POLITECNICO DI TORINO

Department of Electronics and Telecommunications (DET)

VLSI Lab

Master of science course in Electronic Engineering

MASTER DEGREE THESIS

Efficient belief propagation decoding of polar codes: algorithms and architectures

---

Author:
MARINO Mattia

Supervisors:
Prof. MASERA Guido
Prof. MARTINA Maurizio

July, 2019

POLITECNICO DI TORINO

Abstract

Electronic Engineering
Department of Electronics and Telecommunications (DET)

Doctor of Electronic Engineering

Efficient belief propagation decoding of polar codes: algorithms and architectures

by MARINO Mattia

Nowadays considering the large transmission infrastructures of data that travel at very high speeds, they must be transferred with considerable efficiency. Important results have been achieved almost at the limit of the maximum capacity of the transmission channel with different types of Error Correction Code (ECC), although they have never fully saturated it.

The new generation polar codes, as demonstrated, are able, due to their recursion, to reach the maximum capacity of the channel with low complexity coding structures. Instead, in the information decoding process the Belief Propagation, an iterative parallel decoding algorithm which allows to improve low latency performance, has been studied. To eliminate the same latency times due to the iterative convergence procedure, the early stopping criteria have been studied and introduced. The objective of this thesis is the study of their performance, considering the complexity of the methods themselves; the introduction of a new early stopping method based on the Cyclic Redundancy Check (CRC) and finally a hardware implementation of the Frozen Bit Error Rate (FBER) criterion.

We will start with the analysis of the state of the art of the stopping criteria trying to compare the performances weighing them to the computational cost. It will continue with the software implementation of the very low FBER method only. Furthermore, the criterion based on the CRC will be devised and created so as to enter into synergy with the other methods given its construction.

Finally, the FBER method will be implemented in hardware considering an architecture optimized to minimize the cost of complexity by trying to maintain and improve its performance.

Contents

List of Figures

List of Tables

List of Abbreviations

ECC	**E**rror **C**orrection **C**ode
BP	**B**elief **P**ropagation
B-DMC	**B**inary-input **D**iscrete **M**emory-less **C**hannels
LLR	**L**og **L**ikelihood **R**atio
PE	**P**rocessing **E**lement
BER	**B**it **E**rror **R**ate
FER	**F**rame **E**rror **R**ate
SNR	**S**ignal to **N**oise **R**atio
CRC	**C**yclic **R**edundancy **C**heck

List of Symbols

$\mathcal{I}(\mathcal{W})$	Capacity of channel W
$Z(\mathcal{W})$	Bhattacharyya parameter
E_b/N_0	Bit energy over Noise ratio
N	Code length, number of bit that compose a frame
K	Number of information bit in a frame
$\mathcal{A}$	Set of indexes of the bits that transmit information
$\mathcal{A}^c$	Complementary set of $\mathcal{A}$ of bits' indexes that are frozen
G_N	Generator matrix, it is used for coding operations
u	Input frame
x	Encoded frame, channel input
$\hat{u}$	Estimated frame
$\hat{x}$	Estimated encoded frame
s	Scaling factor

Chapter 1

Introduction to polar codes

To realize a data transmission through a noisy channel, it is possible to apply the Error Correction Code (ECC) techniques since they allow to increase the reliability of the transmission with a substantial cheapness. Currently, to consider a reliable transmission it is necessary to have a certain degree of reliability even at low SNR; to do this, redundant information is added to the transmission itself trying to reduce the costs related to its transmission.

In information theory, there are a large number of ECCs (for example Low Density Parity Check (LDPC) or Turbo Codes) but in this work the Polar Codes will be examined. The fundamental equations and their constitutive parameters will be described for explicit mathematical validity. Moreover, its encoding process will be illustrated with a hardware implementation and, subsequently, the decoding algorithm of the Belief Propagation (BP) will be explained with its relative optimizations.

1.1 Polar codes description

Polar codes are a type of ECC created by Arikan in [1] where he formally proves that they ensure a reliable channel for an infinite length code. In his paper, he concentrates on Binary-input Discrete Memory-less channels (B-DMCs) trying to achieve a low encoding and decoding complexity. To make these concepts explicit, mathematical formulas will be introduced first and then the method created by Arikan for the construction of the code itself.

1.1.1 Channel polarization

The polar codes are based on the idea that for a channel $\mathcal{W}$ exist two capacity types $I(\mathcal{W})$:

1: $I(\mathcal{W}) = 1$ then the channel is perfect in transmission and ECC is not needed;

2: $I(\mathcal{W}) = 0$ then the channel is useless and no transmission is allowed.

The ultimate goal of polar codes is that to commute ordinary channel $\mathcal{W}$ into one of these two types. This is obtained through the polarization which requests a variable copies of the channel. Exploiting a transformation 1 to 1, it's obtainable a second set in which the channel capacity tends to 1 or 0.

Mathematically, we start from a B-DMC indicated with $\mathcal{W} : \mathcal{X} \longrightarrow \mathcal{Y}$ where $\mathcal{X}$ is the input with values $[0, 1]$ only, $\mathcal{Y}$ is the output and $\mathcal{W}(y|x), x \in \mathcal{X}, y \in \mathcal{Y}$ is the transition probability. Therefore two parameters are important:

- symmetric capacity of the channel described by

$$I(\mathcal{W}) \triangleq \sum_{y\in\mathcal{Y}} \sum_{x\in\mathcal{X}} \frac{1}{2}\mathcal{W}(y|x) log\left(\frac{\mathcal{W}(y|x)}{\frac{1}{2}\mathcal{W}(y|0)+\frac{1}{2}\mathcal{W}(y|1)}\right) \tag{1.1}$$

- Bhattacharyya parameter described by

$$Z(\mathcal{W}) \triangleq \sum_{y\in\mathcal{Y}} \sqrt{\mathcal{W}(y|0)+\mathcal{W}(y|1)} \tag{1.2}$$

The first one indicates the maximum rate at which the reliable communication can be realized; the second one is an upper bound on the probability of maximum-likelihood (ML) decision error to transmit a 1 or 0. It's simple to understand that both of them can take values in $[0,1]$. $I(\mathcal{W})$ will be equal to the Shannon capacity only when $I(\mathcal{W})$ is a symmetric channel.

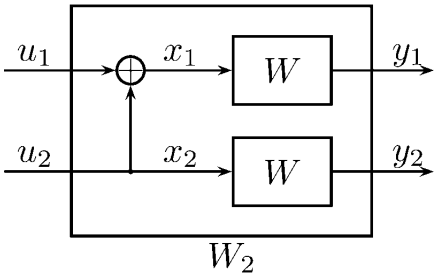


FIGURE 1.1: $\mathcal{W}_2$ basic channel

The channel polarization occurs into two different phases divided in combining and splitting channel.

Combining channel It combines $N = 2^n$ (with $n \geq 1$) independent copies of $\mathcal{W}$ to construct a more complex channel $\mathcal{W}_N : \mathcal{X}^N \longrightarrow \mathcal{Y}^N$. To achieve the $\mathcal{W}_N$ channel it is necessary to start from the basic $\mathcal{W}_2$ arranging the transition probabilities in this way:

$$\mathcal{W}_2(y_1y_2|u_1u_2) = \mathcal{W}(y_1|u_1 \oplus u_2)\mathcal{W}(y_2|u_2) \tag{1.3}$$

The graph in Figure 1.1 shows the basic combining channel with $\mathcal{W}_2$.

Going forward with $\mathcal{W}_4$, it's possible to condense the whole transition probability simplifying the reading with a generator matrix G_4 described as

$$G_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{1.4}$$

In Figure 1.2 we have the graph for $\mathcal{W}_4$ and the overall relation is

$$\mathcal{W}_4(y_1^4|u_1^4) = \mathcal{W}^4(y_1^4|u_1^4G_4) \tag{1.5}$$

The general expression of the combining operation is presented in Figure 1.3 where the sub-block R_N is called *reverse shuffle*. It works on its input to create the input for the two copies of $\mathcal{W}_{N/2}$. The probabilities with N factor is determined by the generator matrix G_N and by the relation

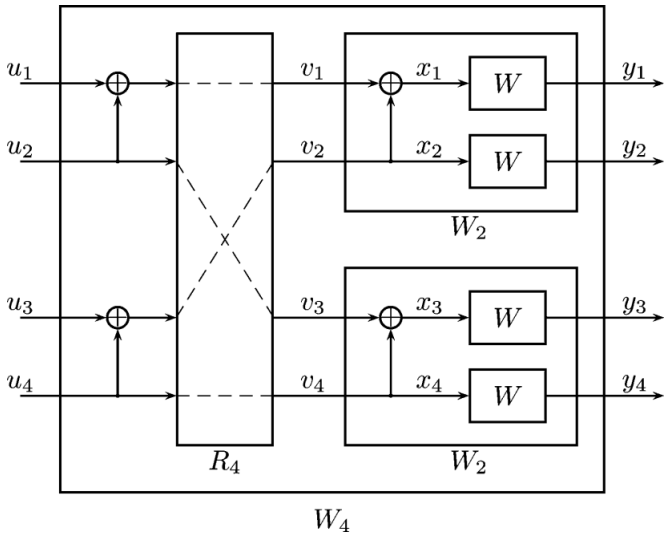


FIGURE 1.2: $\mathcal{W}_4$ channel

$$\mathcal{W}_N(y_1^N|u_1^N) = \mathcal{W}^N(y_1^N|u_1^N G_N) \quad with\ y_1^N \in \mathcal{Y}, u_1^N \in \mathcal{X} \tag{1.6}$$

Splitting channel In the splitting step N different polarized channels must be obtained and it's possible to demonstrate that

$$I(\mathcal{W}_N) = I(\mathcal{U}^N, \mathcal{Y}^N) = \sum_{i=1}^{N} I(\mathcal{U}_i; \mathcal{Y}^N, \mathcal{U}^{i-1}) \tag{1.7}$$

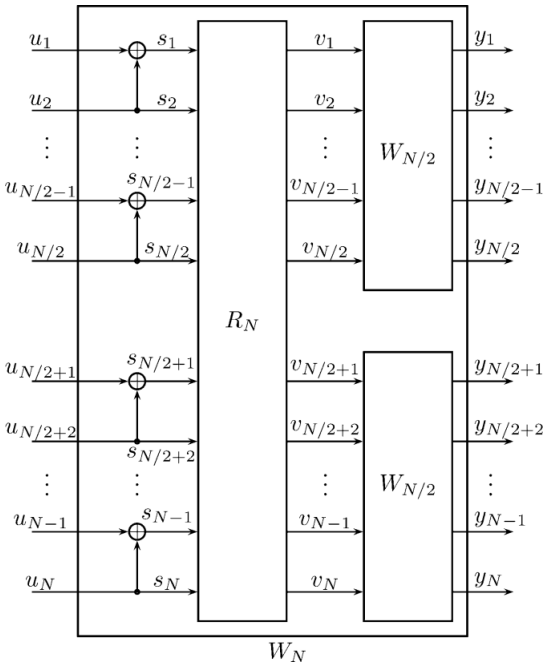


FIGURE 1.3: $\mathcal{W}_N$ channel composed by two copies of $\mathcal{W}_{N/2}$

Thus, defining a new bit line channel $\mathcal{W}_i : \mathcal{U}_i \longrightarrow (\mathcal{Y}^N, \mathcal{U}^{i-1})$, it's possible to conclude that:

$$I(\mathcal{W}_N) = I(\mathcal{U}^N, \mathcal{Y}^N) = \sum_{i=1}^{N} I(\mathcal{U}_i; \mathcal{Y}^N, \mathcal{U}^{i-1}) = \sum_{i=1}^{N} I(\mathcal{W}_i) \tag{1.8}$$

After that it's possible to affirm that the capacitance of the polarized channel is the same of the not polarized one.

It's necessary to notice that a random permutation with a mapping transformation 1 to 1 is most likely a good polarizer but it's not simple to implement. To remedy this, isotropic and step-wise proprieties have to be chosen in order to obtain a low complexity polarizer. Indeed the same Arikan introduces this transformation type as figured in 1.4.

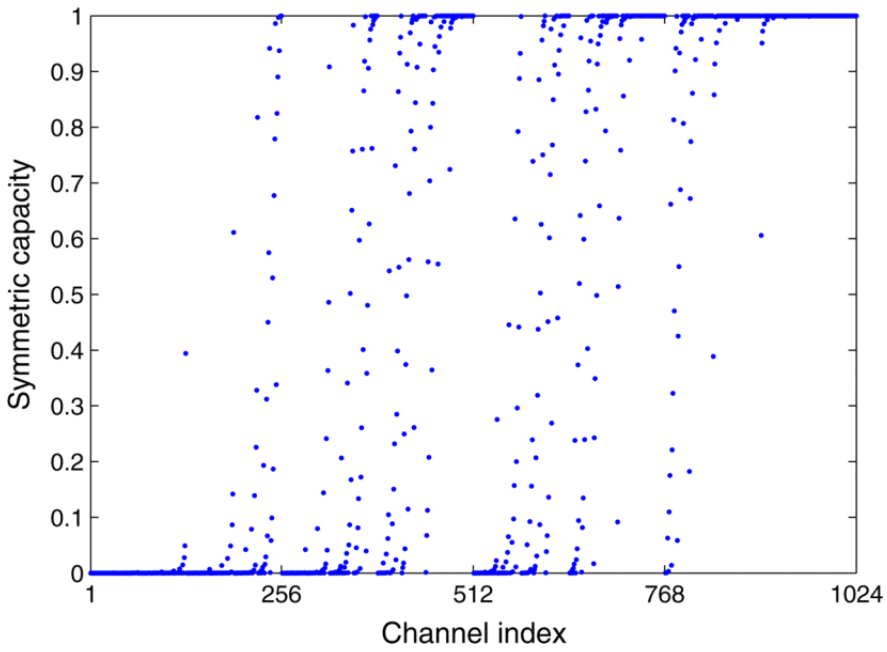


FIGURE 1.4: $\mathcal{W}_N$ channel capacity to vary index

Furthermore, Arikan managed to create a recursive transformation composed by an encoder with $Nlog(N)$ complexity which reaches a good channel polarization. More insights and details can be found in [2].

1.2 Encoding process

After the definition and description of channel polarization formalities, the encoding process can be analyzed. The bits' set which carries the information will be named $\mathcal{A}$ and it is compound by K high capacitance channels; its complementary, the bits' set which does not carry information (then fixed to 0, usually called *frozen bits*), will be $\mathcal{A}^c$. The $\mathbf{x}$ vector is the transmitted codeword; instead $\mathbf{u}$ is the input of the encoder which performs the following operation:

$$x_1^N = u_1^N G_N \tag{1.9}$$

Considering the sets described above, it's possible to rewrite the equation 1.9 dividing the two contributions:

$$x_1^N = u_{\mathcal{A}} G_N(\mathcal{A}) \oplus u_{\mathcal{A}^c} G_N(\mathcal{A}^c) \tag{1.10}$$

However, as explained before, G_N is the generator matrix with $N = 2^n$ dimension with $n \geq 0$ and, generalizing, it can be described as:

$$G_N = R_N(F \otimes I_{N/2})(I_2 \otimes G_{N/2}) = R_N(F \otimes G_{N/2}) \quad for \;\; N \geq 2 \tag{1.11}$$

where:

- I_k is the k-dimensional identity matrix for $k \geq 1$;
- R_N is the *reverse shuffle* operator;
- F is a 2x2 matrix defined as $F \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$;
- the operand $\otimes$ is the Kronecker product of matrix determined as

$$A \otimes B = \begin{pmatrix} A_{1,1}B & A_{1,2}B & \cdots & A_{1,n}B \\ A_{2,1}B & A_{2,2}B & \cdots & A_{2,n}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1}B & A_{m,2}B & \cdots & A_{m,n}B \end{pmatrix}.$$

Substituting for two times $G_{N/2} = R_{N/2}(F \otimes G_{N/4})$ in 1.11 and exploiting the identity propriety $(AC) \otimes (BD) = (A \otimes B)(C \otimes D)$ with $A = I_2$, $B = R_{N/2}$, $C = F$, $D = F \otimes G_{N/4}$, we obtain

$$G_N = B_N F^{\otimes n} \tag{1.12}$$

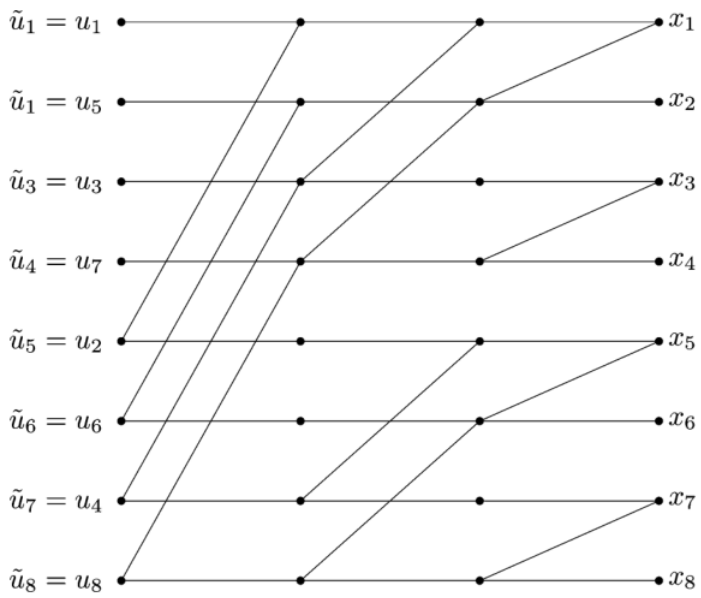


FIGURE 1.5: Encoder hardware implementation for $G_8 = F^{\otimes 3}$, $N = 8$

B_N is a *permutation matrix* since it derives from a product of two permutation matrices and it's defined as:

$$B_N \triangleq R_N(I_2 \otimes R_{N/2})(I_4 \otimes R_{N/4}) \cdots (I_{N/2}R_2) = R_N(I_2 \otimes B_{N/2}) \tag{1.13}$$

In Figure 1.5 it is shown a circuit which realizes the encoding process for $N = 8$. In the implementation B_N is omitted since it is preferable only to use $F^{\otimes n}$ in order to simplify the implementation itself. The total latency is equal to $NlogN$ and it's necessary to notice that the $\mathbf{u}$ bit-input vector is not processed in its natural order only to get the output in the correct index progression.

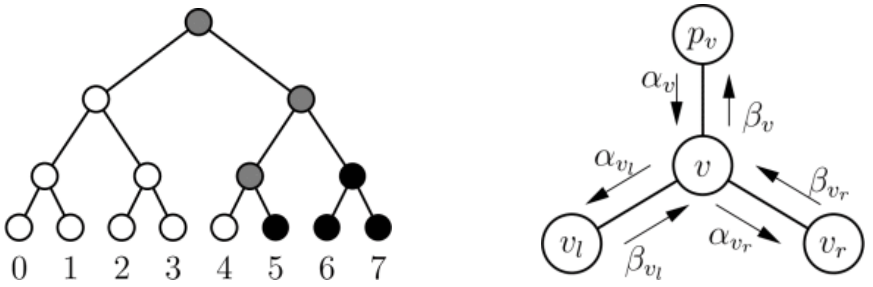
As Arikan exhibits in [1], the latency's implementation could be improved applying in parallel N processors in order to execute the encoding operation *column-by-column* thus reducing the total latency to $logN$. At the end, the encoder can be synthesized using only XOR logic gates.

1.3 Decoding process

Going on to examine the decoding techniques, we can affirm that, in literature, the decoding of the polar codes is essentially formed by two main families:

- Successive cancellation (SC);
- Belief propagation (BP).

The first one is characterized by a sequential algorithm which decodes the frame in a recursive approach involving a finite number steps: starting from the root, the decoder computes the $\hat{u}_i$ step exploiting the channel output y and the previous calculated step $\hat{u}_{i-1}$.



(A) Polar codes tree for successive cancellation decoder $N = 8$

(B) Local node decoder $N = 8$

FIGURE 1.6: Basic elements for successive cancellation

This recursive way of proceeding offers a natural binary representation for polar codes figured in 1.6a; instead the local operative node (illustrated in 1.6b) needs to be fed, in order to start the decoding process, at root with $(\lambda_1, \cdots, \lambda_N)$ channels with the Log Likelihood Ratio (LLR) where

$$\lambda_i = log\left(\frac{P(y_i|x_i = 0)}{P(y_i|x_i = 1)}\right) \tag{1.14}$$

The SC decoder has the advantage of requiring few computational costs for decoding an input frame and, in doing this, it reaches a good error-correcting performance for long code lengths. However, the most significant problem of this decoder

is its latency since every decoding step is not independent from each others. Optimizations can be done but this nature can not be changed. This type of decoder will not be taken into account in this work and more information on the successive cancellation can be found at [3]. We will focus on belief propagation decoder that ensures a low-latency response given from the possibility of performing calculations in parallel.

1.3.1 Belief propagation decoder

The belief propagation decoder manages to eliminate the problem of sequentiality introduced by the SC since the BP algorithm operates in separate stages [4]. It can be used for the decoding of polar codes exploiting the factor graph shown in Figure 1.7.

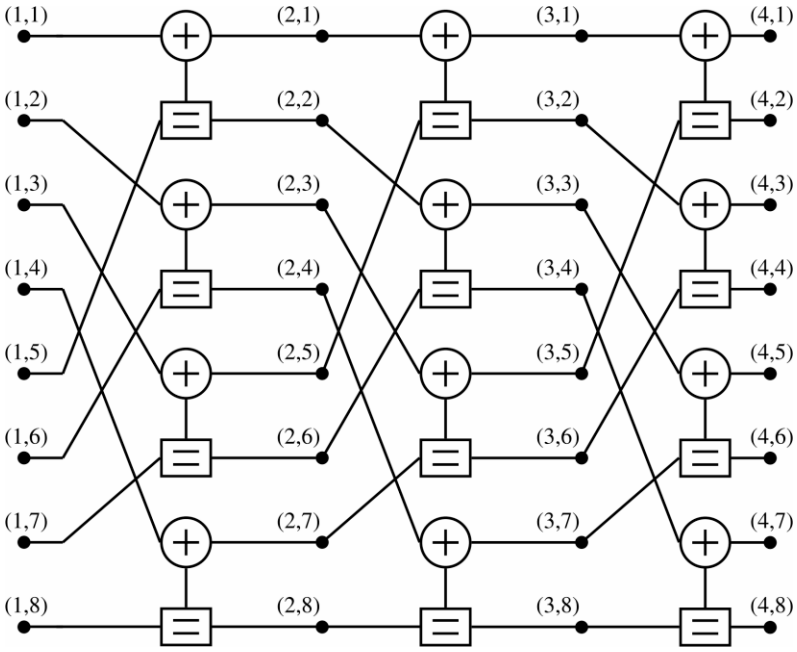


FIGURE 1.7: Factor graph of Belief Propagation decoder for $N = 8$

It's possible to generalize for a N code length with $(logN + 1)N$ nodes (denoted with a couple $(i, j) : 1 \leq i \leq n; 1 \leq j \leq N/2$) which have, for each one, two different message types: L and R. The nodes at first layer are connected with the source vector

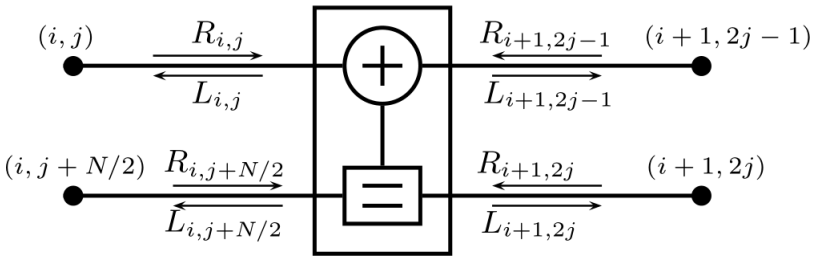


FIGURE 1.8: Basic Processing Element (PE) of Belief Propagation decoder

u and the last $(n+1)$ nodes' layer is linked with the code-word **x**. The calculations are performed by the basic PE in Figure 1.8.

These messages are the LLRs and they propagates and updates them-self among adjoining nodes in according with the following equations:

$$\begin{aligned} L_{i,j} &= g(L_{i+1,2j-1}, L_{i+1,2j} + R_{i,j+N/2}) \\ L_{i,j+N/2} &= g(R_{i,j}, L_{i+1,2j-1}) + L_{i+1,2j} \\ R_{i+1,2j-1} &= g(R_{i,j}, L_{i+1,2j} + R_{i,j+N/2}) \\ R_{i+1,2j} &= g(R_{i,j}, L_{i+1,2j-1}) + R_{i,j+N/2} \end{aligned} \tag{1.15}$$

where

$$g(x,y) = ln\left(\frac{(1+xy)}{(x+y)}\right) \tag{1.16}$$

By the way, the last equation is too complex to be integrated into the circuit, then it's approximated in hardware implementation with the following expression:

$$g(x,y) \approx sgn(x)sgn(y)min(|x|,|y|) \tag{1.17}$$

The $R_{1,j}$ messages generated from the vector **u** are set as

$$R_{1,j} = \begin{cases} 0 & \text{if } j \in \mathcal{A} \\ \infty & \text{if } j \in \mathcal{A}^c \text{and } \mathrm{u_j} = 0 \\ -\infty & \text{if } j \in \mathcal{A}^c \text{and } \mathrm{u_j} = 1 \end{cases}$$

Instead, the messages $L_{N+1,j}$, $1 \leq j \leq N$, that originate from the channel block are given by

$$L_{N+1,j} = log\left(\frac{P(y_j|x_j=0)}{P(y_j|x_j=1)}\right) \tag{1.18}$$

As shown so far and supported by [5], BP polar decoder has the advantage of processing data in parallel and, when compared with SC, offers a certain advantage for all low-latency solutions. Given the iterative nature of the BP polar decoder, the number of iterations performed will be fundamental for the increase's latency and energy consumption. Exactly for this reason, the computational cost is high exactly in the presence of a large number of iterations. To overcome this problem, according to the literature, early stop criteria have been introduced in Chapter 2 while maintaining the same level of error-correcting performance.

1.3.2 Belief propagation optimizations

Over the years, there have been numerous advances and progresses in the improvement of the BP algorithm with the inclusion of a scaling factor [6], in order to balance the approximations, and the exploitation of the frozen bits position to sharpen performance for a finite-length polar codes exposed in [7] and [8].

Scaling factor Given the introduction of an approximation from the equation 1.17 in hardware implementation, it is necessary to take into account the performance losses. In order to minimize them, a scaling parameter s has been introduce to mitigate this errors. Applying s, the set's equations 1.15 becomes:

$$\begin{aligned} L_{i,j} &= s * g(L_{i+1,2j-1}, L_{i+1,2j} + R_{i,j+N/2}) \\ L_{i,j+N/2} &= s * g(R_{i,j}, L_{i+1,2j-1}) + L_{i+1,2j} \\ R_{i+1,2j-1} &= s * g + (R_{i,j}, L_{i+1,2j} + R_{i,j+N/2}) \\ R_{i+1,2j} &= s * g(R_{i,j}, L_{i+1,2j-1}) + R_{i,j+N/2} \end{aligned} \tag{1.19}$$

The correction introduces significantly improves in the decoding performance due to the fact that the g function is more faithful to the original without approximation. In fact, this parameter permits the alignment of the errors allowing a considerable gain comparable with the non-scaled algorithm. Given the tests, we find that the best value to apply is $s = 0.9375 = 1 - 2^{-4}$ with an improvement of $0.5dB$ and, therefore, in the hardware implementation the scaling factor can be realized with a simple shift-sum circuit.

Frozen bit position In accordance with the equation 1.15, it's possible to observe that when the lower left node linked to PE $(i, j + N/2)$ is frozen, the node $(i + 1, 2j)$ is frozen as well; similarly, when the two nodes $(i, j + N/2)$ and $(i, j + N/2)$ are frozen, the two nodes $(i + 1, 2j - 1)$ and $(i + 1, 2j)$ are frozen as well. With this mind, optimizations can be made to avoid the allocation of registers that have to store messages related to frozen nodes, the computation of frozen messages and to allocate simplified PEs when the inputs are frozen.

Taking into account these observations, it is possible to affirm that by analyzing the BP decoding algorithm and referring to the Figure 1.9, the nodes can be classified as *frozen* and *information*. As for the former, the values they can assume are

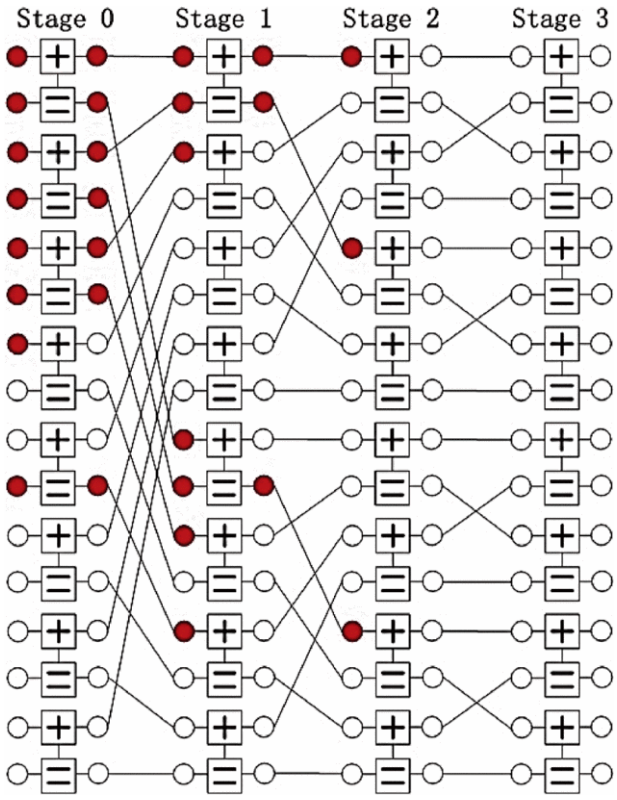


FIGURE 1.9: Simplified BP decoder factor graph with highlighted frozen bits in red, $N = 16$

already known and independent from formulas; the latter necessarily depends on the algorithm. Moreover, a third type of node is introduced, called *check node*, which exchanges messages to simplify the algorithm itself, shortening the path. The new type of node is described by the following formulas:

$$\begin{aligned} L^*_{i,j} &= L_{i,j} + P_{i,j} \\ R^*_{i,j} &= R_{i,j} + P_{i,j} \end{aligned} \tag{1.20}$$

where $P_{i,j}$ value is decisive. In fact, initializing $P = \infty$ we will set the next node as frozen; contrariwise if $P = 0$ the next node will be an information node. For more insights see [7].

However, it is possible to change the point of view using optimized PE rather than varying the inputs. This approach, explained in [8], allows the classification of PEs into 4 categories figured in 1.10:

- all input are frozen nodes;
- lower frozen input nodes;
- only upper right input node is frozen;
- no frozen input nodes.

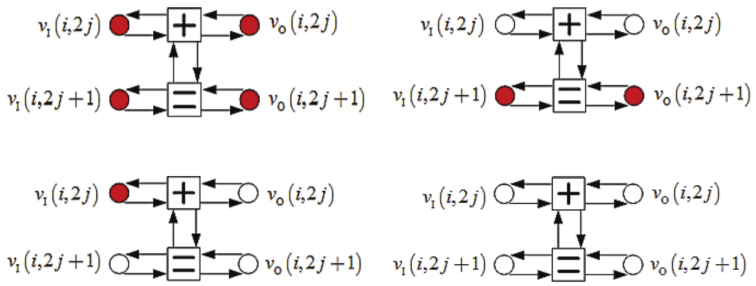


FIGURE 1.10: Types of PE with highlighted frozen bits in red

The savings with this technique are possible in the first three cases because in the last one canonical equations need to be used. The tests show that the cost of complexity is reduced by 19.9% up to 25% maintaining the same error correction performance.

By the way when the whole nodes are frozen the equations become simplified:

$$L_{i,j} = L_{i,j+N/2} = R_{i+1,2j-1} = R_{i+1,2j} = \infty \tag{1.21}$$

In second case equations will be:

$$\begin{aligned} L_{i,j} &= L_{i+1,2j-1} \\ R_{i+1,2j-1} &= R_{i,j} \\ L_{i,j+N/2} &= R_{i+1,2j} = \infty \end{aligned} \tag{1.22}$$

In third case there is the least savings as it is possible to simplify only few elements:

$$\begin{aligned} L_{i,j} &= \infty \\ L_{i+1,2j-1} &= L_{i+1,2j-1} + L_{i+1,2j} \\ R_{i+1,2j-1} &= L_{i+1,2j} + R_{i,j+n/2} \\ R_{i+1,2j} &= L_{i+1,2j-1} + R_{i,j+n/2} \end{aligned} \tag{1.23}$$

Chapter 2

Stopping criteria for belief propagation polar codes decoders

To reduce the decoding process time given by the decoder's iterative nature, it's fundamental to find some early stopping criteria in order to increase performances and throughput, always trying to detect the correct decoding of the frames. However the main goal of early stopping criteria is to find the correct solution before reaching the imposed maximum number of iteration.

In literature distinct stopping criteria exist and afterwards are described the more interesting ones. Although the first two methods G-Matrix and minLLR are efficient criteria, they have expensive computational costs. The third, the H-Matrix criterion, is an improvement of the first one and promises better performances using fewer additional resources. Furthermore, to decrease these computational costs, Worst of Information Bits (WIB) and Frozen Bit Error Rate (FBER) are presented as low complexity early stopping criteria.

2.1 G-Matrix

G-Matrix criterion [9] has the purpose of reducing computational costs exploiting the G generation matrix. The G-Matrix is created as

$$G_N = F^{\otimes n}$$

A pair of (n, k) is needed to create a message: N is the length (in bit) of whole codeword, K are the information bits and $(N - K)$ is the set of frozen bits fixed to "0" value. The $N - bit$ message, hereafter called u, is multiplied with the generator matrix G and the x codeword output is transmitted through the polar code channel.

Considering $x_j = L_{n+1,j} + R_{n+1,j}$ and the formula's approximation just above $\hat{x} = \hat{u}G_N$, if $\hat{x} = \hat{u}$ then we have a successful decoding and the iteration process can be stopped; contrariwise if we do not reach this condition at iteration's maximum value then the decoded frame is incorrect. Accordingly we repeat the process at every iteration.

G-Matrix method is able to reduce iterations, without FER's loss performance, from 23% at 2.5 dB to 42% at 3.5 dB with 40 iteration's maximum threshold.

Computational cost Following the hardware scheme in Figure 2.1, we obtain the computational cost described in Table 2.1: we need $2N$ adders to calculate $\hat{x}$ and $\hat{u}$, $3N$ comparators to check if convergent conditions are set and $N \log N$ XOR to perform the multiply with G-Matrix.

G-Matrix performances		
SNR	*Avg iter*	*Iterations' drop*
2.5 dB	30.8	23.0 %
3.0 dB	26.1	34.7 %
3.5 dB	23.0	42.5 %

G-Matrix complexity costs	
Type	*Quantity*
Adder	$2n$
Comparator	$3n$
XOR	$n \log n$

TABLE 2.1: G-Matrix performances with 40 maximum iterations and complexity costs

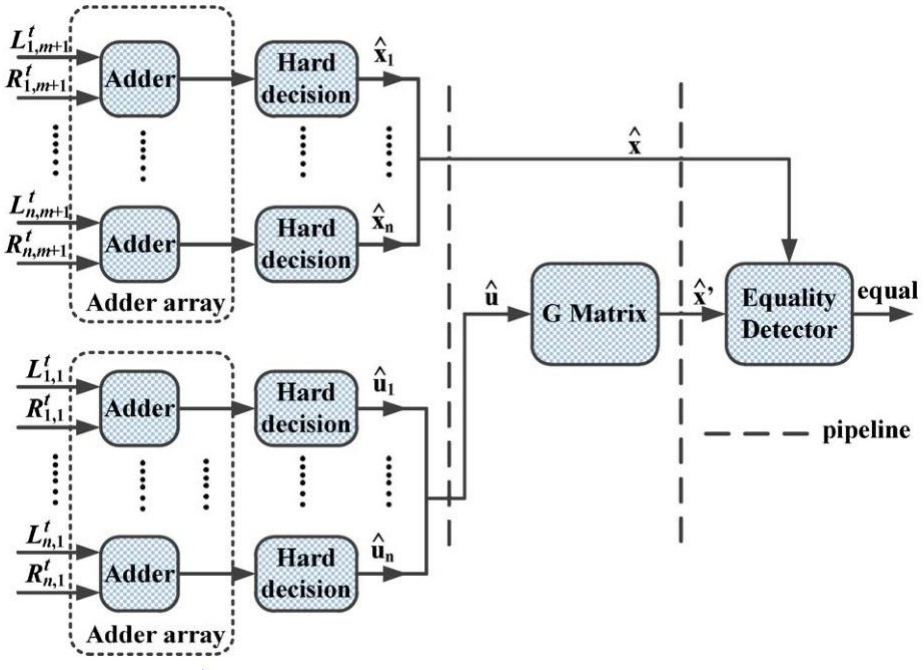


FIGURE 2.1: Hardware architecture of G-Matrix criterion

2.2 minLLR (minimum Log-Likelihood Ratio)

With minLLR criterion [9] we can obtain a measure of the reliability of $\hat{u}_i$ probability of being 0 or 1 considering $\hat{u}_i = sign(LLR^t_{i,1})$. Hence a larger $\left|LLR^t_{i,1}\right|$ means that the corresponding computation has greater reliability. So using the minimum $\left|LLR^t_{i,1}\right|$ corresponds to identify a valid $\hat{u}$. Thus, if

$$min \left|LLR^t_{1,j}\right| > \beta$$

with β tipically 2.5, the $\hat{u}_i$ is likely a valid estimation of u, the real information present on the channel.

Adaptive minLLR minLLR criterion has not a uniform behaviour when varying the SNR. This means that a different threshold of β is needed during the growth of SNR value. To have a better and uniform trend without getting worse performances, modifying β constant is necessary: an higher SNR value needs an higher β constant. The problem can be solved estimating the channel condition using the Hamming

distance between $\hat{u}G$ and $\hat{x}$. The distance, called λ, is equal to 0 when $\hat{u}G = \hat{x}$ and it means that $\hat{u}G$ is an accurate approximation of the output $\hat{x}$; thus β value can be chosen dynamically.

minLLR criterion suffers from confined performance degradation starting from $SNR = 3.5dB$ of <0.05dB. It is due to the fact that the β choice is not always optimal. By the way the criterion reaches a good reduction of iteration keeping in mind computational cost's savings.

Computational cost Regarding adaptive minLLR computational parts are lower than G-Matrix ones since it has a different operations' pattern: a halving of adders, a third less of comparator and absence of XORs.

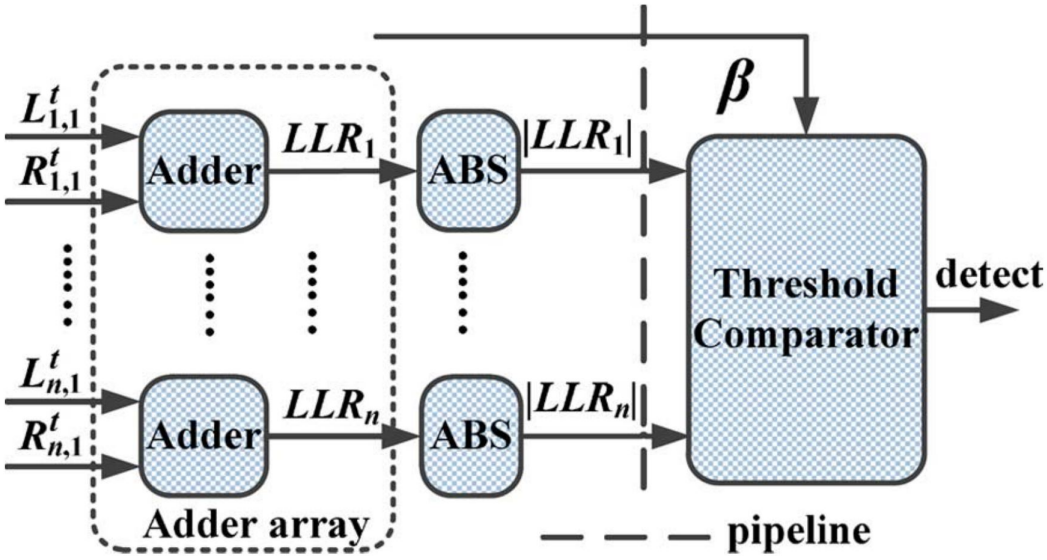


FIGURE 2.2: Hardware architecture of Adaptive minLLR criterion

Adapt. minLLR performances		
SNR	*Avg iter*	*Iterations' drop*
2.5 dB	35.7	10.7 %
3.0 dB	33.9	15.2 %
3.5 dB	30.7	23.2 %

Adapt. minLLR complexity costs	
Type	*Quantity*
Adder	N
Comparator	$2N$
XOR	-

TABLE 2.2: Adaptive minLLR performances with 40 maximum iterations and complexity costs

2.3 H-Matrix

The H-Matrix method, also called "Parity check matrix", has the aim to find a syndrome composed only by zeros which means the decoding is successful and correct. This early stopping criterion can be used only with linear block codes, as explained in [10], since we can obtain the parity check matrix starting from the generator matrix G_N and thus a certain syndrome.

The encoding of polar codes is described as $x = uG_N$, where $u = (u_1, u_2, ..., u_n)$ is container of information bits (coded with 1) or frozen ones (coded with 0); x is the codeword formed as $x = (x_1, x_2, ..., x_n)$. Remembering this steps, H-Matrix is developed as following:

1. from G_N $N \times N$ matrix we derive a G-Matrix $K \times N$ considering only the columns of the information bit's index (always supposing k information bits in the code);
2. through the Gaussian Elimination we get a systematic matrix $G = \begin{bmatrix} P_{K\times(N-K)} & I_{K\times K} \end{bmatrix}$;
3. reversing the last one we find the H-Matrix $H = \begin{bmatrix} I_{(N-K)\times(N-K)} & P^T_{(N-K)\times N} \end{bmatrix}$ from which calculating the syndrome.

If syndrome contains only zeros, early stopping criterion has found the correct codeword $\hat{x}$ and the process is stopped; else if syndrome is not composed by only zeros, $\hat{x}$ is not a valid codeword and the process must continue. If correct $\hat{x}$ is not found within maximum iterations, it is considered failed.

H-Matrix early stopping criterion guarantees minus iterations without loss performances from 11.6% at $SNR = 3dB$ with 20 maximum iterations until 71.7% at $SNR = 3dB$ with 80 iterations.

Computational cost Regarding the fact that we need G-Matrix to compute the parity check matrix, the computational costs are the same respect the G-Matrix case. Moreover we have to include to 2.3 the Gaussian Elimination operations and transposition matrix.

H-Matrix performances		
SNR	*Avg iter*	*Iterations' drop*
2.5 dB	25.6	36.0 %
3.0 dB	21.9	45.3 %
3.5 dB	18.2	54.5 %

H-Matrix complexity costs	
Type	*Quantity*
Adder	$2N$
Comparator	$3N$
XOR	$N \log N$

TABLE 2.3: H-Matrix performances with 40 maximum iterations and complexity costs

2.4 WIB (Worse of Information Bits)

The Worse of Information Bits (WIB), described in [10], is the information cluster with whom it is possible to stop the iteration process detecting a successful decode. The WIB cluster is composed by polarized bits with the highest error probabilities and in this set are present only the frozen-bits. Applying Bhattacharyya bound $Z(\mathcal{W})$ for polar codes (because it has the lowest complexity) the method is created exploiting the proportion of average Bhattacharyya values (PoB) described as

$$PoB = \frac{\frac{1}{n_{WIB}} \sum_{l \in \mathcal{WIB}} Z(\sigma_l^2)}{\frac{1}{k - n_{WIB}} \sum_{l \in \mathcal{WIB}} Z(\sigma_l^2)}$$

where n_{WIB} are the observed necessary bits, k the whole information bits and σ^2 is the Gaussian noise. It's simple to notice that when n_{WIB} increases also the PoB

increases drastically (and vice-versa); in this way a higher value of n_{WIB} means a better behaviour in decoding process. Ultimately it's used $n_{WIB} = 128$.

To simplify the computation, we observe only the LLR's (both left and right) sign mutation of WIB with

$$\hat{u}_i^t = sign(R_{i,1}^t + L_{i,1}^t)$$

After calculating this partial formula, we must control that the following formula

$$\sum_{l \in \mathcal{WIB}} \sum_{v=t-M+1} \hat{u}_i^v \oplus \hat{u}_i^{v-1}$$

has not "0" as result. The M parameter indicates for how many steps results must be stable to "0". If the condition is fulfilled we can say that the method is successful and the iterations can be stopped. Varying M parameter depending on the SNR value we can exploit a kind of adaptive WIB that can still reduce the computational time.

WIB criterion reaches a good iterations' cut from 11.8% at 2.5 dB for both variants to 33.3% at 3.5 dB only for the fixed one. In every test iteration's maximum is set to 40 and $n_{WIB} = 128$. Adaptive WIB has better performances at low SNR where overcomes the fixed variant which has also a performance degradation of 0.05 dB at 3.5 dB.

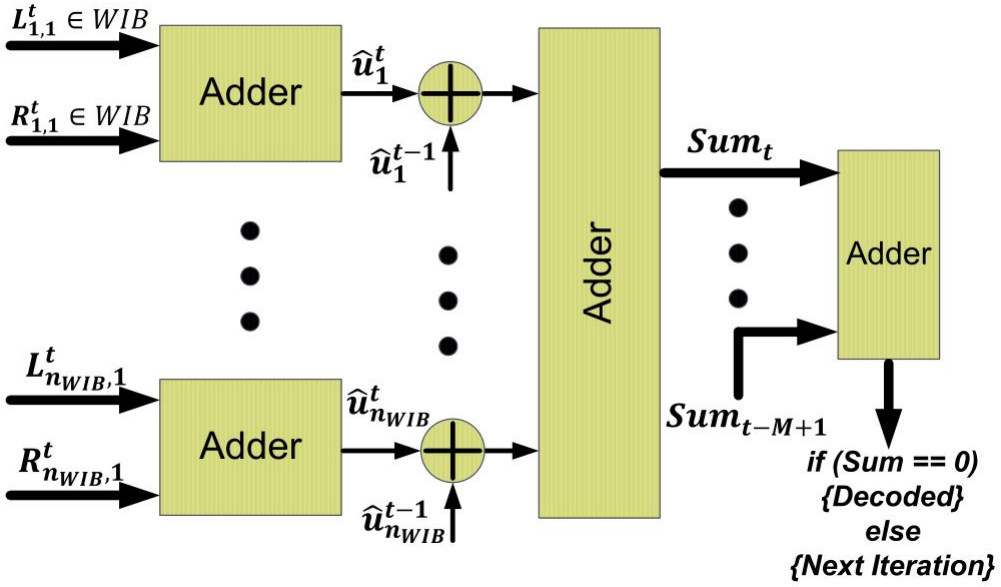


FIGURE 2.3: Hardware architecture of WIB criterion

Computational cost Both variant methods have same computational costs how summarized in 2.4. Synthesis is achieved from hardware architecture figured in 2.3. WIB criterion does not use comparator blocks but obtains good results expending few adders and XOR blocks.

WIB performances adaptive / fixed			
SNR	*Avg iter*	*Iterations' drop*	*M*
2.5 dB	35.3 / 35.3	11.8 %/ 11.8 %	5 / 5
3.0 dB	28.4 / 28.4	29.0 %/ 29.0 %	5 / 5
3.5 dB	28.4 / 26.7	29.0 %/ 33.3 %	7 / 5

WIB complexity costs	
Type	*Quantity*
Adder	$M + 2N/8$
Comparator	–
XOR	$N/8$

TABLE 2.4: WIB performances with 40 maximum iterations and complexity costs

2.5 FBER (Frozen Bit Error Rate)

This method, presented in [11], uses the Frozen Bit Error Rate (FBER) to detect when the belief propagation decoding is successful. The F^t_{BER} at t-th iteration is defined as

$$F^t_{BER} = \frac{1}{|\mathcal{A}^c|} \sum_{i \in \mathcal{A}^c} [(1 + sign(L^t_{i,1}))/2] \oplus u_i$$

where u_i is a frozen bit. The stopping criterion, however, is based on the calculation of

$$F^t_M = \sum_{v=t-M+2}^{t} \left| F^v_{BER} - F^{v-1}_{BER} \right|$$

where M is the iteration number in which F^t_M remains stable. For a complete and successful BP decoding $F^t_M = 0$ is needed.

It's further possible to reduce the computational cost considering $\mathcal{A}^c_p \subset \mathcal{A}^c$ in which there are only the most reliable frozen bits. $\mathcal{A}^c_p$ is a partial index set of frozen bits and then the formulas become

$$F^t_{PBER} = \frac{1}{|\mathcal{A}^c|} \sum_{i \in \mathcal{A}^c} [(1 + sign(L^t_{i,1}))/2] \oplus u_i$$

$$F^t_M = \sum_{v=t-M+2}^{t} \left| F^v_{PBER} - F^{v-1}_{PBER} \right|$$

FBER performances		
SNR	*Avg iter*	*Iterations' drop*
2.5 dB	33	17.5 %
3.0 dB	28	29.8 %
3.5 dB	26.5	34.0 %

FBER complexity costs	
Type	*Quantity*
Adder	$M + N/16$
Comparator	–
XOR	$N/16$

TABLE 2.5: FBER performances with 40 maximum iterations and complexity costs

Computational cost Analysing the Table 2.5, we can conclude that FBER criterion offers a iteration's reduction equal to 17.5% at 2.5 dB until 34% at 3.5 dB. This gain is obtained with a very low complexity cost which includes only a few adders and XORs components cutting out the comparators.

2.6 Comparison between early stopping criteria

Comparing all the early stopping criteria and using at least one of them, we obtain a minimum saving in the order of 10.7% of iterations applying the adaptive minLLR, as shown in Table 2.6 and in Figure 2.4. However it is pretty expensive even taking into account the performances which offers. In fact compared with the two most expensive criteria (G-Matrix and H-Matrix), minLLR only halves the numbers of adders and comparators providing just a third of iteration savings.

Performance comparison between early stopping criteria					
Criterion	*G-Matrix*	*H-Matrix*	*minLLR*	*WIB*	*FBER*
SNR	*Average Iterations / Iterations' drop*				
2.5 dB	30.8/23 %	25.6/36 %	35.7/10.7 %	35.3/11.8 %	33/17.5 %
3.0 dB	26.1/34.7 %	21.9/45.3 %	33.9/15.2 %	28.4/29 %	28/29.8 %
3.5 dB	23/42.5 %	18.2/54.5 %	30.7/23.2 %	26.7/33.3 %	26.5/34 %

TABLE 2.6: Performance comparison between criteria normalized to 40 iterations

G-Matrix and H-Matrix give the best performances among all criteria and they have a comparable similar cost and benefits: between them the minimum reduction is 23% of G-Matrix until 54.5% due from H-Matrix. Although the last one referred criterion needs more computational costs to obtain the parity check matrix, it is still preferred to G-Matrix since it has a better behaviour in the entire SNR scale. Nevertheless they still remain very expensive criteria how is highlighted in Table 2.7 and in Figure 2.5.

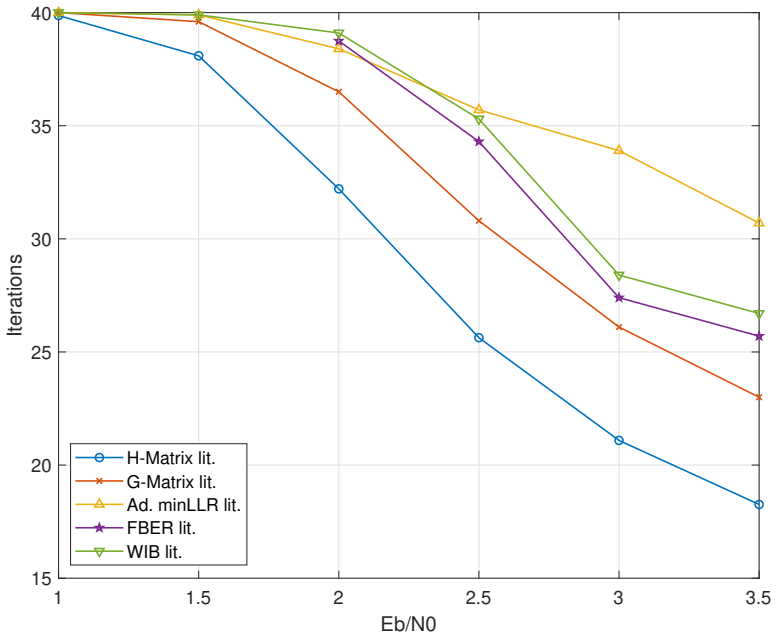


FIGURE 2.4: Performance comparison between criteria normalized to 40 iterations

Complexity costs comparison between early stopping criteria					
Criterion	*G-Matrix*	*H-Matrix*	*minLLR*	*WIB*	*FBER*
Type	*Components' quantity*				
Add	$2N$	$2N$	N	$M+2N/8$	$M+N/16$
Comparator	$3N$	$3N$	$2N$	-	-
XOR	$Nlog(N)$	$Nlog(N)$	-	$N/8$	$N/16$

TABLE 2.7: Complexity costs comparison between criteria

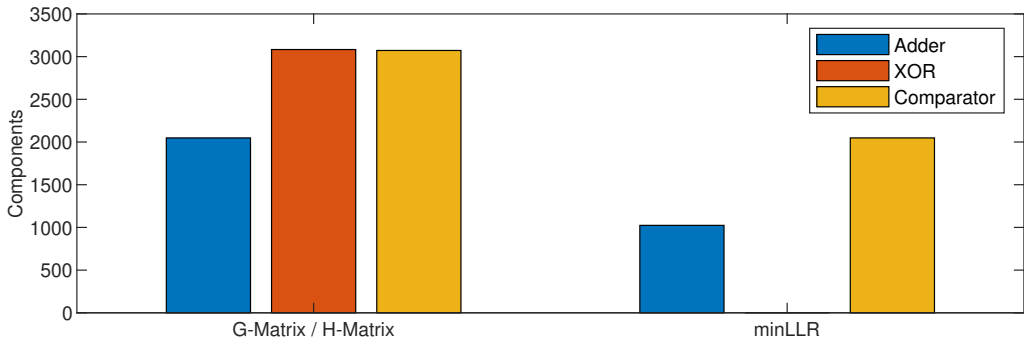


FIGURE 2.5: Complexity cost comparison between high computational costs criteria with $N = 1024$

Finally WIB and FBER are the cheapest presented methods which offer limited benefits in terms of iterations' reduction. They are comparable with minLLR despite being better: WIB has a 11.8% of savings, FBER has 17.5%. They also have a very similar behaviour at high SNR but FBER criterion provides those result with fewer adders and XORs components how can be seen in Figure 2.6.

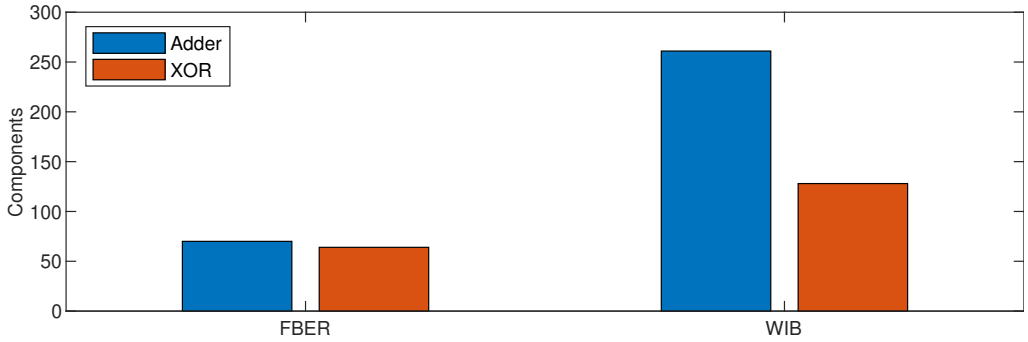


FIGURE 2.6: Complexity costs comparison between simplified criteria with $N = 1024$

Chapter 3

Software simulation

Once introduced a global view of BP in Chapter 1 and defined the stopping criteria in Chapter 2, it is possible to go beyond in testing phase. Starting from a prefixed C code which already includes a BP decoder, custom schedule implementations and G-Matrix stopping criterion, it is necessary to recall the BP's parameters and some basic aspects of the scheduling concepts.

Moreover, it is presented a FBER implementation and a first CRC application in its variants. Finally, test's results are shown with relative analysis and conclusions.

For all the tests have been used computational resources provided by VLSI Lab.

3.1 Software model

It's useful to remind the main parameters of already existing and tested C model of BP decoder. These parameters are:

- N is the size (in bit) composing a single frame;
- K is the size (in bit) of the information contained in a single frame;
- *intrinsic information* refers to how much a message coming from channel can vary;
- *extrinsic information* is the range of how a message can vary internally with a dynamic behavior of the decoder;
- s is the so called *scaling factor* used to improve BP's decoder performances in order to have a better approximation of g function.

In Chapter 4 is described analytically the whole parameters' set to perform the simulations.

3.1.1 Scheduling profiles

Due the intrinsic parallelism of the belief propagation algorithm, it is possible to apply a different operations' order. It is important to point out that a schedule profile is only a way to control the succession of operations relative to LLR messages.

The single PE can propagate both L and R messages at once (bidirectional schedules) or only one of them at the time (unidirectional schedules). Examining the whole PE's set, a PE can be used in parallel or sequentially according to the scheduling logic. In this scope, a iteration is over when all messages belong to a factor graph ($2N \log N$) are updated, in any order.

In this work four schedules are used, divided in basic and advanced.

Basic schedules The simplest one used is *Linear RL* that processes the rightmost nodes first (PE_5 in Figure 3.1). In this way the last PE column to be activated is PE_1. At the end of computation, the LLR restarts with the index n. It also propagates messages bidirectionally.

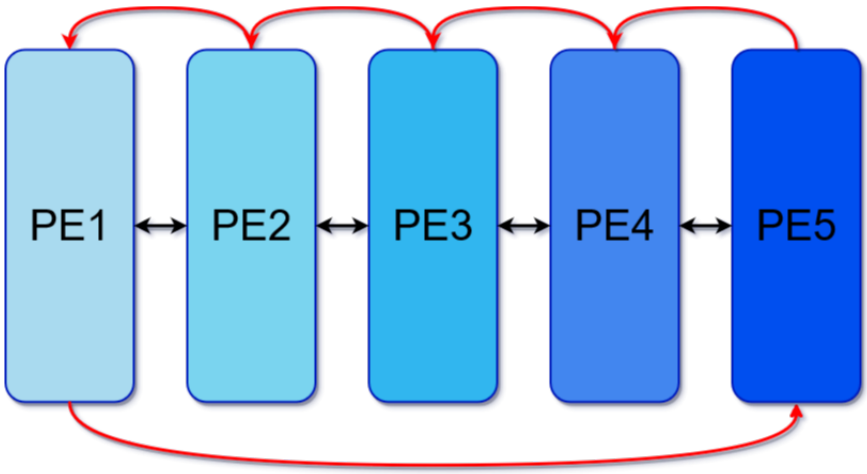


FIGURE 3.1: RL schedule

Overall, this schedule computes for each step $2N$ messages and so a iteration needs $(2NlogN)/2N = logN$ to be completed.

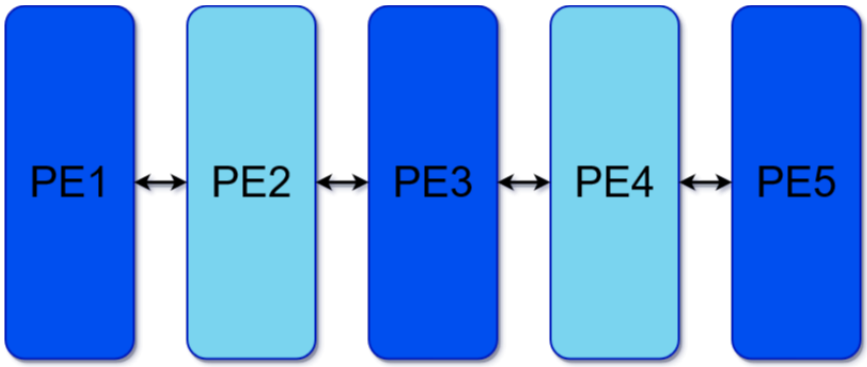


FIGURE 3.2: Stepped schedule

The other basic schedule is *Stepped*. It involves alternately the odd PE columns (blue in Figure 3.2) with the even ones (light blue). The whole $NlogN$ messages are elaborated in only two steps and it propagates them bidirectionally as *Linear RL*.

Advanced schedules The advanced schedules are more complex than the basic ones but they ensure better performances in precision and velocity convergence.

The *Biwave* (shown in Figure 3.3) is composed by *Linear RL* schedule and its opposite *Linear LR* and it runs them at the same time on the factor graph. In its unidirectional implementation, it processes $2N$ messages per step and needs $logN$ in order to finish an iteration.

Moreover, the *Circular LR* in Figure 3.4 starts the computation from first leftmost column up to n rightmost one. The iteration is concluded by completing the path in reverse starting from n rightmost column up to first leftmost one. It calculates N

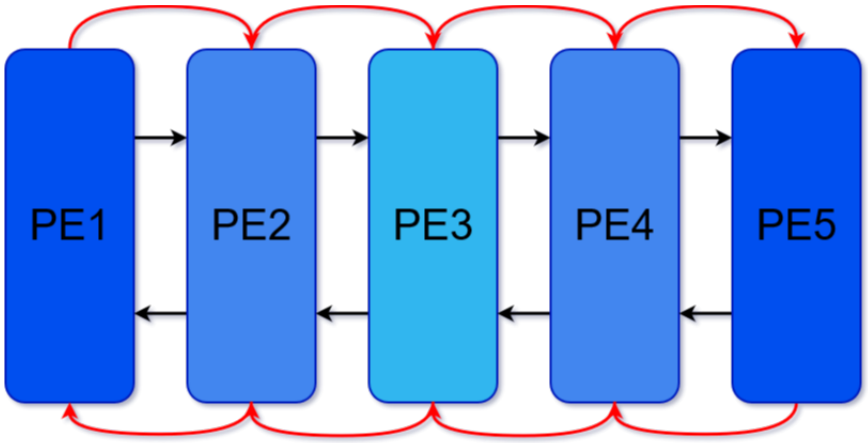


FIGURE 3.3: Biwave schedule

messages for each PE and it needs $2logN$ steps to finish an iteration in its unidirectional implementation.

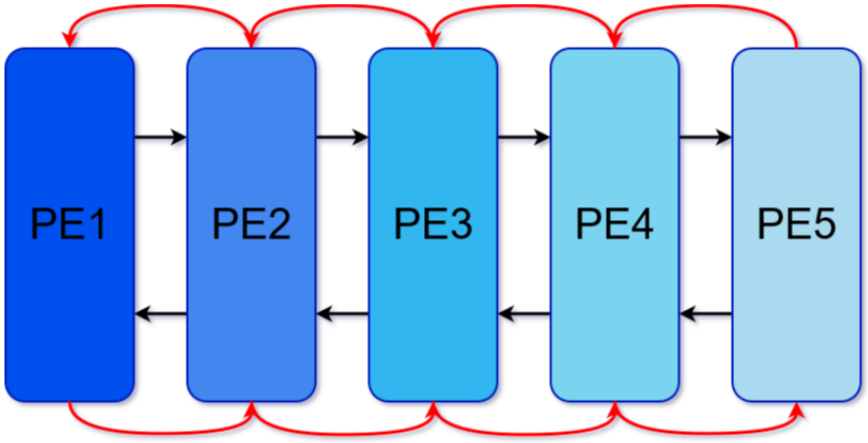


FIGURE 3.4: Circular LR schedule

For further details and clarifications see [5], [12] and [13].

3.2 Software implementations

In view of the above, it has been decided to add to the system the algorithmic simplified early stopping criterion *FBER* and a *CRC* implementation in order to perform a direct control over data. Both applications have been developed and integrated to the main program as defined below in 3.2.1 and 3.2.2.

3.2.1 FBER application

The FBER criterion allows us to obtain relatively important savings of iterations with simple and a few components. The criterion implementation in C language has been developed as precise as possible to the description provided from literature in 2.5. For this purpose, an algorithm has been created to perform the entire operation set and it is presented in the box below.

1: INPUT:

2: $L_{i,1}^{t}$ left messages; $max\_iterations$; x_i coded vector; M parameter;

3: ITERATION PROCESS (t= current iteration)

4: setting: maximum code length N, decision bit $\forall\, i \in 0 < i < N$

5: if($L_{i,1}^{t} \geqslant 0$) $dec\_FBER_i = 0$; else $dec\_FBER_i = 1$;

6: $F_{BER}^{t} += [(1 + dec\_FBER_i)/2] \oplus x_i$;

7: if($t > 1$)

8: $F_M = F_{BER}^{t+1} - F_{BER}^{t}$;

9: if($F_M == 0$) $FBER\_COUNT++$; else $FBER\_COUNT = 0$;

10: if($FBER\_COUNT == M$)

11: stop iteration;

12: $FBER\_COUNT = 0$;

13: else continue iteration ($t++$);

14: if($t = max\_iteration$) $F_M = FBER\_COUNT = 0$;

In the iteration process (from 3 to 6) the data to be processed are prepared and hard decision vector is elaborate. Subsequently, only when a complete iteration is performed, the computations useful for stopping iterations are calculated (from 7 to 13). The parameter M ensures stability to the entire criterion. Only when $FBER\_COUNT$ variable is equal to M then current iteration t will be stopped. In the event that iteration reaches the simulation's imposed limit, then variables will be manually reset.

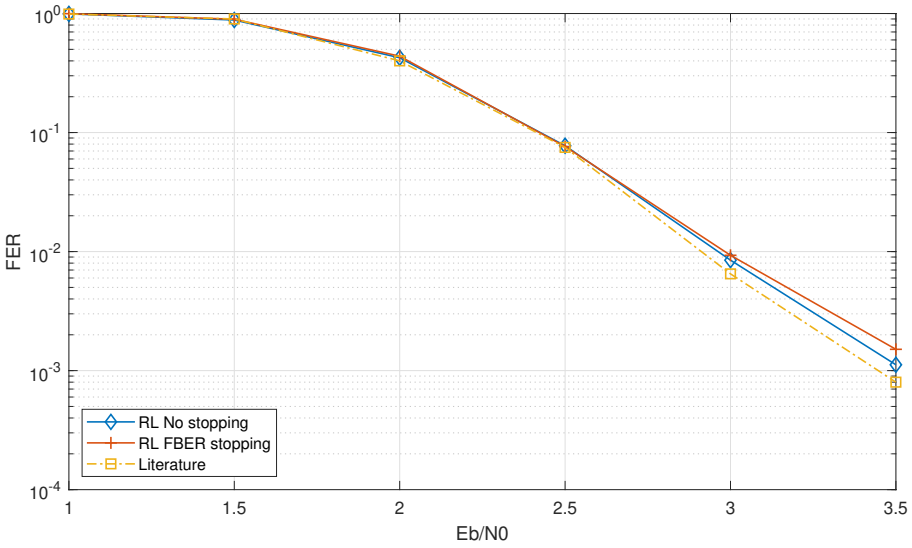


FIGURE 3.5: Software validation for FBER error with *Linear RL* scheduling compared with [5]

Software validation For software validation, it has been used *Linear RL* and *Circular LR* schedules; results are compared with [12] and [5]. Parameters' test are different between them: in first case they are $N = 1024$, $K = 512$ and 40 maximum iterations; in second one maximum iterations is fixed to 60. Obviously only *FBER* criterion was activated. Instead, M is set to 4, the lowest value for which no performance losses were registered.

It can be noticed a similar trend between curves in Figure 3.5. Until $SNR = 2.5dB$ trends of simulated curves and literature ones are practically the same; at higher SNR a slight difference between them can be found.

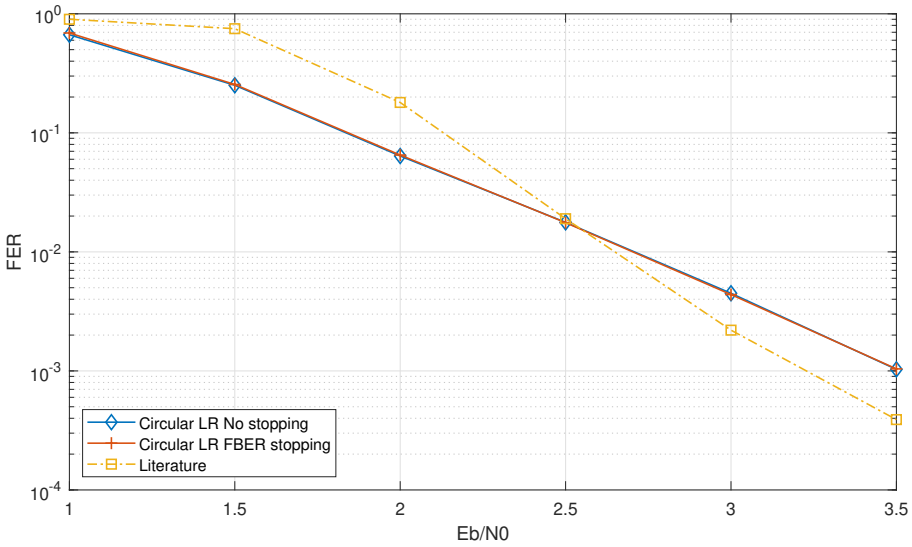


FIGURE 3.6: Software validation for FBER error with *Circular LR* scheduling compared with [12]

A different behavior has been observed regarding *Circular LR* schedule in Figure 3.6 where at lower SNR simulations show a better trend than literature results. Contrariwise, for $SNR > 2.5dB$ the errors evicted are sharply lesser. By the way simulated curves don't evince any differences between them.

These discrepancies can be dictated to the fact that an analysis on the position of frozen bit was performed in order to identify optimizations in terms of memory and processing elements. More information in 1.3.2 and for other tests and insights see Chapter 4.

3.2.2 CRC (Cyclic Redundancy Check) error data control application

The CRC system (presented in [14]) calculates checksum useful for identifying data transmission errors. It is very simple to realize in binary coding since it's composed by only XOR logic gates. In its canonical form, shown in Figure 3.7, CRC system generates a string of control bit usually attached to its relative data sequence and finally transmitted.

In order to compute CRC control a polynomial generator necessarily must be chosen. Data is linked to another polynomial $b(x)$ (degree $n-1$) and assuming that the CRC has a generator $G(x)$, $b(x)$ is shifted of G position to the left achieving $P(x)$ polynomial of $h = n-1+g$ degree. The division $P(x)/G(x)$ takes place exploiting module 2 polynomial long division without carryovers. This last operation

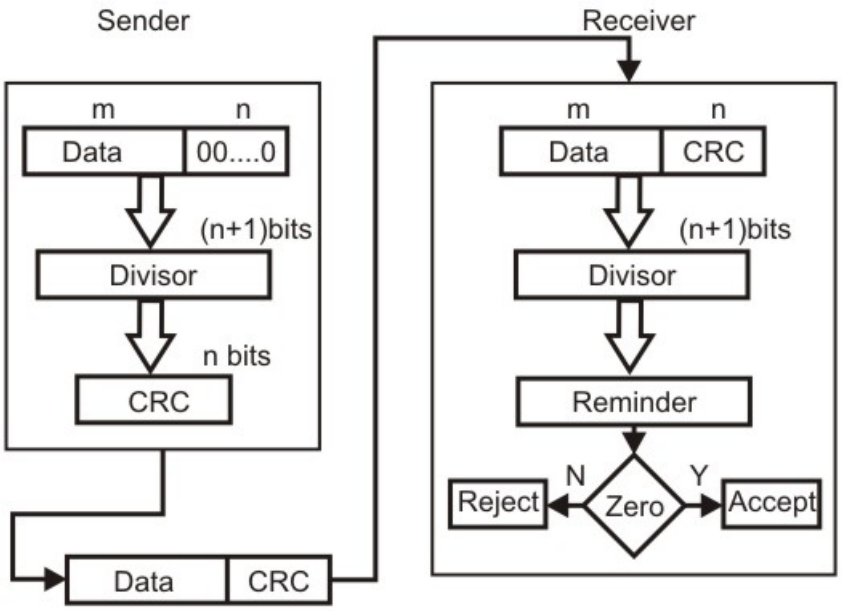


FIGURE 3.7: Basic CRC diagram

in binary arithmetic can be performed using only XOR gates obtaining a quotient $Q(x)$ and a remainder $R(x)$. Whatever polynomial divisor will be used, the fundamental functioning is the same. Finally $R(x)$ is attached to $P(X)$ and sent over transmission channel. Receiver will execute same operations considering data equal to $M(x) = P(X) - R(X)$ and if the new $R'(x)$ will result equal to the sent one then received data will be unchanged.

Every generator polynomial defines a particular kind of CRC code and two versions have been implemented:

- CRC32 with characteristic polynomial equal to $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ or in hexadecimal $0x04C11DB7$;
- CRC16 with characteristic polynomial equal to $x^{16} + x^{12} + x^5 + 1$ or in hexadecimal $0x8005$.

It's highlighted that the basic program of the algorithm that calculates the CRC in C language has been provided by Lambert Bies under open source license and can be found entirely at https://www.libcrc.org/.

The complete implementation is described in the box below. At the beginning, the messages to process are stored in specific vectors. The input data from which we will calculate the CRC are saved in $crc\_input\_uncoded_i^1$ and this one is built only at the first iteration. In the second vector $crc\_input\_coded_i^t$ left messages $L_{i,1}^t$ are stored. Both operations are performed in the iteration process (from 3 to 7). In step 8 and 9 the CRCs of $crc\_input\_uncoded_i^1$ and $crc\_input\_coded_i^t$ (relative to current iteration t) are obtained. In passage 10 the comparison takes place and only if both CRCs are equal then CRC_COUNT will be increased. Still here like in FBER application, a parameter M is necessary to ensure stability but, performing the tests, the minimum and optimal value is $M = 1$ (best iterations' stopping and no performance losses).

The others tested values are $M = 3$, $M = 4$ and $M = 6$. Choosing $M = 1$ we will stop iterations as soon as the first couple of CRCs is identical; otherwise in step 15

iterations' counter is increased and at last iteration CRC_COUNT is manually reset if criterion does not give a positive reply.

1: INPUT:

2: $L_{i,1}^{t}$ left messages; $max\_iterations$; $uncoded_N$ coded vector; M parameter;

3: ITERATION PROCESS

4: setting: maximum code length N, decision bit $\forall\, i \in 0 < i < N$

5: if($t == 1$) $crc\_input\_uncoded_i^1$=$uncodedN_i$;

6: if($L_{i,1}^{t} \geqslant 0$) $dec\_CRC_i = 0$; else $dec\_CRC_i = 1$;

7: $crc\_input\_coded_i^t$=dec_CRC_i;

8: if($t == 1$) $crc\_uncoded^1$=$function\_crc(crc\_input\_uncoded^1, N)$;

9: crc_coded^t=$function\_crc(crc\_input\_coded^t, N)$;

10: if($crc\_coded^t == crc\_uncoded^1$) $CRC\_COUNT++$; else $CRC\_COUNT = 0$;

11: if($CRC\_COUNT == M$)

13: stop iteration;

14: $CRC\_COUNT = 0$;

15: else continue iteration ($t++$);

16: if($t = max\_iteration$) $CRC\_COUNT = 0$;

Software validation Since two versions of the CRC (CRC16 and CRC32) have been implemented, they have been compared to each other in order to find their precise correspondence. As already done in FBER application, for software validation has

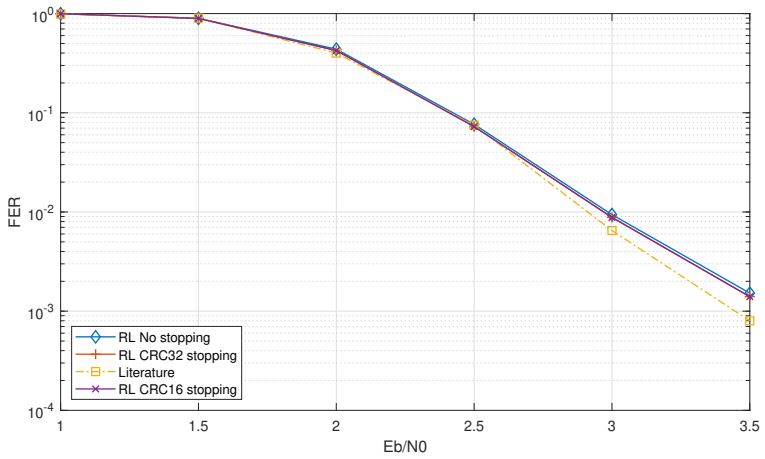


FIGURE 3.8: Software validation for CRCs errors with *Linear RL* scheduling compared with [5]

been used *Linear RL* and *Circular LR* schedules and the results are compared respectively with [12] and [5]. Also in this case parameters are in first case $N = 1024$, $K = 512$ with 40 maximum iterations and in the second one max iterations are 60.

As can be seen in Figure 3.8, CRCs errors are practically identical to the *Linear RL* and no particular differences stand out. Literature's curve has still a better behavior at higher SNR.

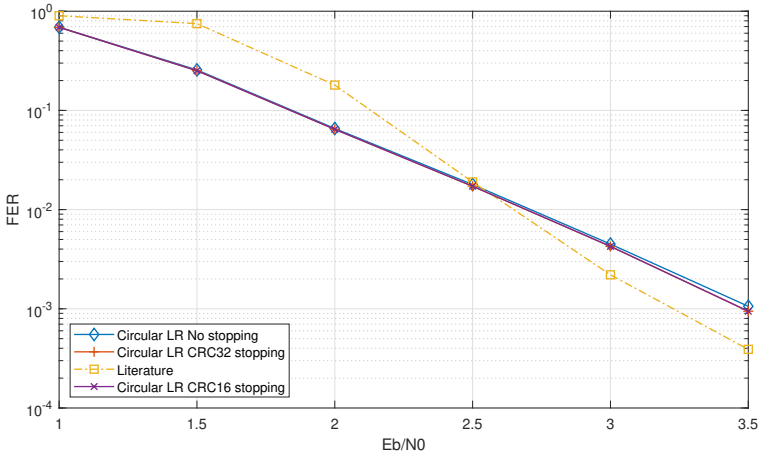


FIGURE 3.9: Software validation for CRCs errors with *Circular LR* scheduling compared with [12]

Instead different trends come out analyzing the graph in 3.9 in which CRCs confirm in all SNR steps the no stopping results. Surprisingly, CRCs errors at $SNR = 3.5dB$ are a little bit less than no stopping ones. Also in this case, at $SNR < 2.5dB$ simulated curves are better respect to the literature one but at higher SNR situation is reversed. As explained above in FBER application, these differences are caused from frozen bit position's optimization (see 1.3.2).

The Figure 3.8 and Figure 3.9 affirm that it does not matter what is the polynomial generator $G(x)$ chosen to compute the CRC because they reach the same results. From here on, for later tests, only CRC32 version will be used in comparisons in order to simplify reading.

Chapter 4

Numerical simulations

After the FBER and CRC systems' validation, the outcomes of numerical tests will be analyzed. More precisely, results will be separated first considering the parameters N and K, then they will be grouped taking into account the stopping criterion and, at the end, which schedule's band has been used. Schedules' groups will be called *basic* and *advanced schedule* as presented in 3.1.1.

Finally, hybrid tests will be compared. In hybrid solutions, both the stopping criterion and the CRC data control have been activated trying to find out a better compromise between performance and computational costs. In the construction of the hybrid methods, precedence has been given to algorithmic criteria rather than to the CRC method. Precisely for this reason, in the case in which both operating criteria reply positively at the same iteration, the algorithmic one will prevail.

The parameters used to perform the whole tests, except N and K, are:

- *max iteration*= 40 (when different is specified);
- *max simulated frames*= 300000;
- *max wrong simulated frames*= 150;
- *intrinsic information fractional bits*= 7.
- *intrinsic information*: $[min, max] = [-2048, 2047]$;
- *extrinsic information*: $[min, max] = [-2048, 2047]$;
- *R messages initialization value*= 2047:
- *s scaling factor*: $[left, right] = [0.937500, 1.00]$

4.1 N=1024, K=512: numerical results

For the following comparisons, reference was made to [12] as regards *Linear RL* and *Stepped* schedules and to [5] for *Circular LR* and *Biwave*. Besides, results from [10] have been introduced in order to achieve a broad spectrum analysis.

Afterwards the complete analysis of the criteria, a general comparison between them is presented in order to understand which has the better stopping behavior varying SNR scale.

4.1.1 G-Matrix criterion

In these tests, the G-Matrix is the only early stopping criterion to have been activated. Great differences between basic and advanced schedules has been detected both in errors' comparison and in iterations' stopping.

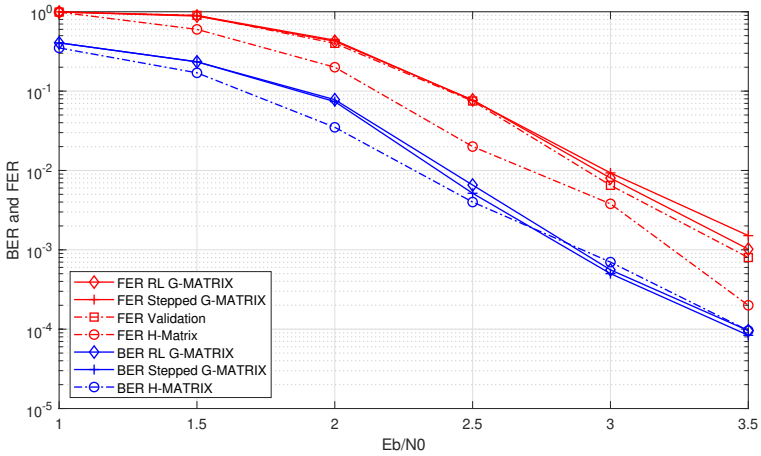


FIGURE 4.1: G-Matrix BER-FER with *Linear RL* and *Stepped* $N = 1024$, $K = 512$

Basic schedules In Figure 4.1 BER performances with *Linear RL* and *Stepped* schedules are aligned with the other ones except at lower SNR where H-Matrix works better. FER's curves follow same validation trend even if *Stepped* is the worst and H-Matrix reaches the lowest FER value.

Instead, analyzing Figure 4.2 and Table 4.1, it's possible to notice that iterations of the two basic schedules are quite efficient reaching 22 (45% savings) and 20.6 (48.5%) average iterations respectively for *Linear RL* and *Stepped*. Also they exceed G-Matrix literature result.

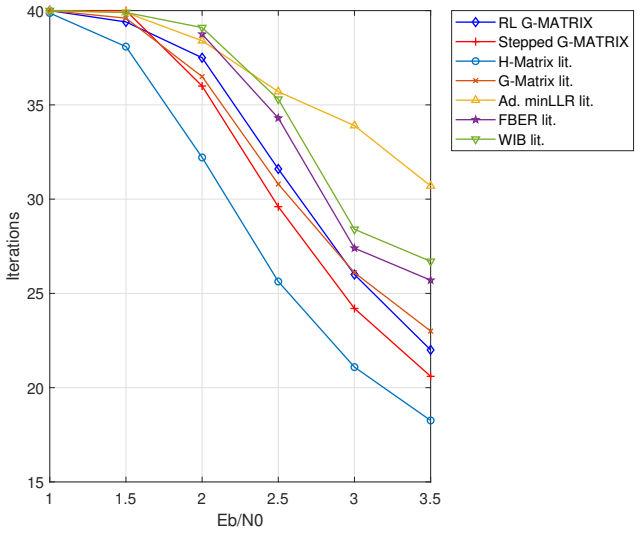


FIGURE 4.2: Iterations with G-Matrix criterion and basic schedules $N = 1024$, $K = 512$

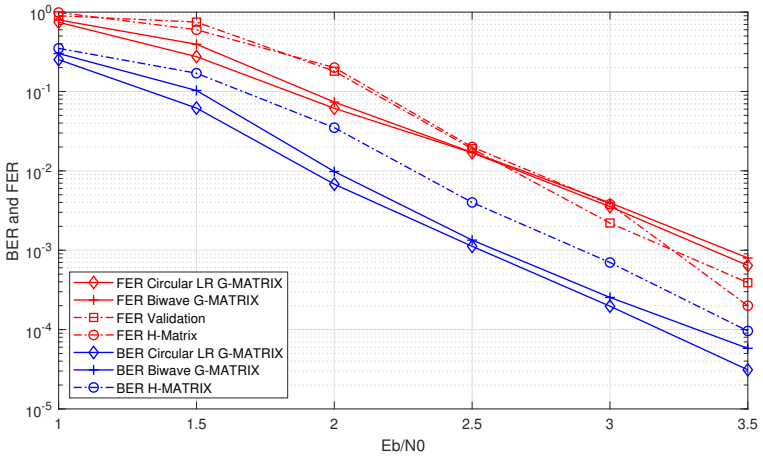


FIGURE 4.3: G-Matrix BER-FER with *Circular LR* and *Biwave* schedules $N = 1024$, $K = 512$

Advanced schedules *Circular LR* and *Biwave* have in all cases best performances if compared with basic schedules. However, FER trends in Figure 4.3 are similar between them and only at $SNR = 2.5dB$ they almost converge in a point. Before this, simulated curves have a lower error; after literature results become better. Talking about BER, simulated cases have a optimal conduct for all SNR spectrum: they always stay below the others curves and they conclude almost close to 10^{-6}.

Viewing iteration results in Figure 4.4 and still in Table 4.1 it can be concluded that G-Matrix criterion with advanced schedules outclasses H-Matrix with a minimum gap of about 7 iterations until a max of about 10. *Circular LR* and *Biwave* reach a iteration drop equal to 80.8% for *Biwave* and 86.3% for *Circular LR*. It's necessary to underline that the advanced schedules with G-Matrix criterion is the most complex case concerning hardware components in no hybrid group.

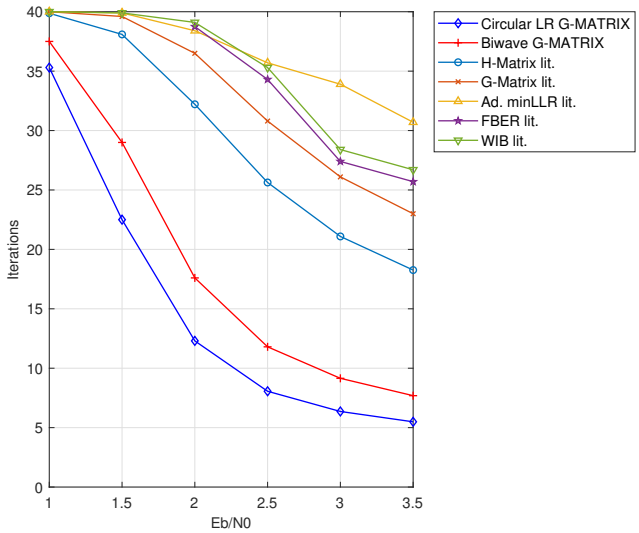


FIGURE 4.4: Iterations with G-Matrix and advanced schedules $N = 1024$, $K = 512$

G-Matrix criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	31.6	21 %
	3.0 dB	26.0	35 %
	3.5 dB	22.0	45 %
Stepped	2.5 dB	29.6	26 %
	3.0 dB	24.2	39.5 %
	3.5 dB	20.6	48.5 %
Circular LR	2.5 dB	8.06	79.9 %
	3.0 dB	6.36	84.1 %
	3.5 dB	5.49	86.3 %
Biwave	2.5 dB	11.8	70.5 %
	3.0 dB	9.15	77.1 %
	3.5 dB	7.69	80.8 %

TABLE 4.1: Average iterations with G-Matrix criterion $N = 1024$, $K = 512$

4.1.2 FBER criterion

Here the implementation explained in 3.2.1 has been put to the test in order to understand how much the system is reliable and performing. Obviously only FBER criterion has been activated as early stopping termination.

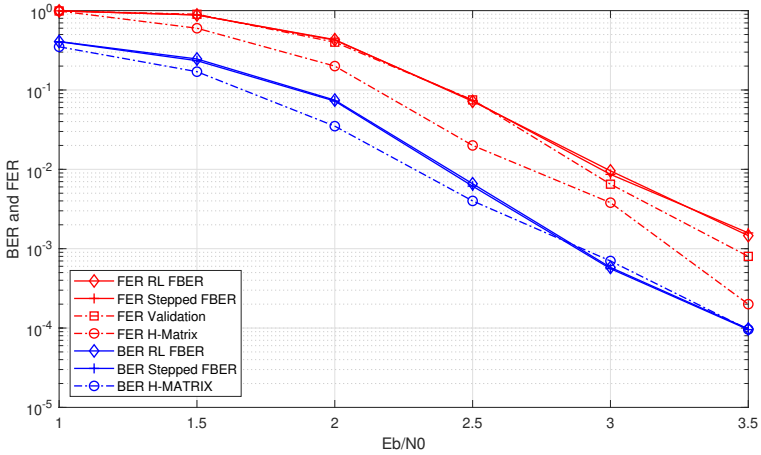


FIGURE 4.5: FBER BER-FER with *Linear RL* and *Stepped* schedules $N = 1024$, $K = 512$

Basic schedules The error diagram shown in 4.5 evinces that the simulations give practically the same outcomes of validation test and basic schedules have the same slight difference in FER with the curves in [5]; this proves and confirms the method's coherence and its actual correctness both in terms of theory and its application. The H-Matrix's error is still the better FER against the basic schedules. Contrariwise the BERs' results end for all tests and literature's data at the same value.

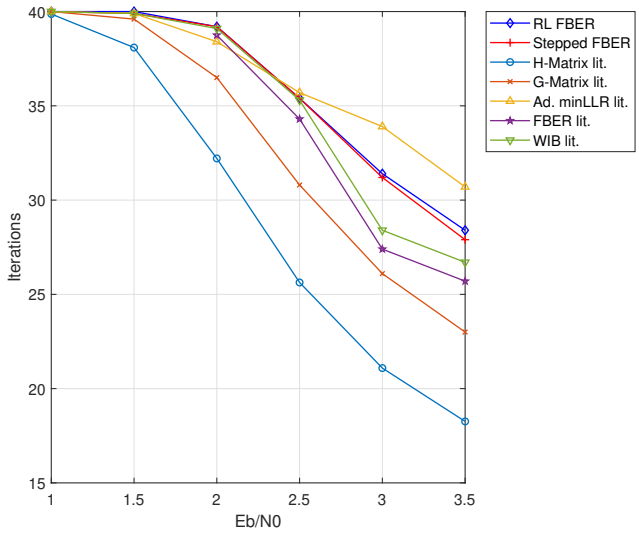


FIGURE 4.6: Iterations with FBER criterion and basic schedules $N = 1024$, $K = 512$

The stopping iterations' attainments are very similar for both basic schedules with a very light benefits for *Stepped* one (see Figure 4.6 and Table 4.2); indeed they distance themselves by only 1.3% @3.5dB in stopping rate, half iteration. The literature's FBER value arrives to 26.5 iterations (already shown in 2.6) with a gap from the simulated tests of about 4% (1.9 iterations).

Advanced schedules The situation changes when we use advanced schedules since FERs' results are quite close to lower $SNRs$ and after $SNR = 2.5dB$ they almost coincide. In Figure 4.7 *Circular LR* and *Biwave* schedules are very similar in FER's terms.

Furthermore, FER's trends and the validation test look alike too and this demonstrate the tests' reliability. The BER parameter is better throughout the SNR scale remaining below enough the H-Matrix literature's values.

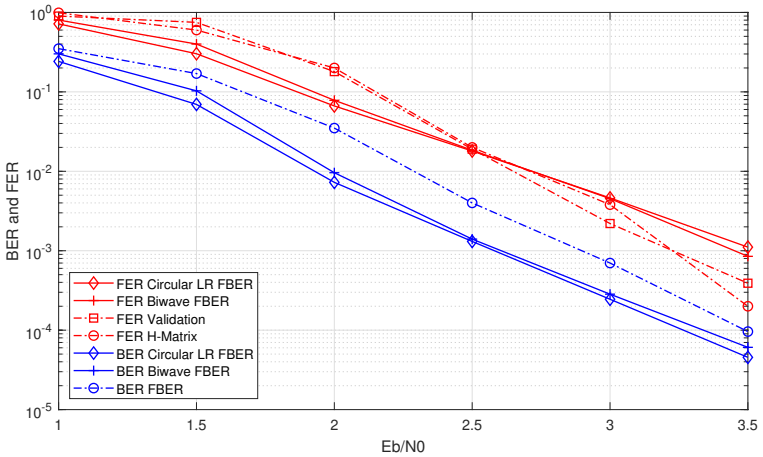


FIGURE 4.7: FBER BER-FER with *Circular LR* and *Biwave* schedules $N = 1024$, $K = 512$

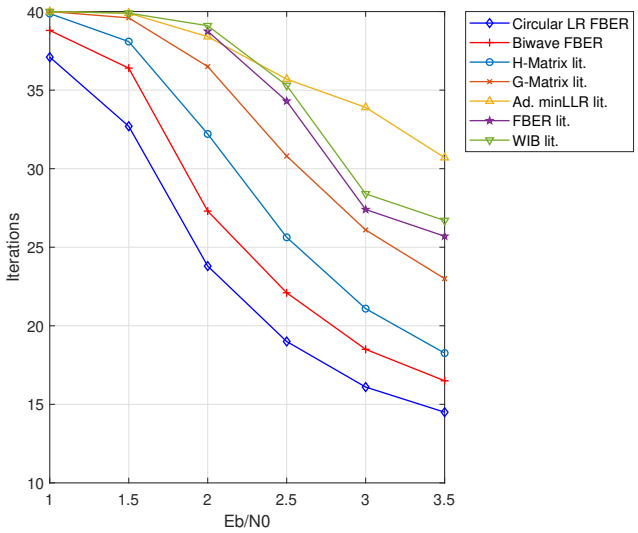


FIGURE 4.8: Iterations with FBER criterion and advanced schedules $N = 1024$, $K = 512$

The advanced schedules achieve about fifteen iterations and they succeed to overtake H-Matrix method reaching at $SNR = 3.5dB$ 16.5 iterations for *Biwave* and 14.5 for *Circular LR* with a savings of respectively 58.8% and 63.8%. Moreover they outdistance from H-Matrix criterion of 1.7 iterations and 3.7 (4.3% and 9.3% less). The whole data are referred to Figure 4.8 and Table 4.2.

FBER criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	35.4	11.5 %
	3.0 dB	31.4	21.5 %
	3.5 dB	28.4	29 %
Stepped	2.5 dB	35.4	11.5 %
	3.0 dB	31.2	22 %
	3.5 dB	27.9	30.3 %
Circular LR	2.5 dB	19	52.5 %
	3.0 dB	16.1	59.8 %
	3.5 dB	14.5	63.8 %
Biwave	2.5 dB	22.1	44.8 %
	3.0 dB	18.5	53.8 %
	3.5 dB	16.5	58.8 %

TABLE 4.2: Average iterations with FBER $N = 1024$, $K = 512$

4.1.3 CRC criterion

As done with the FBER criterion, CRC method has been tested in order to include an all-around analysis and understand its strengths and reliability. Also in this case

only CRC criterion has been put into operation and, as mentioned before in 3.2.2, only CRC32 version's tests will be analyzed just to improve the reading.

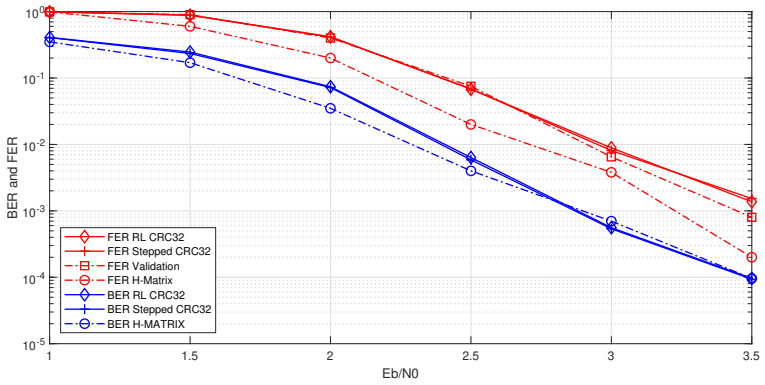


FIGURE 4.9: CRC32 BER-FER with *Linear RL* and *Stepped* schedules

Basic schedules With *Linear RL* and *Stepped*, CRC criterion replies with a perfect alignment between schedules in FERs' performances. Their trends, represented in Figure 4.9, overlap each other for all SNR values compared to the validation test. Besides, the same behavior can be found in BER path attaining, at high SNR, the H-Matrix method's result. Hence we can consider definitively CRC as a reliable method because it proves coherence and convergence.

Under the profile of stopping performances in Figure 4.10 and Table 4.3, CRC criterion with basic schedules sure suprises since they catch the G-Matrix literature's result presented in 2.6: at $SNR = 3.5dB$ the *Stepped* is distant 0.4 iteration, only 1% more, while the *Linear RL* even overtake G-Matrix result with 22.9 iterations against 23 iterations. The gap is really sligth, only the 0.25% less, but this data confirms that CRC system can also provide optimal stopping results.

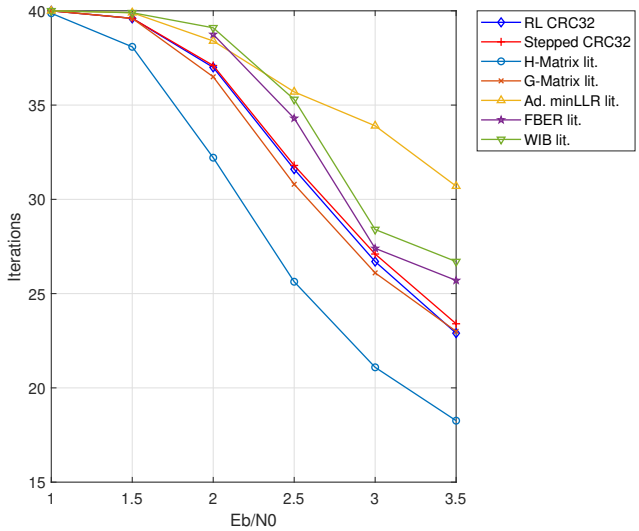


FIGURE 4.10: Iterations with CRC32 criterion and basic schedules
$N = 1024$, $K = 512$

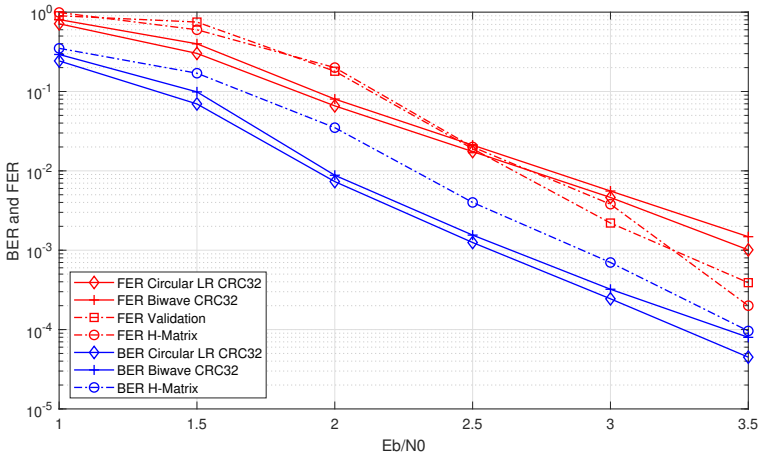


FIGURE 4.11: CRC32 BER-FER with *Circular LR* and *Biwave* schedules $N = 1024$, $K = 512$

Advanced schedules The overcomes with *Circular LR* and *Biwave* validate the argumentation just made for basic schedules. Indeed even the trends figured in 4.11 confirm the CRC method's goodness: it provides a FERs' path completely in line with the validation test. As in the others advanced performed tests, still here BERs' trends are always below the H-Matrix result.

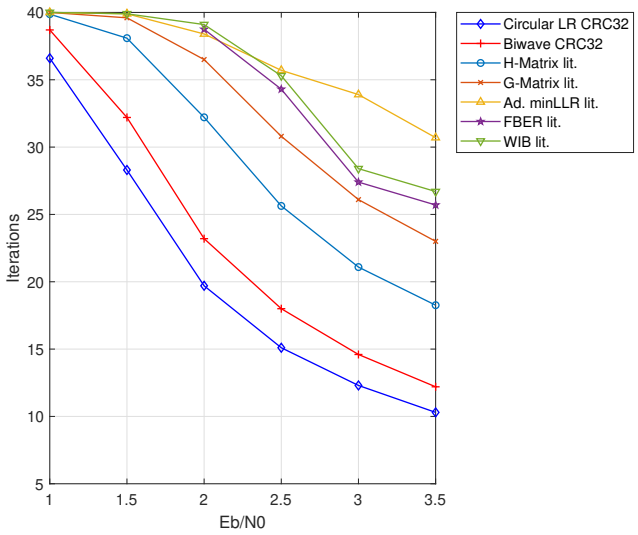


FIGURE 4.12: Iterations with CRC32 criterion and advanced schedules $N = 1024$, $K = 512$

Analyzing Figure 4.12 and the following table, we can notice that CRC32 gives a less steep iterations' drop but it gets off under fifteen iterations fot both schedules. *Biwave* reaches at $SNR = 3.5dB$ 12.2 average iterations (69.5% of savings) and *Circular LR* achieves 10.3 (74.3%). It's an excellent result that supports the idea that CRC can offer optimal performances with seriously poor costs.

CRC32 criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	31.6	21 %
	3.0 dB	26.7	33.3 %
	3.5 dB	22.9	42.8 %
Stepped	2.5 dB	31.8	20.5 %
	3.0 dB	27.1	32.3 %
	3.5 dB	23.4	41.5 %
Circular LR	2.5 dB	15.1	62.3 %
	3.0 dB	12.3	69.3 %
	3.5 dB	10.3	74.3 %
Biwave	2.5 dB	18.0	55 %
	3.0 dB	14.6	63.5 %
	3.5 dB	12.2	69.5 %

TABLE 4.3: Average iterations with CRC32 $N = 1024$, $K = 512$

4.1.4 Hybrid solution: G-Matrix merged with CRC

The first hybrid solution is composed by the synergy of the G-Matrix criterion with CRC32 data error control. As mentioned in Chapter 4, priority has been given to the algorithmic method, in this case the G-Matrix. This method will also be named as *hybrid 1* for brevity if necessary.

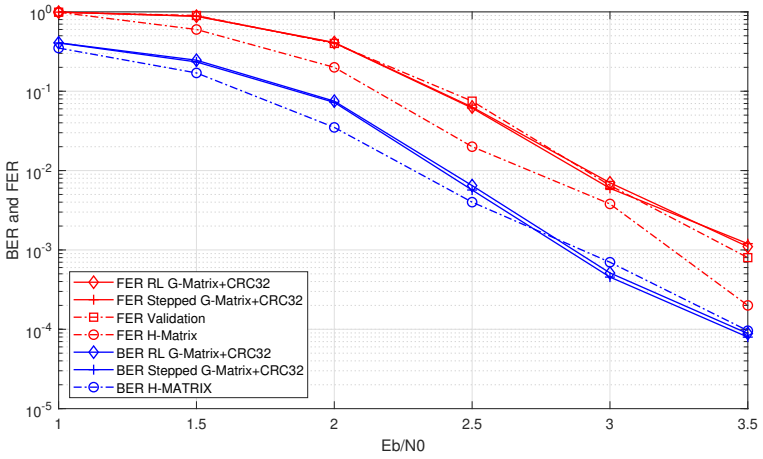


FIGURE 4.13: Hybrid G-Matrix + CRC32 BER-FER with *Linear RL* and *Stepped* schedules $N = 1024$, $K = 512$

Basic schedules Observing the curves in Figure 4.13 relative to FER parameter it's possibile to notice that the trends are very close to the validation FER value and at $SNR = 3.5dB$ both *Linear RL* and *Stepped* schedule considerably approached to literature's curves. The same thing happens for BER parameter's curves that slightly end @3.5dB below the H-Matrix value although at lower SNR they still suffer against the last one.

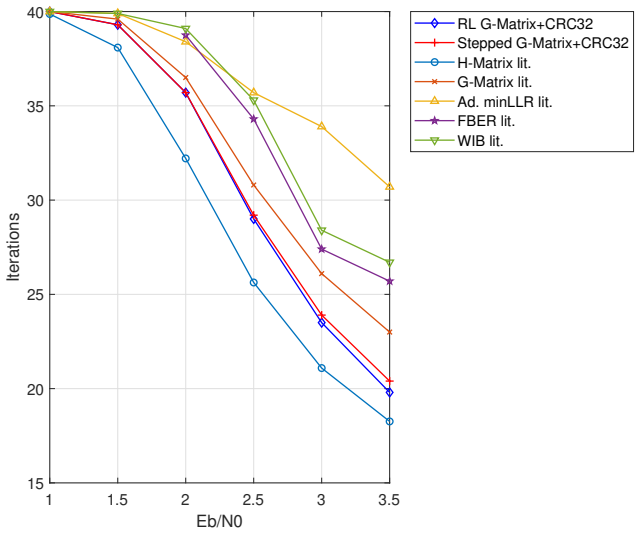


FIGURE 4.14: Iterations with G-Matrix + CRC32 criterion and basic schedules $N = 1024$, $K = 512$

As regards the stopping efficiency in Figure 4.14 and Table 4.4, it highlights a functioning beyond expectations with a solid drop that halves maximum iterations' number. The basic schedules reach 19.8 iterations for *Linear RL* and 20.4 for *Stepped* one (respectively 50.5% and 49% less). They are in any case better then G-Matrix's result in the whole SNR scale (with a minimum and maximum iteration's gap of 1.6 and 3.2) and the elaborated tests are also close to the H-Matrix's outcome with only a maximum diffrence of 2.2 iterations. It's necessary to underline that the CRC32 method adds only a check in cascade to G-Matrix criterion.

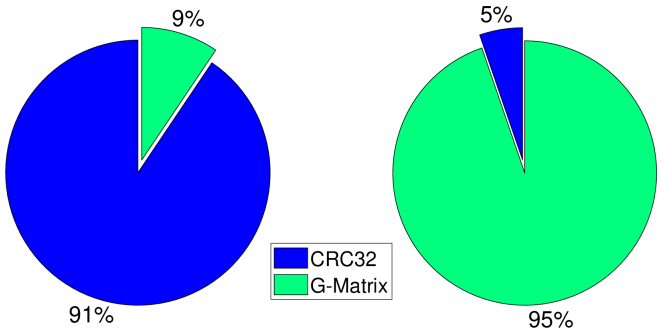


FIGURE 4.15: Percentage of stopping criteria's successes between G-Matrix and CRC32 with basic schedules $N = 1024$, $K = 512$. Left *Linear RL*, right *Stepped*

It has been possible to assess the working of individual criteria regarding hybrid solutions too. Indeed, as shown in Figure 4.15, the data have been summarized and approximated in order to examine discrepancies. In leftmost pie chart the uses for the *Linear RL* schedule are shown and it underlines an absolute predominance of the CRC with a percentage of successes equal to 91%. Contrariwise, in *Stepped* case on right, results are reversed with a G-Matrix's overpowering working with 95% of successes.

Advanced schedules The errors figured in 4.16 describes a situation very similar to validation test since FER's data follows the same validation's path even if the *Circular LR* has a better precision if compared with *Biwave*. Until $SNR = 3.0dB$ the simulated curves have a better trends than H-Matrix one. Also in BER's performance each schedules are below H-Matrix literature curves.

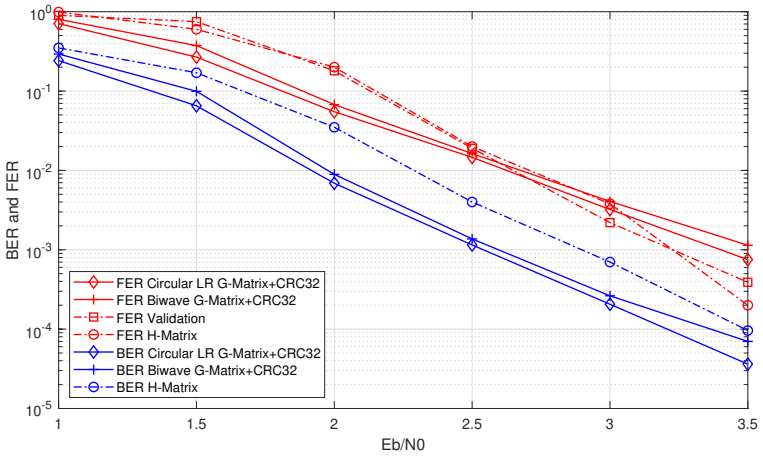


FIGURE 4.16: Hybrid G-Matrix + CRC32 BER-FER with *Circular LR* and *Biwave* schedules $N = 1024$, $K = 512$

In Figure 4.17 is represented the average stopping iterations' yield for the advanced schedules and it displays the best steepness between $SNR = 1.0dB$ to $2dB$ so far encountered. After that threshold, curves become much less steep concluding at $SNR = 3.5dB$ with 7.63 iterations for *Biwave* and 5.48 for *Circular LR* (data are from Table 4.4). These are the lowest obtainable results but we must consider that this hybrid solution is the most complex and expensive criterion between all of them in terms of components.

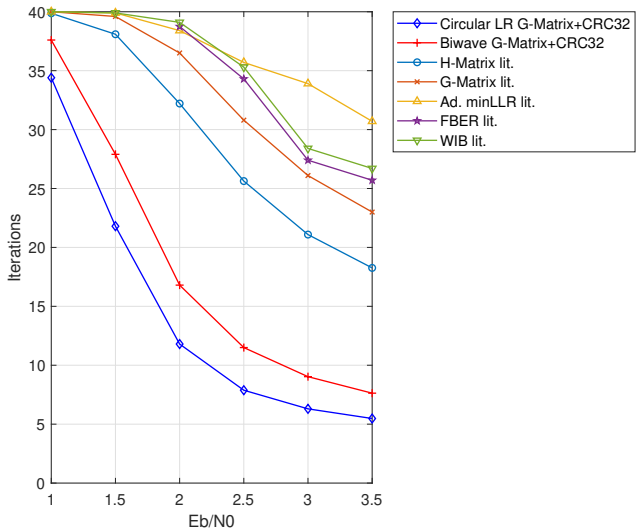


FIGURE 4.17: Iterations with hybrid G-Matrix + CRC32 criterion and advanced schedules $N = 1024$, $K = 512$

Moreover, looking at Figure 4.18, the outcomes obtainted with advanced schedules are very similar among them. In the leftmost pie chart concerning *Biwave*, the

G-Matrix criterion is widely predominant rather than CRC32 with a hits' percentage of 94%. The same trend is obeserved for *Circular LR* schedule on the right with a result even more unbalanced: G-Matrix has a success rate equal to 99% against only 1% of CRC32.

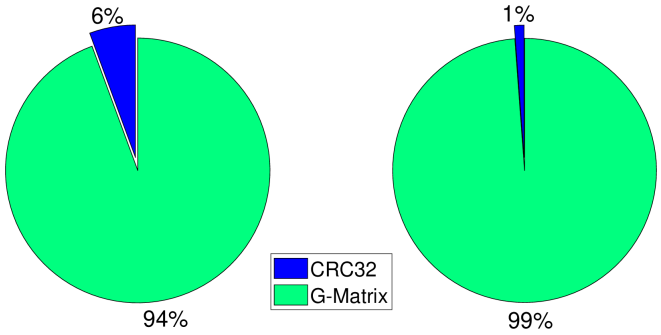


FIGURE 4.18: Percentage of stopping criteria's successes between G-Matrix and CRC32 with advanced schedules $N = 1024$, $K = 512$. Left *Biwave*, right *Circular LR*

G-Matrix + CRC32 hybrid criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	29.0	27.5 %
	3.0 dB	23.5	41.3 %
	3.5 dB	19.8	50.5 %
Stepped	2.5 dB	29.2	27 %
	3.0 dB	23.9	30.3 %
	3.5 dB	20.4	49 %
Circular LR	2.5 dB	7.88	80.3 %
	3.0 dB	6.30	84.3 %
	3.5 dB	5.48	86.3 %
Biwave	2.5 dB	11.5	71.3 %
	3.0 dB	9.03	77.4 %
	3.5 dB	7.63	80.9 %

TABLE 4.4: Average iterations with G-Matrix + CRC32 $N = 1024$, $K = 512$

4.1.5 Hybrid solution: FBER merged with CRC

The second hybrid criterion is the combination between algorithmic standard FBER and CRC data error control. As already done previously, it reiterates that leading has been given to FBER method (the algorithmic one) rather than CRC. This solution will also be named as *hybrid 2* for brevity if necessary.

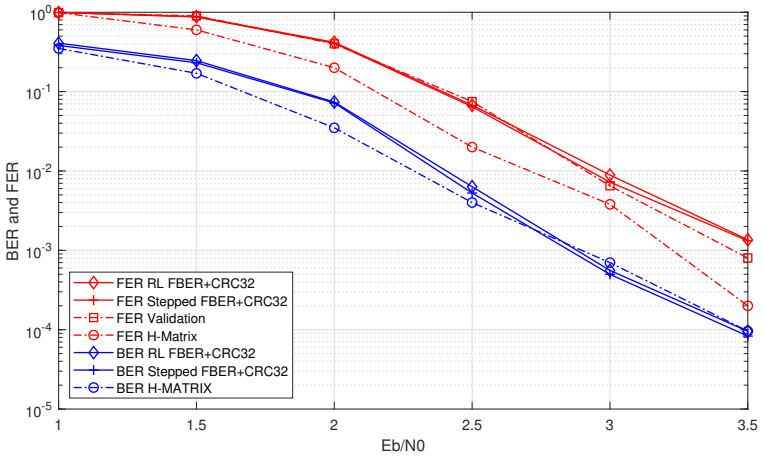


FIGURE 4.19: Hybrid FBER + CRC32 BER-FER with *Linear RL* and *Stepped* schedules $N = 1024$, $K = 512$

Basic schedules The error's curves shown in Figure 4.19 let us guess that FERs' values are very close between the two path of *Linear RL* and *Stepped*. They overlap each other in every SNR point except at $3.0dB$ where the *Stepped* one is a slightly below. The BER's performance is aligned with H-Matrix one and at high SNR all curves are practically the same.

Regarding the stopping iterations' performances, it's possibile to notice that both tested basic schedules exceed G-Matrix literature's result. *Stepped* behaves really

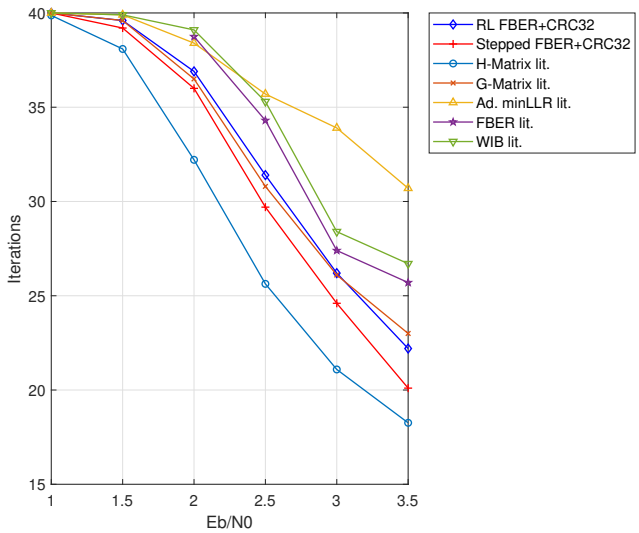


FIGURE 4.20: Iterations with hybrid FBER + CRC32 criterion and basic schedules $N = 1024$, $K = 512$

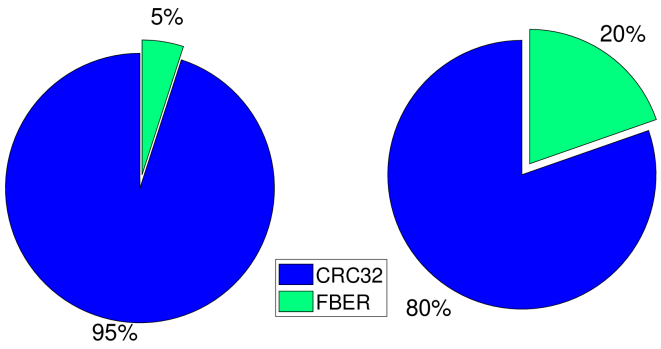


FIGURE 4.21: Percentage of stopping criteria's successes between FBER and CRC32 with basic schedules $N = 1024$, $K = 512$. Left *Linear RL*, right *Stepped*

well achieving at $3.5dB$ 20.1 iterations with a drop of 49.8%; the H-Matrix criterion is not that far with a gap of 1.9 iterations. Instead the *Linear RL* schedule reaches 22.2 iterations with a distance from G-Matrix of only 0.8 iteration. Observing the results altogether, we can say that this hydrid solution offers an optimal iteration reduction considering that it saves a large number of components. All results are in Figure 4.20 and Table 4.5.

How it can be seen from 4.21, the 95% of successes for *Linear RL* (on left side) is given from CRC32 at the FBER's expense; this establishes that CRC32 is absolutely fundamental for the operation of the method. On right is shown the uses' distribution of the two stopping criteria for *Stepped* schedule; here the outcomes are more balanced with an 80% for CRC32 against a 20% of FBER system.

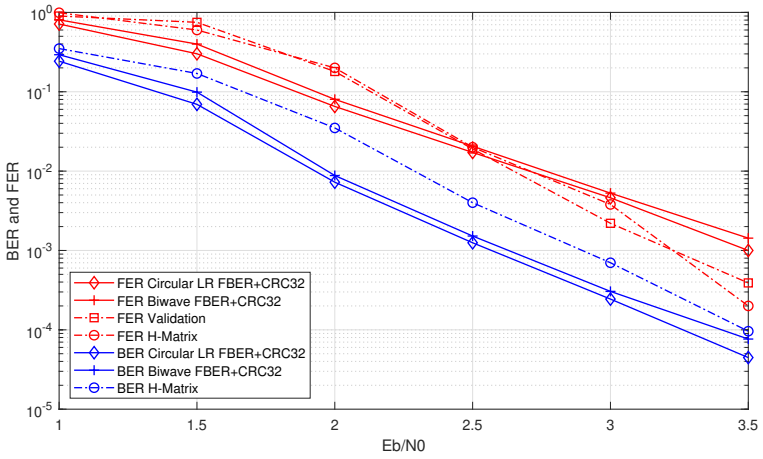


FIGURE 4.22: Hybrid FBER + CRC32 BER-FER with *Circular LR* and *Biwave* schedules $N = 1024$, $K = 512$

Advanced schedules If we observe the graph in Figure 4.22, we can verify that the FERs' trends are aligned with the validation one and the *Circular LR* schedule conclude its path at $3.5dB$ at the lower limit of 10^{-3}. The BER parameter results to be better in every case than the H-Matrix one and, moreover, the *Circular LR* obtains a great outcome reaching about 10^{-4} at $SNR = 3.5dB$.

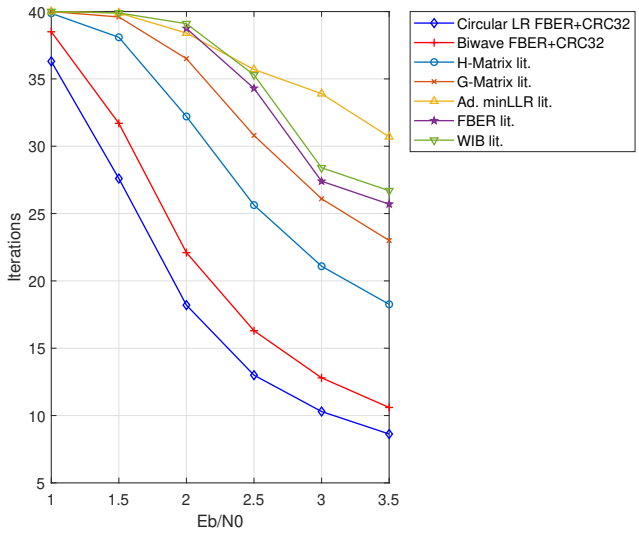


FIGURE 4.23: Iterations with hybrid FBER + CRC32 criterion and advanced schedules $N = 1024$, $K = 512$

The second graph inherent to average iterations in 4.23 describes excellent results' curves because they get very close to 10 iterations and even the *Circular LR* attains 8.63 iterations (a drop of 78.4%). The *Biwave* schedule stops at 10.6 iterations. Details can be found in Table 4.5. Also in this case, it's possible to assert that this hybrid method can get very close to more complex criteria only using a fraction of their computational costs.

Advanced schedules provide in 4.24 outcomes very similar among them since the gap between hits' percentages of CRC32 and FBER criteria is minimal: the difference is less of 1% between the two schedules with values that fluctuate from 94% to 95% for CRC method. The CRC, also in this case, proves to be the key criterion in

FBER + CRC32 hybrid criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	31.4	21.5 %
	3.0 dB	26.2	34.5 %
	3.5 dB	22.2	44.5 %
Stepped	2.5 dB	29.7	25.8 %
	3.0 dB	24.6	38.5 %
	3.5 dB	20.1	49.8 %
Circular LR	2.5 dB	13.0	67.5 %
	3.0 dB	10.3	74.3 %
	3.5 dB	8.63	78.4 %
Biwave	2.5 dB	16.3	59.3 %
	3.0 dB	12.8	68 %
	3.5 dB	10.6	73.5 %

TABLE 4.5: Average iterations with hybrid FBER + CRC32 and advanced schedules $N = 1024$, $K = 512$

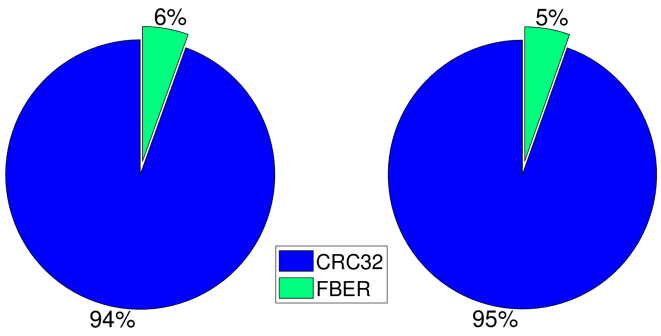


FIGURE 4.24: Percentage of stopping criteria's successes between FBER and CRC32 with advanced schedules $N = 1024$, $K = 512$. Left *Biwave*, right *Circular LR*

order to achieve excellent results.

4.1.6 Comparison between criteria

In Table 4.6 the most relevant results presented so far are grouped in order to compare them with each other. Data are divided by schedules and which early stopping criterion has been applied.

Comparison between early stopping criteria with 40 maximum iterations						
Criterion		*G-Matrix*	*FBER*	*CRC*	*G-Matrix+CRC*	*FBER+CRC*
Schedule	**SNR**	*Average iterations / Iterations' drop*				
Linear RL	2.5 dB	31.6/21%	35.4/11.5%	31.6/21%	29.0/27.5%	31.4/21.5%
	3.0 dB	26/35%	31.4/21.5%	26.7/33.3%	23.5/41.3%	26.2/34.5%
	3.5 dB	22.0/45%	28.4/29%	22.9/42.8%	19.8/50.5%	22.2/44.5%
Stepped	2.5 dB	29.6/26%	35.4/11.5%	31.8/20.5%	29.2/27%	29.7/25.8%
	3.0 dB	24.2/39.5%	31.2/22%	27.1/32.3%	23.9/40.3%	24.6/38.5%
	3.5 dB	20.6/48.5%	27.9/30.3%	23.4/41.5%	20.4/49%	20.1/49.8%
Circular LR	2.5 dB	8.06/79.9%	19.0/52.5%	15.1/62.3%	7.88/80.3%	13.0/67.5%
	3.0 dB	6.36/84.1%	16.1/59.8%	12.3/69.3%	6.3/84.3%	10.3/74.3%
	3.5 dB	5.49/86.3%	14.5/63.8%	10.3/74.3%	5.48/86.3%	8.63/78.4%
Biwave	2.5 dB	11.8/70.5%	22.1/44.8%	18.0/55%	11.5/71.25%	16.3/59.3%
	3.0 dB	9.15/77.1%	18.5/53.8%	14.6/63.5%	9.03/77.4%	12.8/68%
	3.5 dB	7.69/80.8%	16.5/58.8%	12.2/69.5%	7.63/80.9%	10.6/73.5%

TABLE 4.6: Stopping performance comparison between criteria with 40 maximum iterations and $N = 1024$, $K = 512$

The best criterion in terms of absolute performance is certainly the hybrid criterion G-Matrix + CRC which is superior in every test except for the case FBER + CRC with *Stepped* schedule. Benefits range from a minimum of 27% for *Linear RL* up to a maximum of 86.3% of *Circular LR*. By the way viewing the results of G-Matrix pure tests, it can be affirm that there are no important differences among them which justify the CRC execution too. In fact in Figure 4.25 and 4.26, the deviations are practically absent at higher $SNRs$.

The CRC criterion succeeds in overcoming the FBER by distancing itself by a minimum of 9% with *Stepped* schedule at $SNR = 2.5dB$ up to 12.9% with *Linear RL* at

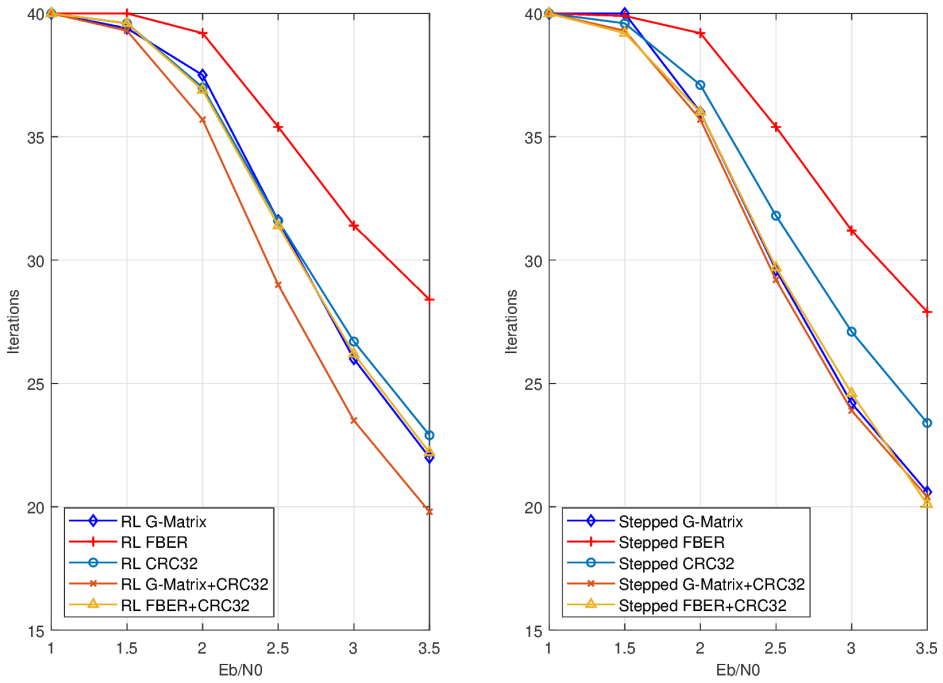


FIGURE 4.25: Comparison between stopping criteria sorted by basic schedules $N = 1024$, $K = 512$

$SNR = 3.5dB$. The higher the SNR value, the more the gap between them increases. Moreover, the CRC criterion often comes close to the results of the G-Matrix one above all in basic schedules.

Instead the FBER criterion fails to reach others finishing last in each tests: it does

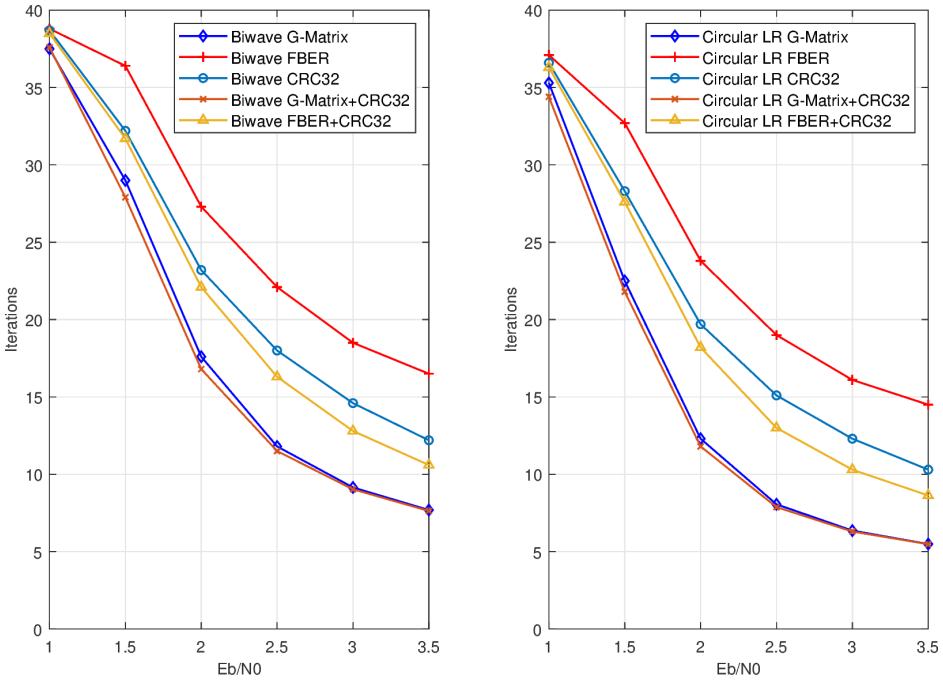


FIGURE 4.26: Comparison between stopping criteria sorted by advanced schedules $N = 1024$, $K = 512$

not achieve a iteration's drop greater than 30.3% and 63.8% with respectively basic and advanced schedules. It can be stated that the FBER criterion is parsimonious from the point of view of complexity costs but does not excel in the performance.

The best compromise between performance and complexity costs is represented by the hybrid solution that combines FBER with the CRC. Indeed it has very similar performances to the G-Matrix criterion even though it does not cost as much as it. This hybrid solution provides outcomes of iteration's drop ranging from 21.5% up to 78.4%. Only for advanced schedules at $SNR = 2.5dB$ it is outdistanced from the G-Matrix criterion and its hybrid version of several percentage points. It is undeniable that it offers excellent performance with savings in terms of complexity costs.

Instead, observing the behaviors of the single schedules it can be noted that all curves in figures have the same trends except for the *Stepped* schedule that overlaps G-Matrix, hybrid G-Matrix + CRC and hybrid FBER + CRC almost on a single line. This behavior is surprising for a simple criterion.

The comparison between advanced and basic schedules under the stop point of view of iterations cannot be made, given the overwhelming victory of the first group.

4.1.7 Comparison between criteria varying maximum iterations

At the end, tests have been carried out to assess how much a stop criterion can be valid when the maximum iterations vary. The same parameters as the other tests were maintained and only the maximum number of iterations was changed.

Comparison between stopping criteria with 20 maximum iterations						
Criterion		*G-Matrix*	*FBER*	*CRC*	*G-Matrix+CRC*	*FBER+CRC*
Schedule	**SNR**	*Average iterations / Iterations' drop*				
Linear RL	2.5 dB	20/-%	20/-%	20/-%	20/-%	20/-%
	3.0 dB	19.9/0.5%	20/-%	19.8/1%	19.7/1.5%	19.9/0.5%
	3.5 dB	19.6/2%	20/-%	19.1/4.5%	18.8/6%	19.1/4.5%
Stepped	2.5 dB	20/-%	20/-%	20/-%	20/-%	20/-%
	3.0 dB	19.9/0.5%	20/-%	19.9/0.5%	19.9/0.5%	19.7/1.5%
	3.5 dB	19.3/3.5%	20/-%	19.5/2.5%	19.2/4%	19.3/3.5%
Circular LR	2.5 dB	7.65/61.8%	13.4/33%	10.4/48%	7.55/62.3%	9.86/50.7%
	3.0 dB	6.27/68.7%	11.8/41%	8.76/56.2%	6.24/68.8%	8.14/59.3%
	3.5 dB	5.48/72.6%	10.7/46.5%	7.56/62.2%	5.47/72.7%	7.00/65%
Biwave	2.5 dB	11.3/43.5%	16.1/19.5%	13.2/34%	11.2/44%	13.0/35%
	3.0 dB	9.04/54.8%	14.1/29.5%	11.0/45%	8.94/55.3%	10.6/47%
	3.5 dB	7.67/61.7%	12.6/37%	9.44/52.8%	7.61/62%	9.01/55%

TABLE 4.7: Stopping performance comparison between criteria with 20 maximum iterations and $N = 1024$, $K = 512$

Starting from Table 4.7, with 20 maximum iterations the FBER does not provide any stop of the iterations for the basic schedules and with those advanced it is difficult to keep pace. G-Matrix fails to offer an important iteration drop for basic schedulers, max 2%, and the application of the CRC is required to raise the iteration stop. Indeed for *Linear RL* and *Stepped* schedule the best solutions are the hybrid ones. By the way, observing altogether the data, the FBER + CRC hybrid solution and pure G-Matrix remain preferable considering performance and costs.

Instead increasing the iterations to a maximum of 60 in Table 4.8, the situation is more relevant and balanced. The FBER remains the worst criterion but it also

Comparison between stopping criteria with 60 maximum iterations						
Criterion		*G-Matrix*	*FBER*	*CRC*	*G-Matrix+CRC*	*FBER+CRC*
Schedule	**SNR**	*Average iterations / Iterations' drop*				
Linear RL	2.5 dB	32.7/45.5%	42.0/30%	36.7/38.8%	29.6/50.7%	35.4/41%
	3.0 dB	26.2/56.3%	35.8/40.3%	30.2/49.7%	23.6/60.7%	28.6/52.3%
	3.5 dB	22.1/63.2%	31.3/47.8%	25.6/57.3%	19.8/67%	24.0/60%
Stepped	2.5 dB	30.4/49.3%	42.4/29.3%	36.8/38.7%	29.8/50.3%	33.4/44.3%
	3.0 dB	24.3/59.5%	36.1/39.8%	30.6/49%	24.0/60%	26.8/55.3%
	3.5 dB	20.6/65.7%	31.8/47%	26.2/56.3%	20.5/65.8%	21.7/63.8%
Circular LR	2.5 dB	8.47/85.9%	24.5/59.2%	19.7/67.7%	8.13/86.5%	16.1/73.2%
	3.0 dB	6.49/89.2%	20.5/65.8%	15.9/73.5%	6.36/89.4%	12.7/78.8%
	3.5 dB	5.54/90.8%	18.3/69.5%	13.1/78.2%	5.49/90.85%	10.5/82.5%
Biwave	2.5 dB	12.3/79.5%	27.6/54%	22.6/62.3%	11.8/80.3%	19.4/67.7%
	3.0 dB	9.3/84.5%	22.9/61.8%	18.2/69.7%	9.11/84.8%	15.2/74.7%
	3.5 dB	6.73/88.8%	19.9/66.8%	15.0/75%	7.66/87.2%	12.4/79.3%

TABLE 4.8: Stopping performance comparison between criteria with 60 maximum iterations and $N = 1024$, $K = 512$

offers a maximum iteration's drop equal to 69.5% with *Circular LR* schedule. In the middle there is CRC criterion which achieves a maximum drop of 78.2%. The best solutions are G-Matrix and G-Matrix + CRC but the differences between them are really minimal: a minimum of 0.05% with *Circular LR* at $SNR = 3.5dB$ up to 5.2% with *Linear RL* at $2.5dB$. Also in this case, the FBER + CRC hybrid criterion is the most balanced solution because it offers optimal performance with the typical complexity savings of the FBER.

4.2 N=2048, K=1024: numerical results

In this set of tests, no comparisons with literature results have been made since no tests with the same parameters were found. Therefore, the comparisons have been carried out among the criteria set out so far to assess their reliability.

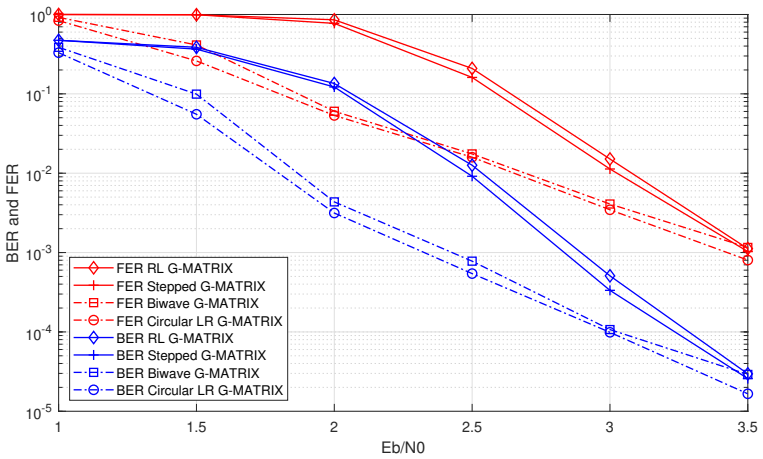


FIGURE 4.27: G-Matrix BER-FER with all schedules $N = 2048$, $K = 1024$

Also in this case, a general comparison between the criteria is done to compare stop performance across the SNR scale.

4.2.1 G-Matrix criterion

Although N and K are doubled, the results of the G-Matrix criterion are not essentially changed as shown in Figure 4.33. The FER parameters for *Linear RL* and *Stepped* (with continuous line) are very close among them and they present the same trends. Indeed, *Circular LR* and *Biwave* schedules deviate slightly at low and high $SNRs$ while retaining a reliable behavior. The BER parameter shows the same trends for

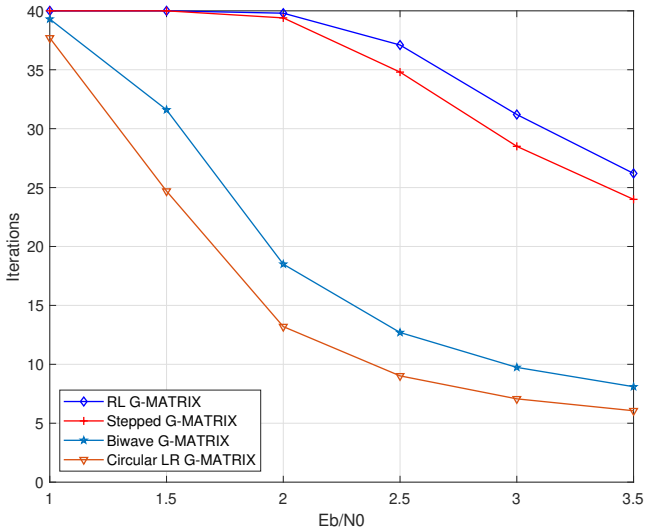


FIGURE 4.28: Iterations with G-Matrix criterion $N = 2048$, $K = 1024$

all schedules obtained for $N = 1024, K = 512$ ensuring reliability with results close to 10^{-5}

G-Matrix criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	37.1	7.3 %
	3.0 dB	31.2	22 %
	3.5 dB	26.2	34.5 %
Stepped	2.5 dB	34.8	13 %
	3.0 dB	28.5	28.8 %
	3.5 dB	24.0	40 %
Circular LR	2.5 dB	9.02	77.5 %
	3.0 dB	7.07	82.3 %
	3.5 dB	6.06	84.9 %
Biwave	2.5 dB	12.7	68.3 %
	3.0 dB	9.74	75.7 %
	3.5 dB	8.10	79.8 %

TABLE 4.9: Average iterations with G-Matrix criterion $N = 2048$, $K = 1024$

In iterations' stop, presented in Figure 4.28 and Table 4.9, the worst schedule is *Linear RL* that reaches only a maximum iterations' drop of 34.5% at $3.5dB$. The *Stepped* schedule is not that far overcoming the first one of 5.5%. With the complex schedules, performances start to get interesting because the *Biwave* reaches a maximum drop of 79.8% and, indeed, *Circular LR* schedule, coming to 6.06 average iterations, is the best with a drop equal to 84.9%.

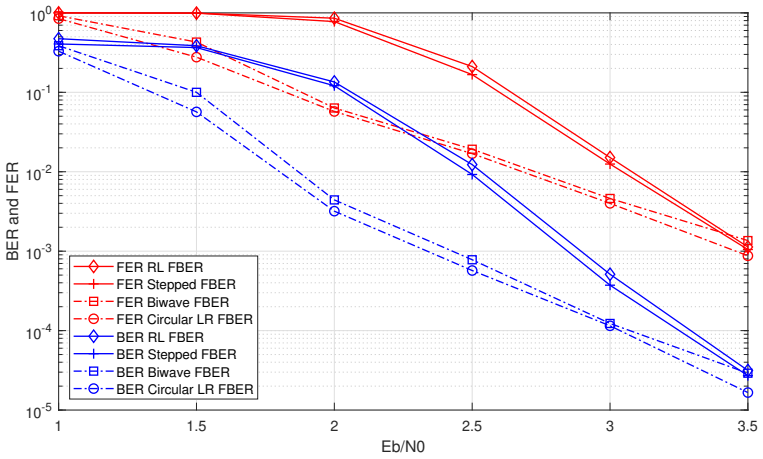


FIGURE 4.29: FBER BER-FER with all schedules $N = 2048$, $K = 1024$

4.2.2 FBER criterion

The FBER criterion achieves a good outcome's error as can be seen in Figure 4.29. For basic schedules, FERs' and BERs' curves are very near between them and they conclude at $3.5dB$ in the same zone of complex schedules. In the same way, *Circular LR* and *Biwave* have the same behaviors both for the FER and BER. Also in this case the results do not differ much from the previous ones confirming their goodness.

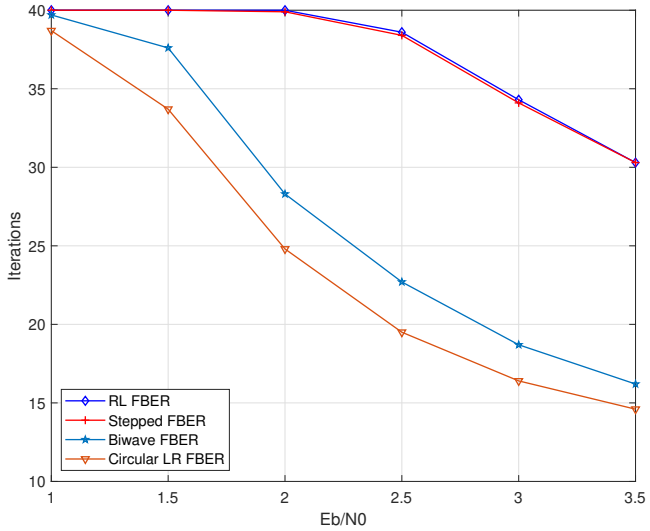


FIGURE 4.30: Iterations with FBER criterion $N = 2048$, $K = 1024$

In stopping iterations' performance, FBER with *Linear RL* and *Stepped* does not shine remaining for simple schedules always above the average 30 iterations, as

highlighted in Figure 4.30 and Table 4.10. They provide the exact same results unless small percentage differences. In reverse, advanced schedules provide good results reaching 63.5% for *Circular LR* and 59.5% for *Biwave*. It's important to underline that FBER is the simplest early stopping criterion tested in terms of complexity cost.

FBER criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	38.6	3.5 %
	3.0 dB	34.3	14.3 %
	3.5 dB	30.3	24.3 %
Stepped	2.5 dB	38.4	4 %
	3.0 dB	34.1	14.8 %
	3.5 dB	30.3	24.3 %
Circular LR	2.5 dB	19.5	51.3 %
	3.0 dB	16.4	59 %
	3.5 dB	14.6	63.5 %
Biwave	2.5 dB	22.7	43.3 %
	3.0 dB	18.7	53.3 %
	3.5 dB	16.2	59.5 %

TABLE 4.10: Average iterations with FBER criterion $N = 2048$, $K = 1024$

4.2.3 CRC criterion

Analyzing the results of the CRC criterion in Figure 4.31, we note that the FER curves are very similar both for the *Linear RL* and *Stepped*. They are quite close together but they do not coincide perfectly as in the case $N = 1024$, $K = 512$. Instead, for the advanced schedules the curves approach and manage to exceed the threshold of 10^{-3}. Regarding BER, there are behaviors beyond expectations because all the

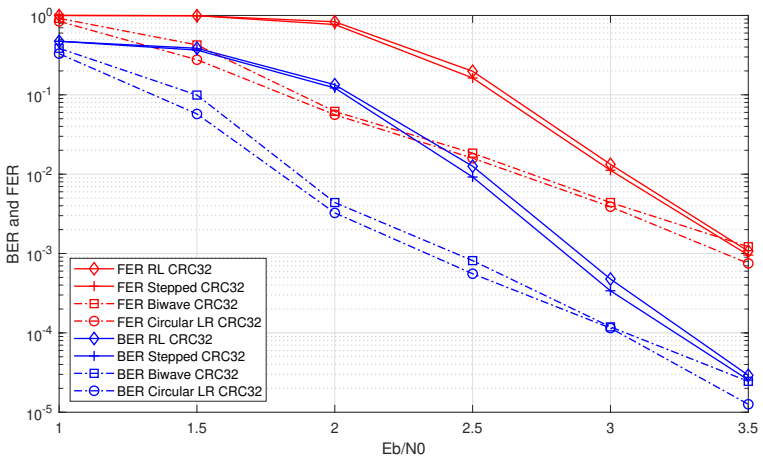


FIGURE 4.31: CRC32 BER-FER with all schedules $N = 2048$, $K = 1024$

trends abundantly exceed the results with the other N and K: the best is *Circular LR* extremely close to 10^{-5}.

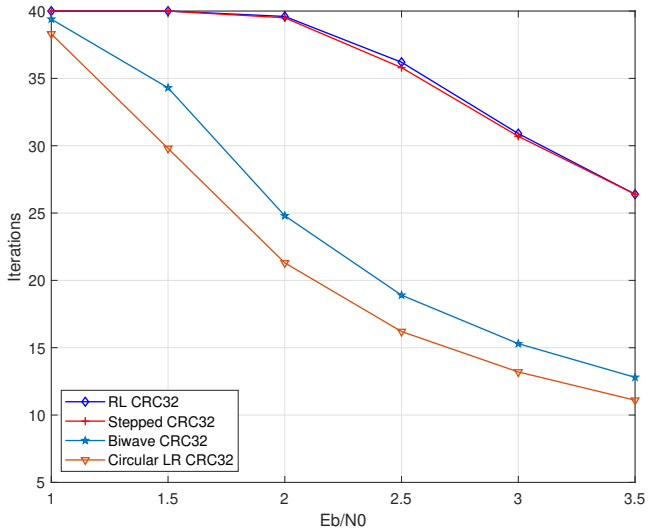


FIGURE 4.32: Iterations with CRC32 criterion $N = 2048$, $K = 1024$

Under the stop performance point of view, we have a similar behavior to that seen in the FBER where the basic schedules have identical trends (except for small percentage variations) and for advanced ones the curves are more smooth and less steep. However, the CRC manages to get good performances, as figured in 4.32 and summarized in Table 4.11, because *Linear RL* and *Stepped* obtain a minimum drop of iterations equal to 9.5% up to 34%; *Circular LR* and *Biwave* double basic schedules performances reaching respectively 72.3% and 68% at $3.5dB$.

CRC32 criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	36.2	9.5 %
	3.0 dB	30.9	22.8 %
	3.5 dB	26.4	34 %
Stepped	2.5 dB	35.8	10.5 %
	3.0 dB	30.7	23.3 %
	3.5 dB	26.4	34 %
Circular LR	2.5 dB	16.2	59.5 %
	3.0 dB	13.2	67 %
	3.5 dB	11.1	72.3 %
Biwave	2.5 dB	18.9	52.8 %
	3.0 dB	15.3	61.8 %
	3.5 dB	12.8	68 %

TABLE 4.11: Average iterations with CRC32 criterion $N = 2048$, $K = 1024$

4.2.4 Hybrid solution: G-Matrix merged with CRC

In the first hybrid solution, composed by G-Matrix criterion with the CRC32 in cascade and also called *hybrid 1*, excellent results are obtained both in terms of errors and in the stops of the iterations.

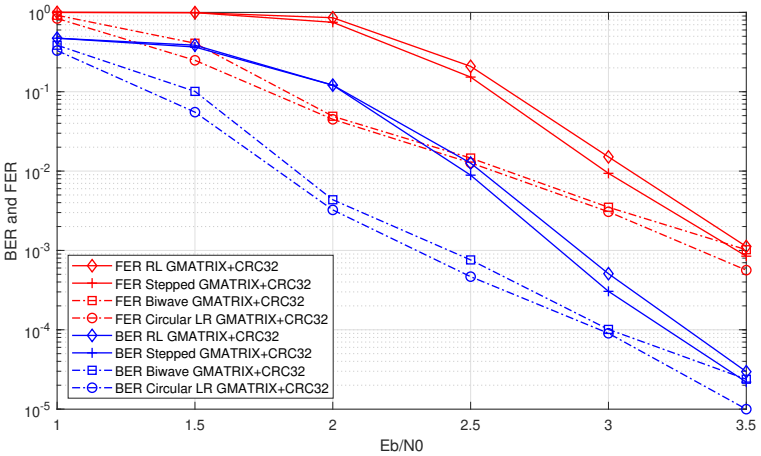


FIGURE 4.33: G-Matrix + CRC32 BER-FER with all schedules $N = 2048$, $K = 1024$

In fact, the FER parameter reaches very low values in all the schedules used and is quite homogeneous throughout the SNR scale. Also the BER assumes very contained values but in *Circular LR* it touches the $1 * 10^{-5}$, an unprecedented outcome. All trends are illustrated in Figure 4.33.

In the stop of the iterations, this method achieves excellent results (Figure 4.34 and Table 4.12) in every schedules too. The lowest saving threshold is obtained from *Linear RL* with a maximum iterations' drop of 34.5% and the *Stepped* schedule is distant only 2.3 iterations (5.8% less). For advanced schedules the best is *Circular LR* that takes the *Biwave* out of 5% (equal to 2 iterations).

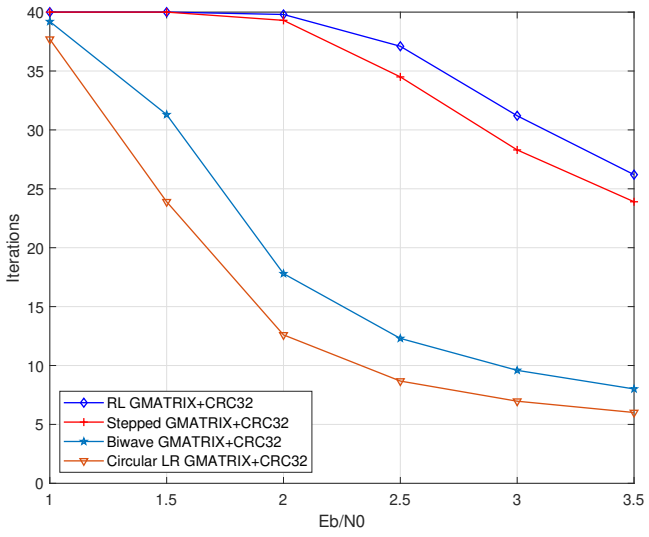


FIGURE 4.34: Iterations with G-Matrix + CRC32 criterion $N = 2048$, $K = 1024$

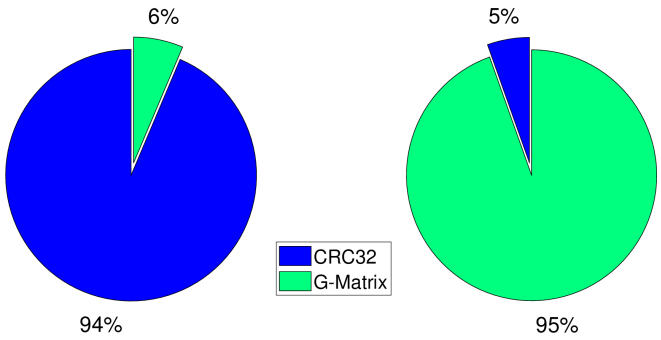


FIGURE 4.35: Percentage of stopping criteria's successes between G-Matrix and CRC32 with basic schedules $N = 2048$, $K = 1024$. Left *Linear RL*, right *Stepped*

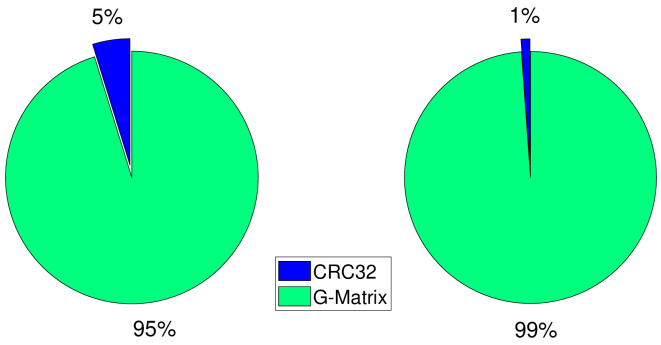


FIGURE 4.36: Percentage of stopping criteria's successes between G-Matrix and CRC32 with advanced schedules $N = 2048$, $K = 1024$. Left *Biwave*, right *Circular LR*

Hybrid G-Matrix + CRC32 criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	37.1	7.3 %
	3.0 dB	31.2	22 %
	3.5 dB	26.2	34.5 %
Stepped	2.5 dB	34.5	13.8 %
	3.0 dB	28.3	29.3 %
	3.5 dB	23.9	40.3 %
Circular LR	2.5 dB	8.68	78.3 %
	3.0 dB	6.97	82.6 %
	3.5 dB	6.01	85 %
Biwave	2.5 dB	12.3	69.3 %
	3.0 dB	9.59	76 %
	3.5 dB	8.01	80 %

TABLE 4.12: Average iterations with G-Matrix + CRC32 criterion $N = 2048$, $K = 1024$

The stop of the iterations in this method is shared, for the basic schedules, as shown in the Figure 4.35: for *Linear RL* almost all of the iterations (95%) are interrupted by the CRC32; contrariwise, for *Stepped* the situation is overturned with 95% for G-Matrix criterion and the rest for the CRC32. In the case of advanced schedules in Figure 4.36, the G-Matrix is *Circular LR* the only leader in the stop of the iterations with 95% for the *Biwave* and even 99% for the *Circular LR*.

4.2.5 Hybrid solution: FBER merged with CRC

The second hybrid method, composed by FBER and CRC32 referenced with *hybrid 2*, provides good results regarding errors and iterations stop.

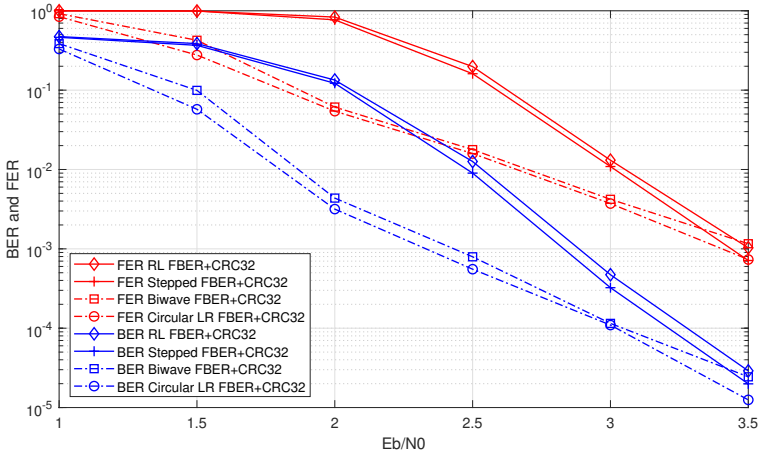


FIGURE 4.37: FBER + CRC32 BER-FER with all schedules $N = 2048$, $K = 1024$

The FER curves of basic schedules are very close among them and they reach the same performance as the advanced ones at $3.5db$. In Figure 4.37 can be noticed that *Circular LR* and *Biwave* have better behavior from $1dB$ up to $3dB$. The BER has even

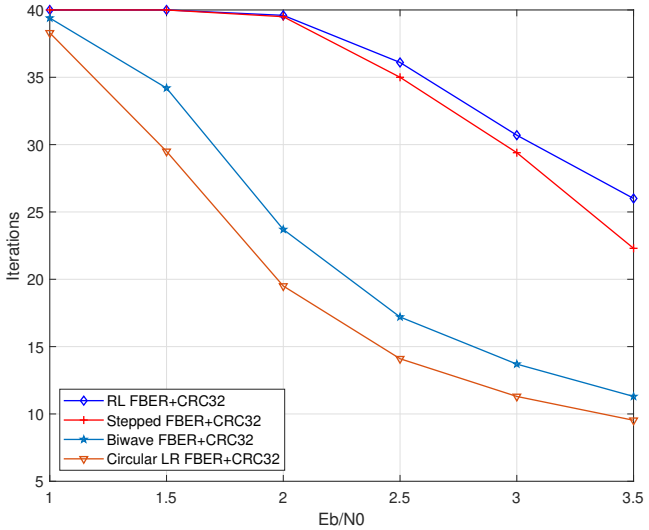


FIGURE 4.38: Iterations with FBER + CRC32 criterion $N = 2048$, $K = 1024$

better trends but in this case only the *Circular LR* behaves better by coming close to $1 * 10^{-5}$.

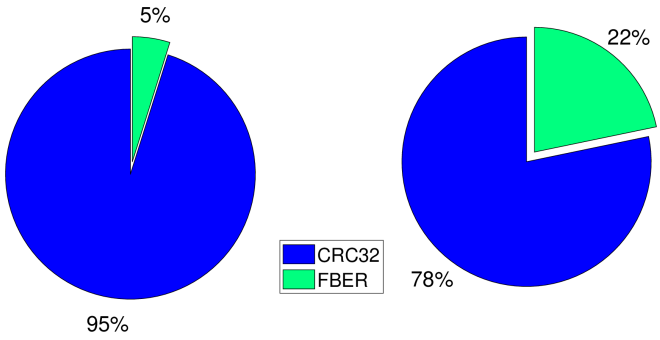


FIGURE 4.39: Percentage of stopping criteria's successes between FBER and CRC32 with basic schedules $N = 2048$, $K = 1024$. Left *Linear RL*, right *Stepped*

The Figure 4.38 and Table 4.13 describe how the hybrid criterion stops iterations with excellent results with both basic and advanced schedules. Indeed, *Linear RL*, even if it's the worst, attains a maximum stop equal to 26 iterations (35% less); also the *Stepped* provides a great result with 44.3% iteration's drop. The situation is even better with advanced schedules because the method stops 71.8% of *Biwave* iterations and 76.2% for the *Circular LR*.

As underlined by the Figure 4.39 and 4.40, in this criterion CRC method is predominant in the *Linear RL* (with 95% of successes) and, to the same extent, the hits for the *Biwave* and *Circular LR* are equal to 94%. The only schedule that differs from the others is *Stepped* which distributes the successes between FBER and CRC respectively in 22% and 78%.

Hybrid FBER + CRC32 criterion			
Schedule	**SNR**	**Avg iter**	**Iter drop**
Linear RL	2.5 dB	36.1	9.8 %
	3.0 dB	30.7	23.3 %
	3.5 dB	26.0	35 %
Stepped	2.5 dB	35.0	12.5 %
	3.0 dB	29.4	26.5 %
	3.5 dB	22.3	44.3 %
Circular LR	2.5 dB	14.1	64.8 %
	3.0 dB	11.3	71.8 %
	3.5 dB	9.53	76.2 %
Biwave	2.5 dB	17.2	57 %
	3.0 dB	13.7	65.8 %
	3.5 dB	11.3	71.8 %

TABLE 4.13: Average iterations with FBER + CRC32 criterion $N = 2048$, $K = 1024$

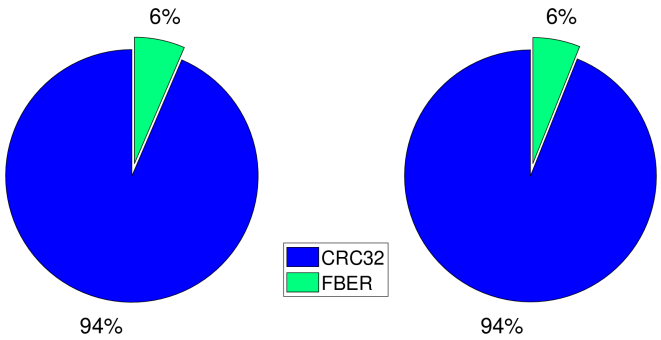


FIGURE 4.40: Percentage of stopping criteria's successes between FBER and CRC32 with advanced schedules $N = 2048$, $K = 1024$. Left *Biwave*, right *Circular LR*

4.2.6 Comparison between criteria

From Table 4.14 we can see that the least performing criterion, in absolute terms, is the FBER which offers minimum gains with all schedules: at $3.5dB$ the minimum decrease in iterations is equal to 24.3% with the *Linear RL* schedule up to the maximum of 63.5% (14.6 iterations) with the *Circular LR*. Although these are a limited savings, we can consider it a good result because it is necessary to take into account the minimum resources used.

Comparison between stopping criteria with 40 maximum iterations						
Criterion		*G-Matrix*	*FBER*	*CRC*	*G-Matrix+CRC*	*FBER+CRC*
Schedule	**SNR**	*Average iterations / Iterations' drop*				
Linear RL	2.5 dB	37.1/7.3%	38.6/3.5%	36.2/9.5%	37.1/7.3%	36.1/9.8%
	3.0 dB	31.2/22%	34.3/13.3%	30.9/22.8%	31.2/22%	30.7/23.3%
	3.5 dB	26.2/34.5%	30.3/24.3%	26.4/34%	26.2/34.5%	26.0/35%
Stepped	2.5 dB	34.8/13%	38.4/4%	35.8/10.5%	34.5/13.8%	35.0/12.5%
	3.0 dB	28.5/28.8%	34.1/14.8%	30.7/23.3%	24.0/29.3%	29.4/26.5%
	3.5 dB	24.0/40%	30.3/24.3%	26.4/34%	20.5/40.3%	22.3/44.3%
Circular LR	2.5 dB	9.02/77.5%	19.5/51.3%	16.2/59.5%	8.68/78.3%	14.1/64.8%
	3.0 dB	7.07/82.3%	16.4/59%	13.2/67%	6.97/82.6%	11.3/71.8%
	3.5 dB	6.06/84.9%	14.6/63.5%	11.1/72.3%	6.01/85%	9.53/76.2%
Biwave	2.5 dB	12.7/68.3%	22.7/43.3%	18.9/52.8%	12.3/69.3%	17.2/57%
	3.0 dB	9.74/75.7%	18.7/53.3%	15.3/61.8%	9.59/76%	13.7/65.8%
	3.5 dB	8.10/79.8%	16.2/59.5%	12.8/68%	8.01/80%	11.3/71.8%

TABLE 4.14: Stopping performance comparison between criteria with 40 maximum iterations and $N = 2048$, $K = 1024$

For the basic schedules, the results are extremely uniform: with the exception of the FBER, all the other criteria vary among them in the stop of the iterations at the most of 10% and at the minimum of 0.3%. The highest level of homogeneity is for the *Stepped* schedule as can be seen in Figure 4.41; in this case the best performances are reached, surprisingly, with the FBER + CRC hybrid method which exceeds the G-Matrix + CRC by 4% (1.8 iterations) at $3.5dB$. The *Linear RL* schedule also behaves similarly but with lower gains.

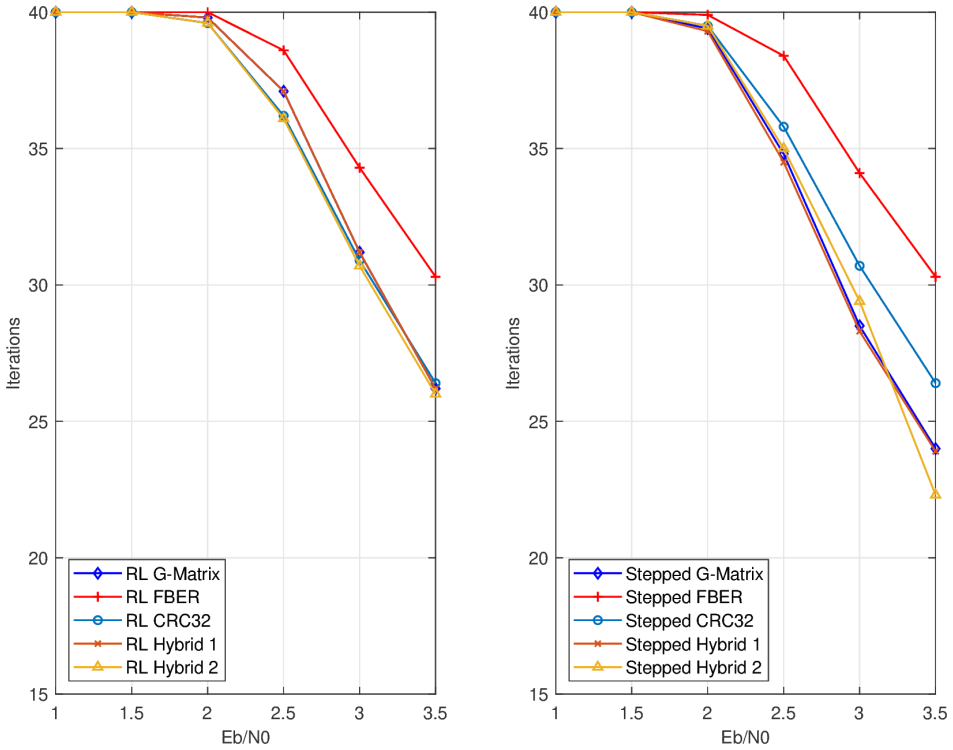


FIGURE 4.41: Comparison between stopping criteria sorted by basic schedules $N = 2048$, $K = 1024$

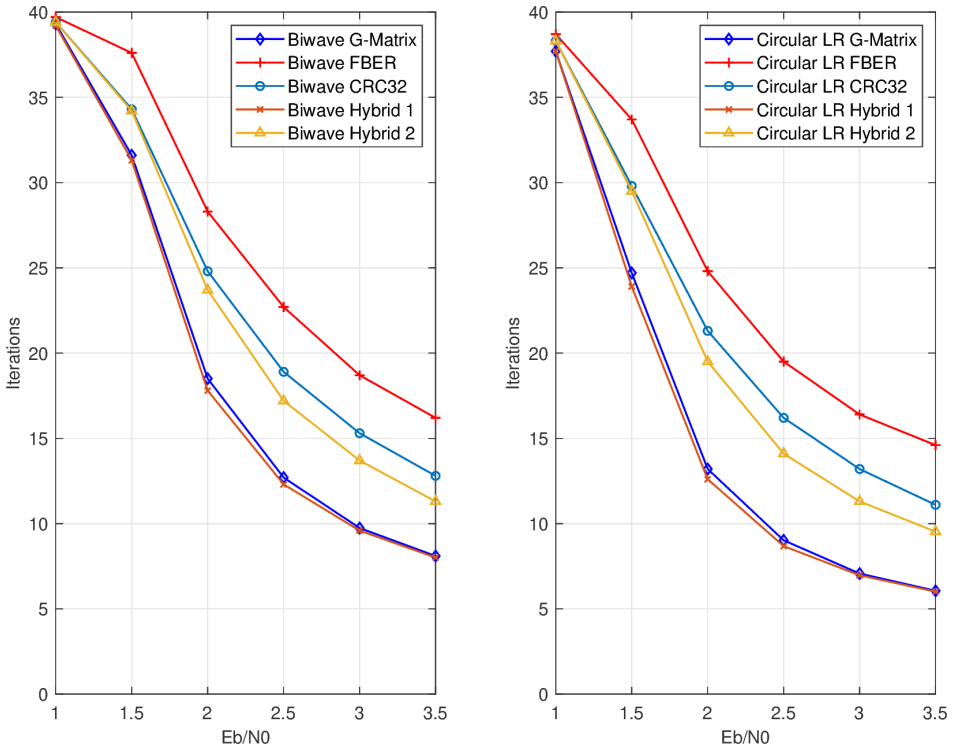


FIGURE 4.42: Comparison between stopping criteria sorted by advanced schedules $N = 2048$, $K = 1024$

Reflecting on the advanced schedules, the performances are more distributed (Figure 4.42) due to the fact that the minimum stop rate occurs with the CRC method (59.5% at 2.5dB) up to the absolute maximum of 85% reached by the G-Matrix + CRC. Despite the performance obtained from this hybrid criterion, it is necessary to consider the computational resources applied: the G-Matrix + CRC is much more expensive than the FBER + CRC which offers inferior but acceptable results. The latter obtains results ranging from 57% in the *Biwave* to 2.5dB of savings up to below the wall of the 10 iterations in the *Circular LR* at 3.5dB (76.2%).

Considering all the analyzes carried out so far, it can be concluded that the best compromise between performance and complexity is certainly given by the hybrid criterion FBER + CRC.

Chapter 5

Hardware implementations

In the last chapter, we will analyze the mathematical formulas of FBER in order to derive a simplified architecture to obtain area savings.

Developed a reduced architecture, it will be realized in hardware in VHDL language and the sub-blocks that compose it will be explored. Furthermore we will present a theoretical integration with the BP decoder.

Finally, the synthesis results of the FBER implementation with the related additions will be listed.

5.1 FBER implemetation

Among all the stopping criteria analyzed so far, the FBER method was chosen for hardware implementation as described in 2.5. It has been selected due to the fact that it offers decent stopping iterations' performance considering the minimum computational cost. Furthermore it has been taken into consideration because it allows a considerable reduction of the classical components that generate latency (for example the adders) and in this work an alternative architecture has been sought, with respect to the simple application of the formulas, to optimize the area expenditure computational cost.

The architectural reductions and the proposed architecture has been directly derived from the mathematical formulas of the criterion and the procedure in detail will be illustrated below.

5.1.1 Derivation of FBER architecture from mathematical formulas

Moving on to the analysis of the mathematical formulas that will lead to the construction of architecture, it is necessary to remember the equations constituting the FBER criterion:

$$F_{BER}^{t} = \frac{1}{|\mathcal{A}^{c}|} \sum_{i \in \mathcal{A}^{c}} [(1 + sign(L_{i,1}^{t}))/2] \oplus u_i \tag{5.1}$$

$$F_{M}^{t} = \sum_{v=t-M+2}^{t} \left| F_{BER}^{v} - F_{BER}^{v-1} \right| \tag{5.2}$$

In the first, the argument of the summation, as can be seen, involves the vector of the encoded information u_i (or as in other papers indicated with x_i) and the vector $sign(L_{i,1}^{t})$. The calculation present between the square brackets of the 5.1 can be reduced to two extreme cases:

1. $sign(L_{i,1}^{t}) = 0$;

2. $sign(L_{i,1}^t) = 1$.

In the first case the operation will be equal to

$$[(1 + sign(L_{i,1}^t))/2]\big|_{sign(L_{i,1}^t)=0} = [(1+0)/2] = 0.5 \tag{5.3}$$

and hence it agrees with the sign of $L_{i,1}^t = 0$ (considering the truncation $0.5 \approx 0$). In the second case instead it will be obtained:

$$[(1 + sign(L_{i,1}^t))/2]\big|_{sign(L_{i,1}^t)=1} = [(1+1)/2] = 1 \tag{5.4}$$

Also in this case the result obtained will agree with the sign of $L_{i,1}^t = 1$

As shown, rather than adding the sum of the term in square brackets, it can be reduced to the calculation of the *sign* only.

$$[(1 + sign(L_{i,1}^t))/2] = sign(L_{i,1}^t) \tag{5.5}$$

Using this reduction it is possible to perform the XOR operation directly with the vector u_i

$$F_{BER}^t = \frac{1}{|\mathcal{A}^c|} \sum_{i \in \mathcal{A}^c} [(1 + sign(L_{i,1}^t))/2] \oplus u_i = \frac{1}{|\mathcal{A}^c|} \sum_{i \in \mathcal{A}^c} sign(L_{i,1}^t) \oplus u_i \tag{5.6}$$

As for the equation 5.2, it is necessary to keep in mind that the subtraction in the argument of the summation is always referred to two consecutive iterations and that the result of the summation itself F_M^t must remain constantly equal to zero for M iterations. In this way, the criterion responds positively.

Observing the subtraction from another angle and bearing in mind that F_M^t must remain constant, it is possible to reduce the subtraction to a continuous XNOR operation on two consecutive iterations to search only when the result of F_M^t is the same on two successive iterations.

$$F_M^t = \sum_{v=t-M+2}^{t} \left| F_{BER}^v - F_{BER}^{v-1} \right| = \sum_{v=t-M+2}^{t} F_{BER}^v \otimes F_{BER}^{v-1} \tag{5.7}$$

The last step is to find a valid alternative to the summation. This operation can be carried out with a counter that resets whenever the input does not return a logical value 1. if F_M^t reaches the threshold M, then the FBER criterion will return an affirmative answer and it will be possible to stop the iterations.

5.2 Proposed architecture

As shown, by assembling all the simplifications performed, the following architectural proposal is obtained:

As can be seen from Figure 5.1, the inputs are exactly the $sign(L_{i,1}^t)$ and $u_i \; \forall i \in [N-1, 0]$ where N is the number of input bits. The next XNOR operation is performed once by iteration comparing the inputs bit by bit and the result is saved in the *splitter* component described in detail in Figure 5.2. The XNOR operation

The *splitter* is composed by a de-multiplexer, two 1-bit registers connected in output with an 2-bits AND. This allows us to check the variable $F_{i,M}^t$ between two successive iterations. All the comparisons between $F_{i,M}^t$ are conveyed in a N-bit AND that allows us to have the definitive feedback on F_M^t for the iteration t.

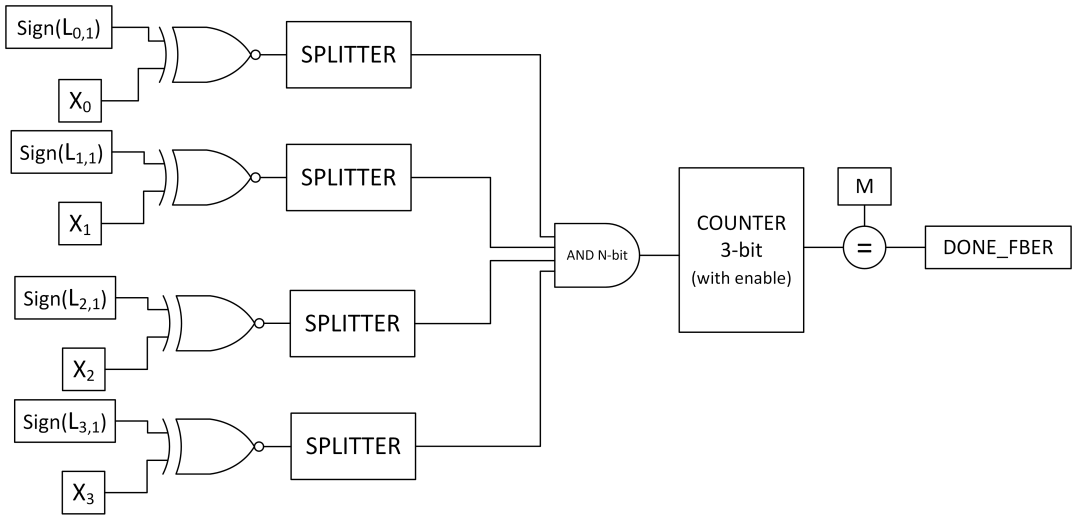


FIGURE 5.1: Proposed architecture with $N = 4$

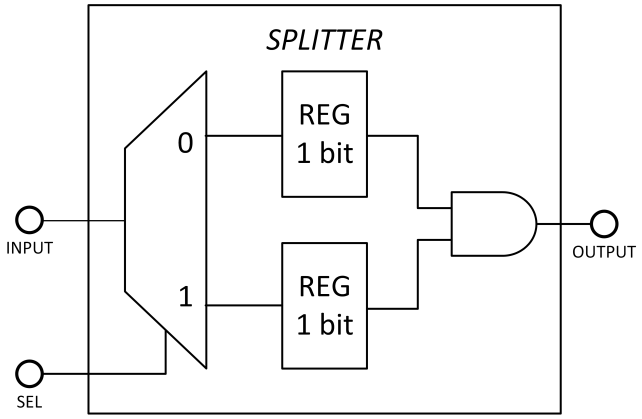


FIGURE 5.2: Schematic of *splitter* component

The response of the N-bit AND is connected to a counter to enumerate the stability of F_M^t and it is reset every time its input, connected to the enable port, does not return a logical value 1. The final equality operation represents the threshold M to be reached to retain the convergent criterion.

5.3 Realization of the proposed architecture

Beginning to illustrate the realization in VHDL of the proposed architecture, we will start from the top-level description of the FBER stopping iterations' system. Two different solutions will be presented: the first the system is totally independent from the BP decoder and in the second one the structure is flanked by a decoder. The FBER entity is therefore equipped with its own FSM and it is used both for carrying out the tests and for the integration with BP decoder. In both cases the top-level representation of the FBER will be the same and it is shown in Figure 5.3.

The architecture relates to a code of length $N = 1024$ but, in general, it is possible to scale the system for any N by modifying exclusively the input size registers and

the number of the components of the data path. However, the signals described in the figure below are associated with:

- CLK (*input*): clock signal;
- START (*input*): start operation signal;
- RESET (*input*): reset signal of components with memory. Active low;
- L,X (*input*): input data vectors N-bits;
- LOAD_NEW (*input*): a new data is ready to be loaded from the external source;
- DONE (*input*): convergence response from the decoder;
- DONE_FBER (*output*): convergence response from the FBER;
- REST (*output*): system on hold, out of all operations.

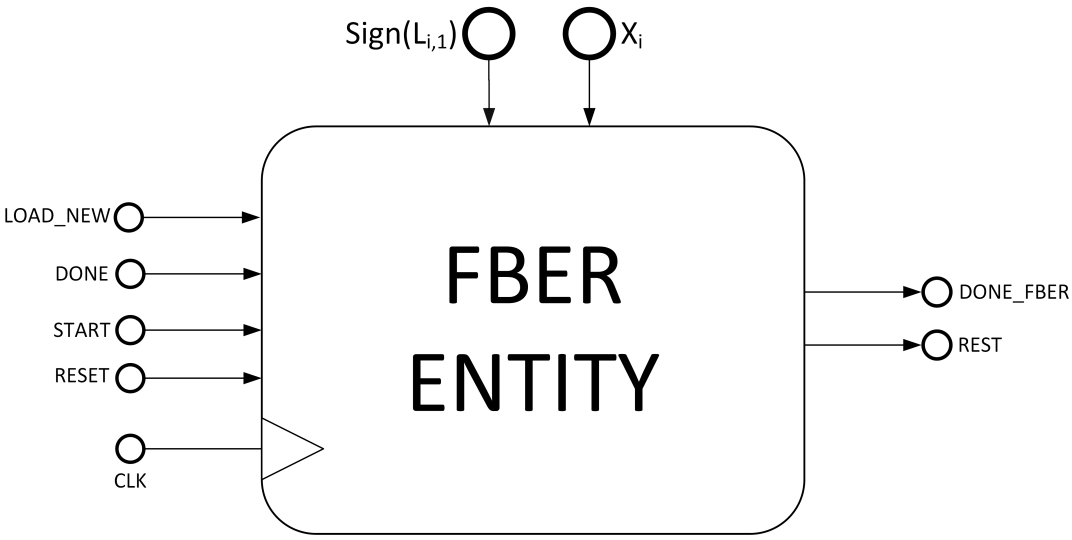


FIGURE 5.3: Top-level of FBER component

5.3.1 FBER hardware implementation

Focusing attention on the implementation of the FBER, we note that, going down a description level, it is possible to distinguish an *execution unit* (EU) and a *control unit* (CU) as explained in Figure 5.4

The *execution unit* generates the output signal DONE_FBER which decrees the convergence of the criterion. On the contrary, the *control unit*, after starting operations with the activation of the START signal, establishes when the system is ready to receive further input data by enabling the REST signal and waiting for a LOAD_NEW signal rise.

The coordination between the two units is carried out by the following signals:

- ENABLE_W (from CU to EU): enabling the writing of input registers;
- ENABLE_S (from CU to EU): enabling the writing of the registers that make up the *splitter* component;

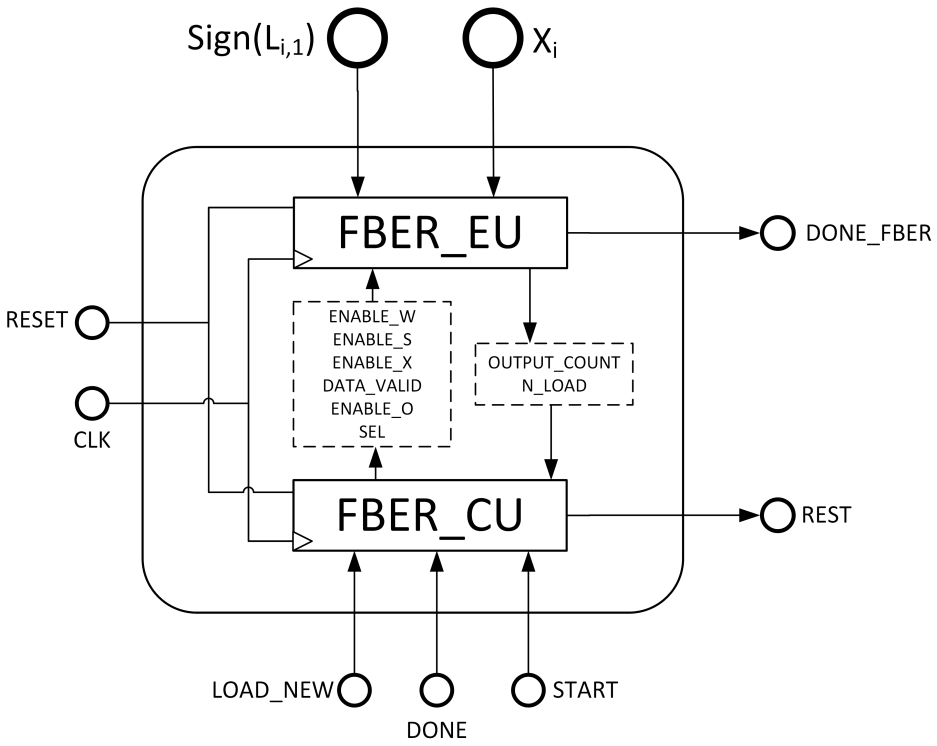


FIGURE 5.4: First level of FBER component

- SEL (from CU to EU): selector for the *splitter*;
- ENABLE_X (from CU to EU): enabling the writing of the registers after the *splitter*;
- DATA_VALID (from CU to EU): enabling data to be saved only after at least two iterations;
- ENABLE_O (from CU to EU): activation for the outgoing counter;
- OUTPUT_COUNT (from EU to CU): counting in output from the EU;
- N_LOAD (from EU to CU): upload number of the minimum incoming data. It is connected with a counter 2-bit with a maximum value equal to 2.

The operations of both the EU and the CU will be explained below. The execution unit is composed of several sections (as shown in the Figure 5.5) and all the operation is regulated on the *rising edge* of the clock.

Execution unit Once the input data are made available, they are stored in the N dimensional registers ($sign(L_{i,1})$ and X_i respectively) via the ENABLE_W=1. Subsequently, bit by bit and in groups of two bits the time, the binary operation XNOR is executed between $sign(L_i, 1)$ and X_i and the result is saved by means of the signal ENABLE_S=1 in the 1-bit *splitter* registers. Since it is necessary to save two successive elaborations, the de-multiplexer, thanks to the SEL signal, will alternate the saving between the two 1-bit registers of the *splitter*. After this, the 2-bits AND operation between these registers of the *splitter* will be carried out and the data will be

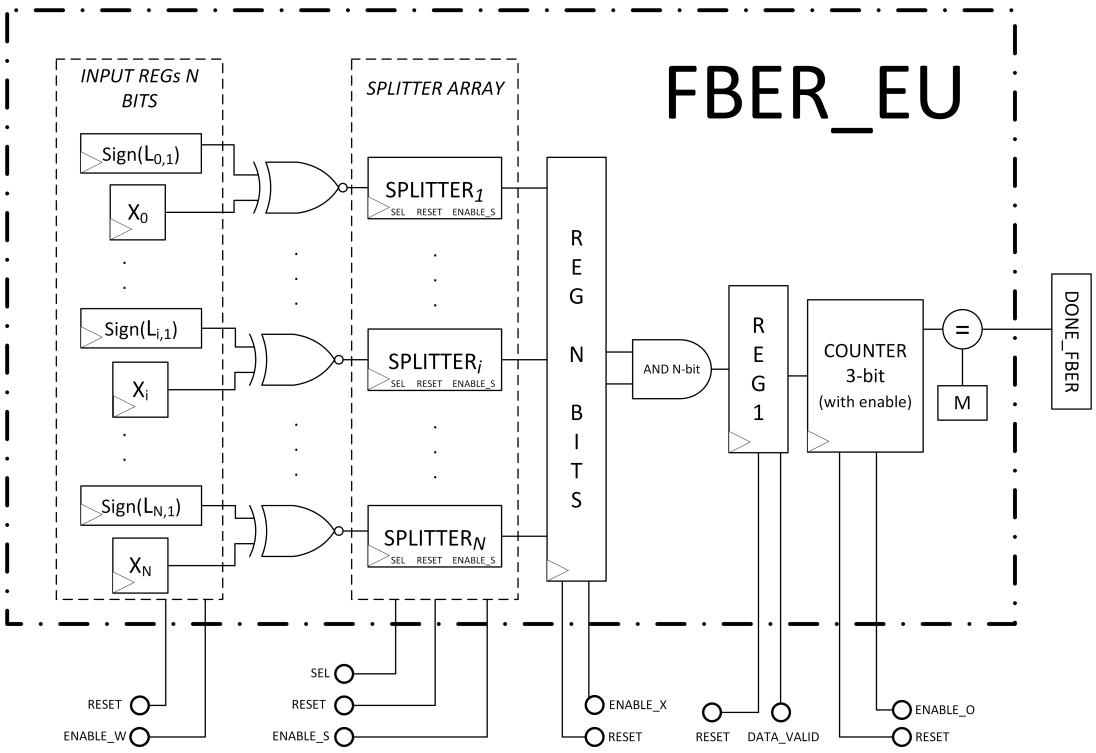


FIGURE 5.5: Execution unit of FBER implementation

kept in an N-bit register maintaining the relative position of the bits by means of the signal ENABLE_X=1.

The bits will be compared in the N-bit AND logic gate and the result will be saved in a further 1-bit register only after a rise of the DATA_VALID signal which decides that at least two input data have been processed and stored. Finally, the REG1_OUT is connected to the ENABLE port of the 3-bit counter and it will be possible to increase it only when the ENABLE_O signal is equal to 1. If the data stored in REG1_OUT is 1, the counter will increase its value; in the opposite case, then it would reset itself so as to achieve the stability required by the FBER criterion. In conclusion, if the output of the counter is equal to 4 (or more generally equal to M), then the signal DONE_FBER will be issued for the stop of the iteration.

Control unit After the presentation of the EU, the CU is designed. Considering all the signals already described, in the Figure 5.6 only their status changes during each passage in the FSM will be highlighted.

The states of the *control unit* are:

- IDLE1: it waits for the signal START=1 to arrive; REST=1, SEL=0 and RESET=0 is active.

  If START=1 then $next\_state \Rightarrow INIT1$, otherwise $next\_state \Rightarrow IDLE1$;

- INIT1: ENABLE_W = 1 is enabled for writing the first input data on the input registers. RESET=1 (not active) and REST=0.

  $next\_state \Rightarrow INIT2$;

- INIT2: ENABLE_S=1 is enabled for writing the first processing in the *splitter* component and ENABLE_W = 0 is disabled. The counter is incremented through N_LOAD++ and the writing is alternated with SEL=$\overline{SEL}$.

  If N_LOAD=2 then $next\_state \Rightarrow ELAB1$, otherwise $next\_state \Rightarrow IDLE2$;

- IDLE2: we await the arrival of a second pair of input data to carry out the first processing since at least two iterations are required. ENABLE_S = 0 and REST=1.

  If LOAD_NEW=1 then $next\_state \Rightarrow INIT1$, otherwise $next\_state \Rightarrow IDLE2$;

- ELAB1: it starts processing by activating ENABLE_X = 1 storing data in N-bits register and deactivating ENABLE_S=0.

  $next\_state \Rightarrow ELAB2$;

- ELAB2: it is possible to compare the data, therefore DATA_VALID=1, ENABLE_X=0.

  $next\_state \Rightarrow ELAB3$;

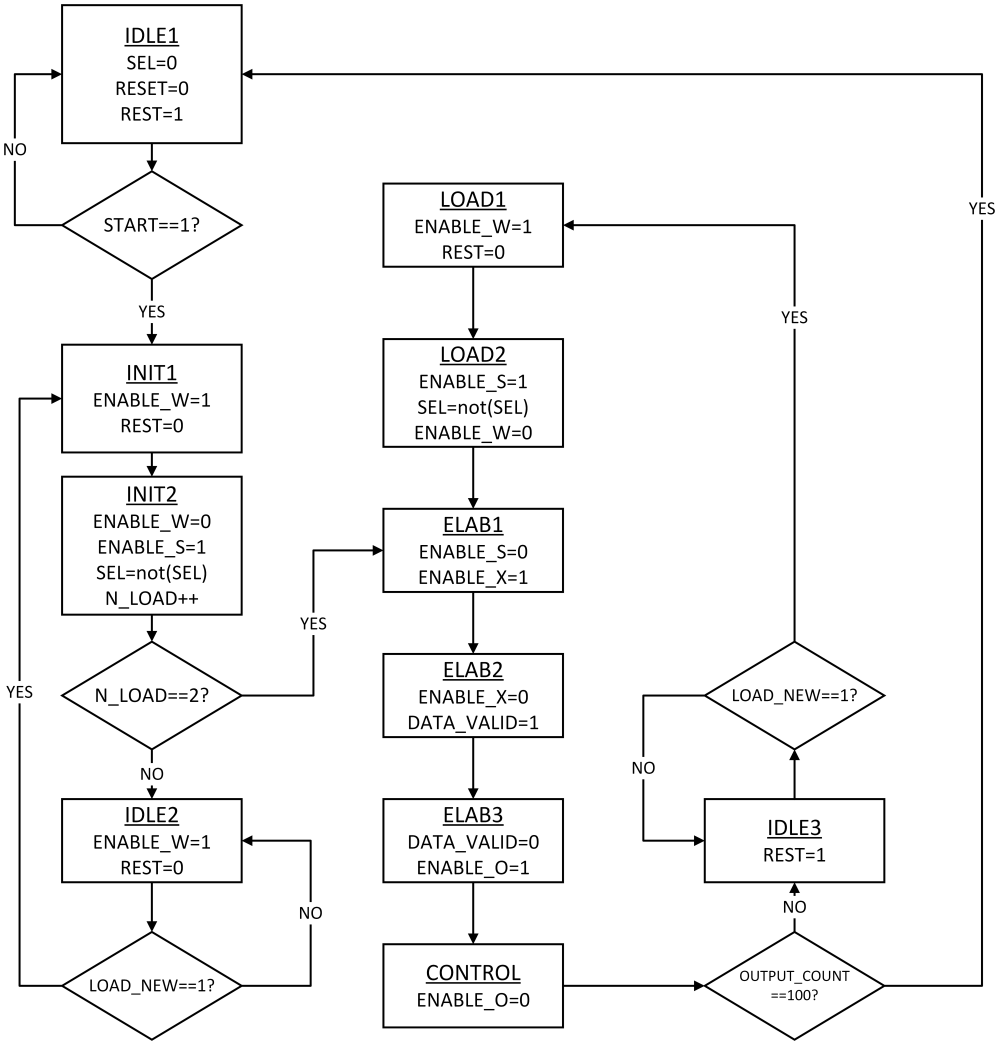


FIGURE 5.6: FSM of FBER control unit implementation

- ELAB3: DATA_VALID is disabled and the writing in the counter is enabled with ENABLE_O = 1.;

  $next\_state \Rightarrow CONTROL$;

- CONTROL: the signal ENABLE_O is set to 0 and the output of the counter is compared to evaluate the state of the system.

  If OUTPUT_COUNT=100 then $next\_state \Rightarrow IDLE1$, otherwise $next\_state \Rightarrow IDLE3$;

- IDLE3: after processing it expects the availability of another pair of incoming data. REST = 1.

  If N_LOAD=2 then $next\_state \Rightarrow LOAD1$, otherwise $next\_state \Rightarrow IDLE3$;

- LOAD1: ENABLE_W = 1 is activated so as to input new data and the rest mode is disabled (REST=0).

  $next\_state \Rightarrow LOAD2$;

- LOAD2: the data is passed to the *splitter* block with ENABLE_S=1 enabled and the alternance of SEL=$\overline{SEL}$. ENABLE_W=0.

  $next\_state \Rightarrow ELAB1$.

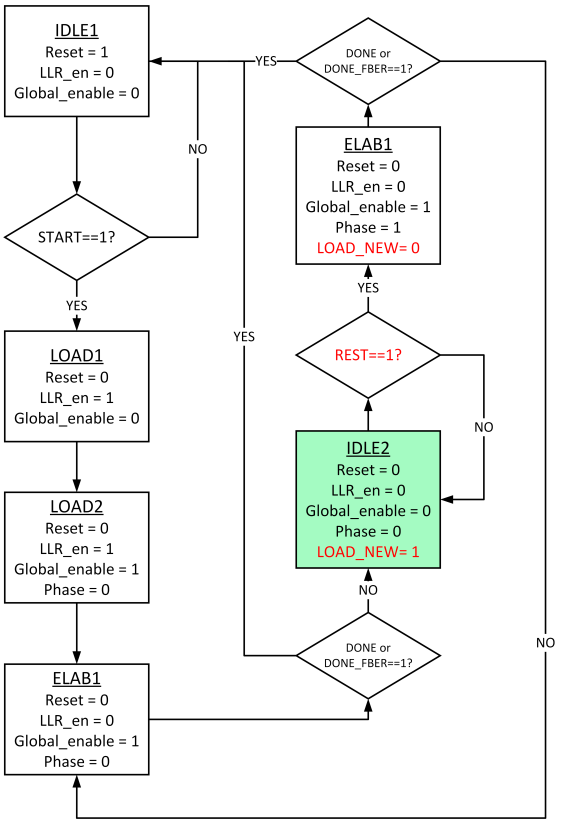


FIGURE 5.7: FSM fully parallel for BP decoder with FBER signal implementation. In green the added state; in red the added signals.

5.3.2 Integration with belief propagation decoder

Once the whole FBER realization in hardware has been explained, we move on to a theoretical integration with the belief propagation decoder. From the analysis of the previous works, it was possible to recover two solutions concerning the control units. In both cases, only a few states have been added to the CUs in order to obtain a unique interfaced system that offers both the performance of the decoder combined and the possibility of interrupting the iterations.

The first FSM, called *fully parallel* (Figure 5.7) which implements the schedule *Stepped* in hardware, has been enriched with a status (highlighted in green) so that it is possible to check the REST signal coming from the FBER and it send back the availability of a new data. The position of the additional state has been chosen because in the FBER it is necessary to load exclusively the data processed by the decoder $sign(L_{i,1})$, therefore relative to the first elaboration. However, at each end of an iteration the FBER is fed by a pair of new data.

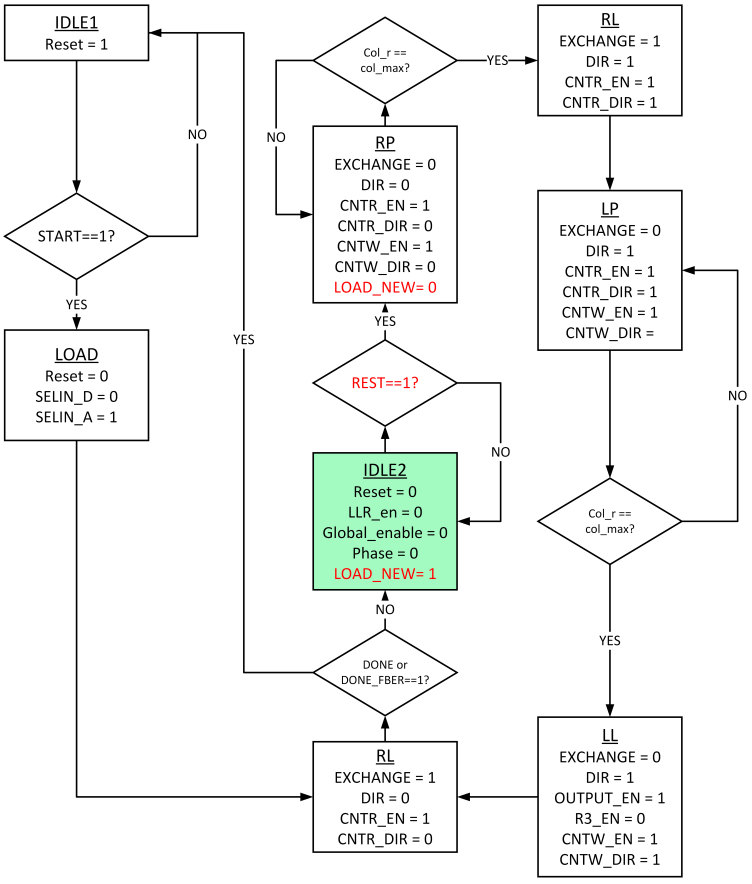


FIGURE 5.8: FSM single column for BP decoder with FBER signal implementation. In green the added state; in red the added signals.

Instead in the second one, the control unit, called *single column* figured in 5.8, is related to the schedule *Circular LR* and, also in this case, the addition of a single state was sufficient. The interchange of the signals is the same as in the previous case allowing the integration much easier for each BP decoder of the polar codes. The latter solution is the best with regards to performance and convergence speed.

In this way a complete system is reached having the classic features of low latency of the BP decoder combined with the FBER stopping performance.

5.3.3 Synthesis results

Following the functional description of the FBER architecture in VHDL, the whole structure was positively validated through the use of Mentor Modelsim. Subsequently, the synthesis was performed with Synopsys Design Compiler to detect area occupancy and maximum clock frequency. For the synthesis it was set $N = 1024$ and the standard cell at 45 nm.

Three tests were carried out: the first involved a low-frequency test (500 MHz = 10 ns) for the validation of the target timing and wide margins of improvement was noted. In fact, in the second test, the clock period was reduced to 0.90 ns (equivalent to 1.11 GHz) and it was likewise reached the target timing with minimal chances for improvements. In the last one, the maximum reachable frequency estimated in 1.16 GHz (0.86 ns) was obtained.

At the same time, tests were also carried out for the occupation of the area and, in all three cases, it remained unchanged since the data path is dependent exclusively on parameter N.

Synthesis area results (45nm)			
Area type	*FBER*	*FBER + fully parallel*	*FBER + Single column*
Combinational	0.021 mm^2	6.301 mm^2	1.813 mm^2
Sequential	0.044 mm^2	3.764 mm^2	1.504 mm^2
Total	0.065 mm^2	10.065 mm^2	3.335 mm^2

TABLE 5.1: Synthesis area results for FBER implementation and for complete integration

For the total estimate of the system, in Table 5.1, the data from previous works were used for the area calculation of the BP decoder with the two implementations analyzed above. The results are combined with the FBER synthesis results presented herein; they show a overpowering use of area from the BP decoder.

Chapter 6

Conclusion

All the expectations of search for the efficiency of the belief propagation decoding of polar codes can be said to be widely satisfied since the stopping algorithms already present in the literature have been tested and verified and a new and unpublished algorithm has been implemented in software. Furthermore, in this work, an architecture optimized for area savings by using an already low-cost algorithm has been proposed, achieving the desired results.

The path to demonstrate what has been said is started by mathematical formulas to introduce the potential of polar codes and the fundamental characteristics of BP decoding.

We continued by examining the literature concerning the algorithms of the early stopping criteria of BP and, after a careful analysis, the Frozen Bit Error Rate method was chosen for the software tests because it was found to be the cheapest under the point of view of computational costs.

The FBER criterion was implemented in C looking for results that validate the literature. The tests proved that the FBER, although it was the worst of the criteria analyzed, it was found to be the most convenient for the performance/cost ratio due to its really minimal costs. Furthermore, the unpublished algorithm based on the Cyclic Reduntant Check has been created and it offers both wide possibilities for integration in cascade with the other algorithms and high results in terms of performance and area occupation.

In the end, we proceeded with the implementation in VHDL of an FBER architecture optimized for the minimization of computational costs. This architecture was derived directly from the formulas of literature and finally an integration was carried out on paper with the BP decoder. The results were excellent both for the reduced area expenditure and for the operating frequencies reached.

A possible future development could concern the expansion of efficiency by integrating the CRC-based algorithm into hardware, as tested and verified by software, to improve early stopping performance.

Bibliography

[1] Erdal Arıkan. "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels". In: *IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 55, NO. 7* (2009), pp. 1–23.

[2] Erdal Arıkan. "Polar codes". In: (2015). URL: https://simons.berkeley.edu/talks/polar-codes.

[3] Amin Alamdar-Yazdi and Frank R. Kschischang. "A Simplified Successive-Cancellation Decoder for Polar Codes". In: *IEEE COMMUNICATIONS LETTERS, VOL. 15, NO. 12* (2011), pp. 1378–1380.

[4] Alptekin Pamuk. "An FPGA Implementation Architecture for Decoding of Polar Codes". In: *8th International Symposium on Wireless Communication Systems, Aachen* (2011), pp. 437–441.

[5] Ji Chen Syed Mohsin Abbas YouZhe Fan and Chi-Ying Tsui. "Low Complexity Belief Propagation Polar Code Decoder". In: *Signal Processing Systems (SiPS), IEEE* (2015).

[6] Bo Yuan and Keshab K. Parhi. "ARCHITECTURE OPTIMIZATIONS FOR BP POLAR DECODERS". In: *ICASSP* (2013), pp. 2654–2658.

[7] Yingxian Zhang. "A Modified Belief Propagation Polar Decoder". In: *IEEE COMMUNICATIONS LETTERS, VOL. 18, NO. 7* (2014), pp. 1091–1094.

[8] Xiaofei Pan Zhan Ye Yingxian Zhang Qingshuang Zhang and Chao Gong. "A Simplified Belief Propagation Decoder for Polar Codes". In: *IEEE* (2014).

[9] Bo Yuan and Keshab K Parhi. "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders". In: *IEEE Transactions on Signal Processing 62.24, IEEE* (2014), pp. 6496–6506.

[10] Shih-Kai Lee Hsin-Fu Yu Huang-Chang Lee. "Early Termination Belief Propagation Decoder with Parity Check Matrix for Polar Codes". In: *The 27th Wireless and Optical Communications Conference (WOCC2018), IEEE* (2018).

[11] Qingshuang Zhang, Aijun Liu, and Xinhai Tong. "Early stopping criterion for belief propagation polar decoder based on frozen bits". In: *ELECTRONICS LETTERS Vol. 53 No. 24* (2017).

[12] Gwan Choi Jingwei Xu Tiben Che. "XJ-BP: Express Journey Belief Propagation Decoding for Polar Codes". In: *2015 IEEE Global Communications Conference (GLOBECOM), IEEE* (2015).

[13] G. Masera A. D. G. Biroli and E Arıkan. "High-throughput belief propagation decoder architectures for polar codes". In: *IEEE SIGNAL PROCESSING LETTERS , VOL. 22, NO. 10* (2015).

[14] D. T. BROWN W. W. PETERSON. "Cyclic Codes for Error Detection". In: *PROCEEDINGS OF THE IRE* (1961), pp. 228–235.