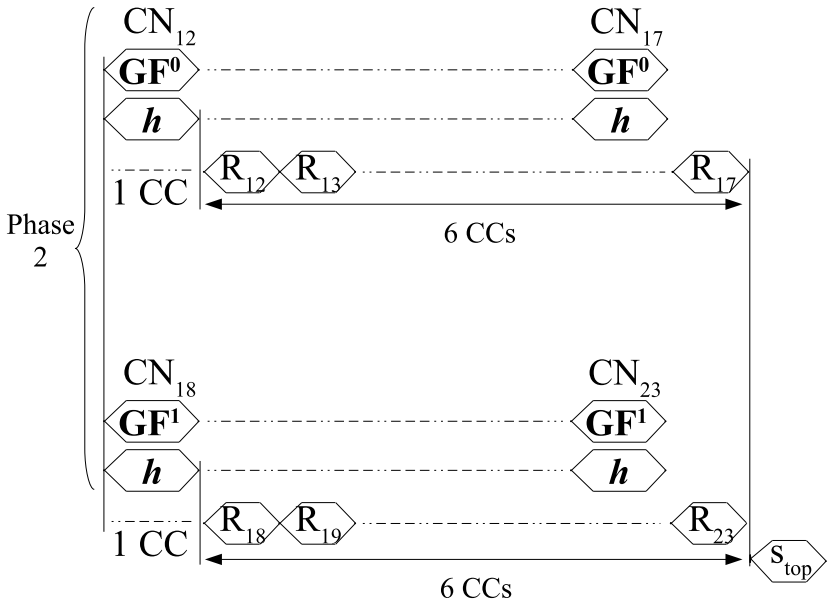

Figure 5.41: Timing diagram of PTB phase 2.

the second iteration of frame 2 as shown in Fig. 5.43.

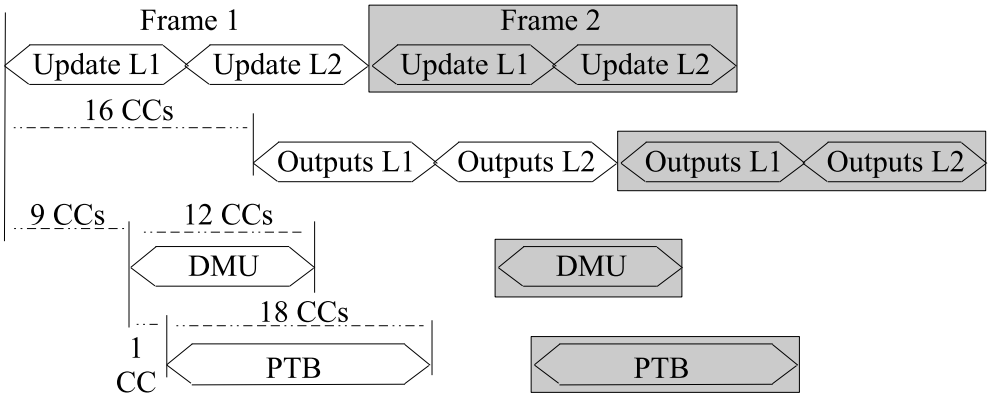

Figure 5.42: Timing diagram of the decoder in case of processing two frames simultaneously.

As we mentioned in section 5.2.2, the required data of the VNs that are connected to CN_0 will be ready after 19 CCs. As Fig. 5.43 shows, the stopping decision of frame 1 is made after $9 + 1 + 18 = 28$ CCs, i.e, the CN-VN unit still have 19 CCs to process frame 2 and hence CN-VN unit can start processing the first iteration of frame 3 right after frame 2. For that, 8 observed symbols are received in parallel to be able to start processing next frame just after the current one. This parallelism in the simultaneous

processing of two consecutive frames requires the duplication of the intrinsic RAMs and extrinsic RAMs to store the data of two frames.

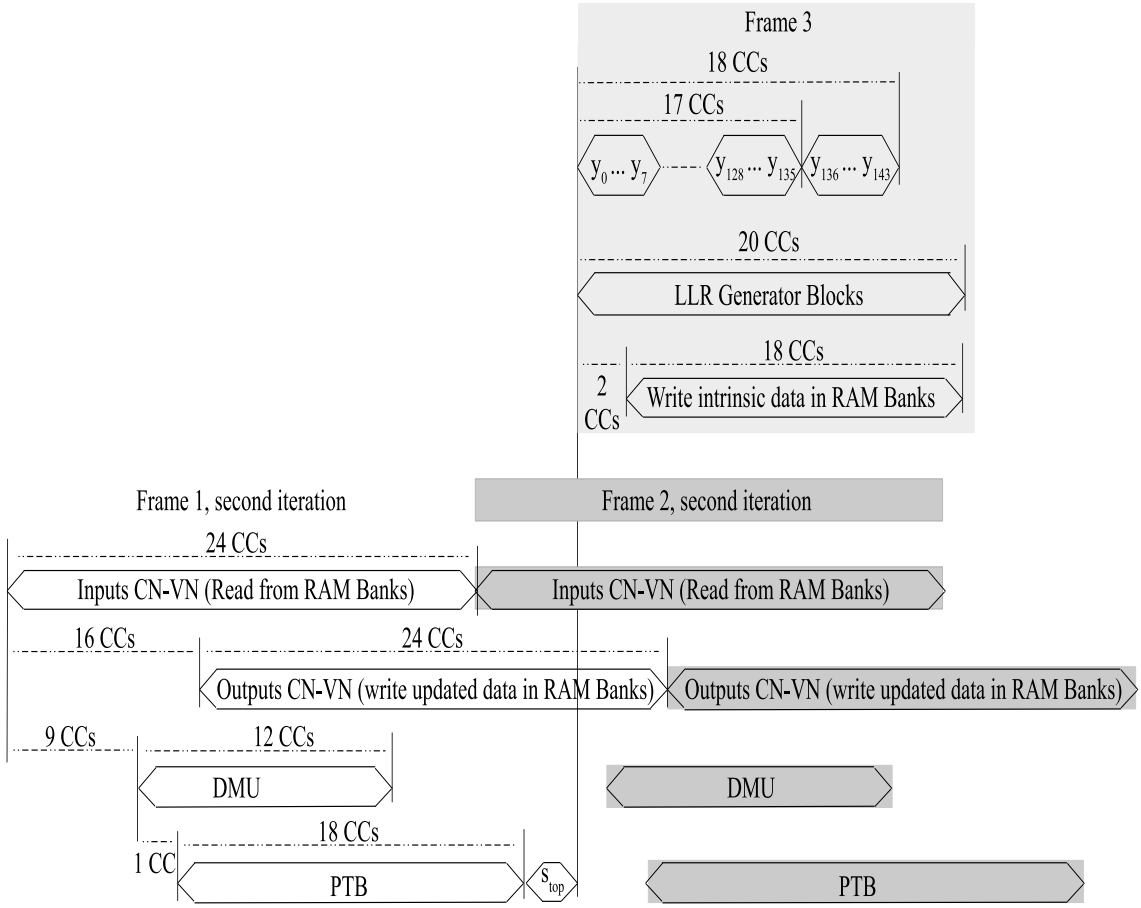

Figure 5.43: Timing diagram of the decoder in case of interleaving frames.

To summarize, looking at the global execution of the decoder, the latency of preparing the data (shown in Fig. 5.16) as well as the 16 CCs latency of the CN are not counted when evaluating the execution time of the decoder that has a direct impact on the throughput rate. We also note that without the duplication of the RAMs, that allowed the parallel processing of two consecutive frames, the 16 CCs latency of the CN have to be counted as part of the execution time at each iteration. This is due to the fact that CN_{23} and CN_0 share the same variable VN_{12}, which prevent the start of the second iteration till the L_2 processing has been finished. Thus, $M = 24$ CCs is the latency of one iteration.

5.5 Implementation results

Table 5.1 shows the synthesis results of this work where it is called Fully Parallel Hybrid Decoder (FPHD) compared to three state-of-the-art decoder architectures

[29, 58, 59]. We chose to compare with these three works since they provide high throughput and adopt the parallel approach in their architectures. Refer to section 2.6 to recall the overview of these works. Note that our discussion of the implementations results and namely the throughput efficiency calculation, we refer to Fig. 5.10 that shows the average number of iterations needed for our decoder and that varies with E_b/N_0.

Table 5.1: COMPARISON OF STATE-OF-THE-ART NB-LDPC DECODERS (ASICs).

	[29]	[58]	[59]	FPHD
Technology	40 nm	90 nm	65 nm	28 nm
Design	Synthesis	Synthesis	Silicon	Synthesis
N (symbols)	3888	837	160	144
CR	8/9	13/15	1/2	5/6
GF	4	32	64	64
Decoding Algorithm	T-EMS	IL-MwBRB	EMS	EMS
Decoding schedule	Layered	-	Flooding	Flooding
Gate Count (NANDs)	4M	4.54M	2.78M	0.79M
Frequency (MHz)	1000	207.04	700	650
Iterations	10	10	10-30	1-30
Throughput (Mb/s)	3600	21661.56	1221	1060-19500
Throughput Efficiency (Mbps/M-gate)	900	4771.27	439.2	1341-24683

Thus, comparing FPHD with:

[29]: FPHD consumes 0.79 M NAND gates while [29] consumes 4 M. On the other hand, FPHD runs at 650 MHz while [29] runs at 1000 MHz. The number of iterations in [29] is fixed to 10 while it varies between 1 and 30 in FPHD. In terms of throughput, FPHD outperforms [29] starting from $E_b/N_0 = 3.7$ db where FER $\approx 10^{-2}$. However, the area efficiency of FPHD is better for all $E_b/N_0 > 3$ db.

[58]: Our proposed decoder allows a saving of NAND gates of a factor equal to 5.74. In addition, a factor gain equal to 3 is obtained in terms of frequency. On the other hand, [58] outperforms FPHD in terms of throughput for all cases of E_b/N_0. However, in terms of throughput efficiency, FPHD starts outperforming [58] from $E_b/N_0 = 3.7$ db where FER $\approx 10^{-2}$ in which the throughput is equal to

about 4 Gbits/s. Hence, the area efficiency is equal to 5063. Again, the number of iteration in [58] is fixed to 10.

[59]: There are 1.99M NAND gates less in FPHD than [59]. While there is 50 MHz difference in terms of frequency in favor of [59]. However, FPHD decoder provides better throughput for all cases of $E_b/N_0 > 3$ db and hence better hardware efficiency in a factor ranging from 4.6 up to 56. The average number of iterations in [59] varies from 10 to 30 while in FPHD it varies from 1 to 30.

Table 5.2 shows the synthesis results on Virtex 6 FPGA target. OS is the number of occupied slices, LUTs is the number of slice look up tables and SR is the number of slice registers. FPHD consumes 27.3$ of LUTs and run at F = 128 MHz.

Table 5.2: Synthesis results on Virtex 6 xc6vlx240t-2ff1156 FPGA device.

OS	LUTs	SR	F (MHz)
22827	65821	25704	128

5.6 Hardware emulation

In order to verify the software simulation results described in section 5.1, we have designed an emulation chain based on a FPGA core Kintex 7. A complete digital chain has been implemented. Some parts were designed using the High Level Synthesis (HLS) tool VIVADO using the System C toolkit and the other parts were coded in VHDL. VIVADO Tool permits to translate the System-C based blocks into VHDL description code that will be synthesized on FPGA device.

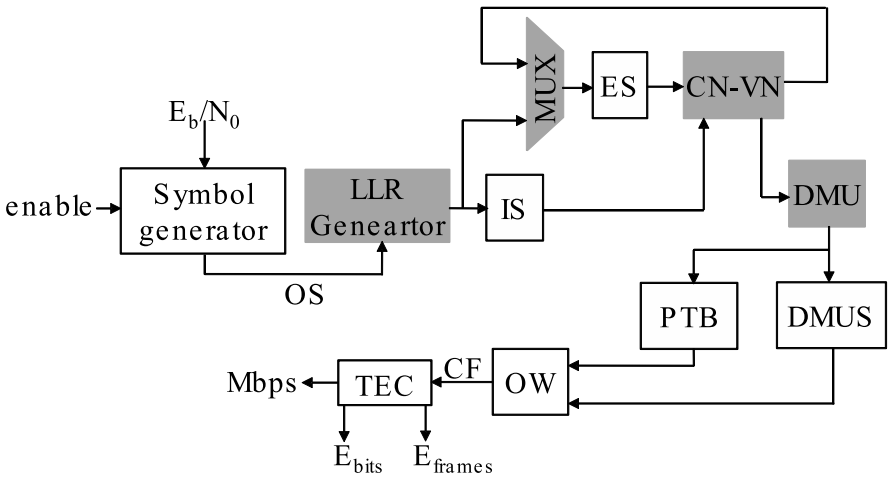

Figure 5.44: Overall hardware emulation architecture.

Fig. 5.44 shows the overall architecture of the hardware emulation design. The blocks in white color are described using systemC language while the blocks that are color are described using VHDL language. The LLR Generator block, the CN-VN block and the DMU block were described in details in this chapter. Thus, the remaining of blocks are:

1. Symbol generator: Fig. 5.45 shows the architecture of the symbol generator block. The enable signal indicates when to begin the emulation and the E_b/N_0 is the energy per bit to noise ratio of the emulation. The randc block generates six random values, then from them, six AWGN noise samples are generated. Then, the noise samples are added with the 6 BSPK bits, i.e, $\{-1.0, 1.0\}$, where six bits are read from the Encoded Modulated Bits ROM (EMB-ROM) block that stores a codeword generated using a LDPC encoder and a BPSK modulator. The content of this EMB-ROM is recorded during the Monte Carlo simulations conducted using the C-based simulator of the chain being run on a PC. Thus, a codewords that has been decoded using the C-simulator will be emulated in hardware. The stored codeword is an array of size $N = 6 \times 144 = 864$ bits. After that, the six noisy bits are quantified by the Q block on 5 bits. Finally, the Input Wrapper (IW) block collects the six quantified noisy bits and send them as Output Symbol (OS). The LLR generator receives OS to generate the $n_m = 4$ intrinsic candidates.

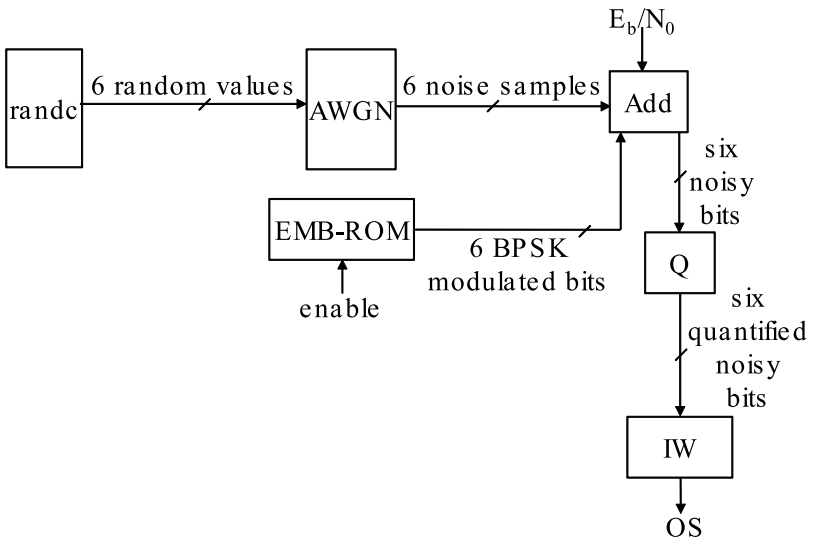

Figure 5.45: Symbol generator architecture.

2. The Intrinsic Storage (IS) block: in this block, the required intrinsic information are stored. See section 5.2, intrinsic RAMs paragraph.

3. The Extrinsic Storage (ES) block: in this block, the $n_m \times N = 4 \times 144 = 576$ updated messages are stored. At the beginning, this block is initialized by the intrinsic messages.

4. The DMU Storage (DMUS) block: this block is to store the decisions made on the 144 VNs.

5. The PTB block: this block is to check if the 24 equations are satisfied. The reason that it is systemC coded is to make it generic for any size NB-LDPC codes.

6. The Output Wrapper (OW) block: this block reads the output from the DMUS block when all the equations are satisfied or after 30 iterations of processing.

7. The Throughput Error Computation (TEC) block: in this unit the throughput of the hardware emulation Mbps, the number of erroneous bits E_{bits} and the number of erroneous frames E_{frames} are generated. The considered codeword is stored in this block to be used in computing E_{bits} and E_{frames}.

The IS, ES and PTB blocks are generic coded so they can be reconfigured for any NB-LDPC codes of $d_c = 12$. Fig. 5.46 and Fig. 5.47 show the simulation and the emulation results of the proposed decoder in case of $n_{it} = 30$ for FER and BER versus E_b/N_0 respectively. We can see that there is no performance loss between the simulation and the emulation results in both FER and BER curves.
The last version of the hardware emulation architecture achieved a throughput Mbps = 500 Mbits/s running at F = 100 MHz in Virtex 7 FPGA target. Improvements are done on IS and ES where their latency is highly reduced and hence better throughput is expected.

5.7 Conclusion

This chapter was dedicated to the proposed fully parallel and pipelined NB-LDPC decoder for $d_c = 12$. The code structure was shown first where the parameters of the considered NB-LDPC code are introduced, $N = 144$, $M = 24$, $q = 64$ and CR = 5/6. Then, the decoding algorithm was described in details, the number of considered bubbles in each ECN was given along with its shape. We showed that there was 0.08 db performance loss when comparing the proposed decoder $n_{it} = 30$ with the FB-CN layered schedule with $n_{it} = 8$. However, we showed that there was 0.05 db performance gain when comparing with the FB-CN flooding schedule with $n_{it} = 8$.

Then, the overall architecture of the proposed decoder was shown. The core of the decoder is the CN-VN block where the CN and VN processing are being performed. The CN-VN block was surrounded by the RAM banks, the DMU block and the PTB block. Then, we showed the structure of the intrinsic and extrinsic RAMs and the ROM block. We presented the timing diagram of the decoder during both launch and

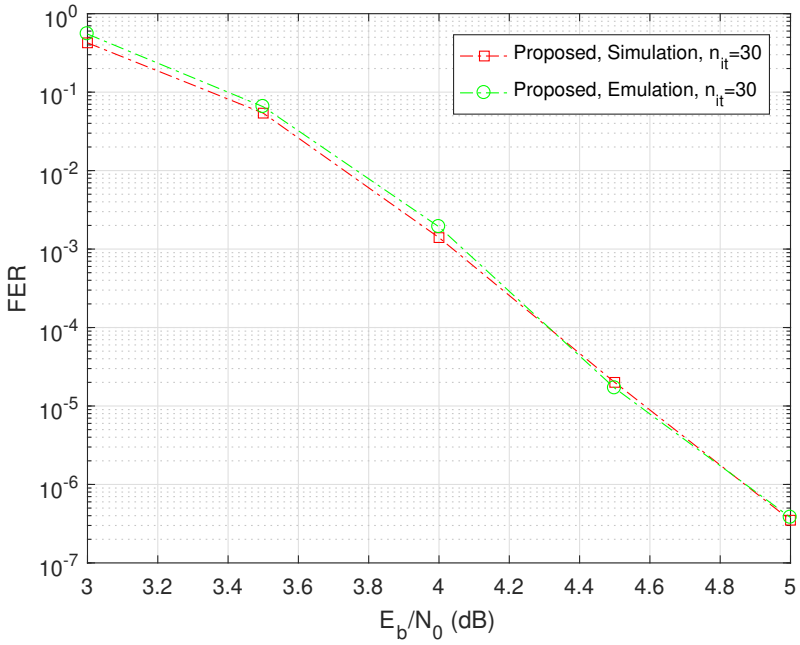

Figure 5.46: Simulation and emulation results of NB-LDPC decoding algorithms for (864, 720) code over GF(64) and $d_c = 12$ under AWGN channel (FER versus E_b/N_0).

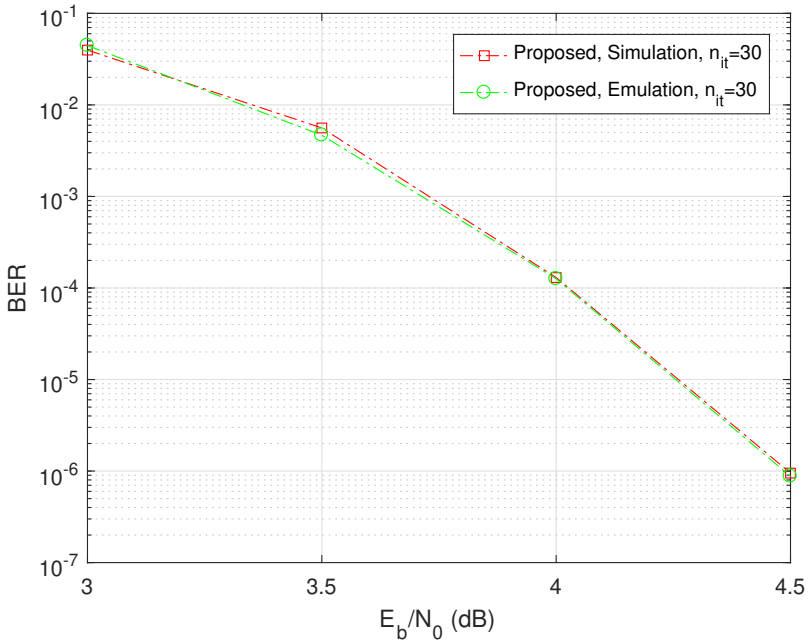

Figure 5.47: Simulation and emulation results of NB-LDPC decoding algorithms for (864, 720) code over GF(64) and $d_c = 12$ under AWGN channel(BER versus E_b/N_0).

updating phases.

Next, we presented the architecture of each component of the decoder. We started with the CN-VN block where the HB(10,0,2) was considered for the CN processing. Then, we showed the architecture of the DMU block where the decisions on the 144 VNs are made based on L_1 in PCM. Finally, the architecture of the PTB block was presented where the satisfaction of the 24 equations are checked.

After that, the timing diagram of the global decoding process was shown. We showed that the processing of the frames was interleaved so that the decoder processes two frames simultaneously. Interleaving the processing of the frames allows to avoid the 19 CCs latency of the launch phase and the 16 CCs latency of the CN-VN block.

The synthesis results showed that the proposed decoder provides important throughput and hardware efficiency. We showed that the throughput and the hardware efficiency are related to the average number of iterations and hence to E_b/N_0. Thus, starting from $E_b/N_0 = 3.7$ db where FER $= 10^{-2}$, the throughput efficiency of the proposed decoder outperforms its counterpart decoder architectures for different N, CR and GF values

Finally, the high level of the hardware emulation design was shown. We presented the AWGN channel along with the quantization operation which systemC coded to have a model that can be implemented in hardware. The design was tested on Kintex 7 FPGA device. We showed that the simulation results are too close to the emulation results. The throughput of the hardware emulation reached 500 Mbits/s.

Chapter 6

Conclusion and perspectives

6.1 Conclusion

This thesis has addressed the hardware design of a high throughput rate NB-LDPC decoder. Knowing that the optimization during the phase of hardware design is not sufficiently efficient, we have carefully reviewed the algorithmic optimization of the most important existing decoding algorithms.
After a careful investigation of the state-of-the-art NB-LDPC decoders, we have considered the EMS-based decoder for two reasons: 1) it is one of the most important sub-optimal decoders; 2) there is still room for algorithmic optimization. The focus was on the reduction of the messages being processed by the decoder, hence enabling the reduction of the hardware complexity and thus guaranteeing the freedom for higher degree of parallelism while designing.
In this context, we reviewed two main approaches for designing a CN: FB-CN and SB-CN, where FB-CN operates serially using a network of S-bubbles while SB-CN is based on parallel processing of messages. Both approaches have been re-implemented using the technique called "pre-sorting" applied to the messages entering the CN. Based on the LLR values, this pre-sorting technique permits to classify the entering symbols into two categories: 1) high reliability candidates and 2) low reliability candidates. The high reliability candidates carry out an inherent high entropy which helps the elimination of a large amount of their competitor candidates. However, the low reliability candidates, due to the low entropy they are carrying, they compete between them and this high competition requires a high computational complexity to identify the most reliable candidate. We showed that the pre-sorting technique allows a high complexity reduction of the FB-CN up to 54.

We then formulated our design strategy as: better algorithmic optimization, less computation, lower hardware complexity and more opportunities for parallelism. From this perspective we have proposed new CN architecture called EF-CN inspired by the SB-CN approach but with reduced complexity, increasing linearly (not exponentially as in SB-CN) with d_c. In order to allow further complexity reduction, we have com-

bined the EF-CN and FB-CN where some ECNs have been implemented allowing the reduction of the number of bubbles being processed and the execution time as well. A hybrid CN (HB) was then proposed and implemented efficiently with different configurations using the pre-sorting technique. The simulation results showed that the HB(6,4,2) gives similar performance as compared to FB-CN when designed over GF(64), while over GF(256) HB(5,5,2) is preferred to obtain same performance. The synthesis results confirmed the lower complexity of the EF-CN and HB-CN as compared to the FB-CN. The selection between the different HB configurations depends on the desired performance, throughput rate and area efficiency.
In order to avoid useless CN processing, we have introduced the "CN skip processing" approach that permits to skip the CN intended to be processed if the parity test of the symbols entering the CN is satisfied.
In addition to the CN optimization, we have proposed a new model of the VN. In this model, the redundancy elimination process is merged with the sorter, which implied some hardware reductions. The proposed VN has been implemented and the ASIC synthesis results showed that it consumes less area and operates at higher frequency compared to the VN proposed in [7].

Being of high importance and of high impact on the throughput of the decoder, the LLR generator and sorter have been carefully re-designed. We have proposed a new parallel pipelined architecture of LLR generator able to generate the n_m potential candidates in only 3 clock cycles offering a gain factors up to 4 in terms of hardware efficiency, and up to 15 in terms of throughput rate. Note that the latency depends on the number of pipeline layers. The specific case for n_m=4 has been implemented in the proposed decoder prototype. We have also proposed a new parallel sorting algorithm to extract the two extrema values among N_s. Compared to the existing architectures, the proposed architecture requires the lowest area and offers the highest frequency, where an area efficiency ranging from 1.17 up to 2 is obtained. We have also considered the generalization of the proposed algorithm to extract more than 2 extrema values.

Finally the global architecture of the NB-LDPC decoder was introduced. We have considered a Quasi-cyclic (120,144) NB-LDPC code, d_v=2 and d_c=6, code rate=5/6. The decoding algorithm has been described along with all the detailed specifications of the CN and VN parameters, and simulated using the flooding schedule. The average number of iterations at different SNRs has been calculated. The layered-schedule was not adopted since it would introduce idle time in the decoding process imposed by the VN and CN dependency. Compared to the FB-CN-based decoder with n_{it}=8 layered schedule (resp. flooding schedule), the proposed decoder, with flooding schedule and n_{it}=30, introduces a performance loss of 0.08 dB (resp. a gain of 0.05 dB). The proposed architecture has been synthesized on 18 nm ASIC technology and compared to three state-of-the-art decoder architectures [29, 58, 59]. In terms of area, i.e. NAND Gates consumption, the proposed decoder offers reduction factors of 5, 5.7

and 3.5 when compared to [29, 58, 59] respectively. In terms of throughput efficiency, the proposed decoder starts to outperform [29] at Eb/N0=3 dB, [58] at 3.7dB and [59] at 3dB. This work has been ended by the design of an emulation chain using the high level synthesis tool VIVADO, where the AWGN channel has been modeled using sytemC. The obtained emulations results matched the software simulation results.

6.2 Perspectives

The work presented in this report does not totally close the topic of efficient NB-LDPC decoding implementation. There are still several development tasks to be considered:

- We believe that the optimization effort of the Hybrid architecture is not fully completed yet. There is still some freedom to optimize the architecture. In this direction, we would particularly focus our effort on the Variable Node architecture.
- Optimization of parallel Hybrid architecture for all value of d_c and GF(q).
- Automatic generation of the hardware architecture from a given NB-LDPC matrix.
- Develop a generic hardware architecture able to process a large variety of NB-LDPC codes in terms of code rate, GF order, size, ..., etc.

α^2	0	12	63	13	37	48	0
α^1	1	2	12	0	0	0	12
α^0	19	54	36	47	1	53	25
0	46	0	0	1	3	64	3

Table 6.1: Example of messages used for T-EMS

α^2	0	12	63	13	37	48	0
α^1	1	2	12	0	0	0	12
α^0	19	54	36	47	1	53	25
0	46	0	0	1	3	64	3

Table 6.2: Example of messages used for EMS

Moreover, there are two competing algorithms to decode at high speed NB-LDPC so far: the Hybrid architecture and the T-MM architecture [51]. If both algorithms target a simplified check node algorithm, they are almost orthogonal in the way of processing the incoming messages. For example, in the example of Table 6.1, the gray

cells show the incoming information used by the Trellis EMS algorithm while Table 6.2 shows the incoming information used by the EMS algorithm for $n_m = 2$. In the T-EMS there are two values used per row, while in the EMS there are two (n_m in the general case) values used per column. A first direction of investigation would be having an accurate comparison between those two architectures in terms of complexity, flexibility, decoding performance, ..., etc. Later, an investigation direction would be to merge those two approaches to propose new and hopefully more efficient algorithms.

Finally, all the work presented in this thesis has been applied for NB-LDPC only. An interesting research direction would be to see if the acquired knowledge can also be applied for other type of Non-Binary code like the Non-Binary Turbo-code [94].

Chapter 7

Appendix A

A.1 Introduction of the Galois field

Modern algebra is characterized by a high level of abstraction. Indeed, classical algebra studies $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{C}$ sets, built with arithmetic operations such as addition and multiplication. As for modern algebra, the notion of operation (or rule of composition) takes on a more complex dimension and it is defined as an application which, in generalized sets, associates two or more symbols with another symbol. Coding theory has taken advantage of this abstraction to extend the definition of error-correcting codes to set new classical sets than the mentioned above. In this thesis manuscript, we are particularly interested in the case of non-binary LDPC codes defined on the Galois Fields (GF). In order to give a complete definition of the Galois fields, we begin by describing the basic algebraic structures with rules of internal composition. The content of this section is mainly extracted from [8], [9], [10] and [11].

A.1.1 Algebraic structures

Let us consider two sets E and K.

Definition 1.1. *A law of internal composition on E is an applied operation on a couple $(x, y) \in E \times E$ that gives an element $z \in E$.*

Definition 1.2. *A law of external composition on E is an applied operation on a couple $(x, y) \in K \times E$ that gives an element $z \in E$.*

A law of composition is generally noted «$*$». We particularly distinguish the additive law noted «$+$» and the multiplicative law noted «.». We call compound of an element x by an element y, the unique element $x * y$ associated to the law «$*$» of the couple (x, y).

Definition 1.3. *A basic algebraic structure is a set provided with one or more laws of internal composition.*

Definition 1.4. *An algebraic structure S is finite if it contains a finite number of elements. The number of elements of S is then denoted $|S|$ and it is called the order of the algebraic structure.*

A.1.2 The groups

Definition 1.5. *A group is a set G with a law of internal composition «$*$» as :*
▷ *«$*$» is associative : $\forall a, b, c \in G, (a * b) * c = a * (b * c)$.*
▷ *«$*$» has a neutral element $e \in G$: $\forall a \in G, a * e = e * a$.*
▷ *$\forall a \in G$ there is a symmetric element $b \in G$: $a * b = b * a = e$.*
Group G becomes abelian (in honor of Niels Abel) if «$*$» is also commutative: $\forall a, b \in G, a * b = b * a$.

The neutral element e is unique. In addition, $\forall a \in G$, its symmetrical b is unique. The associativity of the law of composition guarantees that the expression $a_1 * a_2 * \cdots * a_n$ represents a unique element of G regardless the position of the parentheses.

The group G will be called additive in case of using the additive notation of the law of composition.The symmetric element of a (or the opposite of a) is then noted $-a$ and the neutral element is denoted 0. In case the multiplicative notation is used, the group will be called multiplicative, the symmetrical element of a (or the inverse of a) is denoted a^{-1} and the neutral element is denoted 1.

We use the following definitions to indicate the n-times compound of an x element with itself :
▷ Notation of additive : $nx = x + x + \cdots + x$, n times.
▷ Notation of Multiplicative : $x^n = x.x. \ ... \ x$, n times.

Table TA.1 gives some conventional rules for the two multiplicative and additive notations.

Subtraction and division operations are defined as a function of the symmetric element :
▷ Subtraction : $a - b = a + (-b)$.
▷ division : $\frac{a}{b} = a.b^{-1}$.

Table TA.1: Conventional rules of both multiplicative and additive notations

Notation of Multiplicative	Notation of additive
$a^0 = 1$	$0a = 0$
$a^{-n} = (a^{-1})^n$	$(-n)a = n(-a)$
$a^{n+m} = a^n.a^m$	$(n+m).a = n.a + m.a$
$a^{nm} = (a^n)^m$	$(nm)\,a = n\,(ma)$

A.1.3 The rings

Definition 1.6. *A ring (A,+,.) Is a set with two laws of internal composition «+» and «.» such that :*
▷ *A with «+» is an abelian group.*
▷ *«.» is associative :* $\forall a, b, c \in A, (a.b).c = a.(b.c)$.
▷ *«.» is distributive to «+» :* $\forall a, b, c \in A, (a+b).c = a.c+b.c$ *and* $c.(a+b) = c.a+c.b$.
▷ *«.» has an neutral element.*
▷ *A is commutative if «.» is commutative :* $\forall a, b \in A : a.b = b.a$.

The neutral element of «+» is noted 0 and so for «.» is noted 1. We use «+» and «.» to indicate that the two laws of internal composition of a ring satisfy some of the properties of addition and multiplication of the relative integer numbers. However, we must always keep in mind the definition of a composition law given in subsection 1.1.1.

A.1.4 Congruence and modular arithmetic in $\mathbb{Z}$

Definition 1.7. *Let a and b two integers, and n is strictly positive integer.*

- ▷ *We say that a is congruent to b* modulo *n if n divides a − b. We use the notation* $a \equiv b \pmod{n}$.
- ▷ $a \equiv b \pmod{n}$ *means in an equivalent way that b is the rest of Euclidean division of a by n. We use the notation* $p = a \bmod n$.

We then get the following equivalences:
$a \equiv b \pmod{n} \iff \exists k \in \mathbb{Z}/a = b + k.n \iff b = a \bmod n$.

- ▷ *The additive operation of a and b* modulo *n is of definition :*
  $a \oplus_n b = (a+b) \bmod n$
- ▷ *The multiplicative operation of a and b* modulo *n is of definition :*
  $a \otimes_n b = (a.b) \bmod n$.

Table TA.2: *modulo* 2 addition

$\oplus_2$	0	1
0	0	1
1	1	0

Table TA.3: *modulo* 2 multiplication

$\oplus_2$	0	1
0	0	0
1	0	1

Particularly, the addition and multiplication operations in the set $\mathbb{Z}_2$={0, 1} correspond to the two logical functions XOR and AND as it is shown in Table TA.2 and Table TA.3.

It is simple to prove that, in general, the set $\mathbb{Z}_n$={0, 1, 2, ... , n-1} provided with both laws of internal composition $\oplus$ and $\otimes$ forms a commutative ring.

A.1.5 Galois field

Definition 1.8. *A field $(C, +, .)$ is a set with two laws of internal composition «+» and «.» such that :*
▷ *C with «+» is a commutative ring.*
▷ $\forall a \neq 0 \in C, \exists a^{-1} \in C / a.a^{-1} = 1$.

Definition 1.9. *A finite field is a field having a finite number of elements. A finite field is usually called Galois field and it is noted as GF. The order (or cardinal) of a Galois field is the number of its elements.*

It is easy to demonstrate that the commutative ring $(\mathbb{Z}_2, \oplus_2, \otimes_2)$ forms a Galois field of order 2. In general, we can prove that for any prime number p, the commutative ring $(\mathbb{Z}_p, \oplus_p, \otimes_p)$ is a Galois field of order p and it is denoted GF(p).

Definition 1.10. *Let C is a field and K is a subset of C. If K of laws of internal composition is also a field then we say that K is a subfield of C. Equivalently, C is called extension of body K.*

We can show that for any prime number p and any positive integer m, the commutative ring $(\mathbb{Z}_{p^m}, \oplus_{p^m}, \otimes_{p^m})$ forms a finite field. $\mathbb{Z}_{p^m}$ is an extension of the Galois field GF(p). It is also called Galois field of order $q = p^m$ and it is denoted GF(q). In particular, Galois fields of order $q = 2^m$, with m is a positive integer, are of great interest in practice, particularly in coding theory. Indeed, as we will show below, an element belongs to a Galois field GF(2^m) can be uniquely represented in the form of a binary symbol of m bits.

Definition 1.11. *We say that a set is closed for an operation if this operation always produces an element of the set when it is applied to any element belongs to it.*

A Galois field GF(q) is closed for the two internal compositions $\oplus_q$ and $\otimes_q$.

A.1.6 The polynomials on GF(q)

Definition 1.12. *A polynomial f defined on the Galois field GF(q) is an expression of the form :*

$$f(X) = \beta_n X^n + \beta_{n-1} X^{n-1} + \cdots + \beta_1 X + \beta_0$$

In which the coefficients β_i, $i = 0, 1, \ldots, n$, are elements belong to GF(q) and X is a formal symbol called indeterminate polynomial. the positive integer n is called the degree of the polynomial and it is noted deg(f).

Definition 1.13. *Lets take two polynomials $f(X) = \sum_{i=0}^{n} a_i X^i$ and $g(X) = \sum_{i=0}^{m} b_i X^i$ where $m \leq n$. the polynomial g(X) can be formulated like $g(X) = \sum_{i=0}^{n} b_i X^i$ considering that the coefficients b_i are nulls for all $i > m$. the following definitions will be obtained :*

▷ *f=g if and only if $\forall i \in 0, 1, \ldots, n, a_i = b_i$.*

▷ *The additive operation defined on the polynomials is :*

$$f(X) + g(X) = \sum_{i=0}^{n} (a_i \oplus_q b_i) X^i$$

▷ *The multiplicative operation defined on the polynomials is :*

$$f(X).g(X) = \sum_{k=0}^{n+m} c_k X^k, \text{ where } c_k = \bigoplus_{\substack{i+j=k \\ i\in\{0,1,\ldots,n\} \text{ and } j\in\{0,1,\ldots,m\}}} a_i \otimes_q b_j$$

▷ *The set $F_q[X]$ of polynomials of indeterminate X and coefficients in GF(q) with the multiplication and addition operations are rings.*

The theorem of Euclidean division can be generalized on polynomials. Thus, if g is a non-zero polynomial in $F_q[X]$ then for every polynomial f of $F_q[X]$ there are two polynomials q and r in $F_q[X]$ such that :

$$f(X) = q(X).g(X) + r(X), \text{ where } deg(f) < deg(g)$$

Definition 1.14 *Let f and $g \in F_q[X]$ are two polynomials.*

- ▷ *We say that g is a divider of f if there is a polynomial $q \in F_q[X]$ such that $f(X) = q(X).g(X)$.*
- ▷ *f is irreducible in $F_q[X]$ if deg(f)>0 and f can not be factored by multiplication of two polynomials of degree > 0 each. In other word, if f=q.g then $deg(q) = 0$ or $deg(g) = 0$.*
- ▷ *an irreducible polynomial f of degree m is primitive if $X^n + 1 = f(X).g(X)$ implies that $n \geq 2^m$.*
- ▷ *An element $\beta \in \mathrm{GF}(q)$ is a root of polynomial $f \in F_q[X]$ if f(β)=0. In equivalent way, we can prove that β is a root of f if the polynomial (X-β) is a divider of f.*

A.1.7 Construction of the Galois field GF(2^m)

Let p is a primitive polynomial of degree m and coefficients in GF(2). This polynomial does not have a root in GF(2). However, in abstract algebra, we can imagine that it has a root in another set (by analogy to polynomials with coefficients in $\mathbb{R}$ that may have one or more roots in $\mathbb{C}$). We consider the two elements 0 and 1 in GF(q) and the new element β. Defining the multiplicative operation denoted «.» as the following :
▷ 0 is the absorbent element of the multiplication : $0.\beta = \beta.0 = 0.1 = 1.0 = 0.0 = 0$
▷ 1 is the neuter element of the multiplication : $1.\beta = \beta.1 = \beta$ and $1.1 = 1$
▷ The composition n-times of the element β with itself is noted $\beta_n = \beta.\beta. \ldots .\beta.$. By convention $\beta_0 = 1$.
▷ $\forall i, j \in \mathbb{N},\ \beta_i.\beta_j = \beta_j.\beta_i = \beta_{i+j}$

p is being a primitive polynomial of degree m implies :

$$X^{2^m-1} + 1 = q(X).p(X)$$

By replacing X by β, we will obtain

$$\beta_{2^m-1} + 1 = q(\beta).p(\beta) = q(\beta).0 = 0$$

And then $\beta_{2^m-1} = 1$. Consequently, the set $F = \{0,1,\beta,\beta_2,...,\beta_{2^m-2}\}$ with the law of «.» is a finite set of order 2^m.

Through this section, we will show that the set F, with the law of multiplication «.» and the law of addition «+» forms a Galois field of order 2^m.

We begin by defining the law of addition so that $(F, +)$ forms an abelian group. For this, we observe that each element β_i of F can be represented in a unique way by a nonzero polynomial of degree strictly inferior to m. Indeed, the Euclidean division of the monomial X^i, $i = 0, 1, \ldots, 2^m - 2$, by p gives $X^i = q_i(X).p(X) + a_i(X)$, where $a_i(X) = a_{i0} + a_{i1}X + a_{i2}X^2 + ... + a_{i(m-1)}X^{m-1}$ and the coefficients $a_{ij} \in \{0, 1\}$. The polynomials $a_i(X)$ are necessarily non-zero because X^i and p are prime with each other. Moreover, it is easy to prove that $a_i(X) \neq a_j(X)$ if $i \neq j$. Since β is a root of p so $\beta_i = a_i(\beta), i = 0, 1, ..., 2^m - 2$. We just shown that each non-zero element of F is represented by a polynomial $a_i(X)$. By convention, the element 0 of F is represented by the null polynomial. Each element of F also has a binary representation considering only the coefficients of its polynomial representation. The law of addition is defined as follows :
▷ 0+0=0
▷ 0 is the neutral element of the addition : $0 + \beta_i = \beta_i + 0 = \beta_i, i = 0, 1, \ldots, 2^m - 2$
▷ $\beta_i + \beta_j = a_i(\beta) + a_j(\beta) = \sum_{k=0}^{m-1}(a_{ik} \oplus_2 b_{jk})X^i, 0 \leq i, j \leq 2^m - 2.$

It is easy to prove that the set $(F,+,.)$ is a commutative ring. Moreover, we observe that $\beta_i.\beta_j = \beta_{(i+j)mod(2^m-1)}$. It means that each non-zero element β_i of F has an inverse equal to β_{2^m-1-i}. In conclusion $(F,+,.)$ Forms a Galois field of order 2^m.

Publications

H. Harb, C. Marchand, A. A. Ghouwayel, L. Conde-Canencia, and E. Boutillon, "*Pre-sorted forward-backward NB-LDPC check node architecture,*" in IEEE International Workshop on Signal Processing Systems (SiPS), Oct 2016, pp. 142-147, Dallas, USA.

Titouan Gendron, Hassan Harb, Alban Derrien, Cédric Marchand, Laura Conde-Canencia, Bertand Le Gal and Emmanuel Boutillon, "*Demo: Construction of good Non-Binary Low Density Parity Check codes*", Demo night at SIPS'2017, Lorient, France, Oct. 2017.

C. Marchand, H. Harb, E. Boutillon, A. Al Ghouwayel, and L. Conde-Canencia, "*Extended-forward architecture for simplied check node processing in NB-LDPC de-coders,*" in IEEE International Workshop on Signal Processing Systems (SiPS), October 2017, Lorient, France.

Cédric Marchand, Emmanuel Boutillon, Hassan Harb, Laura Conde-Canencia and Ali Al Ghouwayel, "Hybrid Check Node Architectures for NB-LDPC Decoders", Accepted in IEEE Transactions on Circuits And Systems-I, August 2018.

Hassan Harb, Emmanuel Boutillon, Bertrand Le Gal, "Real-time evaluation of NB-LDPC codes thanks to HLS-based hardware emulation", Demo night at DASIP'2018, Porto, Portugal, Oct. 2018.

Ali Al-Ghouwayel, Member, IEEE, Hassan Harb and Emmanuel Boutillon, Senior Member, IEEE, "*First-Then-Second Extrema Selection,*". Submitted.

Hassan Harb, Ali Al Ghouwayel, Cédric Marchand, Laura Conde-Canencia, Emmanuel Boutillon, "*Throughput Rocket EMS NB-LDPC Decoder Based On A Parallel And Pipelined Architecture,*". In preparation.

Hassan Harb, Ali Al Ghouwayel and Emmanuel Boutillon, "*Parallel pipelined LLR generator,*". In preparation.

Bibliography

[1] R. G. Gallager. *Low-density parity-check codes*, Cambridge, MA: MIT. Press, 1963.

[2] The Digital Video Broadcasting Project. http://www.dvb.org/.

[3] The 3rd Generation Partnership Project. http://www.3gpp.org/.

[4] Jeffrey G Andrews, Arunabha Ghosh, and Rias Muhamed. *Fundamentals of WiMAX : understanding broadband wireless networking.*, Prentice Hall, Upper Saddle River, NJ, 2007.

[5] Claude Elwood Shannon and Warren Weaver. *The mathematical theory of communication.*, University of Illinois Press, Urbana, 1964.

[6] Shu Lin. *Error control coding : fundamentals and applications.*, Pearson-Prentice Hall, Upper Saddle River, N.J, 2004.

[7] Oussama Abassi, Laura Conde-Canencia and Emmanuel Boutillon. *Study of decoders Non-Binary LDPC.* Prepared by UMR 6285 Sud Brittany University Lab-STICC.

[8] Rudolf Lidl and Harald Niederreiter. *Finite fields.*, Cambridge University Press, Cambridge, 2008.

[9] Jean-Pierre Deschamps and Gustavo D Sutter. *Hardware implementation of finite field arithmetic.*, McGraw-Hill, New York, 2009.

[10] John M. Howie. *Fields and Galois Theory (Springer Undergraduate Mathematics Series).*, McGraw-Hill, Springer, 2007.

[11] David Forney. Introduction to finite fields. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-451-principles-of-digital-communication-ii-spring-2005/lecture-notes/chap7.pdf.

[12] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. *In Communications, 1993. ICC '93 Geneva. Technical Program, Conference Record, IEEE International Conference on,* volume 2, pages 1064-1070 vol.2, 1993.

[13] C. E. Shannon. *A mathematical theory of communication.* Bell System Technical Journal, 27 :379-423 and 623-656, 1948.

[14] R. G. Gallager. *Low-density parity-check codes.* PhD thesis, MIT, Cambridge, Mass., September 1960.

[15] D. J. C. MacKay and R. M. Neal. *Near Shannon limit performance of low density parity check codes.* Electron. Lett., 32(18) :1645-1646, August 1996.

[16] D. J C MacKay. *Good error-correcting codes based on very sparse matrices.* Information Theory, IEEE Transactions on, 45(2) :399-431, 1999.

[17] Jorge Moreira. *Essentials of error-control coding.* John Wiley & Sons, West Sussex, England, 2006.

[18] R.M. Tanner. *A recursive approach to low complexity codes.* Information Theory, IEEE Transactions on, 27(5) :533-547, 1981.

[19] M.C. Davey and D. MacKay. *Low density parity check codes over GF(q).* Communications Letters, IEEE, 2(6) :165-167, 1998.

[20] Hongxin Song and J.R. Cruz. *Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording.* Magnetics, IEEE Transactions on, 39(2) :1081-1087, 2003.

[21] L. Barnault and D. Declercq. *Fast decoding algorithm for LDPC over* $gf(2^q)$. In information Theory Workshop, 2003. Proceedings. 2003 IEEE, pages 70-73, 2003.

[22] H. Wymeersch, H. Steendam, and M. Moeneclaey. *Fast decoding algorithm for LDPC over GF(*$q > 2$*).* In Communications, 2004 IEEE International Conference on, volume 2, pages 772-776 Vol.2, 2004.

[23] P Schläfer, N Wehn, M Alles, T Lehnigk-Emden, E Boutillon. *Syndrome based check node processing of high order NB-LDPC decoders.* International Conference on Telecommunications, Apr 2015, Sydney, Australia..

[24] D. Declercq and M. Fossorier. *Decoding Algorithms for Nonbinary LDPC Codes Over GF(*q*).* Communications, IEEE Transactions on, 55(4) :633-643, 2007.

[25] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard. *Low-Complexity, Low-Memory EMS Algorithm for Non-Binary LDPC Codes.* In Communications, 2007. ICC 07. IEEE International Conference on, pages 671-676, 2007.

[26] L. Conde-Canencia, E. Boutillon, and A. Al-Ghouwayel. *Complexity comparison of non-binary ldpc decoders.* In proceedings of ICT Mobile Summit, Spain, June 2009.

[27] V. Savin. *Min-Max decoding for non binary LDPC codes.* In Information Theory, 2008. ISIT 2008. IEEE International Symposium on, pages 960-964, 2008.

[28] Erbao Li, K. Gunnam, and D. Declercq. *Trellis based Extended Min-Sum for decoding nonbinary LDPC codes.* In Wireless Communication Systems (ISWCS), 2011 8th International Symposium on, pages 46-50, Nov 2011.

[29] Erbao Li, D. Declercq, and K. Gunnam. *Trellis-Based Extended Min-Sum Algorithm for Non-Binary LDPC Codes and its Hardware Structure.* Communications, IEEE Transactions on, 61(7) :2600-2611, July 2013.

[30] Design And Versatile Implementation of Non-binary wireless Communications based on Innovative LDPC codes. http://www.ict-davinci-codes.eu/.

[31] E. Boutillon, L. Conde-Canencia, and A. Al Ghouwayel. *Design of a GF(64)-LDPC decoder based on the EMS algorithm.* Circuits and Systems I : Regular Papers, IEEE Transactions on, 60(10) :2644-2656, 2013.

[32] L. Song, Q. Huang, Z. Wang, M. Zhang, and S. Wang. *Two Enhanced Reliability-Based Decoding Algorithms for Nonbinary LDPC Codes.* IEEE Transactions on Communications, vol. 64, no. 2, pp. 479-489, Feb 2016.

[33] C. Y. Chen, Q. Huang, C. C. Chao, and S. Lin. *Two Low-Complexity Reliability-Based Message-Passing Algorithms for Decoding Non-Binary LDPC Codes.* IEEE Transactions on Communications, vol. 58, no. 11, pp. 3140-3147, 2010.

[34] C. Xiong and Z. Yan. *Improved Iterative Hard- and Soft-Reliability Based Majority-Logic Decoding Algorithms for Non-Binary Low-Density Parity-Check Codes.* in 2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), Nov 2011, pp. 894-898.

[35] X. Zhang, F. Cai, and S. Lin. *Low-Complexity Reliability-Based Message-Passing Decoder Architectures for Non-Binary LDPC Codes.* IEEE Transactions on Very Large Scale Integration Systems, vol. 20, no. 11, pp. 1938-1950, 2012.

[36] F. Garcia-Herrero, D. Declercq, and J. Valls. *Non-Binary LDPC Decoder Based on Symbol Flipping with Multiple Votes.* IEEE Communications Letters, vol. 18, no. 5, pp. 749-752, 2014.

[37] A.A. Ghouwayel and E. Boutillon. *A Systolic LLR Generation Architecture for Non-Binary LDPC Decoders.* Communications Letters, IEEE, 15(8) :851-853, 2011.

[38] Youngjoo Lee, *Member, IEEE,* Bongjin Kim, Student *Member, IEEE,* Jaehwan Jung, Student *Member, IEEE,* and In-Cheol Park, Senior *Member, IEEE Low-Complexity Tree Architecture for Finding the First Two Minima.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-II: EXPRESS BRIEFS, VOL. 62, NO. 1, JANUARY 2015.

[39] E. Boutillon and L. Conde-Canencia. *Bubble check: a simplified algorithm for elementary check node processing in extended min-sum non-binary LDPC decoders.* IEEE Electron. Lett., vol. 46, no. 9, pp. 633-634, Apr. 2010.

[40] Voicila, A., Declercq, D., Verdier, F., Fossorier, M., and Urard, P. *Low-Complexity, Low-Memory EMS Algorithm for Non-Binary LDPC Codes.* Proc. of IEEE Int. Conf. on Commun., ICC'2007, Glasgow, United Kingdom, June 2007.

[41] http://www-labsticc.univ-ubs.fr/nb_ldpc/

[42] E. Boutillon and L. Conde-Canencia. *Simplified check node processing in non-binary LDPC decoders.* In Turbo Codes and Iterative Information Processing (ISTC), 2010 6th International Symposium on, pages 201-205, 2010.

[43] Oussama Abassi, Laura Conde-Canencia, Ali Al Ghouwayel and Emmanuel Boutillon *A Novel Architecture for Elementary-Check-Node Processing in Nonbinary LDPC Decoders.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-II: EXPRESS BRIEFS, VOL. 64, NO. 2, FEBRUARY 2017.

[44] Koch, Dirk and Torresen, Jim *A High Performance Sorting Architecture Exploiting Run-time Reconfiguration on FPGAs for Large Problem Sorting.* Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pages 45-54, year 2011.

[45] J. Martinez and R. Cumplido and C. Feregrino *An FPGA-based parallel sorting architecture for the Burrows Wheeler transform.* 2005 Int. Conf. on Reconfigurable Computing and FPGAs, Sept. 2005.

[46] Emmanuel Boutillon and Laura Conde-Canencia. Procédé de commande d'une unité de calcul, tel qu'un noeud de parité élémentaire dans un décodeur de code LDPC non binaire, et unité de calcul correspondante. patent no. FR0952988, May 2009.

[47] C. Marchand and E. Boutillon. *NB-LDPC check node with pre-sorted input.* in 9th International Symposium on Turbo Codes & Iterative Information Processing, September 2016.

[48] Harb, H. and Marchand, C. and Al Ghouwayel and A. Conde-Canencia and L. and Boutillon, E. *Pre-Sorted Forward-Backward NB-LDPC Check Node Architecture.* in SIPS, 2016.

[49] Cedric Marchand, Emmanuel Boutillon, Hassan Harb, Laura Conde-Canencia, and Ali Al Ghouwayel. *Extended-forward architecture for simplied check node processing in NB-LDPC decoders,.* in IEEE International Workshop on Signal Processing Systems (SiPS), October 2017, Lorient, France.

[50] Marchand, C. Harb, H. Boutillon, E. Al Ghouwayel, A. and Conde-Canencia, L. *An Efficient Decoder Architecture for Nonbinary LDPC Codes With Extended Min-Sum Algorithm.* IEEE Transactions on Circuits and Systems II: Express Briefs, VOL. 63, NO. 9, Sept 2016.

[51] J. O. Lacruz, F. Garcia-Herrero, M. J. Canet, and J. Valls *Reduced-Complexity Non-Binary LDPC Decoder for High-Order Galois Fields Based on Trellis Min-Max Algorithm.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 8, pp. 2643-2653, Aug 2016.

[52] Y. L. Ueng and C. Y. Leong and C. J. Yang and C. C. Cheng and K. H. Liao and S. W. Chen. *An Efficient Layered Decoding Architecture for Nonbinary QC-LDPC Codes.* IEEE Transactions on Circuits and Systems I: Regular Papers, VOL. 59, NO. 2, Feb 2012.

[53] C. Marchand and J. B. Dore and L. Conde-Canencia and E. Boutillon. *Conflict resolution for pipelined layered LDPC decoders.* 2009 IEEE Workshop on Signal Processing Systems, Oct 2009.

[54] P. Schläfer. *Implementation Aspects of Binary and Non-Binary Low-Density Parity-Check Decoders.* Technische UniversitÃ¤t Kaiserslautern, 2016.

[55] Amin Farmahini-Farahani, Henry J. Duwe III, Michael J. Schulte, and Katherine Compton *Modular Design of High-Throughput, Low-Latency Sorting Units.* IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, NO. 7, JULY 2013.

[56] C. Poulliat, M. Fossorier, and D. Declercq, *Design of regular (2,dc)-ldpc codes over GF(q) using their binary images,.* IEEE Transactions on Communications, vol. 56, no. 10, pp. 1626-1635, October 2008.

[57] F. Cai and X. Zhang. *Relaxed Min-Max Decoder Architectures for Nonbinary Low-Density Parity-Check Codes.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 11, pp. 2010-2023, Nov 2013.

[58] J. Tian, J. Lin, and Z. Wang. *A 21.66Gbps Non-Binary LDPC Decoder for High-Speed Communications.* IEEE Transactions on Circuits and Systems II: Express Briefs, vol. PP, no. 99, pp. 1-1, 2017.

[59] Y. S. Park, Y. Tao, and Z. Zhang. *A Fully Parallel Nonbinary LDPC Decoder With Fine-Grained Dynamic Clock Gating.* IEEE Journal of Solid-State Circuits, vol. 50, no. 2, pp. 464-475, Feb 2015.

[60] P Schlafer, N Wehn, M Alles, T Lehnigk-Emden, E Boutillon. *Syndrome based check node processing of high order NB-LDPC decoders.* International Conference on Telecommunications, Apr 2015, Sydney, Australia. <hal-01151980>.

[61] Z. Guo and P. Nilsson. *Algorithm and implementation of the K-best sphere decoding for MIMO detection.* IEEE J. Sel. Areas Commun., vol. 24, no.3, pp. 491503, Mar. 2006.

[62] R. M. Pyndiah. *Near-optimum decoding of product codes: block turbo codes.* IEEE Trans. Commun., vol. 46, no. 8, pp. 10031010, Aug. 1998.

[63] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X. Y. Hu, *Reduced-complexity decoding of LDPC codes.* IEEE Trans. Commun., vol. 53, no. 8, pp. 12881299, Aug. 2005.

[64] S. Zezza, S. Nooshabadi, and G. Masera, *A 2.63 Mbit/s VLSI Implementation of SISO Arithmetic Decoders for High Performance Joint Source Channel Codes.* IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 60, no. 4, pp. 951964, Apr. 2013.

[65] S. Papaharalabos, P. T.Mathiopoulos, G.Masera, and M.Martina, *Non-recursive max* operator with reduced implementation complexity for turbo decoding.* IET Commun., vol. 6, no. 7, pp. 702-707, Jul. 2012.

[66] K.E. Batcher. *Sorting Networks and Their Applications.* Proc. AFIPS Proc. Spring Joint Computer Conf., pp. 307-314, 1968.

[67] A. Farmahini-Farahani and al. *Modular Design of High-Throughput, Low-Latency Sorting Units.* Computers, IEEE Trans., vol. 62, no. 7, pp. 13891402, 2013.

[68] G. Xiao, M. Martina, G. Masera, *A Parallel Radix-Sort-Based VLSI Architecture for Finding the First W Maximum/Minimum Values.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-II: EXPRESS BRIEFS, VOL. 61, NO. 11, NOVEMBER 2014.

[69] K. Gunnam, G. Choi, and M. Yeary, *A Parallel VLSI Architecture for Layered Decoding for Array LDPC Codes.* in Proc. IEEE Int. Conf. VLSI Design, 2007, pp. 738743.

[70] C. Condo, M. Martina, and G. Masera, *VLSI Implementation of a Multi-Mode Turbo/LDPC Decoder Architecture.* IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 60, no. 6, pp. 14411454, Jun. 2013.

[71] C. L. Wey, M. D. Shieh, and S. Y. Lin, *Algorithms of Finding the First Two Minimum Values and Their Hardware Implementation.* IEEE Trans. on Circuits and Systems I, vol. 55, no. 11, pp. 34303437, Dec 2008.

[72] G. Masera L.G. Amaru, M. Martina, *High Speed Architectures for Finding the First two Maximum/Minimum Values.* IEEE Transactions on Very Large Scale Integration, vol. 20, no. 12, pp. 23422346, 2012.

[73] D. E. Knuth, The Art of Computer Programming, 2nd ed. New York: Addison-Wesley, 1998, vol. 3, Sorting and Searching, sec. 5.3.3.

[74] D. Zhao, X. Ma, C. Chen, and B. Bai, *A Low Complexity Decoding Algorithm for Majority-Logic Decodable Nonbinary LDPC Codes.* IEEE Commun. Lett., vol. 14, no. 11, pp. 1062-1064, Nov. 2010.

[75] C.-Y. Chen, Q. Huang, and C.-C. Chao, *Low-Complexity Reliability-Based Message-Passing Decoder Architectures for Non-Binary LDPC Codes.* IEEE Trans. Commun., vol. 58, no. 11, pp. 3140-3147, Nov. 2010.

[76] Ying Yu Tai, *Student Member, IEEE*, Lan Lan, Lingqi Zeng, Shu Lin, *Life Fellow, IEEE*, and Khaled A. S. Abdel-Ghaffar, *Member, IEEE*, *Algebraic construction of quasi-cyclic LDPC codes for the AWGN and erasure channels.* IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. 54, NO. 10, OCTOBER 2006.

[77] Shumei Song, Lingqi Zeng, Shu Lin and Khaled Abdel-Ghaffar Department of Electrical and Computer Engineering University of California, Davis, Davis, CA 95616, U.S.A, *Algebraic Constructions of Nonbinary Quasi-Cyclic LDPC Codes.* ISIT 2006, Seattle, USA, July 9 14, 2006.

[78] Lingqi Zeng, Lan Lan, Ying Yu Tai, Bo Zhou, Shu Lin, and Khaled A. S. Abdel-Ghaffar, *Construction of nonbinary cyclic, quasi-cyclic and regular LDPC codes: a finite geometry approach.* IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. 56, NO. 3, MARCH 2008.

[79] S. Azuma, T. Sakuma, T. Takeo, T. Ando, and K. Shirai, *Diaprism Hardware Sorter - Sort aMillion Records within a Second.* http://sortbenchmark.org/Y2000_Datamation_DiaprismSorter.pdf, 2000.

[80] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha, *GPUTeraSort: High Performance Graphics Co-Processor Sorting for Large Database Management,* Proc. Conf. Management of Data, pp. 325-336, 2006.

[81] D. Koch and J. Torresen, *FPGASort: A High Performance Sorting Architecture Exploiting Run-Time Reconfiguration on FPGAs for Large Problem Sorting,* Proc. Symp. Field Programmable Gate Arrays, pp. 45-54, 2011.

[82] D. Pok, C.-I. Chen, J. Schamus, C. Montgomery, and J. Tsui, *Chip design for monobit receiver,* IEEE Trans. Microwave Theory and Techniques, vol. 45, no. 12, pp. 2283-2295, Dec. 1997.

[83] I. Pitas and A.N. Venetsanopoulos, *Nonlinear Digital Filters: Principles and Applications,* Kluwer Academic Publishers, 1990.

[84] J.P. Agrawal, *Arbitrary size bitonic (ASB) sorters and their applications in broadband ATM switching,* Proc. IEEE Int'l Conf. Computers and Comm., pp. 454-458, Mar. 1996.

[85] K. Yun, K. James, R. Fairlie-Cuninghame, S. Chakraborty and R. Cruz, *A self-timed real-time sorting network,* IEEE Trans. Very Large Scale Integration Systems, vol. 8, no. 3, pp. 356-363, June 2000.

[86] A. Colavita, E. Mumolo, and G. Capello, *A novel sorting algorithm and its application to a gamma-ray telescope asynchronous data acquisition system,* Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 394, no. 3, pp. 374-380, 1997.

[87] D.C. Stephens, J.C. Bennett, and H. Zhang, *Implementing scheduling algorithms in high-speed networks,* IEEE J. Selected Areas in Comm, vol. 17, no. 6, pp. 1145-1158, June 1999.

[88] V. Brajovic and T. Kanade, *A VLSI sorting image sensor: global massively parallel intensity-to-time processing for low-latency adaptive vision,* IEEE Trans. Robotics and Automation, vol. 15, no. 1, pp. 67-75, Feb. 1999.

[89] C. Chakrabarti and L.-Y. Wang, *Novel sorting network-based architectures for rank order filters,* IEEE Trans. Very Large Scale Integration Systems, vol. 2, no. 4, pp. 502-507, Dec. 1994.

[90] S.-N. Dong, X.-T. Wang, and X.-B. Wang, *A Novel High-Speed Parallel Scheme for Data Sorting Algorithm Based on FPGA,* Proc. Int'l Cong. Image and Signal Processing, pp. 1-4, Oct. 2009.

[91] K. Ratnayake and A. Amer, *An FPGA Architecture of Stable-Sorting on a Large Data Volume : Application to Video Signals,* Proc. Ann. Conf. Information Sciences and Systems, pp. 431-436, 2007.

[92] A. Gregerson, M. Schulte, and K. Compton, *High-Energy Physics,* Handbook of Signal Processing Systems, pp. 179-211, Springer, 2010.

[93] Titouan Gendron, Hassan Harb, Alban Derrien, Cédric Marchand, Laura Conde-Canencia, Bertand Le Gal and Emmanuel Boutillon, *Demo: Construction of good Non-Binary Low Density Parity Check codes.* Demo night at SIPS'2017, Lorient, France, Oct. 2017.

[94] Rami Klaimi, Charbel Abdel Nour, Catherine Douillard, and Joumana Farah, *Design of Low-Complexity Convolutional Codes over GF(q).* ACCEPTED FOR PUBLICATION IN IEEE-GLOBECOM 2018.