Republic of Iraq
Ministry of Higher Education
and Scientific Research
University of Babylon
College of Engineering
Department of Electrical Engineering

Field Programmable Gate Array Implementation of Turbo Polar Codec Using High-Level Synthesis

A Dissertation

Submitted to the College of Engineering, University of Babylon in Partial Fulfillment of the Requirements for the Degree of Doctorate of Philosophy in Electrical Engineering/Electronic & Communications

By

Wallaa Yaseen Younis Alebady

(B.sc. 2015, M.sc. 2018)

Supervised by

Assist. Prof. Dr. Ahmed A. Hamad

2023 A.D **1444 A.H**

بسم الله الرحمن الرحيم

يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنْكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ

دَرَجَاتٍ وَاللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ ﴿11﴾

صدق اللَّه العلي العظيم

سورة المجادلة

Dedicational

To

My Support and Strength in Life ..... My Father.

To

The Foundation of Love and Tenderness ..... My Mother.

To

The Mate of Road and Life..... My Wife

To

My Jewels in Life..... My Sisters and Brother.

To

My First Teacher ..... Mrs. Batool Hadi.

To

My Dearest Friends.

I Dedicate This Work.

Walla'a

Supervisor's Certification

I certify that this dissertation entitled "**Field Programmable Gate Array Implementation of Turbo Polar Codec Using High-Level Synthesis**" and submitted by the student "**Wallaa Yaseen Younis Alebady"** was prepared under my supervision at the Department of Electrical Engineering, College of Engineering, University of Babylon, as a part of the requirements for the degree of Doctorate of Philosophy in Electronics and Communications Engineering.

Signature**:**

Supervisor**: Asst. Prof. Dr. Ahmed A. Hamad**

Date**:** / / 2023

I certify that this dissertation mentioned above has been completed in Electronics and Communications Engineering in the college of Engineering/ University of Babylon

Signature**:**

Head of Department**: Prof. Dr. Qais Kareem Omran**

Date**:** / / 2023

Examining Committee Certificate

We certify that we have read this dissertation entitled "**Field Programmable Gate Array Implementation of Turbo Polar Codec Using High-Level Synthesis**" and as an examining committee, we examined the student "**Wallaa Yaseen Younis Alebady"**, in its contents and that in our opinion it meets the standard of a dissertation for the degree of **Doctorate of Philosophy** in Electrical Engineering/ Electronics and Communications Engineering.

Signature:
Name: **Prof. Dr. Samir Jasim Al-Muraab**
(Chairman)
Date: / / 2023

Signature:
Name:**Prof. Dr. Ehab AbdulRazzaq Hussein**
(Member)
Date: / / 2023

Signature:
Name: **Prof. Dr. Ahmed Ghanim Wadday**
(Member)
Date: / / 2023

Signature:
Name: **Prof. Dr. Saad Saffah Hassoon**
(Member)
Date: / / 2023

Signature:
Name: **Asst. Prof. Dr.Hilal Abdul-Hussein Abbood**
(Member)
Date: / / 2023

Approval of Head of Department

Signature:
Name: **Prof. Dr. Qais Kareem Omran**
Date: / / 2023

Approval of the Dean of College

Signature:
Name**: Prof. Dr. Hatem Hadi Obeid**
Date: / / 2023

اقرار لجنة المناقشة

نحن اعضاء لجنة المناقشة ، نشهد بأننا اطلعنا على اطروحة الدكتوراه الموسومة:

تنفيذ الترميز القطبي التوربيني بالاعتماد على مصفوفة البوابات المنطقية القابلة للبرمجة في المجال والتوليف العالي المستوى.

وقد ناقشنا الطالب **"ولاء ياسين يونس العبادي"** في محتوياتها وفيهما له علاقة بها ، نؤيد انها جديرة بالقبول لنيل درجة الدكتوراه في الهندسة الكهربائية/الالكترونيك والاتصالات.

رئيس اللجنة

التوقيع:

الأسم: **أ.د. سمير جاسم المرعب**

التاريخ: / /2023

عضو اللجنة

التوقيع:

الأسم: **أ.د. إيهاب عبد الرزاق حسين**

التاريخ: / /2023

عضو اللجنة

التوقيع:

الأسم: **أ.د. أحمد غانم وداي**

التاريخ: / /2023

عضو اللجنة

التوقيع:

الأسم: **أ.د.سعد سفاح حسون**

التاريخ: / /2023

عضو اللجنة

التوقيع:

الأسم: **أ.م.د. هلال عبد الحسين عبود**

التاريخ: / /2023

مصادقة رئيس القسم

التوقيع:

الأسم: **أ.د. قيس كريم عمران**

التاريخ: / /2023

مصادقة عميد الكلية

التوقيع:

الأسم: **أ.د. حاتم هادي عبيد**

التاريخ: / /2023

Acknowledgments

First of all, I praise ALLAH Almighty for providing me this opportunity and granting me the capability to proceed with this dissertation.

I would like to warmly thank and gratitude my father, my mother, my wife, my brother, and my sisters for their spiritual support in all aspects of my life.

I would like to express my profound gratitude and my respect to my supervisor Dr. Ahmed A. Hamad, for his supervision, guidance, and constant support.

I would like to thank all my friends for their friendship and encouragement during the period of study. I also appreciate the help and support from all persons directly or indirectly involved in my project.

Wallaa

Abstract

Since the first invention of the polar code by Arikan, the research field of this code is still active. The main core of this field focuses on improving the performance of the polar codes with a finite code length and improving the hardware implementation of these codes. The hardware implementation can be improved by overcoming the throughput bottleneck created by the trade-off issue between resource utilization and throughput. This dissertation suggests several turbo-polar schemes in addition to their corresponding hardware implementation using the Soft-Cancelation (SCAN) algorithm as a constituent decoder. These turbo schemes and other polar schemes are presented in this study with four scenarios.

First, the original and modified polar codes are simulated and implemented using the Xilinx Kintex-7 Field Programmable Gate Array (FPGA) device with the pipelining directive and arbitrary-precision data types. Second, the Parallel Concatenated Turbo Polar Code (PCTPC) scheme has been presented with Systematic Polar Codes (SPCs) as constituent encoding units. On the decoder side, two SCAN decoders with a single internal iteration are used as Soft-In-Soft-Out (SISO) algorithms to construct the iterative decoding algorithm of the suggested PCTPC. This scheme mitigates the issue of error floor that typical turbo convolutional codes have. At Bit Error Rate (BER) of 10^{-6}, the suggested PCTPC scheme achieves gains of more than 0.5dB over the original turbo convolutional code.

Third, the Parallel Concatenated Turbo Polar Convolutional Code (PCTPCC) has been introduced using the Systematic Polar Code (SPC) and Recursive Systematic Convolutional (RSC) code as constituent units. The proposed polar-convolutional iterative decoding algorithm has been constructed using a SCAN decoder with a single internal iteration for the SPC and the Soft-Output Viterbi Algorithm (SOVA) or the Logarithmic Maximum

A Posteriori (LogMap) decoder for the RSC. The proposed PCTPCC scheme has been suggested to further improve the PCTPC scheme by replacing one of the constituent SPCs with an RSC. At BER & FER of 10^{-6}, the LogMap-PCTPCC(Nc=384, K=128) exceeds the PCTPC(Nc=384, K=128) scheme by gains of about 0.525dB and 0.5dB, respectively.

Fourth, the Serial Concatenated Turbo Polar-Convolutional Code (SCTPCC) has been proposed to eliminate the error floor problems even more. The SCTPCC scheme consists of outer-SPC and inner-RSC codes with rates of 2/3 and 1/2, respectively. This scheme outperforms the parallel turbo polar schemes but has a growing complexity level due to the inner code length being twice as long as the outer code. At FER of 10^{-5}, the SOVA and LogMap SCTPCC schemes exceed the SOVA and LogMap PCTPCC schemes by gains of 0.34dB and 0.36dB, respectively. Although the SCTPCC scheme outperforms the PCTPCC schemes in high SNR regions, the simulation results show that the proposed PCTCC schemes reveal better performance in terms of BER and FER than the proposed SCTPCC schemes in low SNR regions.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
3D-PC	Three-Dimensional Polar Code
3GPP	3rd Generation Partnership Project
APPs	posteriori probabilities
ASICs	Application-Specific Integrated Circuits
ASSPs	Application-Specific Standard Parts
AWGN	Additive White Gaussian Noise
BCH	Bose Chaudhuri Hocquenghem
BCJR	Bahl Cocke Jelinek Raviv
BEC	Binary Erasure Channel
BER	Bit Error Rate
BI-DMC	Binary-Input Discrete Memoryless Channels
BP	Belief Propagation
BPSK	Binary Phase Shift Keying
BTC	Block Turbo Codes
CLB	Configurable Logic Block
CRC	Cyclic-Redundancy-Check
CTC	Convolutional Turbo Codes
CTPC	Concatenated Turbo Polar Code
CCSDS	Consultative Committee for Space Data System
DSP	Digital Signal Processing
eMBB	Enhanced Mobile Broadband
e-RAM	Embedded-RAM
FER	Frame Error Rate
FPGA	Field Programmable Gate Array
GF	Galois Field
HDL	Hardware Description Language
IP	Intellectual Property
ISP	In-System Programmable
LC	Logic Cell
LDPC	Low-Density Parity Check
LLR	Log-Likelihood Ratio

LogMap	Logarithmic Maximum A Posteriori Decoder
MCM	Multi-Chip Module.
ML	Maximum-Likelihood
MLM	Max-Log-MAP Decoder
NSPE	Non-Systematic Polar Encoder
OTP	One Time Programmable
PCCC	Parallel Concatenated Convolutional Codes
PCTPC	Parallel Concatenated Turbo Polar Code
PCTPCC	Parallel Concatenated Turbo Polar-Convolutional Code
PLBs	Programmable Logic Blocks
PAPR	Peak to Average Power Ratio
RSC	Recursive Systematic Convolutional
RTL	Register Transfer Level
SCAN	Soft Cancellation
SCCC	Serial Concatenated Convolutional Codes
SCD	Successive Cancellation Decoder
SCL	Successive Cancellation List
SCTPCC	Serial Concatenated Turbo Polar-Convolutional Code
SDK	Software Development Kit
SISO	Soft-In-Soft-Out
SOVA	Soft-Output Viterbi Decoder
SPC	Systematic Polar Code
SPE	Systematic Polar Encoder
SSC	Simplified Successive Cancellation
SSCL	Soft Successive Cancellation List
SSTPC	SCAN-SCAN Turbo Polar Code
ST-SPE	Serial Turbo- Systematic polar Encoder
TPCs	Turbo Polar Codes
T-SPE	Turbo- Systematic polar Encoder
VA	Viterbi Algorithm
WiMAX	Worldwide Interoperability for Microwave Access

List of Symbols

Item	Description
$\mathcal{A}$	Forward Log Probability Matrix of the LogMap Decoder
$\mathcal{B}$	Backward Log Probability Matrix of the LogMap Decoder
$\mathbb{B}_N$	$N \times N$ Bit-Reversal-Permutation-Matrix
b	LLRs output of a SISO decoder
C	Coded Bits Vector
C_{λ^c}	Sub-Encoded Vector of User Bits.
C_{λ}	Sub-Encoded Vector of Parity Bits
d	User input Bits Vector
d_{min}	Minimum Hamming Distance
E_b	Energy Per Information Bit
$\mathcal{F}$	Frozen Bits Indices Vector
$\mathcal{F}^c$	Information Bits Indices Vector
$\mathrm{F}^{\otimes \mathrm{n}}$	n^{th} Order Hadamard Matrix
$f_{R\|C}(.)$	Likelihood Probability Density Function (LPDF) in an AWGN Channel
G_N	Generator Matrix
$G_{\mathcal{F}}$	Sub-Matrix of the Original Generator Matrix (G) With Row-Indices Corresponding to $\mathcal{F}$.
$G_{\mathcal{F}^c}$	Sub- Matrix of the Original Generator Matrix (G) With Row-Indices Corresponding to $\mathcal{F}^c$
$G_{f^c\,\lambda^c}$	Sub- Matrix of the Original Generator Matrix (G) With Row and Column Indices Corresponding to $\mathcal{F}^c$ and λ^c
$G_{\mathcal{F}\,\lambda^c}$	Sub- Matrix of the Original Generator Matrix (G) With Row and Column Indices Corresponding to $\mathcal{F}$ and λ^c
I_{s_i}	Number of SCAN Iterations
I_{B_i}	Number of BP Iterations
I_{list}	List Size of SCL Decoder
I_{outer}	Outer Iterations Number of the Turbo Codes.
I_{SCAN}	Soft Input Vector of SCAN Decoder
I_{SCAN}^{Fc}	SCAN Input LLRs of Information Sequence.
I_{SCAN}^{F}	SCAN Input LLRs of Frozen Sequence.

$I(\mathcal{W})$	Symmetric Channel Capacity
K	Information Length
L_c	Channel Reliability
m	D-elements Number of RSC Encoder
$\hat{m}_i, \hat{d}_i$	Decoded Message Bits
$\mathcal{M}$	Path Metric Matrix
N	Code Length
N_c	Length of Parallel Turbo Codes
N_R	Encoded Symbols Number of the RSC Encoder
N_{inner}	Length of inner encoded bits
N_{outer}	Length of outer encoded bits
N_o	Noise Power Spectral Density
$O^{inf.}$	Decoded LLRs of Information Sequence.
O^{parity}	Decoded LLRs of Parity Sequence.
O_{scan}	Soft Output Vector of SCAN Decoder
O_{scan}^{Fc}	SCAN Decoded LLRs of Information Sequence.
O_{scan}^{F}	SCAN Decoded LLRs of Parity Sequence.
p	Column index of the Factor Graph
$\mathbb{R}_N$	Odd-Even Permutation Matrix
$\mathfrak{R}$	Code Rate
$\mathfrak{R}^i$	Inner Code Rate of Serial Turbo Codes
$\mathfrak{R}^o$	Outer Code Rate of Serial Turbo Codes
R	LLRs Vector of the Received Symbol
R_d	Received LLRs of the Systematic User's Bits
R_P	Received LLRs of the Systematic Parity Bits
R_d^{π}	Interleaved Version of R_d
S	Input Vector of Polar Encoder
$S_{\mathcal{F}^c}$	K-Information Sub-Vector
$S_{\mathcal{F}}$	Frozen Bits Sub-Vector
SF_1 & SF_2	Scaling Factors
STF	Stooping Test Factor
S_{t^-}	Current State
S_{t^+}	Present State

Thr.	Through Puts (b/s)
$\mathcal{W}^i$	A polarized Channel
Y	Modulated Coded Bits Vector
$Z(\mathcal{W})$	Bhattacharyya-Parameter
λ	Indices Vector of Systematic Frozen Encoded Bits
$\otimes$n	n^{th} Kronecker Product
λ^c	Indices Vector of Systematic Information Bits
$\mathcal{L}_T$	Output LLR Vector
α_N^{n+1}	$(N \times n + 1)$-Alpha Matrix of SC and SCAN Decodes
β_N^{n+1}	$(N \times n + 1)$-Beta Matrix of SC and SCAN Decodes
σ^2	Noise Variance
φ^s	Scaled a-Prior Information
$\emptyset$	Group index of the Factor Graph
Δ	Path Metric Difference Matrix
Γ	Log Metric Matrix
γ	Noise Vector with Noise Variance σ^2
δ	Decoding SOVA-Depth
ε	Extrinsic Information
μ	Branch Metric Matrix
π	Interleaver Indices Vector
φ	A-prior LLRs of Message Bits.
ω	Node index of the Factor Graph

Chapter One

Introduction

Chapter One

Introduction

1.1. Introduction

Shannon left the door of telecommunication research open through his rigorous foundation: "a mathematical theory of communication"[1]. The coding research field has been striving to find a practical coding design that achieves Shannon's channel capacity ever since. Practically, approaching Shannon's limit is the most interesting and oldest challenge within the coding community since that time. This challenge continues to be the main subject of research using various channel models. Until the 1990s, with the creation of the Low-Density Parity Check (LDPC) and turbo codes, all research was fairly modest. These new coding styles can approach the capacity of an Additive White Gaussian Noise (AWGN) channel up to a fraction of a decibel (dB) of SNR. In many practical cases, these codes solve the problem of achieving the channel capacity (Shannon limit). Nevertheless, none of these codes can be proven to reach Shannon's limit, and they can't work well with the finite length regime [2][3][4]. Additionally, the LDPC and turbo codes have other troubles, such as high encoder complexity [5], the latency and complexity of their iterative decoding algorithm are non-deterministic, and error floors[6]–[9]. Moreover, due to their random structure, these codes have many implementation challenges, such as memory-access problems and routing congestion [10][11]. Therefore, the aim of obtaining a practical coding design that achieves the channel capacity is still open on any channel.

Arikan introduced the solution for this seven-decade-old problem by creating the polar code utilizing the channel polarization concept [12]. Over a wide range of Binary-Input Discrete Memoryless Channels (BI-DMC), polar

code is the first code used to prove the asymptotical achievement of the channel capacity. The BI-DMC channel class was the initial target to prove the channel capacity achievement, but later the results were extended to a general Discrete Memoryless Channel (DMC). Polar code has changed the performance equation in the coding theory since its first invention in 2009 [12]. The attention of most researchers in the information theory field has been focused on the polar code. Moreover, the polar code acted as a motivation to discover other proven capacity-achieving codes.

Polar codes have many attractive attributes, such as lack of error floors, low computational and fixed complexity at decoder and encoder, simple implementations, efficient utilization of silicon area, and deterministic structure. Several of these features have been demonstrated in the literature. As a result of these features, the 3rd Generation Partnership Project (3GPP) recommended using polar codes in the 5G system on 17-November 2016. In the Enhanced Mobile Broadband (eMBB) system, these codes are introduced for the 5G applications as a control channel coding scheme [13]. Despite all of the advantages of polar code, implementing it in practical systems still faces several challenges. These challenges include reducing the latency, creating an efficient systematic encoder, enhancing the code performance, and so on [3].

In terms of Bit Error Rate (BER) and Frame Error Rate (FER), the first polar code with a Successive Cancellation Decoder (SCD) performs poorly with finite-code lengths. Two methods can achieve the performance improvement of the original polar code. The first method is based on proposing a new decoder scheme, such as Belief Propagation (BP), Successive Cancellation list (SCL), Soft Cancellation (SCAN), Simplified Successive Cancellation (SSC), Soft Successive Cancellation List (SSCL), Fast-SSC, and others. At the same time, the second method can be achieved

by introducing the Turbo Polar Codes (TPCs). The TPCs can be classified into serial and parallel concatenated codes. This dissertation will present several methods and models to improve the performance of polar codes, especially with a finite code length and a small SNR

1.2. Literature Survey

Since Arkan's seminal paper [12] in 2009 till the present, there have been thousands of research papers and studies on polar codes, making up a sizable portion of the literature. It is impossible to review all these works, so the literature survey in this dissertation will focus on the design of the fundamental polar codes, the practical implementation of the polar codes, and the Turbo Polar Codes(TPCs). The inferior performance of Polar codes at a finite code length; due to the use of a suboptimal SC decoder, a lower minimum distance structure of the code itself, and the sequential handling of the decoder, is the foremost challenge when compared with the state-of-the-art codes. This challenge is the core of a large amount of research which can be divided into two groups, as discussed below.

1.2.1. Polar Decoders Survey

The first strategy involves introducing new polar decoders to improve the code performance in terms of delay and error correction. In **2008**, **Arikan** introduced the belief propagation (BP) decoder as the early attempt in this approach [14]. After that, the step that achieved qualitative leaps in the performance of the polar code was the introduction of the successive cancellation list (SCL) decoder in **2011** by **Tal and Vardy** [15] and the Soft Cancellation (SCAN) decoder by **Fayyaz in 2013** [16][17]. The performance of the SCL decoder is more approach to that of the Maximum-Likelihood (ML) decoder [15]. As stated in [15], the polar codes with the SCL decoder outperform the state-of-the-art Low-Density-Parity-Check codes used in the

WiMAX standard but with a high level of complexity. In contrast, the SCAN decoder offers promising features in terms of complexity and error correction performance. Further decoding algorithms are introduced to improve the performance of the polar code including linear program decoder [18], multi-dimensional decoder [19], ordered-statistic decoder [20], and others.

To overcome the practical challenges mentioned earlier, the hardware implementation of the decoders mentioned above using various processors is the core of several research works, especially the successive cancellation (SC) and successive cancellation list (SCL) decoders. In [21]–[24], **Dizdar 2016**, **Sarkis 2016**, **Berhault 2017**, and **Giard 2018**, repectively, implement the polar codes with different decoding algorithms. The main objective of these works is to obtain high throughput with less consumption of resources..

1.2.2. Concatenated Polar Codes Survey

The second improvement strategy involves introducing parallel and serial concatenated turbo-polar codes. This strategy is more efficient than the first in improving the delay and error correction performance with finite-code lengths. The first serial concatenated polar code is constructed by concatenating the polar code with a Cyclic Redundancy Check (CRC) code in a serial structure [15]. In [15], **Tal and Vardy** use the CRC code with the polar code to considere L decoding paths concurrently at each decoding stage, where L is an integer parameter, where L is an integer parameter. At the end of the decoding process, the most likely among the L paths is selected as the single codeword at the decoder output. Then, the polar code is concatenated serially with another polar code or with other different codes such as Reed-Solomon, BCH, LDPC, and Convolutional codes[25]–[31]. In [26], **Eslami and Pishro-Nik** introduced the Polar-LDPC code to analyzing the performance of finite-length polar codes over the binary erasure channel (BEC). The code analysis along with bit error rate (BER) simulations

demonstrated that finite-length polar codes showed superior error floor performance compared to the conventional capacity-approaching coding techniques. In [28], **Wang and Krishna** showed that the convolutional-polar codes; for the AWGN channel; outperform the stand-alone polar code and RS-polar codes. In addition, Wang and Krishna showed that the BCH-polar codes outperform the stand-alone polar codes with the Binary Erasure Channel(BEC).

Moreover, the second strategy includes the Parallel Concatenated Turbo Polar Codes (PCTPCs). This arrangement showed promising performance, especially at a low signal-to-noise ratio. In **2015, Qingshuang Zhang** and his colleagues proposed the first parallel turbo polar code using the Systematic Polar Codes (SPCs) and the Recursive Systematic Convolutional Code (RSC) as constituent components [32]. This code depends on separating the information bit-stream into multiple short blocks, and then each block is encoded by an individual SPC encoder. Although this code performs well, it is more complex than other models due to the approach of using multiple short blocks. The iterative decoding algorithm of Qingshuang scheme consists of BCJR and BP decoders for RSC and SPC codes, respectively. In **2016**, the same team introduced a new PCTPC scheme consisting of two SPC codes as constituent components [33]. The complexity of this new scheme is less than the old one. In **2017** and **2018**, another team consisting of **Zhenzhen Liu** and her colleagues exploited the second Qingshuang scheme to examine other decoding algorithms [34], [35]. They used Soft Successive Cancellation List (SSCL) and SCAN decoders to construct the iterative decoding algorithms of the 2017 and 2018 schemes, respectively. In addition, they utilized the BP decoder and a scaling factor optimization algorithm in the 2018-scheme. The complexity analysis of previous PCTPC schemes shows that the SCAN decoder offers lower complexity than the other decoding algorithms. Another

scheme of PCTPC is presented in [36], which is the Punctured Turbo Polar Code (PTPC). This scheme is proposed to get a compatible code rate by puncturing some parity bits. As stated in [36], the asymmetric puncturing strategy (i.e., the parity bits of the first branch are more than those of the second branch) outperforms the symmetric puncturing strategy.

Finally, the **Zhenzhen** team has exploited the advantages of serial and parallel concatenated turbo codes to construct a new coding scheme known as the Three-Dimensional Polar Code (3D-PC) [37]. This code consists of Systematic PCTPC as the outer code and Rate-1 RSC as the inner code. The 3D-PC code has been presented to overcome the error-floor problem inherent in the parallel turbo codes.

In this dissertation, the FPGA hardware implementation of the non-systematic and systematic polar codes with the SCAN decoder are implemented by introducing further solutions for the practical challenges. Furthermore, the performance of polar codes in finite-length regime is further improved by introducing new serial and parallel concatenated turbo polar codes. These suggested systems handle the deteriorated performance and the high complexity of some previously presented concatenated turbo polar codes.

1.3. Dissertation Objectives

The main objectives of this study can be summarized as follows:

- Introducing new solutions to polar codes' challenges in terms of code design and hardware implementation.
- Addressing the following problems:
  - ✓ The deteriorated performance of polar codes in a finite code-length regime.
  - ✓ The high complexity of the decoding algorithms.

- ✓ The error-floor problem facing the PCTPC.
- ✓ The hardware challenges.

- Introducing a new FPGA implementation for the non-systematic and systematic polar codes, and also for the concatenation turbo polar codes.

1.4. Contributions Summary

The contributions aspects of this study can be summarized as follows:

1) Enhancement

- Improving the existing SCAN-SCAN Turbo Polar Codes(SSTPCs).

2) Proposals

- Proposing a new Parallel Concatenated Turbo Polar-Convolutional Code (PCTPCC). This code is presented to improve the performance of the polar code at a low SNR. The proposed PCTPCC scheme consists of systematic polar and convolutional codes.
- Proposing a new Serial Concatenated Turbo Polar-Convolutional Code (SCTPCC). The performance of the PCTPC at a high SNR deteriorates because of the error floor problem, so this scheme is suggested to improve polar codes' performance at this ratio.

3) Hardware Implementation

- Suggesting a new FPGA architecture to implement the encoder and decoder of the systematic and Non-Systematic polar codes with a SCAN decoder.
- Introducing FPGA architectures to implement the schemes mentioned previously in the enhancement and proposals sections.

  In both above suggestions, we introduce a trade-off between the latency and resource requirements and choose the best option.

1.5. Dissertation Outline

In this section, the structure of this dissertation is presented as follows:

I) Chapter Two: This chapter shows the main theory of polar code construction and the basics of polar and convolutional codes in terms of encoder and decoder. Moreover, this chapter provides an introduction to the structure of the Field Programmable Gate Array (FPGA).

II) Chapter Three: The Concatenated Turbo Polar Codes (CTPCs) schemes are presented in this chapter for both serial and parallel types. This chapter contains the encoders' and decoders' designs of these codes and the theoretical aspect of some issues related to the concatenated turbo polar codes. Moreover, some case studies are presented in this chapter.

III) Chapter Four: This chapter presents the implemented designs of the proposed schemes with the simulation results of these schemes. The simulation results of these schemes are also compared with other schemes and illustrated in the form of BER and FER graphs. Moreover, the FPGA implementation results of the studied schemes(enhanced & proposed) are shown along with the analysis and discussion of these results.

IV) Chapter Five: The conclusions and some future works are presented in this chapter.

2

Chapter Two

Concatenated Codes Construction and FPGA Architecture

Chapter Two

Concatenated Codes Construction and FPGA Architecture

2.1. Introduction

This chapter illustrates the construction strategy of polar codes and the preliminaries of polar and convolutional codes in terms of encoding and decoding. A brief description of FPGA architecture is also introduced. Several decoding algorithms can be used to decode the coded message of polar and convolutional codes. However, this chapter only presents the SC and SCAN algorithms for polar codes and the SOVA and LogMap algorithms for convolutional codes.

2.2. The Parameters Notations

Generally, three parameters are required to specify a linear block code, which are the information length (K), code length (N), and the minimum hamming distance (d_{min}). These parameters are written as a triple (N,K, d_{min}) for any linear block code and convolutional code. Polar codes also follow this strategy as follows.

Four parameters are required to specify a polar code, namely, information length (K), codeword length (N), frozen bits indices vector ($\mathcal{F}$) of length ($N-K$), and frozen bits vector ($S_{\mathcal{F}}$). The N-input set (S) consists of two subsets, which are a K-information subset ($S_{\mathcal{F}^c}$) and a frozen subset ($S_{\mathcal{F}}$) of length $N-K$. A polar code can be specified by a four-tuple $(N, K, \mathcal{F}, S_{\mathcal{F}})$ or by a triple $(N, K, \mathcal{F})$ if the frozen bits ($S_{\mathcal{F}}$) are set to zeros. The zero frozen bits vector is the default case of polar codes.

The operations $\subset, \cup$, and \ represent the subset relation of two sets, union relation, and the set difference relation on two sets, respectively, for instance: $\mathcal{F} \subset N,\ N = \mathcal{F}^c \cup \mathcal{F}, \text{and } \mathcal{F}^c = N \backslash \mathcal{F}$. In this dissertation, the q_m^n notation is used as a row shorthand vector, i.e. $(q_m, q_{m+1}, \dots\dots, q_n)$, such as $a_0^3 = [a_0, a_1, a_2, a_3]$. Moreover, the Q^M notation is a row vector consisting of M elements. The $A(i_0^{M-1}, n)$ expression is used for matrix notation to identify all rows from index 0 to $(M-1)$ in column n of matrix A. The (i_0^{N-1}) notation defines a range of indices.

2.3. The Channel Modelling

The Binary Input Additive White Gaussian Noise Channel (BI-AWGN) is employed to demonstrate the construction of polar codes. The E_b/N_o expression is used to define the SNR, where N_o and E_b are the noise power spectral density and the energy per information bit, respectively. The coded bits vector (C) is modulated using a BPSK scheme to get the modulated vector (Y), i.e., $1 \rightarrow +\sqrt{\Re E_b}$ and $0 \rightarrow -\sqrt{\Re E_b}$ where $(\Re = \frac{K}{N})$ is the code rate, and the $(\Re E_b)$ is the energy per coded bit. The channel output is depicted in Figure (2.1) and as follows [3]:

$$R = Y + n, \qquad n \sim G\left(0, \frac{N_o}{2}\right) \tag{2.1}$$

Where $G(0, \frac{N_o}{2})$ is a zero-mean Gaussian distribution with variance $\frac{N_o}{2}$, R is the received values across the channel, Y is the modulated input values, and n is the noise values inside the channel. The channel output can be expressed as an LLR of the received symbol as follows [3]:

$$L(r) \triangleq \log \frac{f_{R|C}(r|c=0)}{f_{R|C}(r|c=1)} = \frac{-4r\sqrt{\Re E_b}}{N_o} \tag{2.2}$$

Where $C = [c^0, c^1, \dots, c^N], R = [r^0, r^1, \dots, r^N]$, and $f_{R|C}(.)$ defines the Likelihood Probability Density Functions (LPDF) in an AWGN channel [3]:

$$f_{R|C}(r|c = 0) \sim G\left(-\sqrt{\Re E_b}, \frac{N_o}{2}\right) \quad (2.3)$$

$$f_{R|C}(r|c = 1) \sim G\left(+\sqrt{\Re E_b}, \frac{N_o}{2}\right) \quad (2.4)$$

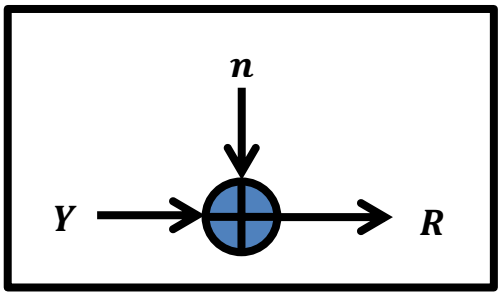


Figure (2.1): The Channel Model.

2.4. Symmetric-Capacity and Bhattacharyya-Parameter

These two parameters are essential for constructing polar codes over a Binary-Input Discrete memoryless Channel (BI-DMC).

Definition-2.1. The Symmetric Capacity $I(\mathcal{W})$ is the highest transmission rate over channel $\mathcal{W}$. At this rate, reliable communication between a transmitter and receiver can be achieved. The Symmetric Capacity is expressed as follows [39] [40]:

$$I(\mathcal{W}) \triangleq \sum_{r \in R} \sum_{c \in C} \frac{1}{2} \mathcal{W}(r|c) \log \frac{\mathcal{W}(r|c)}{\frac{1}{2}[\mathcal{W}(r|c = 0) + \mathcal{W}(r|c = 1)]} \quad (2.5)$$

Where $\mathcal{W}(r|c)$ is the channel transition probability.

Definition-2.2. The Bhattacharyya-Parameter $Z(\mathcal{W})$ is defined as an Upper-Bound (UP) on the probability of ML decision error when a 1 or 0 is transmitted across channel $\mathcal{W}$. This parameter's value, which ranges from zero to one like the value of $I(\mathcal{W})$, is used to estimate the channel reliability.

The mathematical expression of the Bhattacharyya-Parameter can be defined as follows [39] [40]:

$$Z(\mathcal{W}) \triangleq \sum_{r \in R} \sqrt{\mathcal{W}(r|c=0) \times \mathcal{W}(r|c=1)} \quad (2.6)$$

I($\mathcal{W}$) has a value of about one only if $Z(\mathcal{W})$ has a value of roughly zero and vice versa.

2.5. Polar Codes Construction

The polar code's construction is simply the selection of the frozen bits indices set ($\mathcal{F}$). Although Polar codes have several construction methods, only the Bhattacharyya approximation method is presented in this chapter. The polar code construction is based on the principle of channel polarization, which in turn consists of two steps: channel combining and channel splitting [3][12][41].

2.5.1. Channel Polarization

Channel polarization is the process of obtaining N polarized channels ($\mathcal{W}_N^i: 1 \leq i \leq N$) out of N independent BI-DMC channels by recursively combining and splitting these independent channels. When N becomes large, the symmetric capacity of some of these polarized channels is close to 1 (completely noiseless channels), whereas it is close to 0 for others (totally noisy channels). The error-free channels (noiseless channels) are assigned for information bits, and the noisy channels are used to send fixed bits (frozen bits) known to the transmitter and receiver [12][42].

2.5.1.1. Channel Combining

The first channel polarization step combines N independent BI-DMC channels (like AWGN channels) recursively to create a vector channel $\mathcal{W}_N: C^N \rightarrow R^N$, where $N = 2^n;\ n \geq 0$. The first level ($n = 0$) of the channel

combination starts with one copy of the BI-DMC channel $\mathcal{W}$ to create $\mathcal{W}_1$, i.e., $N = 2^0 = 1$, $\mathcal{W}_1 \triangleq \mathcal{W}$. The second level ($n = 1$) creates the channel $\mathcal{W}_2: C^2 \rightarrow R^2$ by combining two independent BI-DMC channels, as illustrated in Figure (2.2). The transition probabilities of the channel $\mathcal{W}_2$ are as follows [12]:

$$\mathcal{W}_2(r_1^2|s_1^2) = \mathcal{W}(r_1|s_1 \oplus s_2)\mathcal{W}(r_2|s_2) \qquad (2.7)$$

Figure (2.2) shows the third level ($n = 2$) of the channel combination by combining two copies of the channel $\mathcal{W}_2$ to create the channel $\mathcal{W}_4: C^4 \rightarrow R^4$. The transition probabilities of this channel are as follows [12]:

$$\mathcal{W}_4(r_1^4|s_1^4) = \mathcal{W}_2(r_1^2|s_1 \oplus s_2, s_3 \oplus s_4)\mathcal{W}_2(r_3^4|s_2, s_4) \qquad (2.8)$$

After the second level of the channel combination, the odd-even permutation matrices ($\mathbb{R}_N$) are required, as shown in Figure (2.3). These matrices are used to connect the recursive channels with each other, such as between the inputs of $\mathcal{W}_4$ & $\mathcal{W}_2$, also between these of $\mathcal{W}_8$ & $\mathcal{W}_4$ and so on. In the fourth level, the matrices $\mathbb{R}_4$ and $\mathbb{R}_8$ are required to construct the channel $\mathcal{W}_8: C^8 \rightarrow R^8$, as shown in Figures (2.3) and (2.4).

Figure (2.5) shows the general form of the channel combination with $\mathbb{R}_N$. To create this general form, two independent copies of the channel $\mathcal{W}_{\frac{N}{2}}$ are combined to create the final channel $\mathcal{W}_N$. As illustrated in general form, the input vector S_1^N is converted to another vector, i.e., X_1^N by applying the second-order polar encoding where $\left\{x_{2i-1} = s_{2i-1} \oplus s_{2i},\ x_{2i} = s_{2i}; for\ 1 \leq i \leq \frac{N}{2}\right\}$. Then, this vector is transformed into an odd-even permutation vector (i.e. $Y_1^N = [x_1, x_3, \dots, x_{N-1}, x_2, x_4, \dots, x_N]$) after being multiplied by the odd-even permutation matrix $\mathbb{R}_N$ [12].

It can be observed that the mapping from the inputs of the created channel $\mathcal{W}_N$ to the inputs of the underlying channel $\mathcal{W}^N$(i.e., $S_1^N \rightarrow C_1^N$) is a

linear process over $GF(2)$. This linear process can be performed as follows [12]:

$$C_1^N = S_1^N . G_N \tag{2.9}$$

Where (G_N) is the generator matrix.

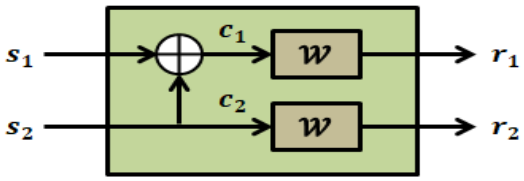


Figure (2.2): Second Level of Channel Combination [42].

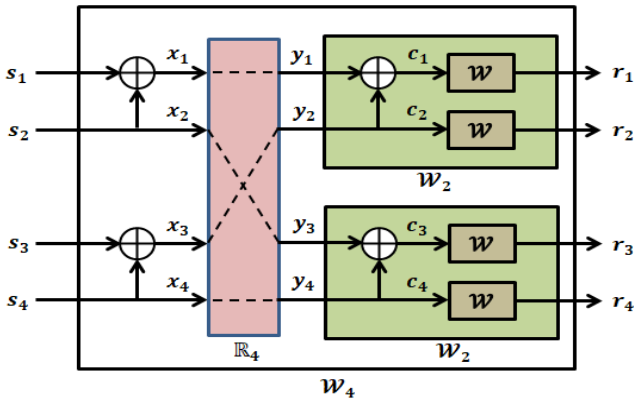


Figure (2.3): Construction of the Channel $\mathcal{W}_4$ [42].

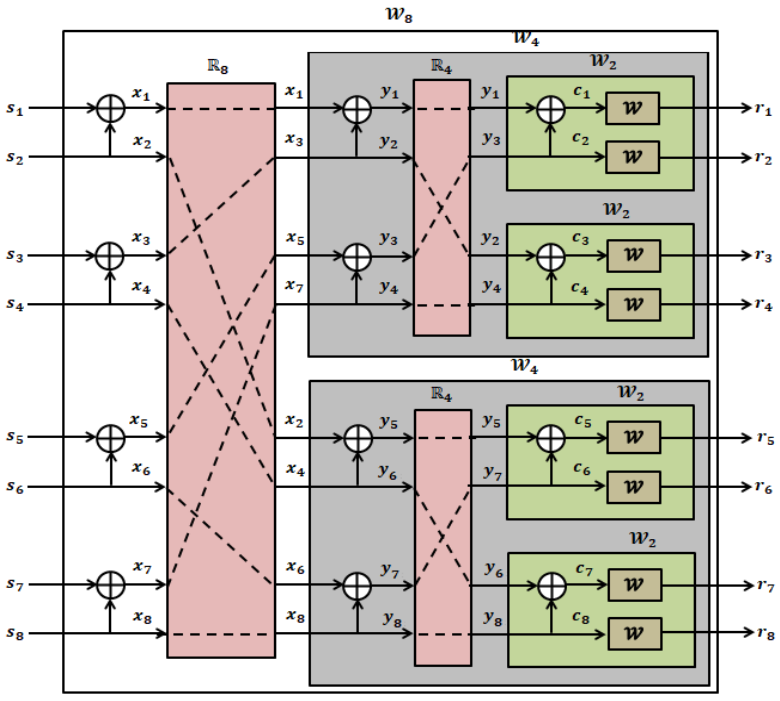


Figure (2.4): Construction of the Channel $\mathcal{W}_8$ [42].

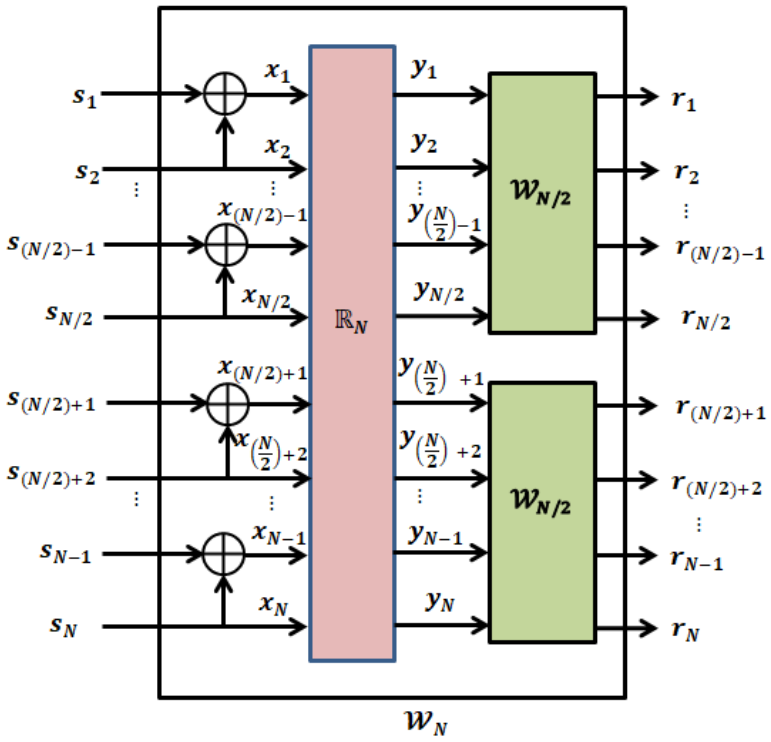


Figure (2.5): The General Form of Channel Combination [42].

2.5.1.2. Channel Splitting

The second step of channel polarization after creating the vector channel ($\mathcal{W}_N$) is channel splitting. This step splits back the channel ($\mathcal{W}_N$) into N different channels ($\mathcal{W}_N^{(l)}: C \to R^N \times C^{l-1}, 1 \le l \le N, C \in \{0,1\}$) with the following transition probabilities [12]:

$$\mathcal{W}_N^{(l)}\left(r_1^N, s_1^{l-1} | s_l\right) \triangleq \sum_{s_{l+1}^N \in C^{N-l}} \frac{1}{2^{N-1}} \mathcal{W}_N(r_1^N | s_1^N) \qquad (2.10)$$

Where the (r_1^N, s_1^{l-1}) and (s_l) are the output and input of the channel $\mathcal{W}_N^{(l)}$, respectively. Figure(2.6) illustrates the binary-input channels $\mathcal{W}_2^{(1)}$ and $\mathcal{W}_2^{(2)}$ which can be called $\mathcal{W}^-$ and $\mathcal{W}^+$ with the following transition probabilities [12]:

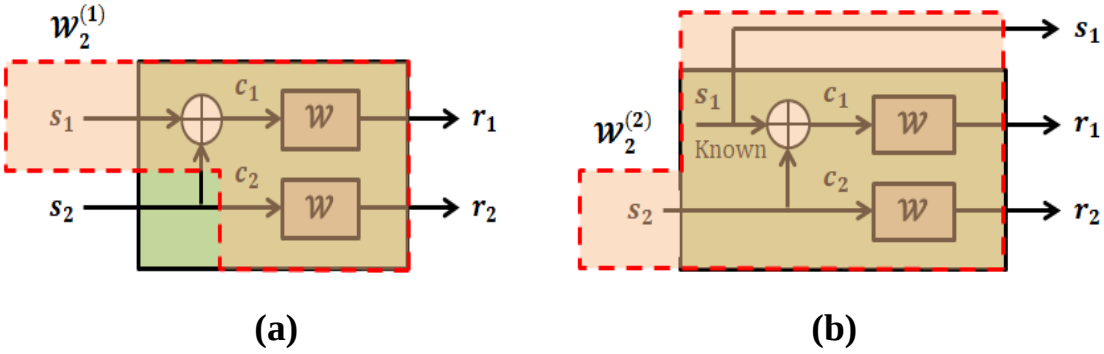
$$\mathcal{W}_2^{(1)}(r_1^2 | s_1) \triangleq \sum_{s_2} \frac{1}{2} \mathcal{W}_2(r_1^2 | s_1^2) = \sum_{s_2} \frac{1}{2} \mathcal{W}(r_1 | s_1 \oplus s_2) \mathcal{W}(r_2 | s_2) \qquad (2.11)$$

$$\mathcal{W}_2^{(2)}(r_1^2, s_1|s_2) \triangleq \frac{1}{2}\mathcal{W}_2(r_1^2|s_1^2) = \frac{1}{2}\mathcal{W}(r_1|s_1 \oplus s_2)\mathcal{W}(r_2|s_2) \qquad (2.12)$$

The new binary-input channels ($\mathcal{W}^-$ & $\mathcal{W}^+$) have the following properties[40], [42], [43]:

- $I(\mathcal{W}^-) + I(\mathcal{W}^+) = 2I(\mathcal{W})$: Where the $I(\mathcal{W})$ is the capacity of the underlying channel ($\mathcal{W}$).
- $Z(\mathcal{W}^-) \leq 2Z(\mathcal{W}) - [Z(\mathcal{W})]^2$
- $Z(\mathcal{W}^+) = [Z(\mathcal{W})]^2$

Thus, the channel capacity of ($\mathcal{W}^+$) is greater than that of the original BI-DMC and also greater than that of ($\mathcal{W}^-$) [i.e., $I(\mathcal{W}^+) \geq I(\mathcal{W}) \geq I(\mathcal{W}^-)$]. As the length of polar codes increases, the channel capacities of ($\mathcal{W}^+$) and ($\mathcal{W}^-$) tend to 1 and 0, respectively, as shown in Figure (2.7) [40], [44], [45].

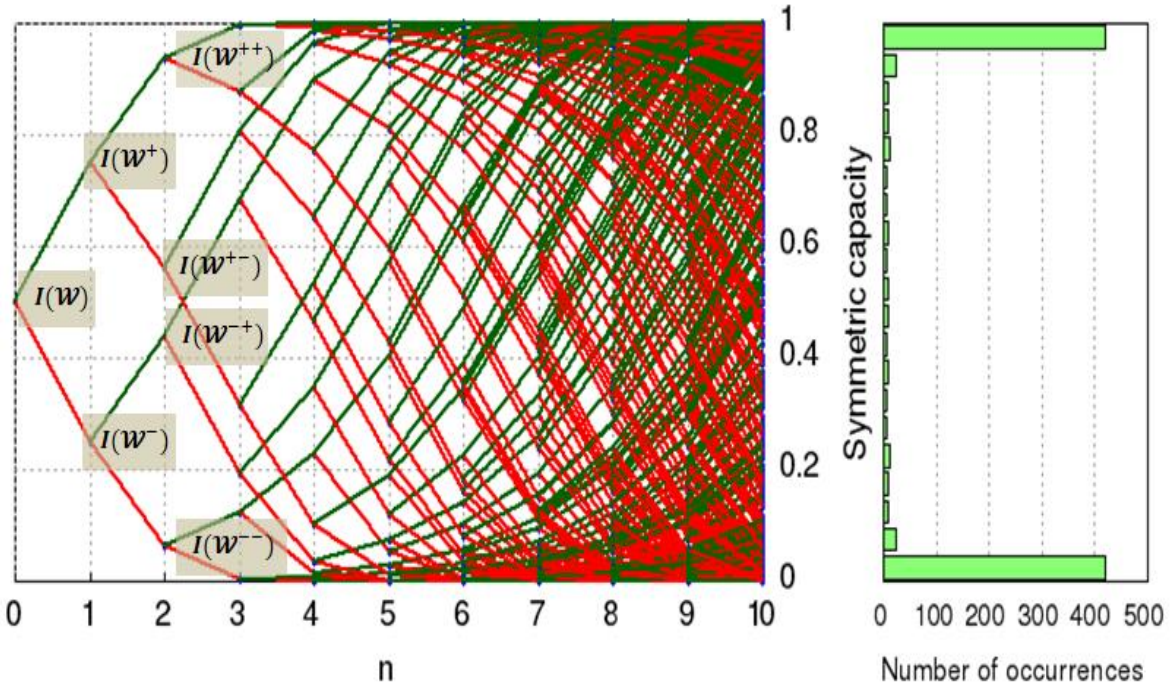


(a) **(b)**

Figure (2.6): The New Binary-Input Channels ($\mathcal{W}_2^{(1)}$&$\mathcal{W}_2^{(2)}$) or ($\mathcal{W}^-$&$\mathcal{W}^+$) [42].

Arikan, in his seminal paper [12], used the Bhattacharyya parameters to find the indices of reliable channels. These parameters can be estimated recursively after n-steps polarizing process, where after an i^{th}-step there are 2^i bit channels with reliabilities $Z^{(i)} = \{z_l^i : l = 0,1,2,\dots,2^i - 1\}$. Figure (2.8) shows this n-steps polarizing process. The initial value of the Bhattacharyya parameter $Z^{(0)} = e^{(-\frac{E_b}{N_0} * \Re)}$, where $Z^{(0)} = e^{(\frac{-1}{2\sigma^2})}$ for the AWGN channel. For the AWGN channel with $\sigma = 0.683$, the construction of PC(N=8, K=4) is illustrated in Figure (2.8), where $\mathcal{F}^c = \{4,6,7,8\}$ is the indices vector of the

more reliable channels.



(a) **(b)**

Figure (2.7): The Martingale/Histogram Plot of the Channel Capacity $I\left(W_N^{(l)}\right)$: $l = 0,1,\dots,N = 2^n, for\ n = 10$ [44].

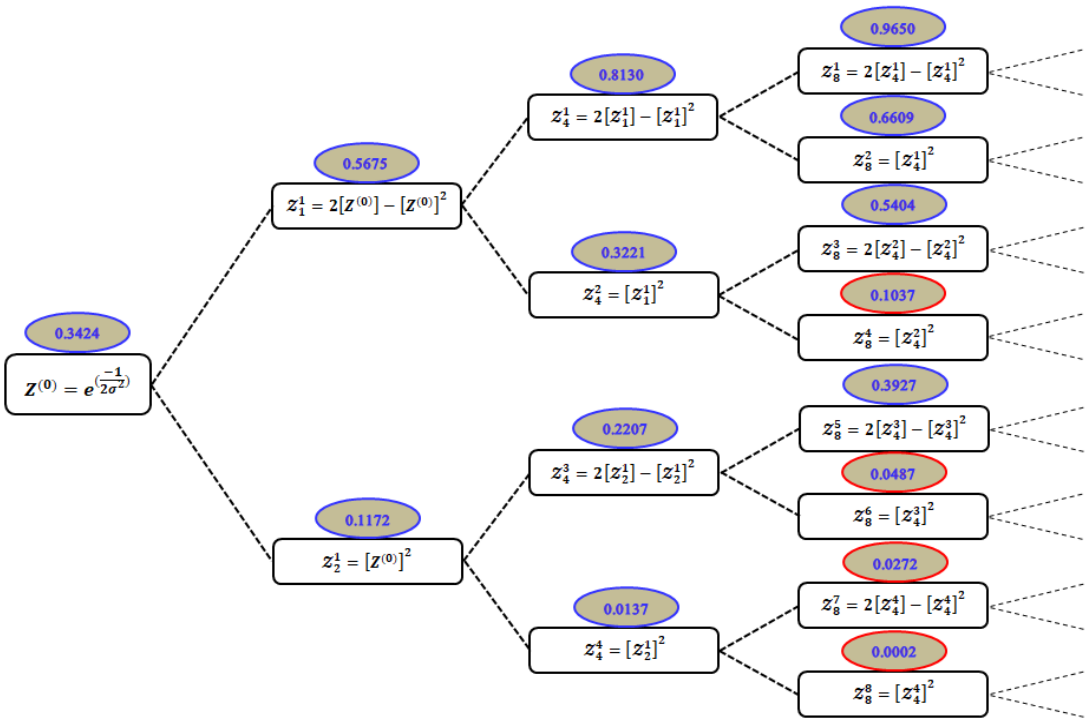


Figure (2.8): The Construction Process of Polar Codes Based on Bhattacharyya Parameters.

2.6. Polar Codes$(N, K, \mathcal{F})$

As a linear code, the Polar codes can be separated into two types: Non-Systematic and Systematic polar codes. In the first, the information bits don't appear in the codewords, while in the latter, they appear at specific locations. Both codes are illustrated below:

2.6.1. Non-Systematic Polar Codes (NSPCs)

A polar code can be defined by three parameters: Code-Length (N), a frozen subset ($\mathcal{F}$), and an information subset length (K). Where $N = 2^n, n > 0$, and $\mathcal{F} \subset \{1,2,\dots,N\}$. If it is assumed that the source bits are $S = (s_1, s_2, \dots\dots, s_N)$ and the user information is $S_{\mathcal{F}^c}$ $(i.e., S_{\mathcal{F}^c} \subset S)$, the codeword C can be generated as[46]–[50]:

$$C = S * G = S * (\mathbb{B}_N * \mathrm{F}^{\otimes \mathrm{n}}) \tag{2.13}$$

Where $\mathbb{B}_N$ is $N \times N$ bit-reversal-permutation-matrix, G is the generator matrix, $\otimes$n is the n^{th} Kronecker product, and $\mathrm{F}^{\otimes \mathrm{n}}$ is an n^{th} order Hadamard matrix as follows [49], [50]:

$$F^{\otimes n} = \begin{bmatrix} F^{\otimes(n-1)} & 0 \\ F^{\otimes(n-1)} & F^{\otimes(n-1)} \end{bmatrix}, where\ \mathrm{F}^{\otimes 1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{2.14}$$

and

$$\mathbb{B}_N = \mathbb{R}_N \left(I_2 \otimes \mathbb{B}_{\frac{N}{2}} \right), where \quad \mathbb{B}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.15}$$

For example:

$$\mathbb{B}_4 = \mathbb{R}_4 (I_2 \otimes \mathbb{B}_2) = \mathbb{R}_4 * \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

There are two encoding models for polar codes: The first assumes

$G = \mathbb{B}_N * \mathrm{F}^{\otimes n}$ as shown in Figure (2.4) for G_8 and the second assumes $G = \mathrm{F}^{\otimes n}$ as shown in Figure (2.9). The second model requires a reverse-indexed input vector instead of a normal-indexed one. Algorithm-1 in appendix-I shows the necessary implementation steps of the Non-Systematic polar encoder using the second model.

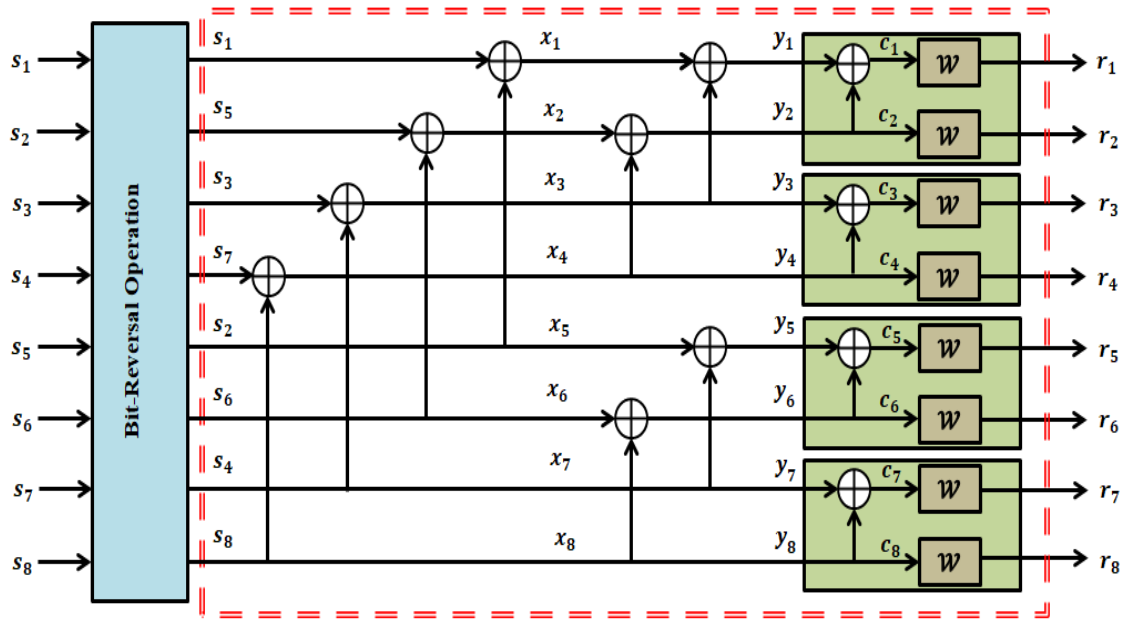


Figure (2.9): Encoder Structure of a Polar Code with N=4 [42].

2.6.2. Systematic Polar Codes (SPCs)

Since the source vector (S) can be separated into two sub-vectors $(S_{\mathcal{F}^c})$ and $(S_{\mathcal{F}})$, the encoded vector (C) in Equation (2.13) can be rewritten as [51]:

$$C = S_{\mathcal{F}^c} * G_{\mathcal{F}^c} + S_{\mathcal{F}} * G_{\mathcal{F}} \qquad (2.16)$$

Where $\mathcal{F} = \{1,2, \dots \dots, N\}/\mathcal{F}^c$, $G_{\mathcal{F}}$ and $G_{\mathcal{F}^c}$ are sub-matrices of the original generator matrix (G) with row-indices corresponding to $\mathcal{F}$ and $\mathcal{F}^c$, respectively.

The systematic polar code can be constructed based on the polar code in [51]. Assume that the subsets (λ^c) and (λ) denote the indexes of user bits and parity bits of the output systematic polar encoded vector (C). So Equation (2.16) can be rewritten as:

$$C_{\lambda^c} = S_{\mathcal{F}^c} * G_{\mathcal{F}^c\,\lambda^c} + S_{\mathcal{F}} * G_{\mathcal{F}\,\lambda^c} \quad (2.17)$$

$$C_{\lambda} = S_{\mathcal{F}^c} * G_{\mathcal{F}^c\,\lambda} + S_{\mathcal{F}} * G_{\mathcal{F}\,\lambda} \quad (2.18)$$

Where $G_{\mathcal{F}^c\,\lambda^c}$ is a sub-matrix of G with row and column indexes in subsets ($\mathcal{F}^c$) and (λ^c) respectively, similarly for the remaining sub-matrices. Since the user bits (C_{λ^c}) of the systematic polar code is known, and the frozen bits ($S_{\mathcal{F}}$) are set to zero and known in both the encoder and decoder, the input bits ($S_{\mathcal{F}^c}$) of the systematic polar code in Equation (2.17) can be determined as [51]:

$$S_{\mathcal{F}^c} = (C_{\lambda^c} - S_{\mathcal{F}} * G_{\mathcal{F}\,\lambda^c}) * (G_{\mathcal{F}^c\,\lambda^c})^{-1} \quad (2.19)$$

So Equation (2.18) can be rewritten as follows to calculate the parity bits (C_{λ}):

$$C_{\lambda} = [(C_{\lambda^c}) * (G_{\mathcal{F}^c\,\lambda^c})^{-1}] * G_{\mathcal{F}^c\,\lambda} \quad (2.20)$$

The systematic polar code is constructed using Equations (2.19) and (2.20), and Algorithm-3 in appendix-I describes the procedure for implementing a systematic polar encoder.

2.6.3. The Factor Graph of Polar Codes

The Factor graph is a creative method to construct a polar code graphically. Polar code factor graph representation comprises $N(n+1)$ individual nodes separated into $(n+1)$ columns labeled by $p \in \{0,1,..,n\}$. Each of these columns consists of 2^p sets represented by $\emptyset \in \{0,1,..,2^p-1\}$. Each set in a factor graph comprises 2^{n-p} nodes indexed by $\omega \in \{0,1,..,2^{n-p}-1\}$. The factor graph sets are represented by $\phi_p^{\emptyset}$ to define the group of nodes at a p-column in a $\emptyset$-set. The factor graph representation of a polar code consists of $2N-1$ such $\phi_p^{\emptyset}$ groups [52].

Figure (2.10) shows the factor graph representation of polar code ($N=8,\ K=4$) with four columns ($p=0,1,2,3$), fifteen sets ($\{\phi_0^0\}$,

$\{\phi_1^0, \phi_1^1\}$, $\{\phi_2^0, \dots, \phi_2^3\}$, $\{\phi_3^0, \phi_3^1 \dots \phi_3^6, \phi_3^7\}$), and 32 nodes. These nodes can be represented by the trio $(p, \emptyset, \omega)$. For example, the first node at the top left side can be indexed by $(p = 0, \emptyset = 0, \omega = 0)$, the third node in the second column by $(p = 1, \emptyset = 0, \omega = 2)$, and so on.

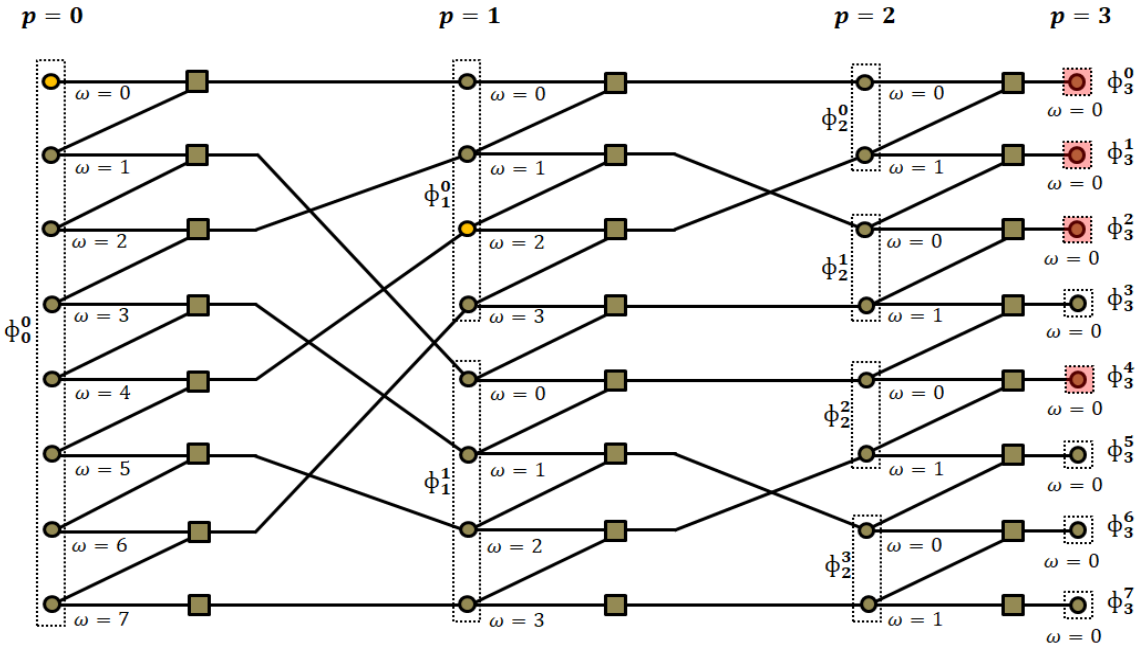


Figure (2.10): The Factor Graph of a Polar Code $(N = 8, K = 4, \mathcal{F} = \{0,1,2,4\})$ [52].

2.6.4. Successive Cancellation Decoder (SCD)

Arikan proposed the SC decoder as the native polar decoding algorithm in 2009 [12]. For polar code $(N, K, \mathcal{F})$, a decoding tree of an n-depth can be used to illustrate the decoding process, where $n = log_2 N$. Figures (2.11) and (2.12) show the decoder structure and 3-depth tree for polar code (8,4), respectively. In the decoding tree and at the t-stage, there is (2^t) nodes. Each node receives and propagates $(\alpha^t_{z:z+2^{n-t}-1})$ and $(\beta^t_{z:z+2^{n-t}-1})$ vectors, respectively, from and to the parent node, where $0 \leq t \leq n$. The received soft information $\alpha^t_{z:z+2^{n-t}-1}$ and the propagated hard information $\beta^t_{z:z+2^{n-t}-1}$ consist of 2^{n-t} values as shown in Figure (2.12) (e.g., the first node of the second stage receives four LLRs $\alpha^1_{0:3}$ $[\alpha^1_0, \alpha^1_1, \alpha^1_2, \alpha^1_3]$ from the root node and propagates four binary bits $\beta^1_{0:3}$ $[\beta^1_0, \beta^1_1, \beta^1_2, \beta^1_3]$). To compute the LLRs

$(\alpha_l^{t+1}: 0 \le l \le 2^n - 1)$ of the left child of the t-node, the following expression must be applied:

$$\alpha_l^{t+1} = f\left(\alpha_l^t, \alpha_{l+2^{n-(t+1)}}^t\right) \qquad (2.21)$$

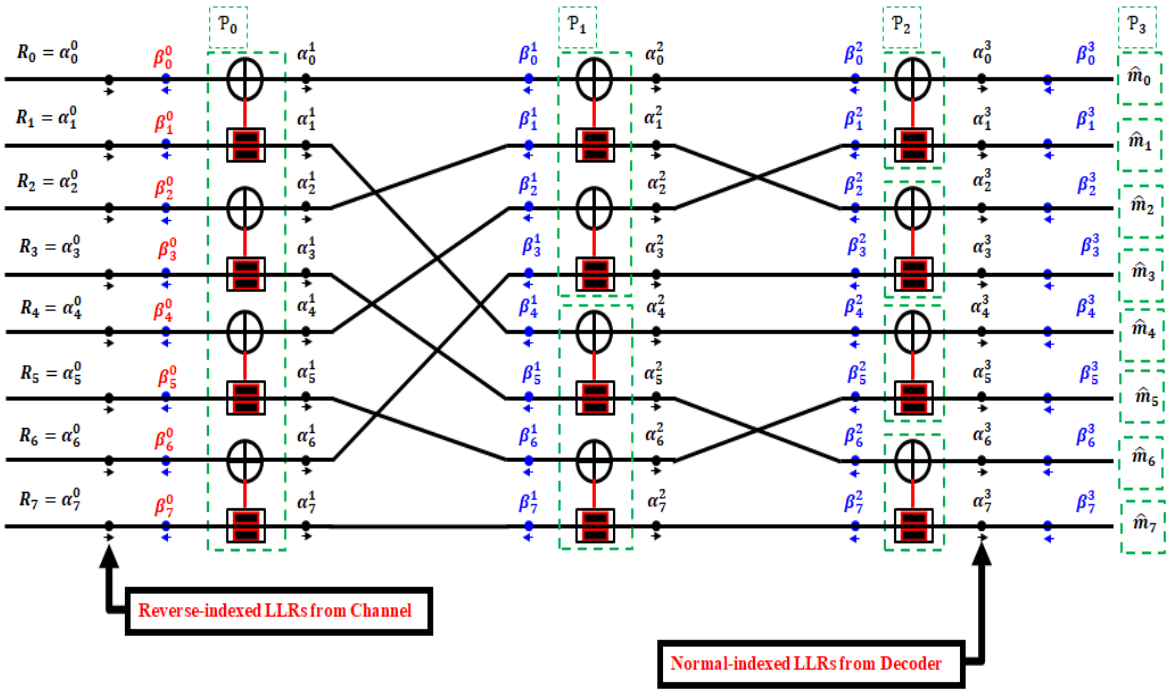


Figure (2.11): The Factor Graph of the Polar Code with N=8.

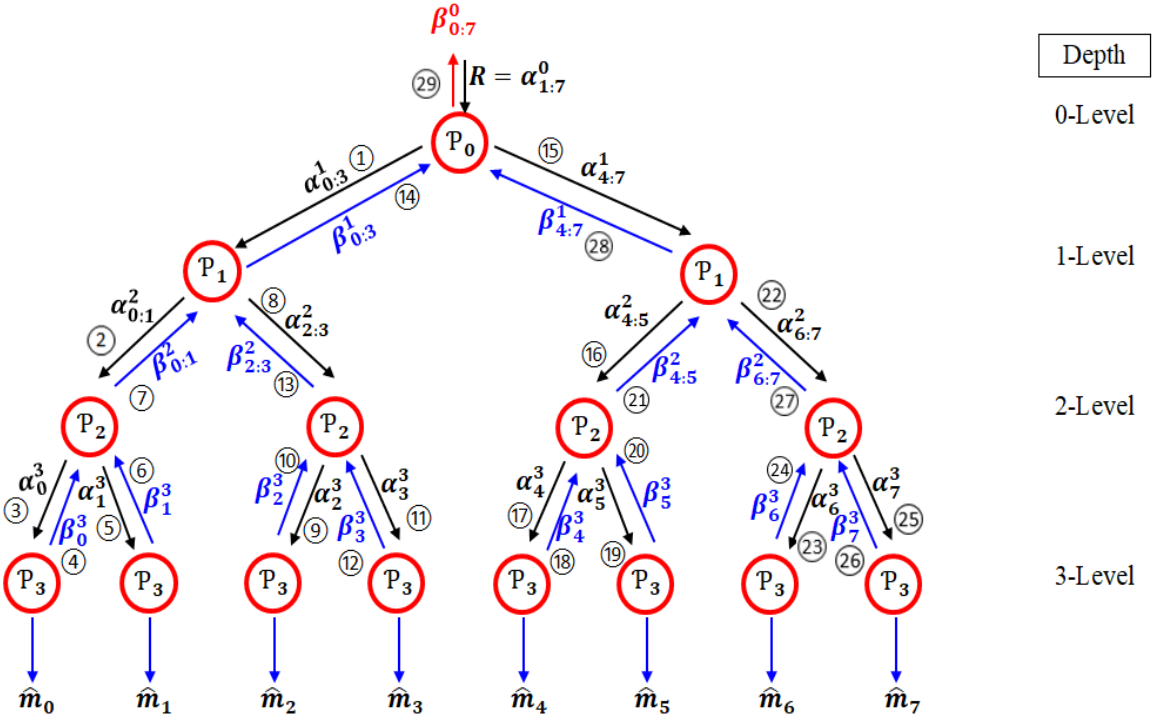


Figure (2.12): Decoding Tree of Polar Code with N=8.

While the LLRs ($\alpha_l^{t+1}: 0 \le l \le 2^n - 1$) of the right child are computed after estimating the hard information ($\beta_l^{t+1}: 0 \le l \le 2^n - 1$) of the left child as [53]–[56]:

$$\alpha_l^{t+1} = g\left(\alpha_{l-2^{n-(t+1)}}^{t}, \alpha_l^t, \beta_{l-2^{n-(t+1)}}^{t+1}\right) \tag{2.22}$$

The message bits ($\hat{m}_i: 0 \le i \le 2^n - 1$) are estimated based on the values of LLRs α_i^n, where these LLRs can be computed as:

$$\alpha_i^n = \log\left(\frac{Pr(R, \hat{m}_{0:i-1} | m_i = 0)}{Pr(R, \hat{m}_{0:i-1} | m_i = 1)}\right) \tag{2.23}$$

Where R is the received Log-Likelihood Ratio (LLR) vector of the transmitted codewords through a channel. Based on the following rule, the message bits $\hat{m}_i$ can be computed as:

$$\hat{m}_i = \begin{cases} 0 & if\, \alpha_i^n \ge 0 \\ 1 & otherwise \end{cases} \tag{2.24}$$

The estimated value of the frozen bit is $\hat{m}_i$=0, regardless of the value of the α_i^n.

The feedback hard-information ($\beta_l^{t+1}: 0 \le l \le 2^n - 1$) can be computed by a linear combination of some decoded hard bits that were computed previously; as illustrated below:

$$\beta_l^{t+1} = \beta_l^{t+2} \oplus \beta_{l+2^{n-(t+2)}}^{t+2} \tag{2.25}$$

$$\beta_{l+2^{n-(t+2)}}^{t+1} = \beta_{l+2^{n-(t+2)}}^{t+2} \tag{2.26}$$

Where $\beta_l^n = \hat{m}_l$.

The functions $\boldsymbol{f}$ and $\boldsymbol{g}$ are defined as follows [53]–[56]:

$$f(a, b) = sign(a * b) * \min(|a|, |b|) \tag{2.27}$$

$$g(a, b, \beta) = \left[(-1)^{1-2\beta} * a\right] + b \tag{2.28}$$

The decoding process needs an extra step for systematic polar codes to obtain the systematic information bits. To estimate the information bits, the outcomes of the decoder ($\widehat{m_l}$) must be re-encoded (i.e., multiplied by generator matrix $[G]$) to obtain the systematic information (β_l^0). This extra step can be avoided by propagating the hard decisions ($\widehat{m_l}$) in the decoding tree.

After the last iteration of the SC decoder, the resulting matrices α and β contain information about noisy received LLRs, recovered encoded bits, and estimated message bits. Where $\beta(i_0^{N-1}, 0)$ contains hard reversed-indexed information for the recovered encoded bits; while, $\alpha(i_0^{N-1}, n)$ contains LLRs for message bits that correspond to the hard decisions at $\beta(i_0^{N-1}, n)$. Noisy received LLRs at $\alpha(i_0^{N-1}, 0)$ are not updated throughout the decoding process. For example, if message bits $m = [0, 0, 1, 0]$ are encoded with a polar encoder $(N = 8, K = 4)$, then the reversed-indexed coded bits are $C = [1, 1, 1, \mathbf{1}, 0, 0, 0, 0]$ (refer to Algorithm-1). Noisy received LLRs R= [-1.27, -0.234, -1.884, **0.26**, 1.311, 2.179, 1.273, 2.051] are received through AWGN channel with $SNR = 2dB$. In the beginning, $\alpha(i_0^7, 0)$ is initialized with noisy received data (i.e., R vector**).** After the last iteration, the SC decoder has the following information: $\beta(i_0^7, 0) = [1, 1, 1, 1, 0, 0, 0, 0]$, $\beta(i_0^7, 3) = [0, 0, 0, 0, 0, 0, 1, 0]$, $\alpha(i_0^7, 3) = [-0.234, 1.038, -0.025, 2.56, 1.50, 4.83, -3.13, 9.94]$. Although the first and third decoded LLRs are negative values, the estimated hard bits at these locations are zeros because the first and third bits are frozen bits.

For systematic polar code, the output of a polar encoder $(N = 8, K = 5)$ is $C = [0, 0, 1, \mathbf{1}, 1, \mathbf{1}, \mathbf{1}, \mathbf{1}]$ (refer to Algorithm-3), if $d = [\mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}]$ is the input, where the vector C in the reversed-indexed form. Noisy received LLRs $R = [1.388, \mathbf{-1.36}, -0.50, -2.52, -1.76, -0.55, -0.91, -0.829]$ are received through the same channel above. The SC decoder has the following

information after finishing the decoding process: $\beta(i_0^7, 0) = [0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]$, $\beta(i_0^7, 3) = [0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1]$, $\alpha(i_0^7, 3) = [-0.50,\ 0.051,\ -0.857,\ 0.5287,\ -0.026,\ 1.77,\ 3.057,\ -7.12]$. If the indices of $\beta(i_0^7, 0)$ are reversed, the result will be $\bar{\beta}(i_0^7, 3) = [0, 1, 1, 1, 0, 1, 1, 1]$, which matches the systematic encoded bits.

2.6.5. Soft Cancellation (SCAN) Algorithm

Based on the SC's schedule, the SCAN decoder massively reduces the decoding complexity compared to the flooding schedule of the belief-propagation decoder [57]. The SCAN algorithm propagates the soft information in both directions (i.e., from right to left and vice versa). Although the propagation of soft information in both directions somewhat increases the complexity and latency compared to the SC decoder, it improves the performance of the decoding process. The SC decoder's polar factor graph in Figure (2.11) can be used with the SCAN decoder. In the beginning, the received noisy LLRs from the channel are assigned to the vector (α_i^0) at left; where $0 \leq i \leq 2^n - 1$. The knowledge of frozen bits' locations can be exploited by the SCAN decoder to initialize the vector β_i^n by a-prior information, where the frozen locations are filled by infinity value because these locations correspond to zero bits. Whereas the locations of information may be zero or one equiprobably, so they are filled with zero LLRs (see Equation (2.23)) as follows [57]–[59]:

$$\beta_i^n = \begin{cases} \infty & if\ i \in \mathcal{F} \\ 0 & otherwise \end{cases} \tag{2.29}$$

While computing α_i^n with $\alpha_{0:i-1}^n$ which has already been computed; α_j^t (where $0 \leq t \leq n, 0 \leq j \leq 2^n - 1$) is partially populated from the top left to the bottom right. This partially computed $\boldsymbol{\alpha}$ matrix can be used as a-prior information to update β from the top right to the bottom left. At the final step,

with a fully populated α_j^t (when $t = n$ and $j = 2^n - 1$), all nodes of the factor graph contain soft extrinsic information (LLRs), which correspond to the values of the fully populated β matrix. Initially, all values of β are zeros except β_i^n, where $0 \leq i \leq 2^n - 1$, so there is no extrinsic information that the SCAN decoder can use at the first iteration. Whereas at the second iteration and other successive iterations, the SCAN decoder exploits the extrinsic information computed with previous iterations [17]. Algorithm-4 in appendix-I describes the procedure for implementing a SCAN decoder.

The updating process can be described as follows based on the factor graph of the basic polar kernel in Figure (2.13) [61][62]:

$$\boldsymbol{\alpha_0^{t-1}} = \boldsymbol{f}\left(\boldsymbol{\alpha_0^t}, \boldsymbol{\alpha_1^t} + \boldsymbol{\beta_1^{t-1}}\right) \tag{2.30}$$

$$\boldsymbol{\alpha_1^{t-1}} = \boldsymbol{\alpha_1^t} + \boldsymbol{f}\left(\boldsymbol{\alpha_0^t}, \boldsymbol{\beta_0^{t-1}}\right) \tag{2.31}$$

$$\boldsymbol{\beta_0^t} = \boldsymbol{f}\left(\boldsymbol{\beta_0^{t-1}}, \boldsymbol{\beta_1^{t-1}} + \boldsymbol{\alpha_1^t}\right) \tag{2.32}$$

$$\boldsymbol{\beta_1^t} = \boldsymbol{\beta_1^{t-1}} + \boldsymbol{f}\left(\boldsymbol{\beta_0^{t-1}}, \boldsymbol{\alpha_0^t}\right) \tag{2.33}$$

Where $\boldsymbol{f}$ is the box-plus operator ($\boxplus$) as follows:

$$x \boxplus y \triangleq \frac{1+e^{x+y}}{e^x+e^y} = 2tanh^{-1}\left[\tanh\left(\frac{x}{2}\right) \times \tanh\left(\frac{y}{2}\right)\right] \tag{2.34}$$

The previous form of box-plus requires more memory resources and execution time, so the hardware-friendly approximation in Equation (2.27) can be used instead of the previous equation. Where the $f(x, y)$ in Equation (2.27) is the expression $x \boxplus y$ [57], [61]–[63].

For the same example mentioned in the previous section with message bit $m = [0, 0, 1, 0]$, the output of the SCAN decoder (Algorithm-4) after five iterations consists of the following outcomes: $\alpha(i_0^7, 3) =$ [1.27, 2.581, 2.514, 5.098, 1.504, 4.828, -3.128, 4.828] and $\beta(i_0^7, 0) =$[-1.858, -2.894, -1.244,-3.388, 3.683, 2.815, 3.555, 2.777]. The calculated message-LLRs can be

extracted from the first vector $\alpha(i_0^7, 3)$ which are [5.098, 4.828, -3.128, 4.828]. The hard decisions for these LLRs are [0, 0, 1, 0], which match the original message bits. The other output vector is $\beta(i_0^7, 0)$ which contains LLRs corresponding to the reverse-indexed coded bits vector $C = [1,1,1,1,0,0,0,0]$.

For systematic polar codes, the message bits appear in the coded vector. Therefore, the resulted-extrinsic LLRs of the coded bits $\beta(i_0^7, 0)$ can be exploited by the decoder to decode the received LLRs after the last iteration of the SCAN. For the same systematic example illustrated previously for the systematic SC decoder, the noisy LLRs $R = [1.388, \mathbf{-1.36}, -0.50, -2.52, -1.76, -0.55, -0.91, -0.829]$ received from the channel are allocated to $\alpha(i_0^7, 0)$. Finally, if the values of $\beta(i_0^7, 0)$ are summed with $\alpha(i_0^7, 0)$ and reversed, the estimated hard decisions for the resulted values are $[0, 1, 1, 1, 0, 1, 1, 1]$, which match the normal-indexed systematic encoded bits.

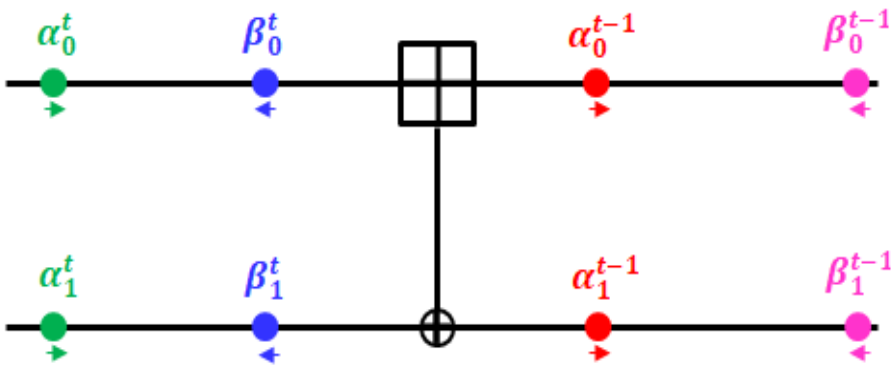


Figure (2.13): The Factor Graph of the Basic Polar Kernel.

2.7. Convolutional Codes

Convolutional codes are linear codes that handle the input sequence as a bits stream rather than discrete blocks, as in block codes. Convolutional codes are preferred in practical applications over bock codes due to their excellent performance and use of a soft-decision decoding algorithm. The convolutional code receives K bit-symbols at a time and produces N bit-symbols, so the

code rate $\Re = \frac{K}{N}$.

2.7.1. Convolutional Encoder

The encoder of convolutional codes consists of several digital elements (i.e., D-registers) and XOR operations. The output of the convolutional encoder depends on the past and present inputs. Thus, the D-registers are used to save past inputs. The message and encoded bits relate to each other by a bijective relationship. Therefore, every encoded bit is associated with a single message bit (i.e., one-to-one). The convolutional encoders are classified into two types based on the generator matrix: Recursive and Non-recursive codes [64].

The recursive convolutional codes have a generator matrix comprising only polynomials entries (e.g., $G = [1 + x^2 + x^3 \quad 1 + x + x^3]$). The encoders of these codes also can be called feedforward encoders and Finite-Impulse-Response (FIR) encoders. The encoded output ($C = [c_0, c_1, \dots, c_N]$) of the message input $d = [d_0, d_1, \dots, d_k]$ can be expressed as follows [64]–[66]:

$$C = d.G \tag{2.35}$$

The other type is the Non-recursive convolutional codes. The generator matrix of these codes contains rational functions rather than just polynomials. A Non-recursive encoder is known as a feedback encoder or Infine-Impulse-Response (IIR) encoder (e.g., $G = \left[\frac{1+x}{1+x+x^3}\right]$). By using Equation (2.35), a Non-recursive encoder has an infinite-bit-stream as encoded output instead of the finite-bit-stream due to the rational functions of the generator matrix [64], [65].

Both previously mentioned types can be systematic or non-systematic codes. The generator matrix of the systematic one contains a K-unity matrix. In contrast, the non-systematic generator matrix has no K-unity matrix, where

K is the number of the input stream bits. Figure (2.14) shows the encoder's structure of the Recursive Systematic Code (RSC) with $G = \left[1 \quad \frac{1+x+x^3}{1+x^2+x^3}\right]$.

The polynomials of the generator matrix are commonly expressed by a binary vector representing the existence of the polynomial coefficients or by the equivalent octal form. For example, $G = [1 + x^2 + x^3 \quad 1 + x + x^3]$ can be expressed by $g^1 = [1011]$ and $g^2 = [1101]$ or $g^1 = (13)_8$ and $g^2 = (15)_8$.

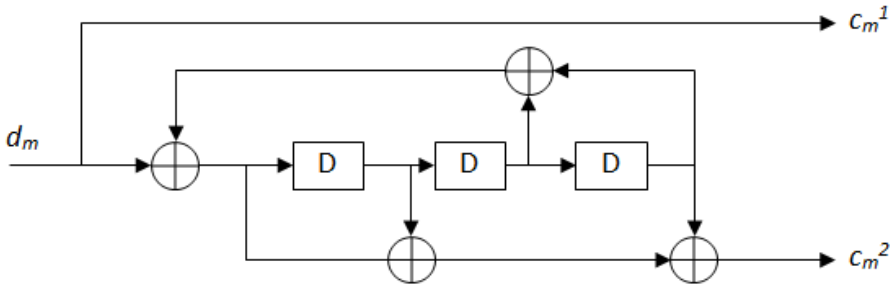


Figure (2.14): The Encoder Structure of $RSC(13,15)_8$ with $\mathfrak{R}$=1/2.

2.7.2. LogMAP Algorithm

This algorithm is the modified version of the MAP algorithm (i.e., the BCJR algorithm). The BCJR algorithm uses probability values rather than Log-Likelihood Ratios (LLRs) and requires many multiplication operations. Also, the BCJR algorithm requires a normalization operation to calculate the final probability metrics. These issues make implementing this algorithm more complex and require more resources and latency, especially for codes with large codeword lengths. The estimation of metrics in logarithmic form can be used instead of probability metrics to overcome the high complexity of the BCJR algorithm. Therefore LogMap algorithm has less complexity compared to the BCJR algorithm. The LogMap algorithm exploits the logarithmic representation to use the addition operations rather than the multiplication operations [64], [67].

All received information values (R) may be used to decode the received bit at time $\boldsymbol{t}$ since the output of the convolutional encoder at a time $\boldsymbol{t}$ depends

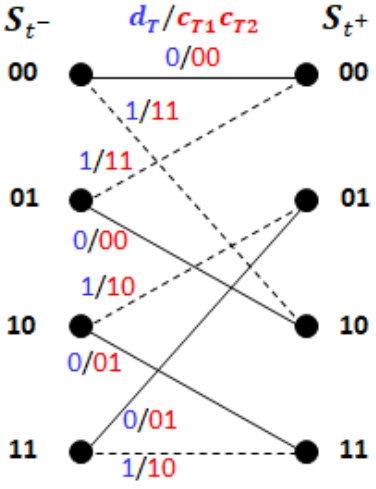
on the outputs at earlier times and impacts the outputs at later times. The LogMap algorithm uses the forward and backward passes to incorporate the information of bits transmitted before and after time $\boldsymbol{t}$.

Since the inputs of the convolutional encoder are the message bits, the received bits can be decoded by tracking which state transition occurred and then finding which bits were sent before the channel. The set (S^+) contains the state transitions that correspond to 1-input bits, while the set (S^-) contains the state transitions that correspond to 0-input bits [67]. For example, the encoder of RSC(5,7) in Figure (2.15) has the following:

$$S^-(S_{t^-}, S_{t^+}) = \{(S_0, S_0), (S_1, S_2), (S_2, S_3), (S_3, S_1)\}$$

$$S^+(S_{t^-}, S_{t^+}) = \{(S_0, S_2), (S_1, S_0), (S_2, S_1), (S_3, S_3)\}$$

Where: $0 \leq t^- < 2^m, 0 \leq t^+ < 2^m$, and m =number of D-registers.



Current state S_{t^-}		Input d_T	Next State S_{t^+}		Output c_T^1	Output c_T^2
S_0	00	0	S_0	00	0	0
S_0	00	1	S_2	10	1	1
S_1	01	0	S_2	10	0	0
S_1	01	1	S_0	00	1	1
S_2	10	0	S_3	11	0	1
S_2	10	1	S_1	01	1	0
S_3	11	0	S_1	01	0	1
S_3	11	1	S_3	11	1	0

Figure (2.15): The Trellis Diagram and the State Transition Table of $RSC(7,5)_8$ with $\mathfrak{R}$=1/2.

The procedure of the LogMap algorithm can be summarized as follows [64], [67]:

i) The inputs of the LogMap algorithm are a posteriori probabilities (APPs) of received information and APPs for the message bits (a-prior information).

ii) Calculate the log metrics (Γ^T) for all given T where $T = 1,2,\dots,k$.

$$\Gamma^T[S_{t^-}, S_{t^+}] = \ln\left[p\left(d^T = d_{t^-,t^+}\right)\right] + \ln\left[p\left(r^T|c_{t^-,t^+}\right)\right] \quad (2.36)$$

Where:

- $\ln\left[p\left(d^T = d_{t^-,t^+}\right)\right]$: Is the APP of the message-bits for all T. where: $p\left(d^t = d_{t^-,t^+}\right)$**:** is the a-prior source probabilities.
- $\ln\left[p\left(r^T|c_{t^-,t^+}\right)\right]$: Is the APP of the channel transition probabilities. For the AWGN channel:

$$\ln\left[p\left(r^T|c_{t^-,t^+}\right)\right] = \ln\left[\left(\sqrt{\frac{1}{2\pi\sigma^2}}\right)^q e^{\left(-\frac{1}{2\sigma^2}\|r^T-c^T\|^2\right)}\right] \quad (2.37)$$

$$\ln\left[p\left(r^T|c_{t^-,t^+}\right)\right] = \left(-\frac{1}{2\sigma^2}\|r^T - c^T\|^2\right) + \ln\left(\sqrt{\frac{1}{2\pi\sigma^2}}\right)^q \quad (2.38)$$

$\|r^T - c^T\|$ is the euclidean distance between $r^T$and c^T. For BPSK, $q = 2$.

iii) Forward Recursion: Calculate the log probability $\mathcal{A}^T(S_{t^-})$, which is the logarithmic representation of the probability that the encoder is in the state (S_{t^-}) at time T based on the received information at the time before T.

$$\mathcal{A}^T(S_{t^-}) = \ln \sum_{j=0}^{2^m-1} e^{\mathcal{A}^{T-1}(S_j)+\Gamma^T(S_j,S_{t^-})} \quad (2.39)$$

$$\mathcal{A}^T(S_{t^-}) = \overline{\underset{j}{\mathrm{MAX}}}\left[\mathcal{A}^{T-1}\left(S_j\right) + \Gamma^T\left(S_j, S_{t^-}\right)\right] \quad (2.40)$$

$$\overline{MAX}[x,y] = \ln[e^x + e^y] \quad (2.41)$$

Where: $\mathcal{A}^0(S_0) = 0$, $T = 1,2,\dots,k$, and $t^- = 0,1,\dots,2^m - 1$.

iv) Backward Recursion: Calculate the log probability $\mathcal{B}^T(S_{t^+})$, which is the logarithmic representation of the probability that the encoder is in the state (S_{t^+}) at time T based on the received information at a time after T.

$$\mathcal{B}^{T}(S_{t^+}) = \ln \sum_{j=0}^{2^{m-1}} e^{\mathcal{B}^{T+1}(S_j)+\Gamma^{T+1}(S_{t^+},S_j)} \quad (2.42)$$

$$\mathcal{B}^{T}(S_{t^+}) = \overline{\underset{j}{\mathrm{MAX}}}\left[\mathcal{B}^{T+1}(S_j) + \Gamma^{T+1}(S_{t^+}, S_j)\right] \quad (2.43)$$

Where: $\mathcal{B}^{k+1}(S_0) = 0$, $T = k, k-1, \ldots 2,1$, and $t^+ = 0,1, \ldots, 2^m - 1$.

v) Calculate the state transition metrics ($\mu^{T}(S_{t^-}, S_{t^+})$) for both $S^-$and S^+ as follows:

$$\mu^{T}(S_{t^-}, S_{t^+}) = \mathcal{A}^{T-1}(S_{t^-}) + \Gamma^{T}[S_{t^-}, S_{t^+}] + \mathcal{B}^{T}(S_{t^+}) \quad (2.44)$$

Where: $T = 1,2, \ldots, k$.

vi) Calculate the output LLRs ($\mathcal{L}_T$) of the LogMap algorithm as follows:

$$\mathcal{L}_T = \mathcal{L}(d^T|r^T) = \ln \sum_{S^-} e^{\mu^{T}(S_{t^-},S_{t^+})} + \ln \sum_{S^+} e^{\mu^{T}(S_{t^-},S_{t^+})} \quad (2.45)$$

$$\mathcal{L}_T = \mathcal{L}(d^T|r^T) = \overline{\underset{S^-}{\mathrm{MAX}}}[\mu^{T}(S_{t^-}, S_{t^+})] - \overline{\underset{S^+}{\mathrm{MAX}}}[\mu^{T}(S_{t^-}, S_{t^+})] \quad (2.46)$$

Where $\mathcal{L}(d^T|r^T) = \ln \frac{p(d^T=0|r^T)}{p(d^T=1|r^T)}$.

vii) Calculate the hard output decision of the LogMap algorithm as follows:

$$\hat{d}_T = \begin{cases} 0 & if\ \mathcal{L}_T \geq 0 \\ 1 & if\ \mathcal{L}_T < 0 \end{cases} \quad (2.47)$$

The LogMap decoder can handle the LLRs inputs and outputs to facilitate the decoding process, as shown in Algorithm-12 (refer to Appendix-I).

2.7.3. Soft Output Viterbi Algorithm (SOVA)

The SOVA algorithm is an alternative to the highly complex MAP decoding family. This algorithm has two differences from the traditional Viterbi Algorithm (VA): First, the Soft Output Viterbi Algorithm requires a-prior probabilities about the message bits to calculate modified path metrics. Secondly, SOVA can use its soft output to indicate the output decision

reliability. This makes the SOVA decoder suitable for an iterative turbo decoding algorithm [65].

The low complexity of the Viterbi family came from the fact that the Viterbi family doesn't require calculating the full path probabilities for each possible codeword. The best path is chosen at each trellis node as a survivor path, and the other paths are ignored. Thus the complexity of the Viterbi family is drastically reduced compared to the MAP family. For SOVA, the metrics of the Maximum Likelihood (ML) path are used in addition to the metrics of the competition paths at each node of the ML path to calculate the path metric difference (Δ_i), which is the output LLR of the SOVA, as shown below.

The SOVA procedure can be summarized as follows [65], [68]–[70]:

i) The inputs of the SOVA algorithm are the soft received information and APPs for the message bits (a-prior information). These inputs can be in an LLR expression or a log expression.

ii) Calculating the LLR branch metrics (μ^T) for all given T, where $T = 1,2,\dots,k$. The LLR branch metrics (μ^T) correspond to the log branch metrics (Γ^T) but in the LLR expression rather than the log expression.

$$\Gamma^T[S_{t^-}, S_{t^+}] = \log\left[p\left(d^T = d_{t^-,t^+}\right)\right] + \log\left[p\left(r^T | c_{t^-,t^+}\right)\right] \qquad (2.48)$$

$$\mu^T[S_{t^-}, S_{t^+}] = \frac{1}{2} d^T \varphi^T + \frac{L_c}{2} \sum_{i=1}^{n} r_i^T c_i^T \qquad (2.49)$$

Where:

$\varphi^T = \ln \frac{p(d^T=0)}{p(d^T=1)}$: Is LLR of the a-prior information of message bits.

n: The output lines' number of the convolutional encoder.

r_i^T: The LLRs of the received sequence.

L_c: The channel reliability.

$$L_c = \frac{E_b}{2\sigma^2} * 4a * \Re \tag{2.50}$$

a: The fading amplitude

E_b: The transmitted energy per bit.

iii) Calculating the LLR path metrics ($\mathcal{M}^T$) for all given T where $T = 1,2,\dots,k$.

$$\mathcal{M}_j^T = \mathcal{M}_i^{T-1} + \mu^T[S_{t^-}, S_{t^+}] \tag{2.51}$$

where: $\mathcal{M}_0^0 = 0,\ 0 \le j < 2^m$, and $0 \le i < 2^m$.

iv) Estimating the maximum likelihood (ML) path by tracking the maximum path metric ($\mathcal{M}_j^T$) from $T = k$ to $T = 0$. If there is trellis termination, the first state of the maximum likelihood path is the initial or the zero state, i.e., $ML(T = k) = S_0$.

v) Calculating the path metric difference ($\Delta_{S_j}^T$), where $0 \le j < 2^m$. As shown in Figure (2.16), if two paths $\left(\text{e.g.}, \mathcal{P}_T^{(1)}\ and\ \mathcal{P}_T^{(2)}\right)$ reach the state (S_j) at time T with the LLR path metrics $\mathcal{M}_{(j1)}^T$ and $\mathcal{M}_{(j2)}^T$, and the first path is selected as the survivor path $\left(\text{i.e.}, \mathcal{M}_{j1}^T > \mathcal{M}_{(j2)}^T\right)$, then the path metric difference (Δ_j^T) will be calculated as follows:

$$\Delta_{S_j}^T = \mathcal{M}_{(j1)}^T - \mathcal{M}_{(j2)}^T \tag{2.52}$$

vi) Calculating the a-posteriori LLR output of the SOVA ($\mathcal{L}_T$) as:

$$\mathcal{L}_T = \mathcal{L}(d^T|r^T) = d^T * \min_{\substack{i=T,T+1,\dots,T+\delta \\ d^T \neq d_i^T}} \left(\Delta_{S_{ml}}^i\right) \tag{2.53}$$

Where

- d^T: The trellis input-bit of the ML path at time T. Where, $d^T = +1$ for 0-

bit and $d^T = -1$ for 1-bit.

- d_i^T: The values of trellis input-bit at time T for paths that have merged with the ML path and discarded at one of the states that the ML path goes through.
- δ: The decoding depth, which is usually equal to $5(m+1)$.
- s_{ml}: A state through which the ML path passes.
- $\Delta_{s_{ml}}^i$: The path metric difference in the state (s_{ml}) at time i.

vii) Calculate the hard output decision of the SOVA algorithm as follows:

$$\hat{d}_T = \begin{cases} 0 & if\ \mathcal{L}_T \geq 0 \\ 1 & if\ \mathcal{L}_T < 0 \end{cases} \qquad (2.54)$$

Viterbi algorithm observation has shown that the merged paths at a state (S_j^T) typically come from the same path at some time before T. This time is estimated to equal the time of (δ) transitions through the trellis. The value of (δ) is estimated to be approximately five times the constrained length of the convolutional encoder. This value is known as the window or decoding depth [66], [71]. Thus, the SOVA calculates the LLR of a bit (d^T) at a time ($T+\delta$). Algorithm-11 in appendix-I describes the procedure for implementing a SOVA decoder with LLRs for inputs and outputs.

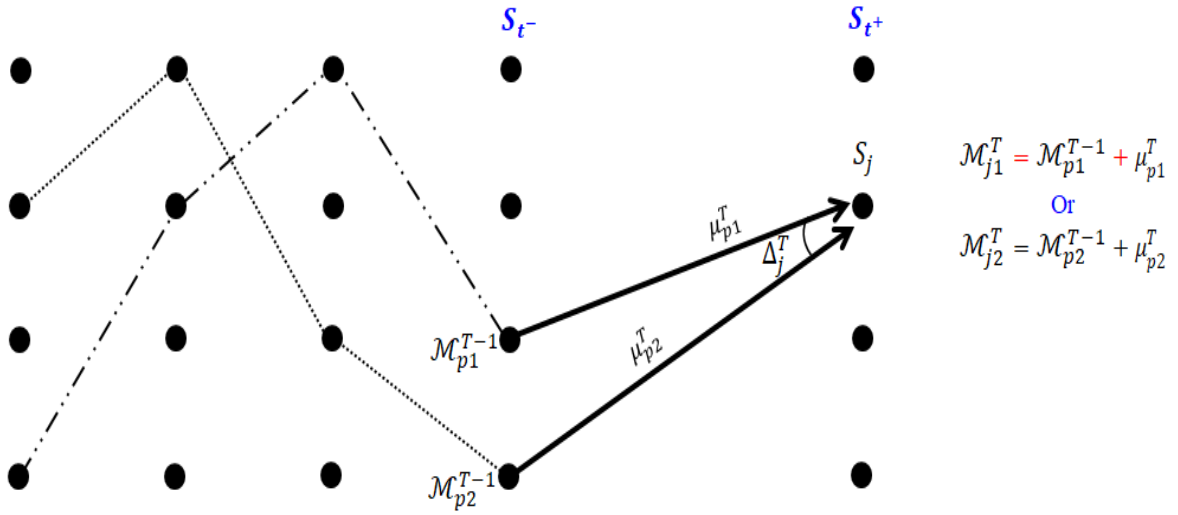


Figure (2.16): Trellis with Two Merged Paths.

Just for instance, if we consider $\delta = 4$ in Figure (2.17), then the LLR estimation of d^2 can be done at T=6. At this moment, the SOVA algorithm checks which of the following trellis input bits are different from the bit (d^2): $d_6^2, d_5^2, d_4^2, d_3^2, and\ d_2^2$. After the comparison, $d_6^2, d_4^2, and\ d_2^2$ are ignored because they are similar to the bit (d^2). We can notice that the paths of both (d_6^2) and (d_2^2) are identical to the ML path at T=2, so they are discarded. Finally, only the path metric differences associated with bits (d_5^2) and (d_3^2) are used to estimate the LLR of bit (d^2) as follows:

$$\mathcal{L}_2 = \mathcal{L}(d^2|r^2) = +1 * \min\left(\Delta_2^5, \Delta_0^3\right)$$

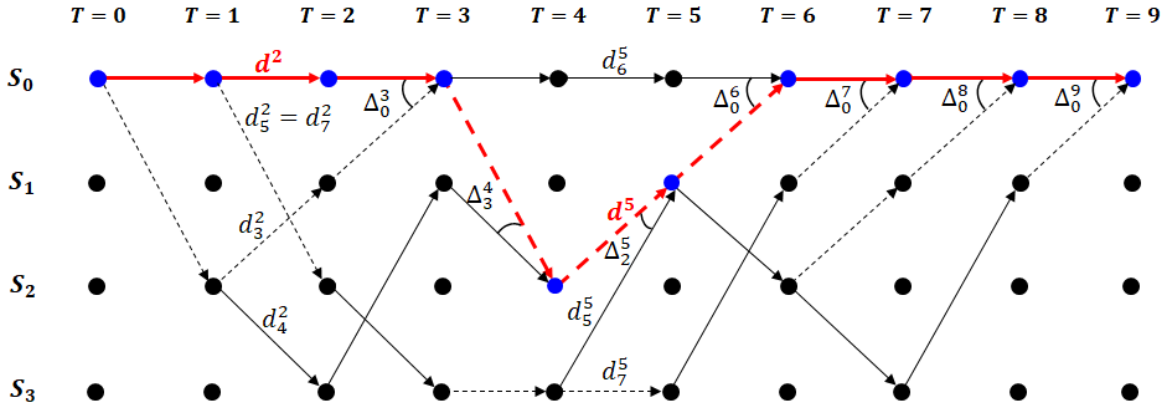


Figure (2.17): Simplified Trellis Diagram for the Output LLR Estimation.

2.8. Concatenated Turbo Codes

Channel coding is a crucial part of new communication systems predominated by severe restrictions on bandwidth and power. Channel coding can be achieved by introducing redundant bits into the information sequence, thus allowing the receiver to correct errors during channel transmission. In 1948, Shannon proved in his landmark paper the theoretical presence of good codes could be used as channel coding [72]. Shannon claimed in this paper that these good codes allow the transmitting of data virtually with no error at rates up to the maximum channel capacity and with surprisingly low transmitted power, contrary to popular belief at that time [73]. Shannon's

paper has not presented the methodology for constructing such codes. The question of "how to construct the capacity-achieving codes?" is left unanswered. This question has motivated intensive research endeavors for decades and has led to the discovery of several codes. However, there is still a large gap (about 3dB or more) between the practical performance and what the theory promised until the early 90s. In 1993, this performance gap was narrowed by the invention of the Convolutional Turbo Codes (CTC) [74] and quickly followed in 1994 by the introduction of Block Turbo Codes (BTC) [75].

Dave Forney introduced the first idea of concatenated codes in 1965 by concatenating two relatively simple codes as inner and outer codes [76]. This step has constituted a significant turning point in the channel coding field since 1965. All proposals that attempted to get close to Shannon's limit before Forney's proposal were unmanageable due to the enormous complexity of the decoding algorithms. Forney introduced a manageable decoding algorithm using relatively simple codes as constituent codes. The complexity of this algorithm increases with the block length algebraically rather than exponentially [77]. This section presents the serial and parallel concatenated turbo codes in terms of encoders and decoders.

2.8.1. Parallel Concatenated Codes

The parallel concatenated codes made the first genuine attempt to approach the Shannon limit with moderate complexity [74]. These codes perform very well in regions of low SNRs and produce high coding gains. However, their performance deteriorates in regions of high SNRs due to error floor problems. Classically, the parallel concatenated codes encoder can be formed by concatenating two or more identical (or not) constituent codes separated by an interleaver, as depicted in Figure (2.18). For example, the original turbo code proposed by Berrou, Glavieux, and Thitimajshima

consists of two identical Recursive Systematic Convolutional (RSC) codes with matrix interleaver where the information bits are inserted row by row and then read pseudo-randomly [74]. Many other parallel turbo codes are constructed using nonsystematic recursive convolutional codes as constituent codes instead of RSC codes [78], [79]. Perez and Benedetto demonstrated in [87] and [88] that recursive convolutional codes can yield codewords with higher weight than non-recursive convolutional codes, even when the weight of input information is low [82]. This property is a crucial benefit of parallel turbo codes since error occurrences are dominated by codewords with low weight.

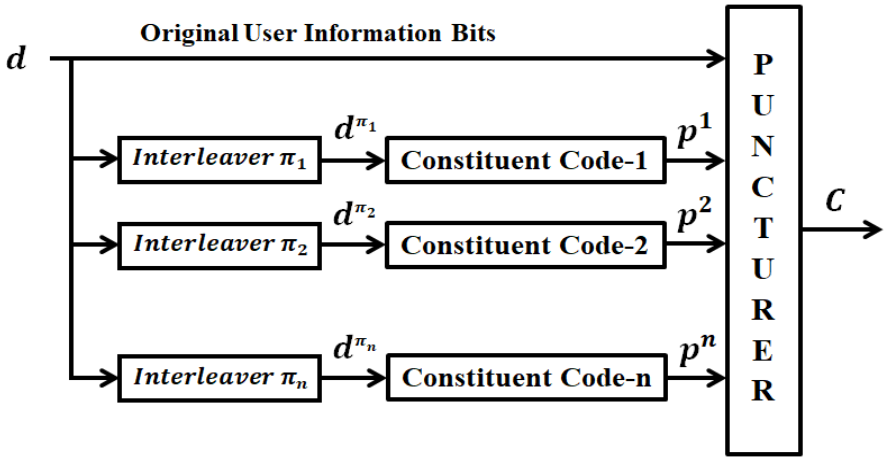


Figure (2.18): The Construction of Parallel Concatenated Code Encoder.

The iterative decoding algorithms of parallel concatenated codes are constructed using SISO algorithms. The SISO decoding algorithms vary depending on the constituent codes. The first convolutional iterative decoding algorithm is presented as follows:

Figure (2.19) depicts the overall structure of a convolutional iterative decoding algorithm. This algorithm consists of two SISO decoders linked together through an interleaver similar to that used in a turbo encoder [71]. As shown in Figure (2.19), the inputs of each decoder include three branches: received LLRs of the systematic encoded message bits, received LLRs of

parity bits of the associated constituent encoder, and a-prior information from the other SISO decoder regarded as a decoder decision about the systematic user bits. The SISO decoders must exploit both soft inputs that include received information through the channel and the a-prior information to yield the soft outputs associated with the decoded user's bits. The soft output of the SISO decoders represents the probability that each bit has been successfully decoded. Soft outputs are commonly expressed by Log-Likelihood Ratios (LLRs), whose magnitude represents the bit's sign, and the amplitude represents the probability of a correct decision.

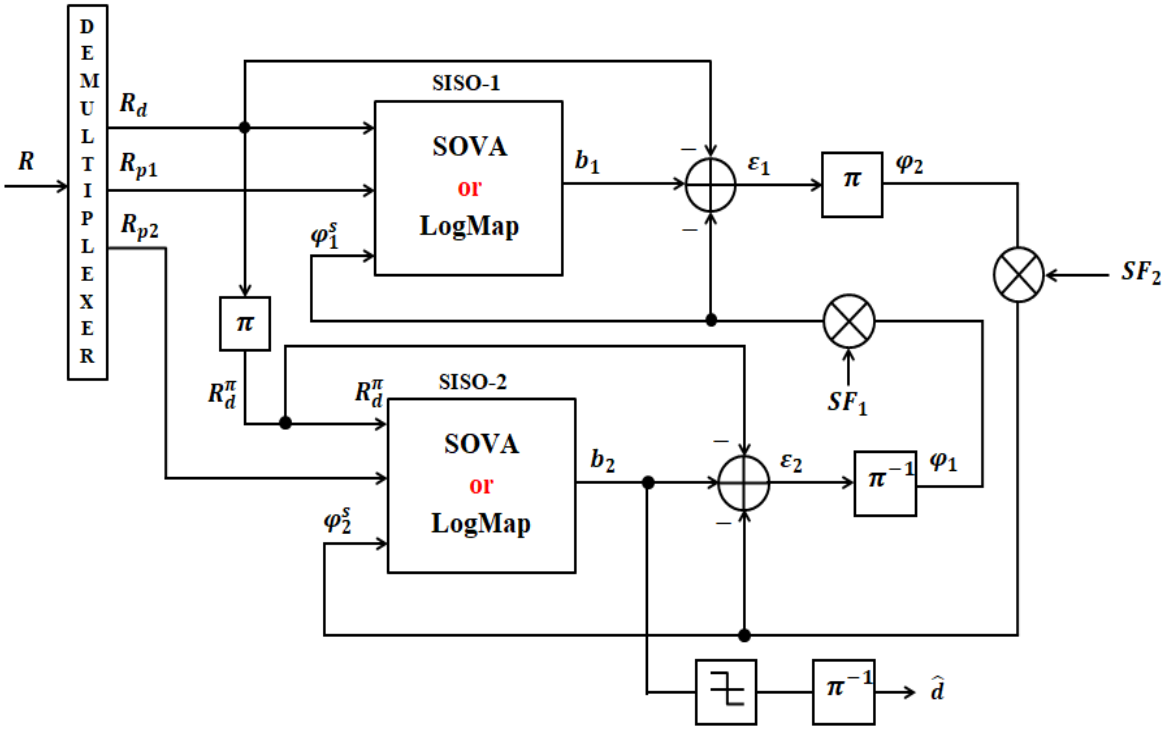


Figure (2.19): Construction of the Convolutional Iterative Decoding Algorithm.

The SISO decoders of the convolutional iterative decoding algorithm can be the Maximum A-Posteriori (MAP) algorithm[83], LogMap algorithm [84], Max LogMap algorithm, or the Soft Output Viterbi Algorithm (SOVA) [69]. The MAP and Max LogMap algorithms are outside the scope of this dissertation, while the LogMap algorithm and SOVA are presented previously

in the this chapter. The MAP algorithm outperforms all remaining algorithms but with a high degree of complexity. In contrast, the SOVA has less complexity but with about 0.6dB degradation compared to the MAP algorithm [85].

In Figure (2.19), the decoding algorithm works iteratively. In the first iteration, the first SISO decoder delivers a soft output as its estimation of the user's bits, depending only on the received information from the channel because the a-prior information from the second SISO decoder at the first iteration is zero. This soft output of the first decoder is then fed to the second SISO decoder as a-prior information. The second SISO decoder uses this information along with the received information to determine its estimation of the user's bits. After completing the first iteration, the second iteration can now start. In the second iteration, the first SISO decoder decodes the same received LLRs of the encoded bits, but currently, with a-prior information provided in the first iteration from the second decoder about the user's bits. The soft output of the first decoder in the second iteration is more accurate due to the a-prior information. Now, the second decoder can use this soft output as a-prior information. This process is repeated with each iteration; thus, the bit error rate of the decoded bits tends to decline. However, the performance gains achieved with increasing iterations numbers decline as the number of iterations is grown. As a result, around five iterations are often employed for complexity and performance considerations [71].

The SOVA and LogMap algorithm can receive a-prior information and produce a-posteriori LLR per input bit in the received sequence, as illustrated in Sections 2.7.2 and 2.7.3. Therefore, this feature qualifies SOVA and LogMap algorithm for a convolutional iterative decoding algorithm. The convolutional iterative decoding algorithm comprises two SISO decoders linked serially in a closed loop, as illustrated in Figure (2.19). In this

algorithm, each SISO decoder estimates the user's bits using a different parity check sequence. The turbo iterative decoding algorithm handles the channel outputs on a frame basis.

As stated in [71], [86]–[88], the soft outputs of the SISO decoders can be decomposed into three unique terms as follows:

$$b = \varphi + \varepsilon + R_d \qquad (2.55)$$

Where:

b: The soft output of a SISO decoder (a-posteriori information).

R_d: The received weighted channel output of the message sequence.

ε: The extrinsic information created by a SISO decoder.

φ: The a-prior information from a SISO decoder.

The extrinsic information that is passed between the SISO decoders can be computed as follows:

$$\varepsilon = b - \varphi - R_d \qquad (2.56)$$

R_d and (φ) are subtracted to prevent sending information that the alternate decoder already has.

The following steps are a summary of the convolutional iterative decoding algorithm [70]:

1. The inputs of the first SISO decoder are:
   i) R_d: received LLRs of the systematic user's bits scaled by the channel reliability constant (L_c).
   ii) R_{P^1}: received LLRs of the parity check bits of the first constituent encoder scaled by the channel reliability constant (L_c).
   iii) φ_1: a-prior information received from the second SISO decoder after de-interleaving the extrinsic information (ε_2). In the first iteration, this information is zero (i.e., $\varphi_1 = 0$).

The a-posteriori LLR (b_1) of the first SISO decoder can be estimated by using Equations (2.46) and (2.53) for the LogMap algorithm and SOVA, respectively.

2. The extrinsic information (ε_1) from the first SISO can be determined by:

$$\varepsilon_1 = b_1 - R_d - \varphi_1 \qquad (2.57)$$

3. The inputs of the second SISO decoder are:
   i) R_d^{π}: interleaved received LLRs of the systematic user's bits scaled by the channel reliability constant (L_c).
   ii) R_{P^2}: received LLRs of the parity check bits of the second constituent encoder scaled by the channel reliability constant (L_c).
   iii) φ_2: a-prior information received from the first SISO decoder after interleaving the extrinsic information (ε_1).

   The output of the second SISO decoder is the a-posteriori LLR (b_2), which is calculated as in the first decoder.
4. The extrinsic information (ε_2) from the second SISO can be determined by:

$$\varepsilon_2 = b_2 - R_d^{\pi} - \varphi_2 \qquad (2.58)$$

5. The sequence (ε_2) is de-interleaved and returned to the first SISO decoder as a-prior information (φ_1) for the next iteration.
6. The estimated bits ($\hat{d}$) are obtained by interleaving the second decoder's output (b_2) before using a simple threshold detector.

The a-prior information of both SISO decoders can be scaled by specific factors (SF_1 & SF_2) to improve the performance of the iterative decoding algorithm, as shown in Figure (2.19). Sometimes the a-prior information contains exaggerated information about the user's bits, especially at the initial iterations. Therefore, these factors can play significant roles in the decoding process.

2.8.2. Serial Concatenated Codes

Serially concatenated codes consist of an outer code with an interleaver and an inner code. The interleaver permutes the codewords bits of the outer code, and then they are fed to the inner code. A serially concatenated code can be constructed with a number (h) of cascaded codes separated by ($h-1$) interleavers. David Forney wanted in [76] to develop a family of codes whose error probability reduced exponentially at rates below capacity while decoding complexity increased algebraically. Forney came up with concatenated code as a multilayer coding structure. This code is the first version of the serial concatenated codes. In Forney's approach, a short convolutional code is used as an inner code with a maximum likelihood decoding algorithm. At the same time, a long high-rate algebraic nonbinary Reed-Solomon code is used as an outer code with a strong error-correction algorithm that could use information about the reliability of the inner decoder as a-prior information [81].

In 1996, S. Benedetto came up with the idea of Serial Concatenated Convolutional Codes (SCCC) [81]. The SCCC incorporates the properties of Forney's serial concatenated and Turbo codes in a convenient signal-to-noise ratio range to produce outstanding decoding performance through iterative decoding. CCSDS131.2-O-1 standard was introduced in September 2007 by employing the serial concatenated convolutional turbo codes (i.e., SCCC codes) in telemetry systems to get performance very close to Shannon's limit. Furthermore, at high SNR, it was discovered that SCCCs performed better than Parallel Concatenated Convolutional Codes (PCCCs) in terms of both the frame error rate and the bit error rate [89]. The major dissimilarity between SCCC code and Forney's concatenated code is that SCCC employs a random interleaver to randomize the burst errors and allow an iterative interchange of soft information between the inner and outer decoders.

The methodology of the SCCC code can be used with other constituent codes such as block codes [90]–[92] (e.g., Reed-Solomon and BCH [89], [93], [94]), polar codes [25], [95], [96] and others to construct modified serially concatenated codes. In this section, the methodology, construction, and iterative decoding algorithms of Serial Concatenated Convolutional Codes (SCCC) is explained as follows:

The ground-breaking concept of parallel turbo codes presented in [74] has sparked an enormous amount of interest in studying serial concatenated schemes. The encoding technique of SCCC differs from that of PCCC schemes. Where the outer encoder first encodes the original user message (d). Unlike PCCC, the second encoder (i.e., inner encoder) receives the outer encoder's interleaved output as input rather than the interleaved user message (d) as in PCCC. The SCCC technique, like the PCCC scheme, can achieve a higher code rate by puncturing a low-rate component encoder or employing high-rate constituent codes. Without code termination, the overall code rate for the SCCC model in Figure (2.20) is:

$$\mathfrak{R} = \mathfrak{R}^{i} * \mathfrak{R}^{o} \qquad (2.59)$$

Where $\mathfrak{R}^{i}$and $\mathfrak{R}^{o}$ are the inner and outer code rates, respectively. Both constituent codes in Figure (2.20) are systematic and recursive convolutional codes. However, it is observed that just the inner encoder is essential to be RSC in SCCC to obtain high interleaving gain. At the same time, the outer code is responsible for correcting burst errors and enhancing performance in the error floor region[89]. Table (2.1) shows the generator polynomials of several convolutional codes.

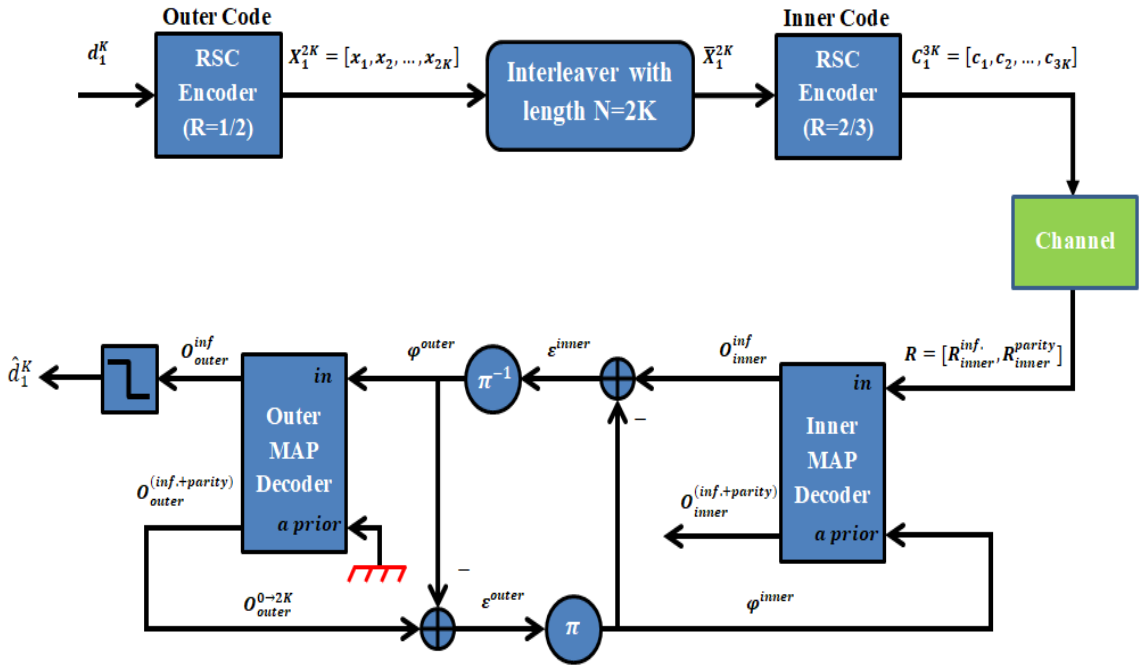


Figure (2.20): Serial Concatenated Convolutional Code.

In Figure (2.20), the first code (i.e., inner code) is an RSC code with a 1/2 rate, while the second code (i.e., outer code) is an RSC code with a 2/3 rate. Accordingly, the overall code rate is 1/3, as indicated by Equation (2.59). A uniform interleaver of length N links these constituent codes with each other.

Table (2.1): Generator polynomials of Convolutional Codes [81].

$G(x)$	Description
$\left[1, \frac{1+x^2}{1+x+x^2}\right]$	Recursive code with $1/2$ rate
$[1+x+x^2, \quad 1+x^2]$	Non-Recursive code with $1/2$ rate
$\begin{bmatrix} 1 & 0 & \frac{1+x^2}{1+x+x^2} \\ 0 & 1 & \frac{1+x}{1+x+x^2} \end{bmatrix}$	Recursive code with $2/3$ rate
$\begin{bmatrix} 1+x & x & 1 \\ 1+x & 1 & 1+x \end{bmatrix}$	Non-Recursive code with $2/3$ rate

Figure (2.20) shows the first iterative decoding algorithm of the SCCC, which has a complexity that is not much more than that required to decode the two constituent codes separately. In this algorithm, Benedetto used two maximum a posteriori (MAP) decoding algorithms applied to both constituent codes [81]. This Map algorithm is clearly different from those Map algorithms of PCCC. In PCCC, the MAP algorithm provides only soft outputs for information sequences, while the SCCC requires soft outputs for information and parity sequences. Therefore the MAP algorithm of the PCCC is modified to produce soft outputs for information and parity sequences.

The mechanism of the iterative decoding algorithm shown in Figure (2.20) can be summarized as follows [81]:

1) During the first iteration, the inner MAP decoder is fed with the received soft information (R) of the inner encoder output without a-prior information from the outer MAP decoder (i.e., $\varphi^{inner} = 0$).
2) The soft inner output of the information sequence ($O_{inner}^{inf.}$) is used to compute the inner extrinsic information (ε^{inner}) by subtracting the inner a-prior information (φ^{inner}) from it, as follows:

$$\varepsilon^{inner} = O_{inner}^{inf.} - \varphi^{inner} \qquad (2.60)$$

3) The inner extrinsic information (ε^{inner}) is de-interleaved to determine the outer a-prior information (φ^{outer}) which is routed to the outer decoder as input.
4) The second input of the outer decoder labeled ($a\ prior$) is permanently fed with a zero vector, as shown in Figure (2.20).
5) The first output of the outer decoder ($O_{outer}^{inf.}$) is routed to a hard decision element that provides the decoded information bits ($\hat{d}_1^K$).

6) The second output of the outer decoder ($O_{outer}^{inf.+parity}$) is used to compute the outer extrinsic information (ε^{outer}) by subtracting the input information (φ^{outer}) from it, as follows:

$$\varepsilon^{outer} = O_{inner}^{inf.+parity} - \varphi^{outer} \tag{2.61}$$

7) The first iteration is complete at this point, and the second iteration begins by repeating steps 1-6, but this time with a-prior information from the outer decoder fed into the inner decoder.
8) After the last iteration, the output in step (5) is used to recover the information bits.

The inner and outer MAP algorithms in the iterative decoding algorithm shown in Figure (2.20) can be replaced by any convolutional decoding algorithm that can provide soft output for information and parity sequences such as SOVA, LogMap, and Maximum LogMap algorithms [89], [97], [98].

2.9. Field Programmable Gate Arrays (FPGAs)

Field Programmable Gate Arrays (FPGAs) are semiconductor devices with several distinctive features that make them more suitable for communications applications. The fundamentals, applications, and architecture of FPGAs will be illustrated in this section.

2.9.1. Fundamentals and Applications

FPGAs are digital integrated circuits consisting of an enormous number of Programmable Logic Blocks (PLBs) with programmable interconnections between them. These interconnections determine the programmable logic blocks needed to perform a specific task. A design engineer can perform various tasks by programming or configuring these devices. Based on the implementation technique, FPGAs can be classified into one-time and multi-time programmable FPGAs. As indicated by the name, the former can only be

programmed once, while the latter can be re-programmed as often as the designer desires [99], [100].

On the other side of the spectrum, two technologies could replace an FPGA: Application-Specific Standard Parts (ASSPs) and Application-Specific Integrated Circuits (ASICs). Although these devices can perform extremely large and complex functions and hold hundreds of millions of logic gates, they are still the second choice after the FPGA, especially in small and medium innovative design companies. FPGAs are better than ASSPs and ASICs in several points, such as low cost, very simple design changes, and faster time-to-market. In addition, ASSPs and ASICs are specifically designed to handle a particular application and can't be used with other applications [99], [100].

Currently, high-performance FPGAs are available in markets, where they contain millions of gates and feature several embedded elements such as high-speed input/output (I/O) devices, microprocessor cores, and others. Thus today's FPGAs can be employed in several fields, such as image processing, radar, DSP applications, software-defined radio, and communications devices.

Based on programming technology, FPGAs can be categorized into three categories [99], [100]:

- Anti-fuse FPGAs: These devices are programmed by utilizing a specific programmer that burns a set of fuses. After burning these fuses, the configured device cannot be reprogrammed again. Therefore the anti-fuse FPGAs are One Time Programmable (OTP) devices used for special functions and cannot be altered anymore, making them unsuitable for prototyping and development purposes. These devices are more suitable for military and aerospace applications because they are nonvolatile (i.e., they save their configuration data after a power outage), rad-hard (i.e., their

radiation immunity is very high, unlike SRAM-based devices that are affected by radiation), and anti-reverse-engineered devices.

- Flash FPGAs: These devices are also programmed offline by a device programmer and can be reprogrammed many thousands of times in the future. Flash FPGAs are available in In-System Programmable (ISP) forms but with a longer programming time. These devices consume high power because of the numerous pull-up resistors connected internally. However, Flash FPGAs have several advantages, such as retaining their configuration data after a power outage (i.e., nonvolatile), using the security concept of a multi-bit key, and requiring little physical space.
- SRAM FPGAs: These devices are unlimitedly reprogrammed by loading the circuit configuration in SRAMs. SRAM-based FPGAs offer extremely quick reconfiguration with the opportunity for partial reconfiguration offered by some devices like the Xilinx, so they are commonly used. As a result, the implementation and testing of the new designs become more rapid and flexible, allowing the way to accommodate evolving protocols and standards easily. The main downside of these devices is related to security issues and the volatility of the configuration data with every powered-up. The former can be scaled using bit-stream encryption but still has some security issues, while the latter demands either a specific external memory element or an onboard microprocessor.

2.9.2. FPGA Architecture

The architecture of the FPGA is different from one company to another, but most of the architectural features are the same in all FPGAs. Figure (2.21) shows the Xilinx FPGA architecture. Sometimes, the DSP blocks are designed to be close and aligned to the embedded RAM blocks, as will be shown later. Several common architectural features of various FPGAs are

illustrated as follows:

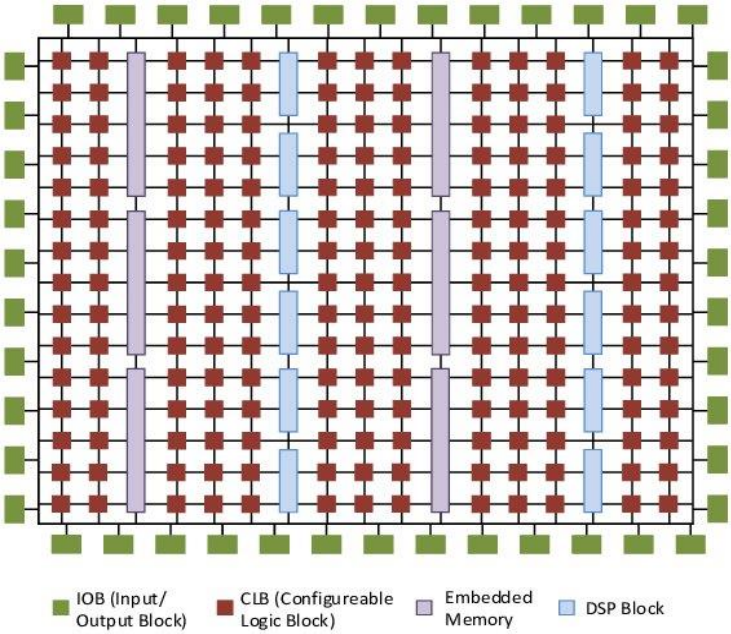


Figure (2.21): The Full Xilinx FPGA Underlying Fabric [101].

2.9.2.1. Granularity

Based on the granularity, there are two FPGA types, either fine-grained or coarse-grained. Before illustrating these types, the underlying fabric of an FPGA must be presented to understand the distribution of the enormous number of Configurable Logic Blocks (CLBs) in a sea of programmable interconnects. Figures (2.21) and (2.22) show the full and partial underlying fabrics [99].

Firstly, each CLB in the fine-grained architecture is responsible for a straightforward process. For instance, this logic block can be programmed to act as a 3-input function, as a logic process (e.g., OR, AND, NOR, etc.), or as a storage process (e.g., D-FlipFlop, D-latch, etc.).

Secondly, a single CLB of the coarse-grained architecture has many logical processes, such as four multiplexers, four 4-input LUTs, four D-FlipFlops, and others.

The number of connections required for each fine-grained CLB is larger than that of the coarse-grained block. This number of connections is relatively

large compared to the size of functionality that a fine-grained CLB can accomplish. Compared to the functionality amount, this number decreases as the order of the granularity increases from medium to high granularity. The number of programmable interconnects is very important because it is closely associated with the amount of propagation delay through an FPGA [99].

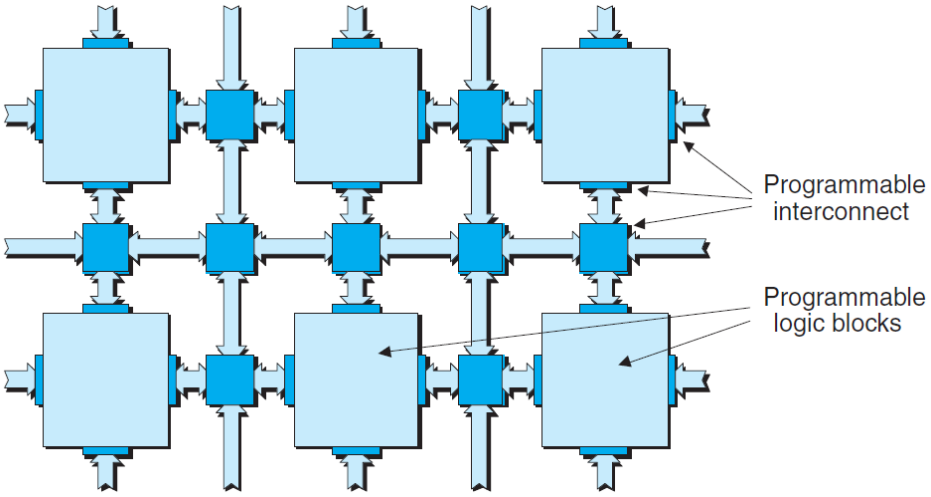


Figure (2.22): The Partial FPGA Underlying Fabric [99].

2.9.2.2. Logic Blocks

The logic block is the basic inner structure of an FPGA. This block is repeated several times and linked with each other by configurable interconnects to perform complicated logic processes, execute memory functions, and others. A configurable logic block (CLB) structure varies based on the company that configured it. The Xilinx configurable logic block consists of two or four slices, where each slice includes two logic cells, as shown in Figure (2.23). A logic cell consists of a 4-input Lookup-Table (LUT), a register, and a multiplexer, as illustrated in Figure (2.24), where the LUT can be acted as a 16-bit shift register or a 16x1 RAM [99], [100].

Figure (2.25) shows the structure of a Xilinx CLB. The inner blocks of the CLB (between LCs and between Slices) are connected by configurable interconnects, but these interconnections are not presented in Figure (2.25) for simplicity. The logic-blocks hierarchy from LC to slice and then to CLB is

linked with another type of hierarchy for configurable interconnects. The interconnects between LCs are faster than those between slices, which in turn is faster than those between CLBs [99], [100].

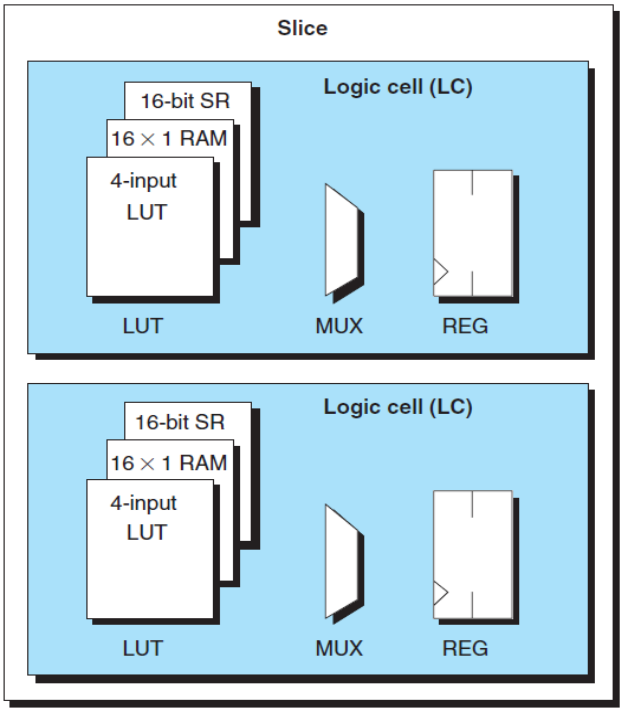


Figure (2.23): A Simple Xilinx Slice with Two Logic Cells [99].

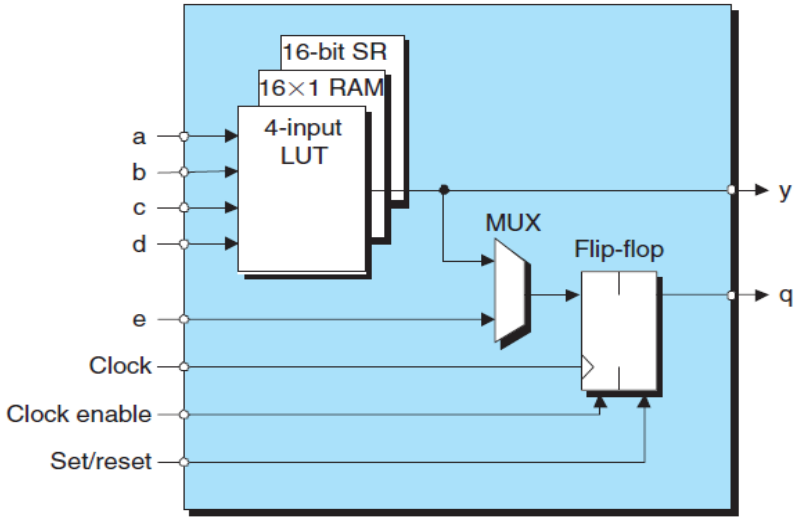


Figure (2.24): The Construction of a Simple Xilinx Logic Cell (LC) [99].

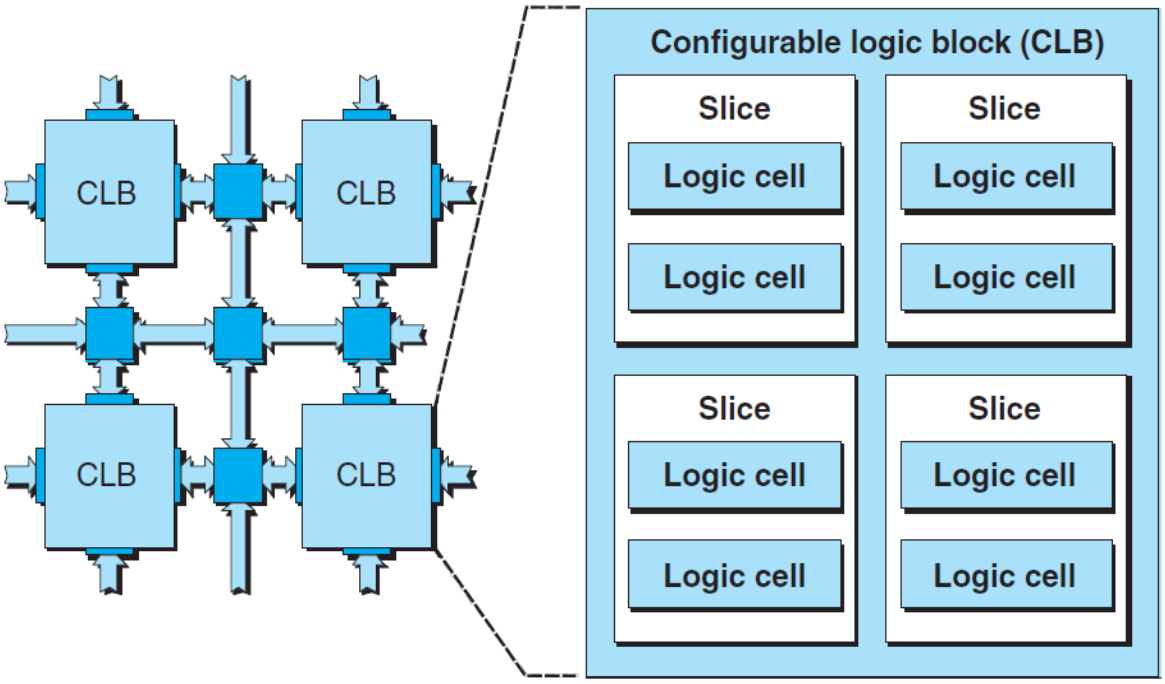


Figure (2.25): A Simple Xilinx CLB with Four Slices [99].

2.9.2.3. Embedded RAMs

Several applications need to employ memory; thus, current FPGAs contain comparatively large blocks of Embedded-RAMs known as e-RAM. Based on the FPGA architecture, these blocks are either placed around the FPGA's periphery or scattered in columns across the FPGA chip, as shown in Figure (2.21). An FPGA may contain tens to thousands of these chunks to provide enormous memory for various applications [99].

2.9.2.4. Embedded Multipliers and Adders

The software multipliers and adders constructed using a large number of CLBs are very slow compared to the hardware multipliers and adders. Therefore, these components with other devices are added as DSP blocks in the form of scattered columns in the FPGA chip, as shown in Figure (2.21). For instance, Some Xilinx FPGAs possess a DSP-Block called a DSP-48 which can perform a 25-bit pre-adder, an 18-bit×25-bit multiplier, and a 48-bit accumulator [99], [102].

2.9.2.5. Embedded Processor Cores

Practically any part of an electronic design can be implemented either as hardware by employing registers, logic gates, and others, or as software by using instruction codes executed on a microprocessor. In the same way, processor cores can be implemented as hardware or software [99].

A hardware microprocessor core can be implemented either in a stripe or inside an FPGA fabric. As shown in Figure (2.26), the stripe of the first is located on the side of the primary FPGA fabric. The Stripe and the FPGA fabric can be fabricated either on a single silicon chip or double chips and packaged as a Multi-Chip Module (MCM). A microprocessor Stripe also includes multipliers, e-RAMs, I/O devices, and others, apart from those in the main FPGA frame. Secondly, the microprocessor cores can be implanted within the FPGA fabric without requiring a second fabric to build these cores. In this case, the e-RAMs and I/O blocks must be able to take into account the existence of these cores. Depending on the device's features, an FPGA fabric can include one, two, or four microprocessor cores.

The second option is the soft microprocessor cores. These cores can be implemented by dedicating a set of CLBs to configure them without needing to implant these cores within the FPGA fabric. Soft-Cores are slower and more primitive than their hard counterparts, so a hardware microprocessor is preferred for complex functions.

2.9.2.6. Clock Managers

The clock signal is an important requirement for any synchronous element within an FPGA. This signal is commonly generated outside FPGA and then fed to a special hardware block called a clock manager to generate several pictures of a clock signal. These pictures are first processed by the clock manager to handle the signal troubles such as frequency and phase shifts and then fed to the special clock's input pins. The copies of the clock

signal at these pins can be used to derive many synchronous elements[99].

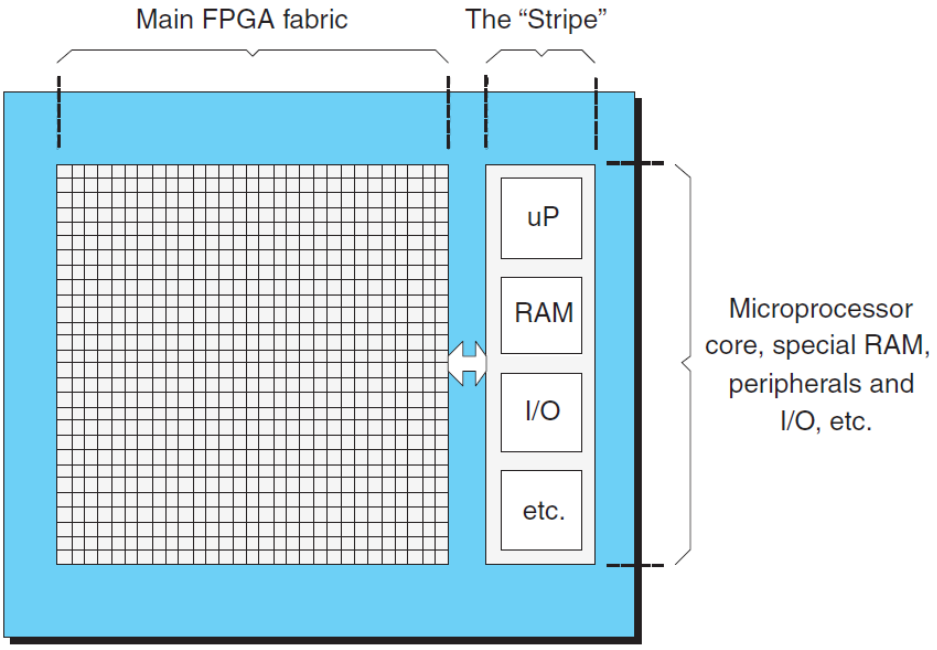


Figure (2.26): The Main FPGA Fabric with the Stripe of Hard-Core [99].

2.9.2.7. General-Purpose I/O

The I/O blocks are illustrated around the circumference of the FPGA fabric in a ring, as shown in Figure (2.21). These I/O blocks have various standards, voltage thresholds, and terminating resistors to prevent signals-reflecting-back [99].

2.9.2.8. Gigabit Transceivers

These blocks are special hardware devices used to send and receive data at incredibly high speeds permitting them to deal with a huge amount of data bits per second. The gigabit transceivers employ one pair of differential signals to receive (RX) information and another to send (TX) information. In fact, a single transceiver block supports multi-transceivers (say four), and each FPGA board may include many such transceiver blocks [99].

2.9.2.9. Intellectual Property

It is impractical to construct every FPGA design from scratch in light of today's large and complicated designs. The common functions can be

designed as programmable functional blocks that can be used with every FPGA design. This idea can help designers to focus on the novel portion of the design that differentiates their designs from other competing designs and reduces design time and resource requirements. Each block of these functional blocks is known as Intellectual Property (IP) [99].

IP can be classified as hard and soft, where each FPGA vendor delivers its own IPs. Firstly, the hard IPs are pre-implemented blocks like gigabit interfaces, adders, multipliers, microprocessor cores, and others. These blocks are designed to be compatible with FPGA devices, providing efficient performance and power consumption. Secondly, the soft IPs are used as a source-level library of high-level functions that can be embedded in the design code of a user. These soft libraries can be defined at the Register Transfer Level (RTL) of abstraction by utilizing a Hardware Description Language (HDL), like VHDL or verilog [99].

3

Chapter Three

Proposed Concatenated Turbo Polar Codes

Chapter Three

Proposed Concatenated Turbo Polar Codes

3.1. Introduction

The original concatenated codes can be updated using the modern codes as constituent codes rather than the convolutional or BCH codes. Furthermore, recent research on modified concatenated codes presents encouraging performance results, motivating the coding community to continue developing in this field. In this dissertation, the polar codes have been used as constituent codes to construct the proposed turbo polar codes. This chapter presents the proposed parallel and serial concatenated turbo polar codes in terms of encoders, decoders, complexity, and some case studies. These codes are designed by using a polar code

3.2. Parallel Concatenation Turbo Polar Code (PCTPC)

The encoder scheme of the proposed PCTPC has been designed by using two identical systematic polar codes as constituent codes as shown in Figure (3.1). In this scheme, The source bits (d) are first fed to the first SPC encoder (i.e., $SPC(N,K)$) that will encode them. Then, the same bits (d) are interleaved via an Srandom interleaver (π) to be encoded by the second SPC encoder. The coded output ($C \in \{0,1\}$) of the PCTPC encoder comprises three vectors multiplexed together: the source vector ($c^1 = d$), the parity vector of SPC-1 ($c^2 = p^1$), and the parity vector of SPC-2 ($c^3 = p^2$). These vectors are multiplexed to yield the overall coded vector (C) that includes N_c bits (i.e., $N_c = 2N - K$). The code rate of the PCTPC is given by: $\Re = \frac{K}{2N-K}$. The coded vector (C) is modulated by the BPSK modulator and transmitted

through the channel, where $C \xrightarrow{BPSK} Y; Y = (y_1, y_2, \dots\dots, y_{N_c}),\ y_i \in \{+1, -1\}$ and is given by:

$$y_i = 1 - 2c_i \qquad (3.1)$$

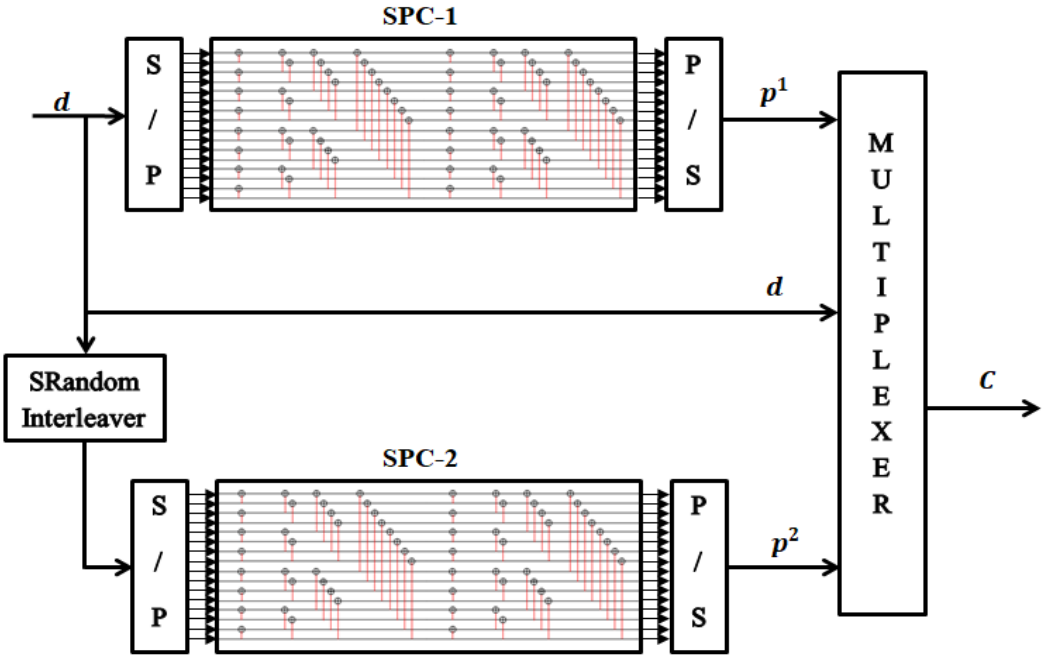


Figure (3.1): The Encoder of the Proposed PCTPC Scheme.

On the other side, Figure (3.2) depicts the overall structure of the proposed polar iterative decoding algorithm. Despite the differences in the construction of the SISO decoders, this structure is comparable to that of the convolutional iterative decoding algorithm. The SISO polar decoders have distinct internal architecture based on the methodology of polar codes. There are several decoding algorithms designed for polar codes, including Successive Cancellation (SC), Successive Cancellation List (SCL), Successive Cancellation Stack (SCS), Belief Propagation (BP), Soft Cancellation (SCAN), Soft Successive Cancellation list (SSCL), and so on. However, only SCAN, BP, and SSCL decoders can accept and produce soft information; thus, only these decoders can be used with a polar iterative decoding algorithm. In this study, the SCAN algorithm (refer to Algorithm-4) is employed with all studied schemes.

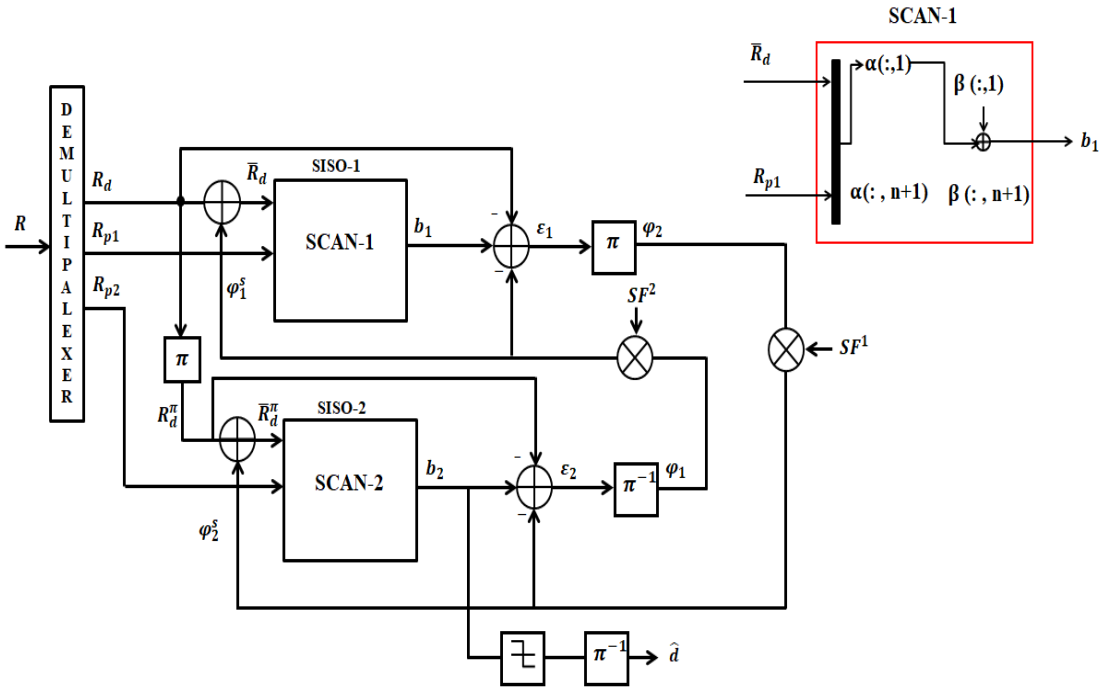


Figure (3.2): Construction of the Polar Iterative Decoding Algorithm.

The mechanism of the proposed polar iterative decoding algorithm can be summarized as follows (refer to Algorithm-8):

1) The received LLRs from the channel are separated into three vectors, namely, $R_d, R_{p1},$ and R_{p2}, which are the LLRs of the systematic message bits, the parity bits of the first constituent encoder, and the parity bits of the second constituent encoder, respectively.
2) R_d , $R_{p1},$ and the scaled a-prior information (φ_1^s) are the inputs of the SCAN-1 decoder that has one output (b_1). The internal procedure of SCAN-1 is shown in Figure (3.2) (The small red box on the top right). Based on the ($\mathcal{F}^c$) and ($\mathcal{F}$) sets, $\bar{R}_d$ is merged with R_{p1} and used to initialize the first column of α matrix (i.e., $\alpha_i^0 | \, i = 0,1,..,N-1$) of SCAN-1 decoder, where the vector ($\bar{R}_d$) corresponds to $\mathcal{F}^c$ indices while the vector (R_{p1}) corresponds to $\mathcal{F}$ indices. The modified information sequence ($\bar{R}_d$) of the SCAN decoder can be computed as follows:

$$\bar{R}_d = R_d + \varphi_1^s \qquad (3.2)$$

3) At the first iteration, the scaled a-prior information (φ_1^s) is zero due to the a-prior information (φ_1) from the second SISO decoder is zero.
4) The extrinsic LLR (ε_1) can be calculated as follows [34], [36]:

$$\varepsilon_1 = b_1 - \frac{2 * R_d}{\sigma^2} - \varphi_1^s \qquad (3.3)$$

Where σ^2 is the noise variance.

5) The extrinsic LLR (ε_1) is interleaved and passed to the SCAN-2 decoder as a-prior information (φ_2).
6) R_d^π , R_{p2}, and the scaled a-prior information (φ_2^s) are the inputs of the SCAN-2 decoder that has one output (b_2). The internal procedure of SCAN-2 is the same as SCAN-1 (see Figure (3.2): The red box).
7) The extrinsic LLR (ε_2) can be calculated as follows [34], [36]:

$$\varepsilon_2 = b_2 - \frac{2 * R_d^\pi}{\sigma^2} - \varphi_2^s \qquad (3.4)$$

8) The extrinsic LLR (ε_2) is de-interleaved and passed to the SCAN-1 decoder as a-prior information (i.e., φ_1). At this point, the first iteration ends, and steps (2 to 8) are repeated for the second iteration, and so on.
9) After the last outer iteration, the estimated bits ($\hat{d}$) are obtained by applying a hard decision to the output of the SCAN-2 decoder (b_2) and then de-interleaving it.
10) The stopping mechanism of the iterative decoding process can be developed based on two criteria. The first one depends on the number of outer iterations, where the iterative decoding process is finished after the last iteration. The second criterion depends on a particular rule. If this rule is satisfied, the iterative decoding process is terminated. Immediately after each iteration, the estimated vector ($\hat{x}$) is calculated to be used for estimating the vector ($\hat{S}$), as follows [32]:

$$\hat{S} = \hat{x} * [G_N]^{-1} = \hat{x} * G_N \quad (3.5)$$

The frozen bits of the vector (S) are fixed with polar codes and known at the receiver, so these bits with the frozen bits of the vector ($\hat{S}$) can be used to satisfy the second criterion of the stopping mechanism through the following:

$$STF = S_{\mathcal{F}} + \hat{S}_{\mathcal{F}} \quad (3.6)$$

If the STF factor equals zero, the iterative decoding process stops. Otherwise, the decoding process continues [32].

3.3. Parallel Concatenation Turbo Polar-Convolutional Code (PCTPCC)

As mentioned earlier, the parallel concatenated codes can be constructed using two non-identical codes, for instance, convolutional and polar codes, BCH and convolutional codes, and so on. Therefore, in this case, polar and convolutional codes are used to construct the proposed PCTPCC scheme. Figure (3.3) shows the encoder structure of the proposed PCTPCC. The presented scheme consists of the SPC and RSC codes concatenated parallelly. The source vector (d) of length K bits is first fed to the RSC encoder (i.e., $RSC(13,15)$) that will encode it to produce the first parity sequence (p^1) of length K. The same bits (d) are then interleaved via an LTE interleaver (π) to be encoded by the SPC encoder (i.e. $SPC(N,K)$). As mentioned earlier with the PCTPC, the encoded output ($C \in \{0,1\}$) comprises three vectors multiplexed together: the source vector ($c^1 = d$), the parity vector of RSC ($c^2 = p^1$), and the parity vector of SPC ($c^3 = p^2$). These vectors are multiplexed to yield the overall coded vector (C) that includes N_c bits (i.e., $N_c = 2N - K$). Therefore, the code rate of the proposed PCTPCC can be determined by: $\Re = \frac{K}{2N-K}$.

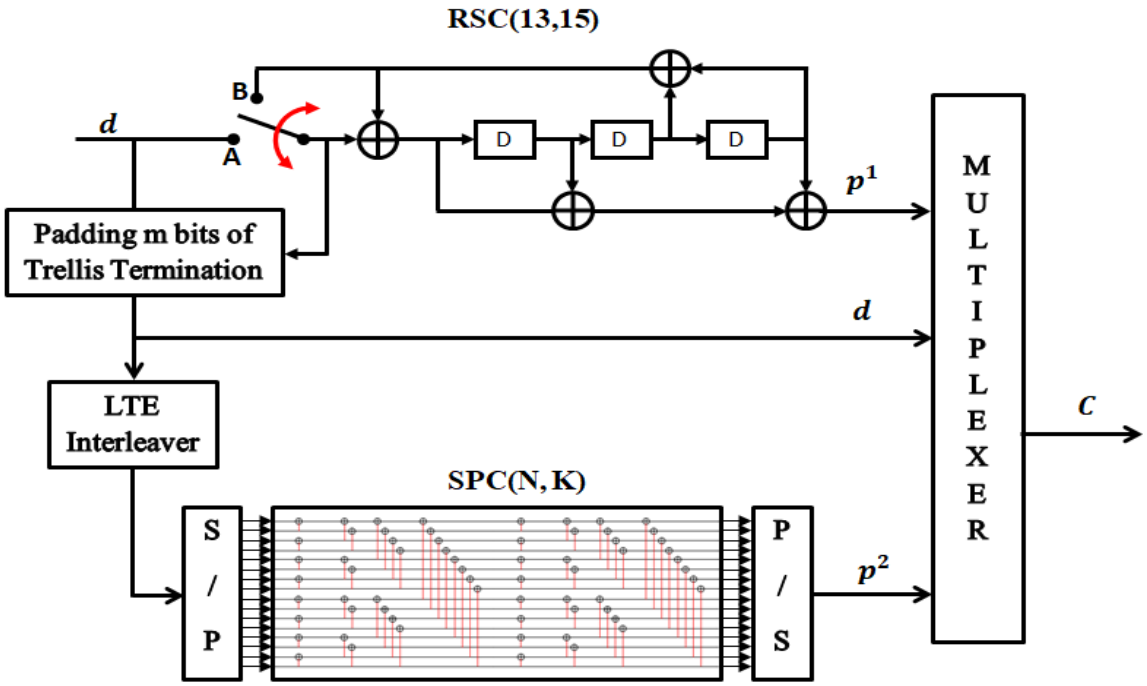


Figure (3.3): The Encoder of the Proposed Parallel Concatenated Turbo Polar-Convolutional Code.

The proposed polar-convolutional iterative decoding algorithm of proposed PCTPCC is a hybrid algorithm between the convolutional and polar iterative decoding algorithms, which are presented in sections 2.8.1 and 3.2, respectively. This algorithm consists of two SISO decoders, one for convolutional codes and the other for polar codes. This study uses SOVA (or LogMap) and SCAN decoders for the RSC and SPC codes, respectively, to construct this algorithm (refer to Algorithm-13).

If the received LLRs $R = (r_1, r_2, \dots\dots, r_{N_c})$ are received through an AWGN channel, the expression of the received message is:

$$R = Y + \gamma = (2C - 1) + \gamma \qquad (3.7)$$

Where C is the encoded sequence and $\gamma = (n_1, n_2, n_3 \dots, \dots, n_{N_c})$ is a noise vector with noise variance σ^2. The received vector (R) is demultiplexed into three parts: $R_d, R_{p1}, and\ R_{p2}$ as shown in Figure (3.4). The first SISO decoder receives three inputs: the received LLRs of the source bits (R_d), the LLRs of the parity bits of the RSC encoder (R_{p1}), and a-prior information

(φ_1) from the second SISO decoder. This a-prior information is zero at the first iteration. Then, the output of the first decoder (b_1) is used to estimate the extrinsic information (ε_1) as follows [34], [36]:

$$\varepsilon_1 = b_1 - \frac{2 * R_d}{\sigma^2} - \varphi_1 \tag{3.8}$$

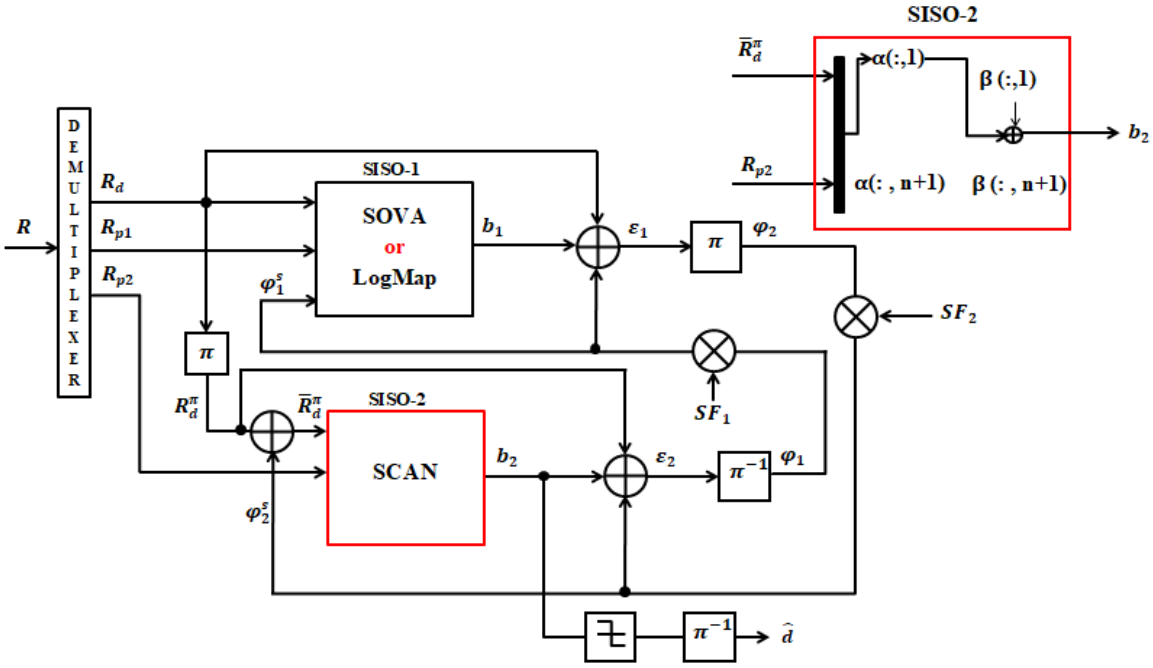


Figure (3.4): Construction of the Polar-Convolutional Iterative Decoding Algorithm.

The a-prior information (φ_2) is estimated by interleaving the extrinsic information (ε_1) of the first SISO decoder. This a-prior message is summed with the interleaved LLRs of the user bits (R_d^{π}) and then fed to the SCAN decoder as input with the LLRs of the parity bits of the SPC encoder (R_{p2}). These LLRs are multiplexed with each other based on the ($\mathcal{F}^c$) and ($\mathcal{F}$) sets of the polar code and then used to initialize the first column of the matrix $\boldsymbol{\alpha}$ of the SCAN decoder, as illustrated in Figure (3.4) (see the red box). After the SCAN decoder has finished its work, the output is determined by summing the first columns of the matrices α and β. For more explanation, the full detail of the SCAN decoder is explained in Section 2.6.5 and Algorithm-4.

The extrinsic information of the second SISO decoder is calculated as follows by helping the output of the SCAN decoder [34], [36]:

$$\varepsilon_2 = b_2 - \frac{2 * R_d^{\pi}}{\sigma^2} - \varphi_2 \qquad (3.9)$$

This extrinsic sequence is de-interleaved and fed to the first SISO decoder as a-prior information. After the last iteration, the estimated bits ($\hat{d}$) of the user information are calculated from the output of the second SISO decoder by taking the hard decision of the LLRs (b_2) and de-interleaving them. As mentioned in Section 2.8.1, the a-prior information of both SISO decoders can be scaled by specific factors (SF_1 & SF_2) to improve the performance of the iterative decoding algorithm, as shown in Figure (3.4).

3.4. Serial Concatenation Turbo Polar-Convolutional Codes (SCTPCC)

Arikan's proposal of polar codes is a significant advancement in coding theory. It has attracted much interest because these codes have been proven to operate exceptionally well over BI-DMC. Moreover, under SC decoding, they offer a low level of complexity. However, since the successive cancellation algorithm is prone to error propagation, the performance of polar codes in the finite length regime deteriorates. This problem has sparked enormous research in studying polar codes' improvements. In the previous sections, the parallel turbo polar codes are introduced as one of these improvements. The parallel concatenation polar codes outperform the original ones decoded with the SC decoder and still have a competing level of complexity. However, the performance of these codes deteriorates in high signal-to-noise regions due to error floor problems. Therefore, serial concatenation polar-convolutional codes are introduced to enhance the parallel concatenated polar code's error floor performance.

Figure (3.5) illustrates the structure of the proposed SCTPCC scheme. This code consists of a systematic polar code and a recursive systematic convolutional code as constituent codes. The SPC code is used as the outer code ($\mathcal{C}_o$) with a rate $\Re^o = K/N_{outer}$ and the RSC code is used as the inner code ($\mathcal{C}_i$) with a rate $\Re^i = N_{outer}/N_{inner}$, where K , N_{outer} , and N_{inner} are the lengths of the message bits, outer coded bits, and inner coded bits, respectively. These codes are joined by an interleaver of length N_{outer} to generate a serial concatenated code with a rate $\Re = K/N_{outer}$. The length of the interleaver must be compatible with the length of the outer codewords. Commonly, the code rates of the constituent codes are carefully chosen to get a rate-compatible code, such as $2/3$ and $1/2$ to get $1/3$ overall code rate.

As illustrated in Figure (3.5), the outputs of the outer code are arranged in the normal-indices order of the polar construction, where the message and parity bits are placed in the information and frozen locations of the polar code structure, respectively. As for the codewords bits of the inner code, they are placed in the odd-even pattern.

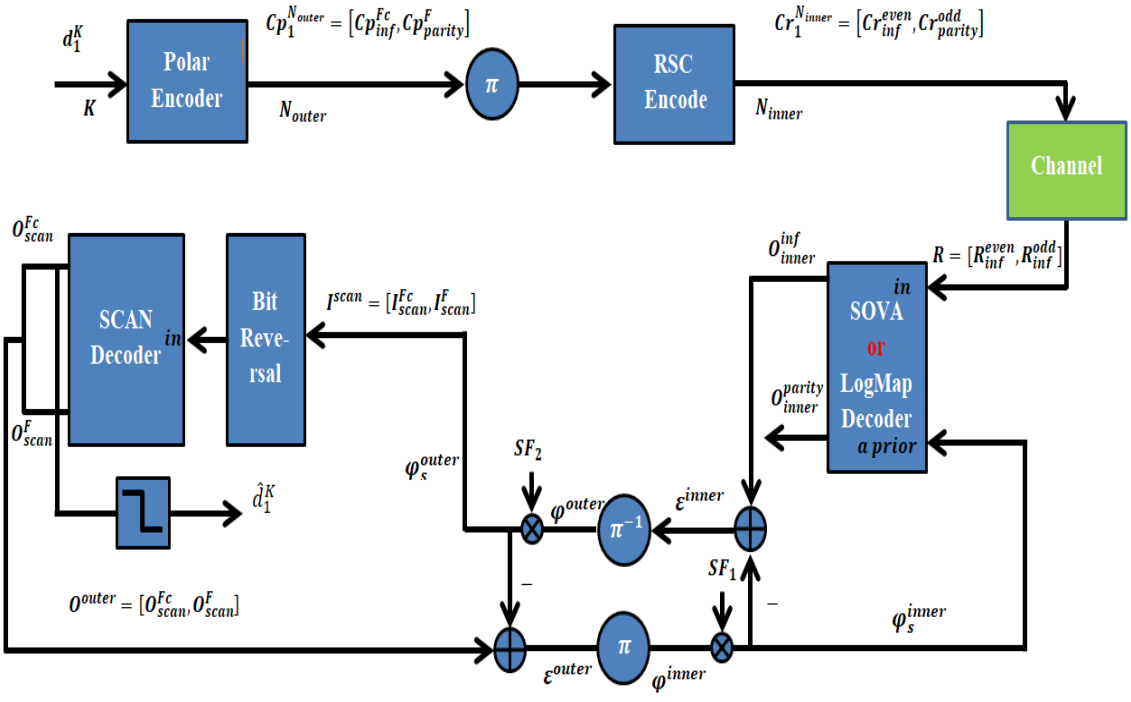


Figure (3.5): Serial Concatenation Turbo Polar-Convolutional Codes.

Serial concatenated polar-convolutional codes are decoded similarly to parallel concatenated polar-convolutional codes using the SOVA decoder (or LogMAP) for the convolutional code and the SCAN algorithm for the polar code, as shown in Figure (3.5). The mechanism of the serial iterative decoding algorithm can be summarized as follows:

a) The odd-even indexed received LLRs (R) drive the inner decoder (SOVA or LogMap) with the scaled a-prior information (φ_s^{inner}), which is received from the outer decoder (SCAN).

b) At the first iteration, the inner a-prior information (φ^{inner}) from the SCAN decoder is zero.

c) The output of the inner decoder (O_{inner}^{inf}) is used to determine the inner extrinsic information as follows:

$$\varepsilon^{inner} = O_{inner}^{inf} - \varphi_s^{inner} \tag{3.10}$$

d) After de-interleaving and scaling the inner extrinsic information (ε^{inner}), the scaled outer a-prior information is defined as follows:

$$\varphi_s^{outer} = SF_2 * \pi^{-1}\left(\varepsilon^{inner}\right) \tag{3.11}$$

e) The scaled outer a-prior information (φ_s^{outer}) drives the outer SCAN decoder after its indices are reversed, as shown in Figure (3.5).

f) By using the output of the SCAN decoder, the outer extrinsic information can be computed as follow:

$$\varepsilon^{outer} = O_{outer} - \varphi_s^{outer} \tag{3.12}$$

g) The outer extrinsic information is interleaved and scaled to generate the scaled inner a-prior information (φ_s^{inner}) which is used to drive the inner decoder, as follows:

$$\varphi_s^{inner} = SF_1 * \pi(\varepsilon^{outer}) \tag{3.13}$$

h) After the last iteration, the estimated bits ($\hat{d}_1^K$) are obtained by applying the hard decision to the output of the SCAN decoder (O_{scan}^{Fc}).

3.5. Complexity Analysis

The previous iterative decoding algorithms include two SISO decoders, as illustrated in the Figures mentioned earlier (i.e., Figures (2.19, 2.20, 3.2, 3.4, and 3.5)). These decoders consume the greatest amount of complexity, while the other components (e.g., Encoder, multiplexer, interleaver, and so on) consume less complexity compared to the complexity of SISO decoders. As a result, it is possible to analyze the complexity of iterative decoding without considering the complexity of these components. The following complexity analysis is based on calculating the number of addition and subtraction operations. The complexity of the individual turbo SISO decoders is presented first, followed by an explanation of the overall complexity of the concatenated codes.

For convolutional codes, the RSC codes can be decoded by several algorithms such as MAP, Log-MAP (LMAP), Max-Log-MAP (MLM), and SOVA algorithms. The following expressions describe the worst-case complexity of these decoders [103]:

$$f_{SOVA} = 2^{m+1}(N_R + 1.5) + 3 \quad (3.14)$$

$$f_{MLM} = 2^{m+1}(N_R + 5) + 3 \quad (3.15)$$

$$f_{LMAP} = 2^{m+1}(N_R + 11) - 3 \quad (3.16)$$

$$f_{MAP} = 2^{m+1}(N_R + 14) + 33 \quad (3.17)$$

Where N_R is the symbols' number of the RSC encoder, and m is the number of the encoder memory elements. The previous expressions can be bound as follows:

$$f_{SOVA} \leq 2^{m+1}(N_R + 1.5) + 3 = O[2^{m+1} * N_R] \quad (3.18)$$

$$f_{MLM} \leq 2^{m+1}(N_R + 5) + 3 = O[2^{m+1}(N_R + 5)] \quad (3.19)$$

$$f_{LMAP} \leq 2^{m+1}(N_R + 11) - 3 = O[2^{m+1}(N_R + 11)] \quad (3.20)$$

$$f_{MAP} \leq 2^{m+1}(N_R + 14) + 33 = O[2^{m+1}(N_R + 14)] \quad (3.21)$$

From the previous expressions, the MAP algorithm is considered the most complex, followed by LMAP, MLM, and the SOVA algorithm is the least complex.

As mentioned previously, only the SCAN, BP, and SSCL decoders of polar codes can be used with the turbo iterative decoding algorithms. Therefore, the total complexity formulas of these decoders are estimated as follows [33], [34], [36]:

$$f_{SCAN} = I_{s_i} * (N * logN) \quad (3.22)$$

$$f_{BP} = I_{B_i} * (N * logN) \quad (3.23)$$

$$f_{SSCL} = I_{list} * (N * logN) \quad (3.24)$$

Where N is the code length of the SPC code; $I_{s_i}$and I_{B_i} represent the inner iterations numbers of the SCAN and BP decoders, respectively, whereas I_{list} is the list size of the SSCL decoder. The BP decoder requires sixty inner iterations ($i.e.,\ I_{B_i} = 60$) to approach the performance of the SCAN decoder when working with only two inner iterations (i.e., $I_{s_i} = 2$) [17], [36]. Obviously, the inner iterations of the SCAN decoder are smaller than the inner iterations of the BP decoder. For the SSCL decoder, the list size (I_{list}) is always 32 or 64 [15], [34]. Therefore, the complexity of the SCAN decoder is smaller than the others.

The total complexity of the concatenated schemes is estimated by considering the complexity of each individual SISO algorithm and adding them together. For example, the complexity expression of the Parallel

Concatenated Turbo Polar-Convolutional Code (PCTPCC) scheme, which consists of SOVA and SCAN algorithms, is:

$$f_{SOVA\_SCAN\_parallel} = O\left(I_{outer}\left[I_{s_i} * NlogN + 2^{m+1} * N_R\right]\right) \qquad (3.25)$$

where I_{outer} is the outer iterations' number between SOVA and SCAN decoders.

The expressions of the complexity for different codes are listed in Table (3.1). The SCAN and SOVA algorithms provide a medium complexity level for the concatenated schemes compared to other turbo-polar codes. The complexity level of the SCAN decoder is much less than that of the BP and SSCL decoders since the inner iterations (I_{s_i}) of the SCAN are less than the inner iterations (I_{B_i}) of BP and the list size (I_{list}) of SSCL.

On the other side of the spectrum, the expressions of the complexity of serial concatenated turbo codes are also based on the same expressions of the individual decoding algorithms mentioned earlier (e.g., SOVA, LogMap, SCAN). However, the total complexity of the serial concatenated turbo schemes is larger than that of the parallel concatenated turbo codes schemes. This considerable complexity comes from the fact that the input to the inner code is the coded sequence generated by the outer code, not the original information. Thus, the complexity of the inner SISO algorithm of the serial concatenated turbo scheme will increase, increasing the total complexity. Moreover, the outer code should not only produce a-prior information about the information but also on the party bits.

Referring to Figure (3.5), if the code rate of both $SPC(N,K)$ and $RSC(N_R,N)$ is $1/2$, then the complexity of the SCTPCC scheme is:

$$f_{SOVA\_SCAN\_Serial} = O\left(I_{outer}\left[I_{s_i} * NlogN + 2^{m+1} * N_R\right]\right) \qquad (3.26)$$

Where $N_R = N\ Symbols$; each symbol has two coded bits. Although the expressions (3.25) and (3.26) are the same, the complexity of expression (3.26) is greater than that of expression (3.25) because the N_R in Equation (3.26) is twice that in Equation (3.25) (i.e., in Equation (3.26), $N_R = N$, while in Equation (3.25), $N_R = N/2$).

Table (3.1): Expressions of Complexity for Several Polar Codes.

Code Scheme	Complexity Expression	Complexity level
Parallel Concatenated Turbo Polar Code with BP [35]	$O[I_{outer} * (2 * I_{B_i} * NlogN)]$	Level-1(Most Complicated)
Parallel Concatenated Turbo Polar Code with SSCL[34]	$O[I_{outer} * (2 * L_{list} * NlogN)]$	Level-2
Parallel Concatenated Turbo Polar Code with SCAN[35], [36]	$O[I_{outer} * (2 * I_{s_i} * NlogN)]$	Level-3
Parallel Concatenated Turbo Polar-Convolutional Code with (SOVA & SCAN)	$O(I_{outer}[I_{s_i} * NlogN + 2^{m+1} * N_R])$	Level-4
Parallel Concatenated Turbo Polar-Convolutional Code with (LMAP&SCAN)	$O(I_{outer}[I_{s_i} * NlogN + 2^{m+1}(N_R + 11)])$	Level-5
Parallel Concatenated Turbo Convolutional Code with SOVA	$O(I_{outer}[2^{m+2} * N_R])$	Level-6
Parallel Concatenated Turbo RS-Polar Code [27]	$O(N(\log N)^2 * loglog\ N + Nlog\ N)$	Level-7
Standalone SSCL	$L_{list} * NlogN$	Level-8
Standalone BP	$I_{B_i} * NlogN$	Level-9
Standalone SCAN	$I_{s_i} * Nlog\ N$	Level-10
Standalone SC	$O[Nlog\ N]$	Level-11 (Less Complicated)

3.6. Case Studies

Two straightforward examples are considered to clarify the details of the iterative decoding algorithms, which are covered in Sections 3.3 and 3.4. The first one is presented to explain the iterative decoding of the parallel

concatenated turbo polar-convolutional code, while the second is presented to explain the iterative decoding of the serial concatenated turbo polar-convolutional code. These case studies are presented as follows:

3.6.1. Case Study-I

In this sub-section, a simple example is considered to explain the iterative decoding algorithm of a parallel concatenated turbo polar-convolutional code. Table (3.2) lists the properties of this concatenated code, and Figures (3.6) and (3.4) depict its encoder and decoder structures, respectively. The input sequence (x) is assumed to be (001110), and the encoder trellis's initial state is zero. After applying the trellis termination, the two bits required to end the trellis at the zero state are {0, 0}, as shown in Table (3.3). The received sequence in Table (3.3) was created by adding AWGN with variance $\sigma^2 = 0.796$ to the transmitted channel sequence (which has E_b/N_o = 4dB). Therefore, according to Equation (2.50), the channel reliability (L_c) equals 2.5, given that the fading amplitude a=1.

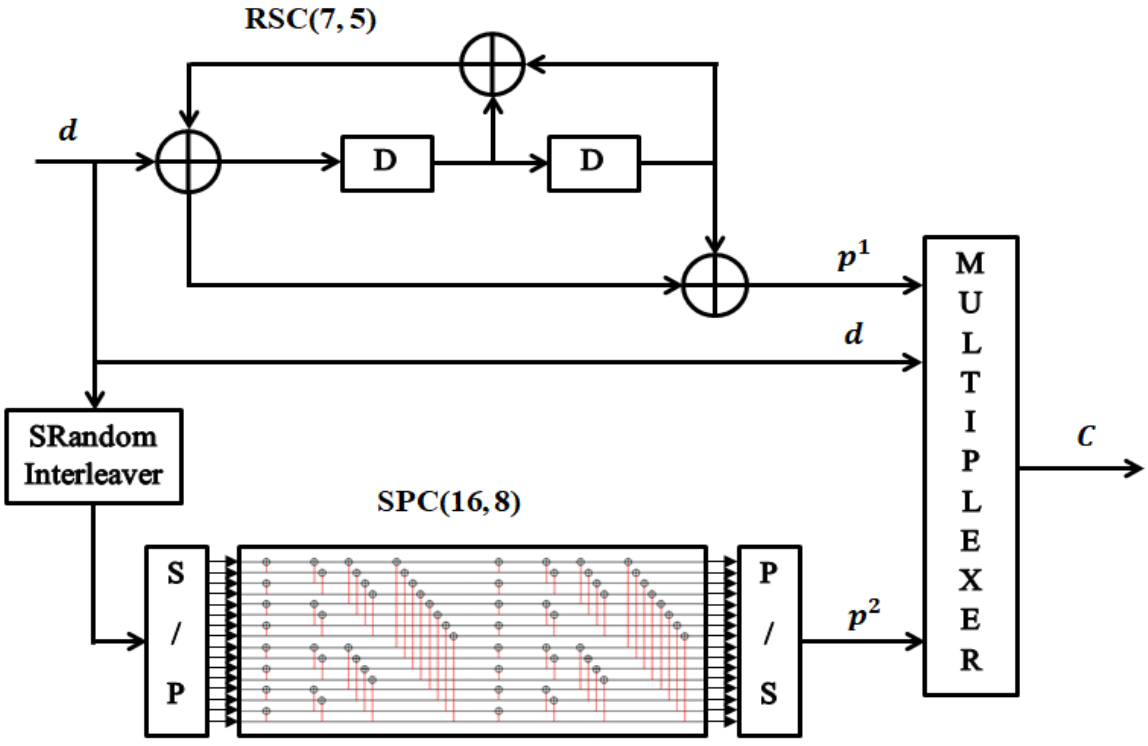


Figure (3.6): The Turbo Encoder of the Considered Code in Case Study-I.

Table (3.2): The Code Characteristics in Case Study-I

Component encoders	RSC(N=16, K=8) and SPC(N=16, K=8)
RSC parameters	n=2, k=n-1=1, *Constraint length*=3, g=$(7,5)_8$
SPC parameters	$\mathcal{F}^c = \{10,11,13,8,12,14,15,16\}$, $K = 8$
Interleaver	S-Random➔{6,3,0,1,4,7,2,5}
Trellis termination	Applied to the first encoder by adding two trellis terminating bits
Puncturing	No
Component decoders	SOVA for RSC and SCAN for SPC $I_{si} = 2$, $I_{outer} = 2$
Modulation	Binary Phase Shift Keying (*BPSK*)

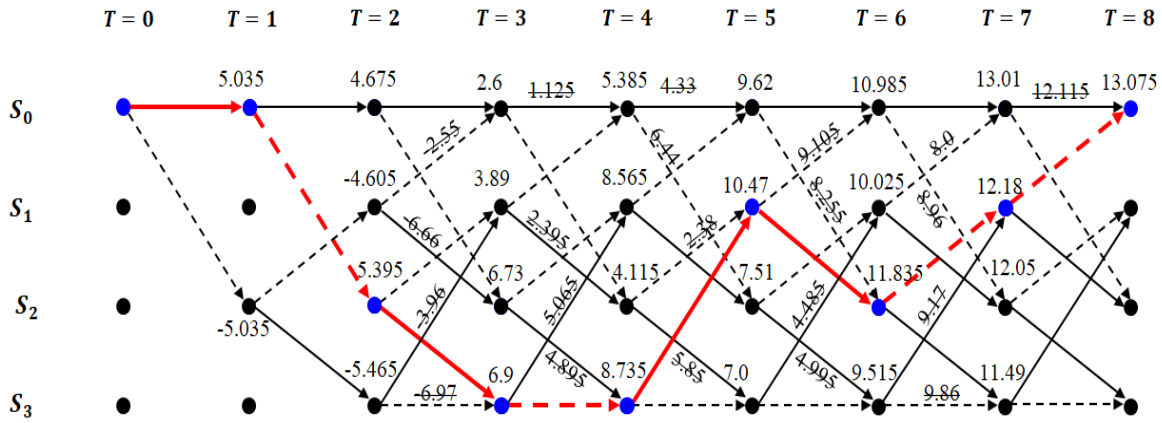
Table (3.3): Input and Transmitted Sequences of the Considered Code in Case Study-I.

Input bits	Systematic bits	Parity Bits Coder1 Coder2	Transmitted Sequence	Received Sequence($L_c * r_i$)
0	0	0 1	-1,-1,1	-3.55, -6.52, 0.13
0	0	0 1	-1,-1,1	0.79, -0.07, 3.37
1	1	1 0	1,1,-1	0.55, 3.56, -5.55
1	1	0 0	1,-1,-1	3.33, -0.34, 2.18
1	1	1 1	1,1,1	-0.68, 2.79, 0.28
0	0	0 0	-1,-1,-1	1.15, -3.88, -1.86
0	0	0 1	-1,-1,1	-1.68, -2.37, 2.56
0	0	0 1	-1,-1,1	2.55, -0.76, -0.38

I) First iteration

a) SOVA^{1st}

As stated in the previous sections, in the first iteration, there is no a-prior information from the second SISO decoder to the first SISO decoder, and hence we have $\varphi_1 = 0$ for all T, corresponding to an a-prior probability of 0.5. Figure (3.7) illustrates the trellis of the SOVA decoder in the first iteration. According to Equation (2.51), the metrics indicated at each node in Figure (3.7) are determined by cross-correlation of the expected and received channel sequences for a given path. The SOVA decoder maximizes these metrics to discover the Maximum Likelihood (ML) route, represented by the bold line in Figure (3.7). This procedure resembles the Viterbi algorithm's mechanism. Figure (3.8) is a simplified version of Figure (3.7) that displays the Maximum Likelihood path and the eliminated competing paths. In addition, the metric differences ($\Delta_{S_j}^{T}$) defined in Equation (2.52) are also shown in this Figure. Table (3.4) summarizes the values given in Figures (3.7) and (3.8).



Input $\frac{L_c * R^T}{1}$	(-3.55, -6.52)	(0.79, -0.07)	(0.55, 3.56)	(3.33, -0.34)	(-0.68, 2.79)	(1.15, -3.88)	(-1.68, -2.37)	(2.55, -0.76)
A-prior $\frac{\varphi^T}{2}$	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
Decoded bits	(0)	(1)	(0)	(1)	(0)	(0)	(1)	(1)

Figure (3.7): Trellis of SOVA Decoder in the First Iteration (Case Study-I).

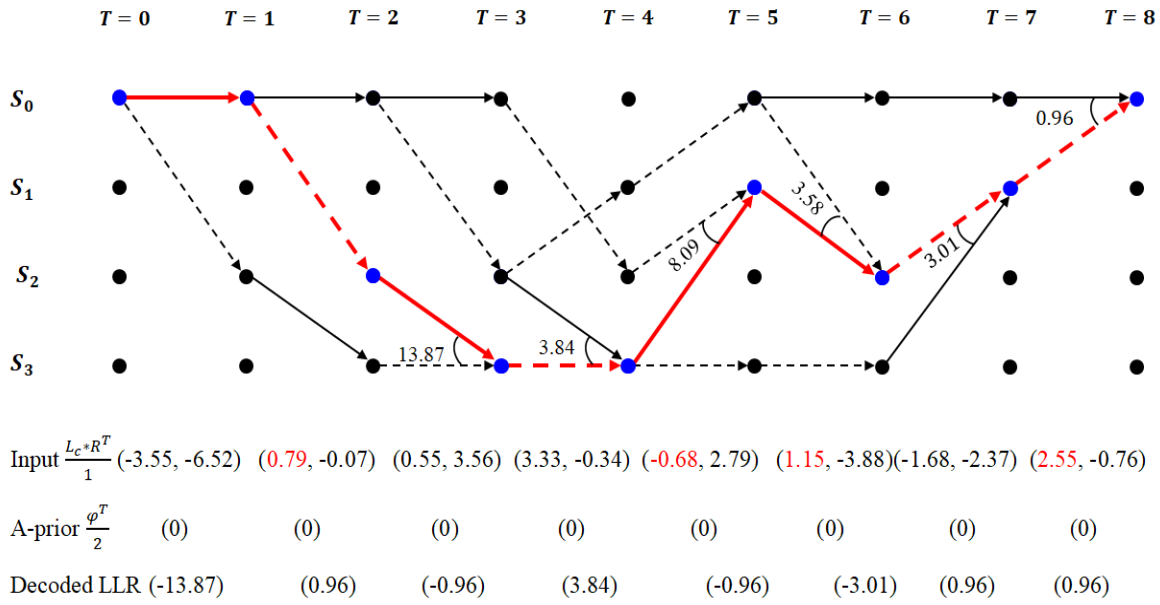


Figure (3.8): Simplified Trellis of SOVA Decoder in the First Iteration(Case Study-I).

Table (3.4): The SOVA Outputs in the First Iteration.

Trellis Stage T	Decoded Bit d^T	Matric Difference $\Delta^i_{s_{ml}}$	Update Sequence	Decoded LLR ($\mathcal{L}_T$)
1	0	-	-	-13.87
2	1	-	-	0.96
3	0	13.87	111	-0.96
4	1	3.84	1110	3.84
5	0	8.09	10010	-0.96
6	0	3.58	110110	-3.01
7	1	3.01	1110000	0.96
8	1	0.96	11010110	0.96

At T=1 & 2, there are no competing paths merged with the maximum likelihood path, so there are no matric differences and update sequences. The update sequence indicates whether the ML and competing paths are different or the same. If these paths are different at a certain T, the update sequence at this T is one, and if they are similar, the update sequence is zero. At stage T of the trellis, the Least Significant Bit (LSB) represents d^T, the next bit represents d^{T-1} and so on until the Most Significant Bit (MSB) represents d^1. The outputs ($\mathcal{L}_T$) of the SOVA decoder is estimated in Figure (3.8) and Table (3.4) according to Equation (2.53).

Table (3.5) depicts the first decoder's extrinsic information generated by Equation (3.8), which is then interleaved by the S-random interleaver given in Table (3.2) and then utilized as a-prior information for the SCAN decoder, as illustrated in Table (3.6).

Table (3.5): The Extrinsic Information of the SOVA in the 1st Iteration.

Trellis Stage T	Decoder LLR Output (b_1^T)	A-Prior Information (φ^T)	Received Systematic Info. ($L_c * R^T$)	Extrinsic Info. (ε_1)
1	-13.87	0	-3.55	-10.32
2	0.96	0	0.79	0.17
3	-0.96	0	0.55	-1.51
4	3.84	0	3.33	0.51
5	-0.96	0	-0.68	-0.28
6	-3.01	0	1.15	-4.16
7	0.96	0	-1.68	2.64
8	0.96	0	2.55	-1.59

Table (3.6): The SCAN Inputs in the 1st iteration.

Interleaved Sys. Received Info. (R_d^π)	A-prior Info. ($\varphi_2^T = \pi(\varepsilon_1)$)	Received 2nd parity info. ($R_{P^2}^\pi$)	$I_{SCAN}^{Fc}(\bar{R}_d^\pi = R_d^\pi + \varphi_2^T)$	Normal-Indexed I_{SCAN}	Inverse-Indexed I_{SCAN}
-1.68	2.64	0.13	0.96	0.13	0.13
3.33	0.51	3.37	3.84	3.37	-0.38
-3.55	-10.32	-5.55	-13.87	-5.55	0.28
0.79	0.17	2.18	0.96	2.18	-0.96
-0.68	-0.28	0.28	-0.96	0.28	-5.55
2.55	-1.59	-1.86	0.96	-1.86	-13.87
0.55	-1.51	2.56	-0.96	2.56	2.56
1.15	-4.16	-0.38	-3.01	0.96	-0.96
				-0.38	3.37
				3.84	3.84
				-13.87	-1.86
				0.96	0.96
				-0.96	2.18
				0.96	0.96
				-0.96	0.96
				-3.01	-3.01

b) SCAN$^{1^{st}}$

The SCAN decoder employs the information from the SOVA decoder as a-prior information to support its operation. So, we hope to get a better decoding performance than the performance of the first decoder. The input vector of the SCAN decoder consists of two sets corresponding to the frozen and unfrozen indices of polar construction (i.e., $\mathcal{F}^c$ & $\mathcal{F}$). The first set that corresponds to the ($\mathcal{F}^c$) set contains information about the encoded user's bits. This information is generated by adding the interleaved version of the received systematic information sequence (R_d) with the a-prior information (φ_2^T) from the SOVA decoder. On the other hand, the received information that corresponds to the parity sequence of the second component encoder is the second set corresponding to the ($\mathcal{F}$) set. Both sets are merged to generate the input vector (I_{SCAN}) of the SCAN decoder, as shown in Table (3.6). The SCAN decoder accepts the negative bit-reversal-indices version of this vector. Thus, this vector must be applied to a bit-reversal-indices algorithm and then multiplied by -1 before applying it to the SCAN decoder.

The SCAN decoder can be constructed by two matrices, namely α & β. The first (i.e., α) is used to save the LLRs corresponding to the nodes' values of the factor graph of the polar code, while the second (i.e., β) contains soft decision information about the nodes' bits. Initially, the first column of the matrix (α) is initialized by the negative inverse-indexed version of the input vector (I_{SCAN}) and the last column of the matrix (β) is initialized by a vector consisting of zero values in the unfrozen indices ($\mathcal{F}^c$) and infinity values (i.e., a large value such as 1000) in the frozen indices ($\mathcal{F}$), as illustrated in Table (3.7).

According to Equations (2.30-2.33), the matrices α & β are updated concurrently. Table (3.8) shows the timeline of updating the matrices α & β

Table (3.7): The Initial Values of α & β Matrices of the SCAN Decoder in the First Iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\alpha(4,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$	$\beta(4,:)$
-0.13	0	0	0	0	0	0	0	0	1000
0.38	0	0	0	0	0	0	0	0	1000
-0.28	0	0	0	0	0	0	0	0	1000
0.96	0	0	0	0	0	0	0	0	1000
5.55	0	0	0	0	0	0	0	0	1000
13.87	0	0	0	0	0	0	0	0	1000
-2.56	0	0	0	0	0	0	0	0	1000
0.96	0	0	0	0	0	0	0	0	0
-3.37	0	0	0	0	0	0	0	0	1000
-3.84	0	0	0	0	0	0	0	0	0
1.86	0	0	0	0	0	0	0	0	0
-0.96	0	0	0	0	0	0	0	0	0
-2.18	0	0	0	0	0	0	0	0	0
-0.96	0	0	0	0	0	0	0	0	0
-0.96	0	0	0	0	0	0	0	0	0
3.01	0	0	0	0	0	0	0	0	0

Table (3.8): The Schedule for Updating Matrices α & β.

Colors																
Time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}

according to the colours of the cells. The matrix α is updated from the upper-left corner to the lower-right corner, while the matrix β is updated from the upper-right corner to the lower-left corner, as illustrated in Table (3.9). At the first iteration of the SCAN decoder, there is no extrinsic information that the SCAN decoder can use because the matrix β is initialized by zero values except the last column. Whereas at the second iteration of the SCAN decoder and other successive iterations, the SCAN decoder exploits the extrinsic information computed with previous iterations, as illustrated by Table (3.10). After the last iteration, the output of the SCAN decoder is computed by adding the first column values of the matrix α with the first column values of the matrix β. This output vector is computed in the negative inverse-indexed

Table (3.9): The Values of α & β Matrices After 1st Iteration of the SCAN Decoder in the First Iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\alpha(4,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$	$\beta(4,:)$
-0.13	-0.13	0.13	-0.13	-0.13	-0.3	6.72	997.12	1000	1000
0.38	-0.28	-0.96	0.96	0.83	-0.81	6.87	998.21	999.87	1000
-0.28	5.55	-0.96	-0.83	-0.83	0.71	1.04	998.21	998.08	1000
0.96	-0.96	-0.96	-1.92	-2.75	-0.53	7.55	998.21	999.17	1000
5.55	3.37	-0.41	-0.41	-0.41	1.04	3.22	7	1000	1000
13.87	-0.96	4.59	0	-0.41	0.79	7.55	2	999.59	1000
-2.56	0.96	2.41	4.18	2.41	1.21	5.63	4.18	2.41	1000
0.96	-0.96	0	2.41	6.59	-2.31	7.55	6.59	4.18	0
-3.37	0.25	0.25	-0.25	-0.25	-3.22	-0.68	-0.9	0.9	1000
-3.84	0.68	-1.6	0.9	0.65	-2.97	-0.25	0.25	-0.25	0
1.86	14.91	-0.9	-1.35	1.35	-1.21	-0.25	0.25	0	0
-0.96	-1.6	-2.05	-1.8	-1.8	1.61	0.25	0.25	0	0
-2.18	-7.06	0.43	-0.43	-0.43	-0.711	0.25	0	0	0
-0.96	0.9	-1.35	0.65	0.65	-1.93	-0.25	0	0	0
-0.96	-3.14	0.65	-1.35	-1.35	2.76	0.25	0	0	0
3.01	2.05	1.8	1.8	1.8	-1.21	-0.25	0	0	0

Table (3.10): The Values of α & β Matrices After 2nd Iteration of the SCAN Decoder in the First Iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\alpha(4,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$	$\beta(4,:)$
-0.13	0.13	0.13	0.13	0.13	-0.3	5.97	1000	1000	1000
0.38	-0.28	0.79	2.97	3.1	-0.81	6.38	1000.13	1000.13	1000
-0.28	5.55	2.97	0.92	0.92	0.71	0.55	1000.13	1000	1000
0.96	-1.21	0.71	3.68	4.6	-0.53	7.31	1003.89	1000.92	1000
5.55	3.37	-0.15	-0.15	-0.15	0.55	2.73	6.25	1000	1000
13.87	-1.21	4.34	2.16	2.01	0.3	7.31	1.76	999.85	1000
-2.56	0.71	2.16	4.19	1.91	1.21	5.39	3.94	1.91	1000
0.96	-0.96	-0.25	1.91	6.1	-2.31	7.06,	6.35	4.19	0
-3.37	0.25	0.25	-0.25	-0.25	-2.73	-0.68	-0.9	0.9	1000
-3.84	0.68	-1.6	0.9	0.65	-2.48	-0.25	0.25	-0.25	0
1.86	14.42	-0.9	-1.35	1.35	-1.21	-0.25	0.25	0	0
-0.96	-1.6	-2.05	-1.8	-1.8	1.61	0.25	0.25	0	0
-2.18	-6.57	0.43	-0.43	-0.43	-0.71	0.25	0	0	0
-0.96	0.9	-1.35	0.65	0.65	-1.93	-0.25	0	0	0
-0.96	-3.14	0.65	-1.35	-1.35	2.76	0.25	0	0	0
3.01	2.05	1.8	1.8	1.8	-1.21	-0.25	0	0	0

version, as illustrated in Table (3.11). Therefore, it must be converted to the positive normal-indexed version to estimate the extrinsic information (ε_2) of the second SISO decoder according to Equation (3.9), as illustrated in Table (3.12). The hard output ($\hat{d}$) of the SCAN decoder is estimated by applying the soft output (O_{SCAN}^{Fc}) to threshold element after de-interleaving it, as illustrated in Table (3.11). After the first iteration of the iterative decoding algorithm, there are still two errors in the decoded bits, as demonstrated in this output (i.e., $\hat{d}$).

Table (3.11): The SCAN Outputs in the 1st iteration.

Inverse-Indexed O_{SCAN} (Neg)	Normal-Indexed O_{SCAN} (Neg)	O_{SCAN}^{Fc} (Neg)	$\pi^{-1}[O_{SCAN}^{Fc}]$ (Neg)	Decoded user bits ($\hat{d}$)
-0.43	-0.43	1.8	14.17	0
-0.43	-6.1	-6.32	-2.89	1
0.43	6.1	14.17	-1.35	1
0.43	-2.89	-2.89	-6.32	1
6.1	0.43	0.43	0.43	0
14.17	0.65	0.65	1.8	0
-1.35	-1.35	-1.35	1.8	0
-1.35	1.8	1.8	0.65	0
-6.1	-0.43			
-6.32	-6.32			
0.65	14.17			
0.65	-2.89			
-2.89	0.43			
-2.89	0.65			
1.8	-1.35			
1.8	1.8			

Table (3.12): The Extrinsic Information of the SCAN Decoder in the First Iteration.

Stage T	O_{SCAN}^{Fc}	A-prior Info. (φ_2^T)	Interleaved Sys. Received Info. (R_d^{π})	Extrinsic Info. (ε_2)	A-prior Info. ($\varphi_1^T = \pi^{-1}(\varepsilon_2)$)
1	-1.8	2.64	-1.68	-2.76	-0.3
2	6.32	0.51	3.33	2.48	1.93
3	-14.17	-10.32	-3.55	-0.3	2.31
4	2.89	0.17	0.79	1.93	2.48
5	-0.43	-0.28	-0.68	0.53	0.53
6	-0.65	-1.59	2.55	-1.61	1.21
7	1.35	-1.51	0.55	2.31	-2.76
8	-1.8	-4.16	2.55	1.21	-1.61

II) Second iteration

a) SOVA$^{2^{nd}}$

In the second iteration and subsequent iterations, the first SISO decoding algorithm (i.e., SOVA) can employ extrinsic information from the second SISO algorithm (i.e., SCAN) as a-prior information. This extrinsic information is de-interleaved and then utilized as a-prior information (φ_1^T) for the SOVA decoder from the SCAN decoder, as illustrated in Table (3.12) and Figure (3.9).

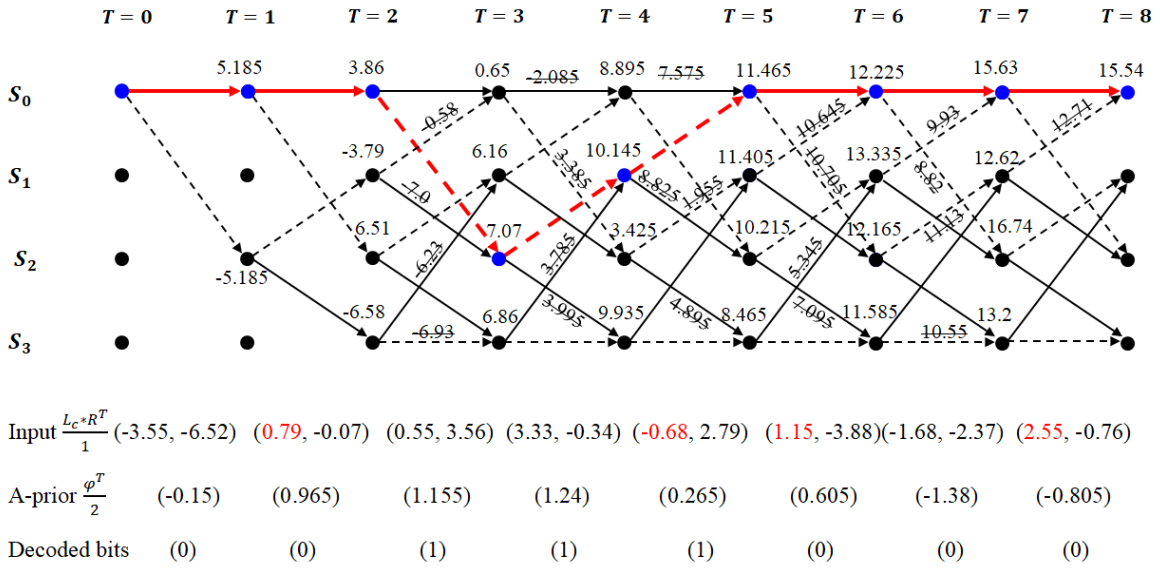
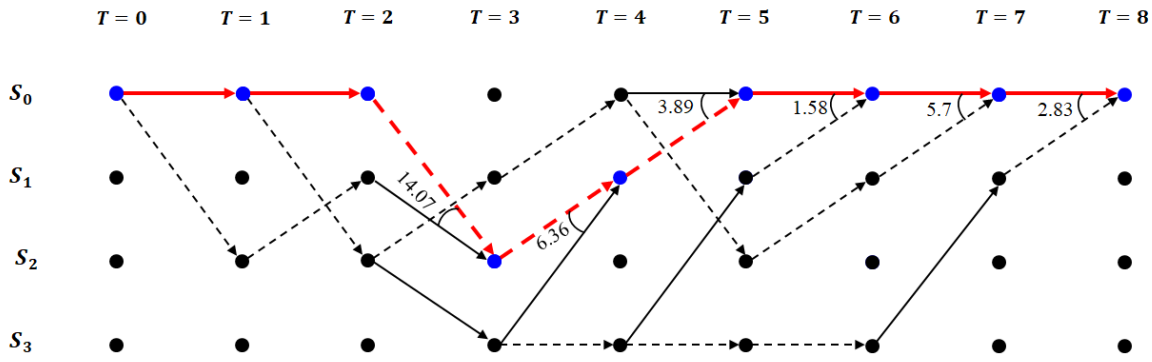


Figure (3.9): Trellis of SOVA Decoder in the 2nd Iteration(Case Study-I).

Figures (3.9) and (3.10) illustrate the trellis that is utilized by the SOVA decoder in the second iteration. It is clear that this decoder makes use of the same received information sequence that it did in the previous iteration. On the other hand, in contrast to Figures (3.7) and (3.8), Figures (3.9) and (3.10) now possess a-prior information, which will aid the SOVA decoder in selecting the appropriate path through the trellis and enable it to do it more efficiently. Once more, a bold line indicates the chosen maximum likelihood path, and it is plain to see that the correct path (i.e., 00111000) has been selected this time. Table (3.13) summarizes the values given in Figures (3.9) and (3.10).

For the second iteration, Table (3.14) depicts the first decoder's extrinsic information generated by Equation (3.8), which is then interleaved by the S-random interleaver given in Table (3.3) and then utilized as a-prior information for the SCAN decoder, as illustrated in Table (3.15). Figure (3.9) and Table (3.13) show that the SOVA decoder in the second iteration succeeded in estimating the correct path (i.e., 00111000).



Input $\frac{L_c * R^T}{1}$	(-3.55, -6.52)	(0.79, -0.07)	(0.55, 3.56)	(3.33, -0.34)	(-0.68, 2.79)	(1.15, -3.88)	(-1.68, -2.37)	(2.55, -0.76)
A-prior $\frac{\varphi^T}{2}$	(-0.15)	(0.965)	(1.155)	(1.24)	(0.265)	(0.605)	(-1.38)	(-0.805)
Decoded LLR	(-14.07)	(-1.58)	(1.58)	(6.36)	(1.58)	(-1.58)	(-5.7)	(-2.83)

Figure (3.10): Simplified Trellis of SOVA Decoder in the Second Iteration (Case Study-I).

Table (3.13): The SOVA Outputs in the Second Iteration.

Trellis Stage T	Decoded Bit d^T	Matric Difference $\Delta^i_{s_{ml}}$	Update Sequence	Decoded LLR ($\mathcal{L}_T$)
1	0	-	-	-14.07
2	0	-	-	-1.58
3	1	14.07	111	1.58
4	1	6.36	1110	6.36
5	1	3.89	10010	1.58
6	0	1.58	110110	-1.58
7	0	5.7	1100010	-5.7
8	0	2.83	10100110	-2.83

Table (3.14): The Extrinsic Information of the SOVA Decoder in the 2nd iteration.

Trellis Stage T	Decoder LLR Output (b_1^T)	A-Prior Info. (φ^T)	Received Systematic Info. ($L_c * R^T$)	Extrinsic Info. (ε_1)
1	-14.07	-0.3	-3.55	-10.22
2	-1.58	1.93	0.79	-4.3
3	1.58	2.31	0.55	-1.28
4	6.36	2.48	3.33	0.55
5	1.58	0.53	-0.68	1.73
6	-1.58	1.21	1.15	-3.94
7	-5.7	-2.76	-1.68	-1.26
8	-2.83	-1.61	2.55	-3.77

b) SCAN^{2nd}

As the SCAN's steps in the first iteration, the same steps are repeated in the second iteration, but this time with different a-prior information. Tables (3.15)-(3.20) illustrate the procedure of the SCAN decoder in the second iteration, from calculating the a-prior information to estimating the extrinsic information and decoded bits.

Table (3.19) demonstrates that the SCAN decoder in the second iteration also selects the correct path (i.e., 00111000). This result brings us to the end of our example of the iterative decoding algorithm of the parallel concatenated turbo polar-convolutional code utilizing the SOVA and SCAN algorithms.

Table (3.15): The SCAN Inputs in the 2nd iteration.

Interleaved Sys. Received Info. (R_d^{π})	A-prior Info. ($\varphi_2^T = \pi(\varepsilon_1)$)	Received 2nd parity info. ($R_{P^2}^{\pi}$)	$I_{SCAN}^{Fc}(\bar{R}_d^{\pi} = R_d^{\pi}+\varphi_2^T)$	Normal-Indexed I_{SCAN}	Inverse-Indexed I_{SCAN}
-1.68	-1.26	0.13	-2.94	0.13	0.13
3.33	0.55	3.37	3.88	3.37	-0.38
-3.55	-10.22	-5.55	-13.77	-5.55	0.28
0.79	-4.3	2.18	-3.51	2.18	1.05
-0.68	1.73	0.28	1.05	0.28	-5.55
2.55	-3.77	-1.86	-1.22	-1.86	-13.77
0.55	-1.28	2.56	-0.73	2.56	2.56
1.15	-3.94	-0.38	-2.79	-2.94	-0.73
				-0.38	3.37
				3.88	3.88
				-13.77	-1.86
				-3.51	-1.22
				1.05	2.18
				-1.22	-3.51
				-0.73	-2.94
				-2.79	-2.79

Table (3.16): The Initial Values of $\alpha\&\beta$ of the SCAN Decoder in the 2nd Iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\alpha(4,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$	$\beta(4,:)$
-0.13	0	0	0	0	0	0	0	0	1000
0.38	0	0	0	0	0	0	0	0	1000
-0.28	0	0	0	0	0	0	0	0	1000
-1.05	0	0	0	0	0	0	0	0	1000
5.55	0	0	0	0	0	0	0	0	1000
13.77	0	0	0	0	0	0	0	0	1000
-2.56	0	0	0	0	0	0	0	0	1000
0.73	0	0	0	0	0	0	0	0	0
-3.37	0	0	0	0	0	0	0	0	1000
-3.88	0	0	0	0	0	0	0	0	0
1.86	0	0	0	0	0	0	0	0	0
1.22	0	0	0	0	0	0	0	0	0
-2.18	0	0	0	0	0	0	0	0	0
3.51	0	0	0	0	0	0	0	0	0
2.94	0	0	0	0	0	0	0	0	0
2.79	0	0	0	0	0	0	0	0	0

Table (3.17): The Values of α & β Matrices After 1st Iteration of the SCAN Decoder in the Second Iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\alpha(4,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$	$\beta(4,:)$
-0.13	-0.13	-0.13	0.13	0.13	-0.95	10.3	998.31	998.78	1000
0.38	0.28	-0.73	-1.22	-1.09	-1.46	9.89	998.91	1000.13	1000
-0.28	5.55	1.22	-0.86	-0.86	-0.8	4.62	996.96	999.04	1000
-1.05	-0.73	-2.18	-0.96	-1.82	-0.03	10.9	1000.36	999.14	1000
5.55	3.37	0.15	0.15	0.15	4.62	6.8	10.02	1000	1000
13.77	1.22	4.82	0.61	0.76	4.37	8.95	5.35	1000.15	1000
-2.56	-2.18	4.59	4.97	4.97	0.98	12.35	5.58	5.2	1000
0.73	2.79	0.61	5.2	10.17	-2.31	7.38	9.56	4.97	0
-3.37	0.25	-0.25	0.25	-0.25	-3.63	-1.33	1.33	-1.33	1000
-3.88	-1.33	-1.83	-1.33	-1.08	-3.12	0.25	0.25	0.25	0
1.86	18.39	-3.08	-1.58	-1.08	0.97	-0.25	0.25	0	0
1.22	-1.83	1.33	1.08	1.08	1.61	0.25	-0.25	0	0
-2.18	-7.25	-1.08	1.08	1.08	3.26	0.25	0	0	0
3.51	3.08	-1.58	2.83	2.83	-2.43	-0.25	0	0	0
2.94	1.33	2.83	-1.58	-1.58	2.54	-0.25	0	0	0
2.79	5.73	5.48	5.48	5.48	2.69	-0.25	0	0	0

Table (3.18): The Values of α & β Matrices after 2nd Iteration of the SCAN Decoder in the Second Iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\alpha(4,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$	$\beta(4,:)$
-0.13	0.13	0.13	0.13	0.13	-0.95	9.55	1000	1000	1000
0.38	0.28	4.37	3.37	3.5	-1.46	9.4	1000.13	1000.13	1000
-0.28	5.55	3.37	4.5	4.5	-0.8	4.13	1000.13	1000	1000
-1.05	-0.98	-2.18	1.19	5.69	-0.03	10.66	1007.87	1004.5	1000
5.55	3.37	0.41	0.41	0.41	4.13	6.31	9.27	1000	1000
13.77	0.97	4.57	4.34	4.75	3.88	8.71	5.11	1000.41	1000
-2.56	-2.18	4.34	4.98	4.7	0.98	11.86	5.34	4.7	1000
0.73	2.54	0.36	4.7	9.68	-2.31	7.14	9.32	4.98	0
-3.37	0.25	-0.25	0.25	-0.25	-3.63	-1.33	1.33	-1.33	1000
-3.88	-1.33	-1.83	-1.33	-1.08	-3.12	0.25	0.25	0.25	0
1.86	17.9	-3.08	-1.58	-1.08	0.97	-0.25	0.25	0	0
1.22	-1.83	1.33	1.08	1.08	1.61	0.25	-0.25	0	0
-2.18	-7.25	-1.08	1.08	1.08	3.26	0.25	0	0	0
3.51	3.08	-1.58	2.83	2.83	-2.43	-0.25	0	0	0
2.94	1.33	2.83	-1.58	-1.58	2.54	-0.25	0	0	0
2.79	5.73	5.48	5.48	5.48	2.69	-0.25	0	0	0

Table (3.19): The SCAN Outputs in the 2nd iteration.

Inverse-Indexed O_{SCAN} (Neg)	Normal-Indexed O_{SCAN} (Neg)	O_{SCAN}^{Fc} (Neg)	$\pi^{-1}[O_{SCAN}^{Fc}]$ (Neg)	Decoded user bits ($\hat{d}$)
-1.08	-1.08	5.48	17.65	0
-1.08	-7.0	-7.0	1.08	0
-1.08	9.68	17.65	-1.58	1
-1.08	1.08	1.08	-7.0	1
9.68	-1.08	-1.08	-1.08	1
17.65	2.83	2.83	5.48	0
-1.58	-1.58	-1.58	5.48	0
-1.58	5.48	5.48	2.83	0
-7.0	-1.08			
-7.0	-7.0			
2.83	17.65			
2.83	1.08			
1.08	-1.08			
1.08	2.83			
5.48	-1.58			
5.48	5.48			

Table (3.20): The Extrinsic Info. of the SCAN Decoder in the 2nd iteration.

Stage T	O_{SCAN}^{Fc}	A-prior Info. (φ_2^T)	Interleaved Sys. Received Info. (R_d^{π})	Extrinsic Info. (ε_2)
1	-5.48	-1.26	-1.68	-2.54
2	7.0	0.55	3.33	3.12
3	-17.65	-10.22	-3.55	-3.88
4	-1.08	-4.3	0.79	2.43
5	1.08	1.73	-0.68	0.03
6	-2.83	-3.77	2.55	-1.61
7	1.58	-1.28	0.55	2.31
8	-5.48	-3.94	1.15	-2.69

3.6.2. Case Study-II

A simple example is used in this subsection to describe the iterative decoding algorithm of a Serial concatenated turbo polar-convolutional code. The characteristics of this concatenated code are listed in Table (3.21), and its encoder and decoder structures are seen in Figure (3.5). The input sequence (x) is assumed to be (101101), and the encoder trellis's initial state is zero.

The received sequence in Table (3.22) was created by adding AWGN with variance $\sigma^2 = 0.637$ (i.e., E_b/N_o=4dB) to the transmitted channel sequence. Therefore, according to Equation (2.50), the channel reliability (L_c) equals 3.14, given that the fading amplitude a=1.

Table (3.21): The Code Characteristics in Case Study-II.

Component encoders	RSC(N=16, K=8) and SPC(N=8, K=5)
RSC parameters	*n*=2, *k*=*n*-1=1, *Constraint length*=3, g=$(7,5)_8$
SPC parameters	$\mathcal{F}^c = \{4,5,6,7,8\}$, $K = 5$
Interleaver	S-Random➔{6,3,0,1,4,7,2,5}
Component decoders	SOVA for RSC and SCAN for SPC $I_{si} = 1$, $I_{outer} = 2$
Modulation	Binary Phase Shift Keying (*BPSK*)
Scaling Factors	$SF_1 = 1$, $SF_2 = 1$

Table (3.22): Input and Transmitted Sequences of the Considered Code in Case Study-II.

Input bits	Outer O/P	Interleaved Outer O/P	Inner O/P	Transmitted Sequence	Received Sequence ($L_c * r_i/2$)
1	1	1	1 1	+1 +1	2.25, 1.144
0	0	1	1 0	+1 -1	5.50, 0.99
1	0	1	1 1	+1 +1	-1.61, 4.84
1	1	0	0 0	-1 -1	-5.22, -3.76
0	0	0	0 0	-1 -1	-4.28, -5.33
	1	0	0 0	-1 -1	-7.63, -3.78
	1	0	0 0	-1 -1	-5.65, -0.64
	0	1	1 1	+1 +1	-0.78, -2.18

I) First iteration

a) SOVA^{1st}

Figure (3.11) illustrates the trellis of the SOVA decoder in the first iteration with zero a-prior values from the second SISO decoder. The metrics given at each node are generated by cross-correlation of the expected and

received channel sequences for a given path, according to Equation (2.51). The SOVA decoder maximizes these metrics to find the Maximum Likelihood (ML) path, which is seen in Figure (3.11) by the bold line. This procedure resembles the Viterbi algorithm's mechanism. Figure (3.12) is a simplified version of Figure (3.11) that displays the maximum likelihood path and the eliminated competing paths, as well as the metric differences ($\Delta_{S_j}^{T}$) defined in Equation (2.52). The values in Figures (3.11) and (3.12) are summarized in Table (3.23).

The output of the SOVA decoder in Table (3.23) shows that there is still one decoding error in the last bit position. Hopefully, the SOVA decoder can correct it at the next iteration. Table (3.24) depicts the SOVA decoder's extrinsic information generated by Equation (3.10), which is de-interleaved by the S-random interleaver given in Table (3.21) and then utilized as a-prior information for the SCAN decoder in the same iteration. These a-prior values are LLRs about the inner encoder inputs, which are the outputs of the outer encoder at the same time. Therefore they can be used as inputs for the outer SCAN decoder.

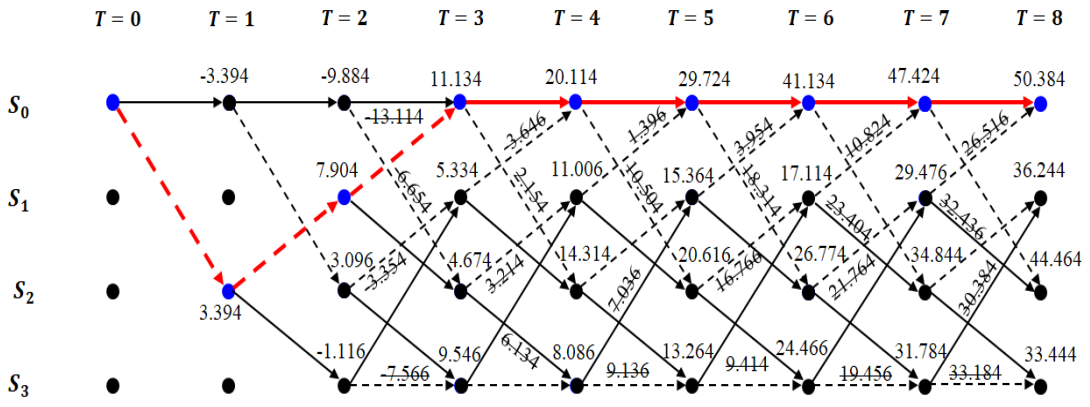


Figure (3.11): Trellis of SOVA Decoder in the First Iteration (Case Study-II).

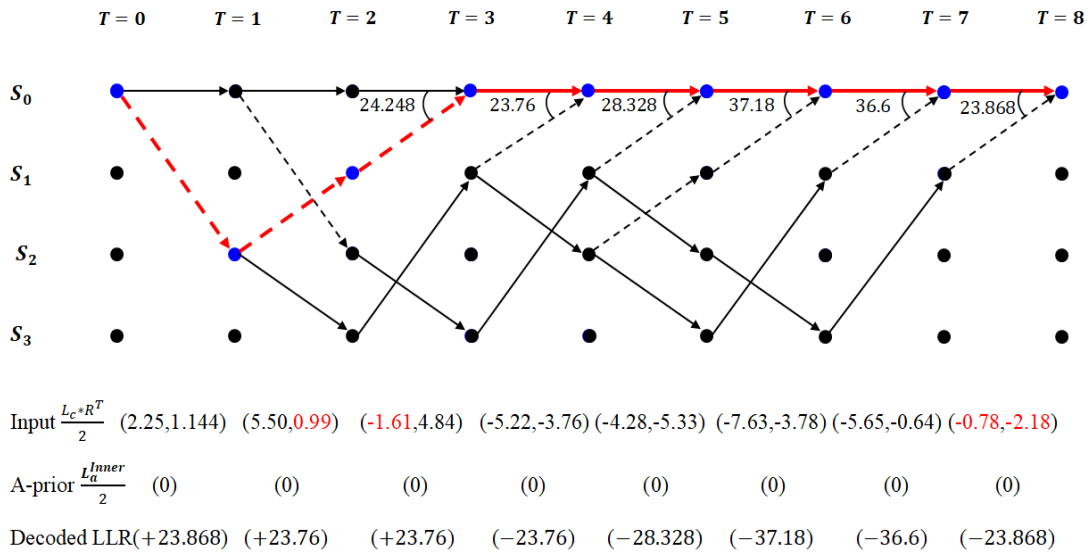


Figure (3.12): Simplified Trellis of SOVA in the 1st Iteration(Case Study-II).

Table (3.23): The SOVA Outputs in the First Iteration.

Trellis Stage T	Decoded Bit d^T	Matric Difference Δ_{sml}^{i}	Update Sequence	Decoded LLR ($\mathcal{L}_T$)
1	1	-	-	+23.868
2	1	-	-	+23.76
3	1	24.248	111	+23.76
4	0	23.76	1110	-23.76
5	0	28.328	10101	-28.328
6	0	37.18	110110	-37.18
7	0	36.6	1000110	-36.6
8	0	23.868	10000001	-23.868

Table (3.24): The Extrinsic Information of the SOVA Decoder in the 1st Iteration.

Trellis Stage T	SOVA output (O_{inner}^{inf})	A-Prior Info. (φ^{inner})	Extrinsic Info. (ε^{inner})	Outer a-Prior Info. $\varphi^{outer} = \pi^{-1}(\varepsilon^{inner}) = I_{SCAN}$	Inverse-Indexed I_{SCAN}
1	+23.868	0	+23.868	+23.76	23.76
2	+23.76	0	+23.76	-23.76	-28.328
3	+23.76	0	+23.76	-36.6	-36.6
4	-23.76	0	-23.76	+23.76	+23.868
5	-28.328	0	-28.328	-28.328	-23.76
6	-37.18	0	-37.18	-23.868	-23.868
7	-36.6	0	-36.6	+23.868	+23.76
8	-23.868	0	-23.868	-37.18	-37.18

b) SCAN$^{1^{st}}$

The outer SISO decoder in an SCTPCC uses an extrinsic sequence from the inner SISO decoder as input after de-interleaving it. To be accepted as input for the outer SCAN decoder, this normal-indexed information must be converted to the reverse-indexed version, as illustrated in Table (3.24). Since the SCAN decoder operates with negative LLR values, the inverse-indexed LLRs (I_{SCAN}) are multiplied by -1 and used as initial values for α matrix of the outer SCAN decoder, as shown in Table (3.25). This SCAN decoder follows the same procedure in Case Study-I, except that it employs a single inner iteration rather than two.

Tables (3.26) and (3.27) show the outputs and extrinsic information of the outer SCAN decoder, respectively. The hard output of the first iteration shows that the iterative decoding algorithm requires another iteration because there is still one decoding error in the third position, as depicted in Table (3.27).

Table (3.25): The Initial Values of α & β of the SCAN Decoder in the 1st iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$
-23.76	0	0	0	0	0	0	1000
+28.328	0	0	0	0	0	0	1000
+36.6	0	0	0	0	0	0	1000
-23.868	0	0	0	0	0	0	0
+23.76	0	0	0	0	0	0	0
+23.868	0	0	0	0	0	0	0
-23.76	0	0	0	0	0	0	0
+37.18	0	0	0	0	0	0	0

In the Serial Concatenated Turbo codes, the outer SISO decoder must be able to produce soft outputs for the information and parity sequences because these sequences are required to calculate the a-prior information for the inner SISO decoder. In this case study, the inner a-prior information (φ^{inner}) is

determined by the outer extrinsic information (ε^{outer}) after interleaving it according to Equation (3.13), as illustrated in Table (3.28).

Table (3.26): The Values of α & β Matrices After 1st Iteration of the SCAN Decoder in the First Iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$		$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$
-23.76	-23.76	23.76	23.76		-23.868	-23.868	976.24	1000
+28.328	-23.868	-23.76	0		23.76	-23.76	1023.76	1000
+36.6	23.76	-47.628	0		23.76	-71.388	0	1000
-23.868	-23.76	0	-47.628		-23.76	-23.868	-47.628	0
+23.76	52.088	-47.628	-0.108		-23.868	0	0	0
+23.868	-47.628	0.108	0.108		-23.76	0	0	0
-23.76	0.108	-47.628	-47.628		-23.868	0	0	0
+37.18	60.94	60.94	60.94		23.76	0	0	0

Table (3.27): The SCAN Outputs in the 1st iteration.

Inverse-Indexed O_{SCAN} (Neg)	Normal-Indexed O_{SCAN} (Neg)	O_{SCAN}^{Fc} (Neg)	Decoded user bits ($\hat{d}$)
-47.628	-47.628	-47.628	1
52.088	-0.108	52.088	0
60.36	60.36	0.108	0
-47.628	-47.628	-47.628	1
-0.108	52.088	60.94	0
0.108	0.108		
-47.628	-47.628		
60.94	60.94		

Table (3.28): The Extrinsic Information of the SCAN Decoder in the 2nd iteration.

Stage T	O_{SCAN}	A-prior Info. (φ^{outer})	Extrinsic Info. (ε^{outer})	Inner a-prior Info. $\varphi^{inner} = \pi(\varepsilon^{outer})$
1	47.628	+23.76	23.868	23.76
2	0.108	-23.76	23.868	23.868
3	-60.36	-36.6	-23.76	23.868
4	47.628	+23.76	23.868	23.868
5	-52.088	-28.328	-23.76	-23.76
6	-0.108	-23.868	23.76	-23.76
7	47.628	+23.868	23.76	-23.76
8	-60.94	-37.18	-23.76	23.76

II) Second Iteration

a) SOVA^{2nd}

The inner SISO decoding algorithm (i.e., SOVA) can use extrinsic information from the outer SISO algorithm (i.e., SCAN) as a-prior information in the second and subsequent iterations. As shown in Table (3.28) and Figure (3.13), this extrinsic information is interleaved and used as a-prior information for the inner SOVA decoder from the outer SCAN decoder.

Figures (3.13) and (3.14) show the trellis used by the SOVA decoder in the second iteration. Evidently, this decoder uses the same received information sequence as it did in the previous iteration, but this time with a-prior information. This a-prior information will help the SOVA decoder to choose the correct path through the trellis and enable it to do so more efficiently. Once more, the bold line in Figures (3.13) and (3.14) indicates the chosen maximum likelihood path, and it is plain to see that the correct path (i.e., 11100001) has been selected this time. The inner extrinsic information can be computed using Equation (3.10) and LLR values in Table (3.29) and then used as input for the outer SCAN decoder, as illustrated in Table (3.30).

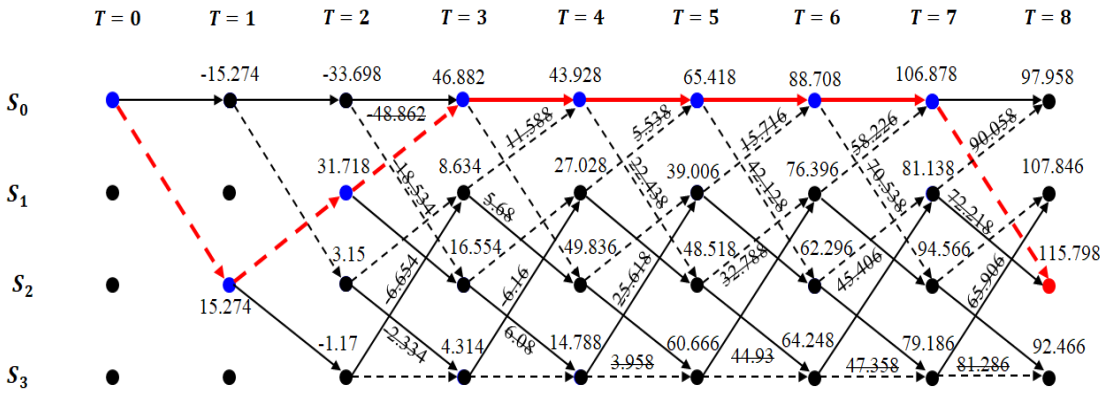


Figure (3.13): Trellis of SOVA Decoder in the 2nd Iteration (Case Study-II).

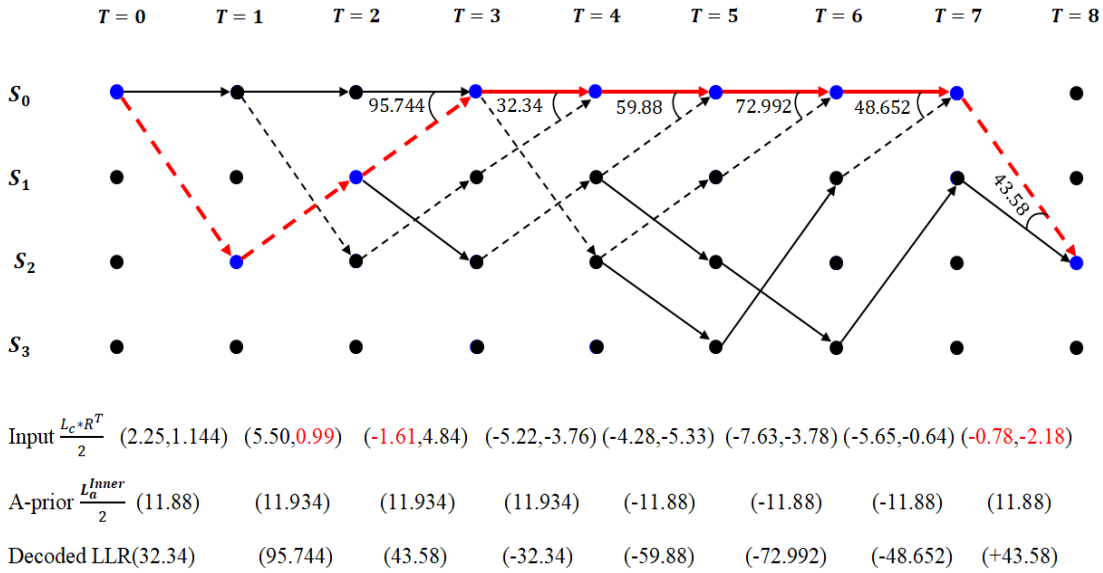


Figure (3.14): Simplified Trellis of SOVA in the 2nd Iteration(Case Study-II).

Table (3.29): The SOVA Outputs in the Second Iteration.

Trellis Stage T	Decoded Bit d^T	Matric Difference $\Delta_{s_{ml}}^{i}$	Update Sequence	Decoded LLR ($\mathcal{L}_T$)
1	1	-	-	32.34
2	1	-	-	95.744
3	1	95.744	111	43.58
4	0	32.34	1001	-32.34
5	0	59.88	11100	-59.88
6	0	72.992	111000	-72.992
7	0	48.652	1001000	-48.652
8	1	43.58	10001100	+43.58

Table (3.30): The Extrinsic Information of the SOVA in the 2nd Iteration.

Trellis Stage T	SOVA output (O_{inner}^{inf})	A-Prior Info. (φ^{inner})	Extrinsic Info. (ε^{inner})	Outer a-Prior Info. $\varphi^{outer} = \pi^{-1}(\varepsilon^{inner}) = I_{SCAN}$	Inverse-Indexed I_{SCAN}
1	32.34	23.76	8.58	19.712	19.712
2	95.744	23.868	71.876	-56.208	-36.12
3	43.58	23.868	19.712	-24.892	-24.892
4	-32.34	23.868	-56.208	71.876	8.58
5	-59.88	-23.76	-36.12	-36.12	-56.208
6	-72.992	-23.76	-49.232	19.82	19.82
7	-48.652	-23.76	-24.892	8.58	71.876
8	+43.58	23.76	19.82	-49.232	-49.232

b) SCAN^{2nd}

The same steps of the SCAN decoder in the previous iteration are repeated in the second iteration but with a different input sequence. Tables (3.30)-(3.34) illustrate the procedure of the SCAN decoder in the second iteration, from calculating the outer a-prior information (φ^{outer}) to estimate the extrinsic information and decoded bits. Table (3.33) demonstrates that the SCAN decoder in the second iteration has corrected all bit errors and has obtained the correct transmitted user bits (i.e., 10110). This result brings us to the end of our example of the iterative decoding algorithm of the serial concatenated turbo polar-convolutional code utilizing the SOVA and SCAN algorithms.

Table (3.31): The Initial Values of α & β of the SCAN Decoder in the 1st iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$
-19.712	0	0	0	0	0	0	1000
36.12	0	0	0	0	0	0	1000
24.892	0	0	0	0	0	0	1000
-8.58	0	0	0	0	0	0	0
56.208	0	0	0	0	0	0	0
-19.82	0	0	0	0	0	0	0
-71.876	0	0	0	0	0	0	0
49.232	0	0	0	0	0	0	0

Table (3.32): The Values of α & β Matrices After 1st Iteration of the SCAN Decoder in the Second Iteration.

$\alpha(0,:)$	$\alpha(1,:)$	$\alpha(2,:)$	$\alpha(3,:)$	$\beta(0,:)$	$\beta(1,:)$	$\beta(2,:)$	$\beta(3,:)$
-19.712	-19.712	8.58	8.58	-36.12	-77.632	1000	1000
36.12	-8.58	19.82	28.4	19.712	-88.764	1008.58	1000
24.892	-19.82	-28.292	28.292	8.58	-77.524	-69.052	1000
-8.58	-49.232	-69.052	-97.344	-24.892	-48.112	-28.292	0
56.208	55.832	-33.472	33.472	19.82	0	0	0
-19.82	-33.472	-76.028	-76.028	-56.208	0	0	0
-71.876	-76.028	-33.472	-33.472	-48.112	0	0	0
49.232	97.344	97.344	97.344	48.112	0	0	0

Table (3.33): The SCAN Outputs in the 2nd iteration.

Inverse-Indexed O_{SCAN} (Neg)	Normal-Indexed O_{SCAN} (Neg)	O_{SCAN}^{Fc} (Neg)	Decoded user bits ($\hat{d}$)
-55.832	-55.832	-119.988	1
55.832	76.028	55.832	0
33.472	33.472	-76,028	1
-33.472	-119.988	-33.472	1
76.028	55.832	97.344	0
-76.028	-76,028		
-119.988	-33.472		
97.344	97.344		

Table (3.34): The Extrinsic Information of the SCAN Decoder in the 2nd iteration.

Stage T	O_{SCAN}	A-prior Info. (φ^{outer})	Extrinsic Info. (ε^{outer})	Inner a-prior Info. $\varphi^{inner} = \pi(\varepsilon^{outer})$
1	55.832	19.712	36.12	24.892
2	-76.028	-56.208	-19.82	48.112
3	-33.472	-24.892	-8.58	36.12
4	119.988	71.876	48.112	-19.82
5	-55.832	-36.12	-19.712	-19.712
6	76,028	19.82	56.208	-48.112
7	33.472	8.58	24.892	-8.58
8	-97.344	-49.232	-48.112	56.208

4

Chapter Four

Simulation and Practical Results

Chapter Four

Simulation and Practical Results

4.1. Introduction

In This Chapter, several schemes of polar codes are simulated and implemented to overcome some challenges, such as achieving high throughput and low complexity, improving finite-length codes performance, and enhancing the error floor performance of Turbo Polar Codes. These schemes are designed using Matlab R2020b and C language\Visual Studio 2017 on PC-Cori7-16GbyteRAM. In addition, they are implemented using the Kintex-7 FPGA device from Xilinx. The FPGA implementation of these schemes is performed based on Vivado HLS and SDK platforms.

The simulation results of the proposed schemes are compared with other state-of-art polar codes in terms of complexity, resource consumption, throughput, Bit-Error-Rate (BER), and Frame-Error-Rate (FER) performance. In this chapter, the AWGN channel has been considered to simulate these codes with an SNR range from 0dB to 5dB. The studied schemes in this chapter can be classified into three categories: Original Systematic and Non-Systematic polar codes, Parallel Concatenated Turbo Polar codes, and Serial Concatenated Turbo Polar codes.

4.2. Non-Systematic and Systematic Polar Codes

According to Equations (2.13), (2.19), and (2.20), the Non-Systematic and Systematic polar codes can be implemented. The construction of these codes is based on determining the sets of frozen and free bits (i.e., $\mathcal{F}$ & $\mathcal{F}^c$). In this section, the polar construction method is designed on a Signal-to-Noise ratio of 0.3dB. Furthermore, all encoded sequences presented in this section are decoded using the SCAN decoder.

4.2.1. FPGA Implementation of the SPC and NSPC

The FPGA implementation of polar codes is based on constructing the IPs of the encoder and decoder. These IPs are built based on algorithms 1-4 in Appendix-I using Vivado HLS and Vivado IP integrator. Figure (4.1) shows the practical design of the systematic polar code, including the IPs of the systematic encoder, SCAN decoder, AWGN channel, and pseudo-random binary source. For the Non-Systematic polar codes, the same circuit in Figure (4.1) is used with the required modifications to the encoder and decoder's IPs. The FPGA implementation of each system suffers from a trade-off between maximizing performance and minimizing resource utilization. This gap between performance and recourse utilization has been minimized in this study by using pipeline directives and arbitrary precision data types.

The arbitrary precision data types include integer and fixed-point types. The integer type is defined as ap_[u]int<W>. The signed integer type is defined as “ap_int<W>”, while the definition “ap_uint<W>” is used to define the unsigned integer number. The bit-size <W> in the previous definitions is an integer number ranging from 1 to 1024. On the other side, the fixed-point type is defined as: “ap_[u]fixed<W, I, Q, O, N>”. The W and I are the bits number used to represent an entire float number and the integer part of this number, respectively. Moreover, the Q and O values are used to define the Quantization and Overflow modes. Finally, the N value defines the saturation bits number in overflow wrap modes. The user guide in [104] can be consulted for further information.

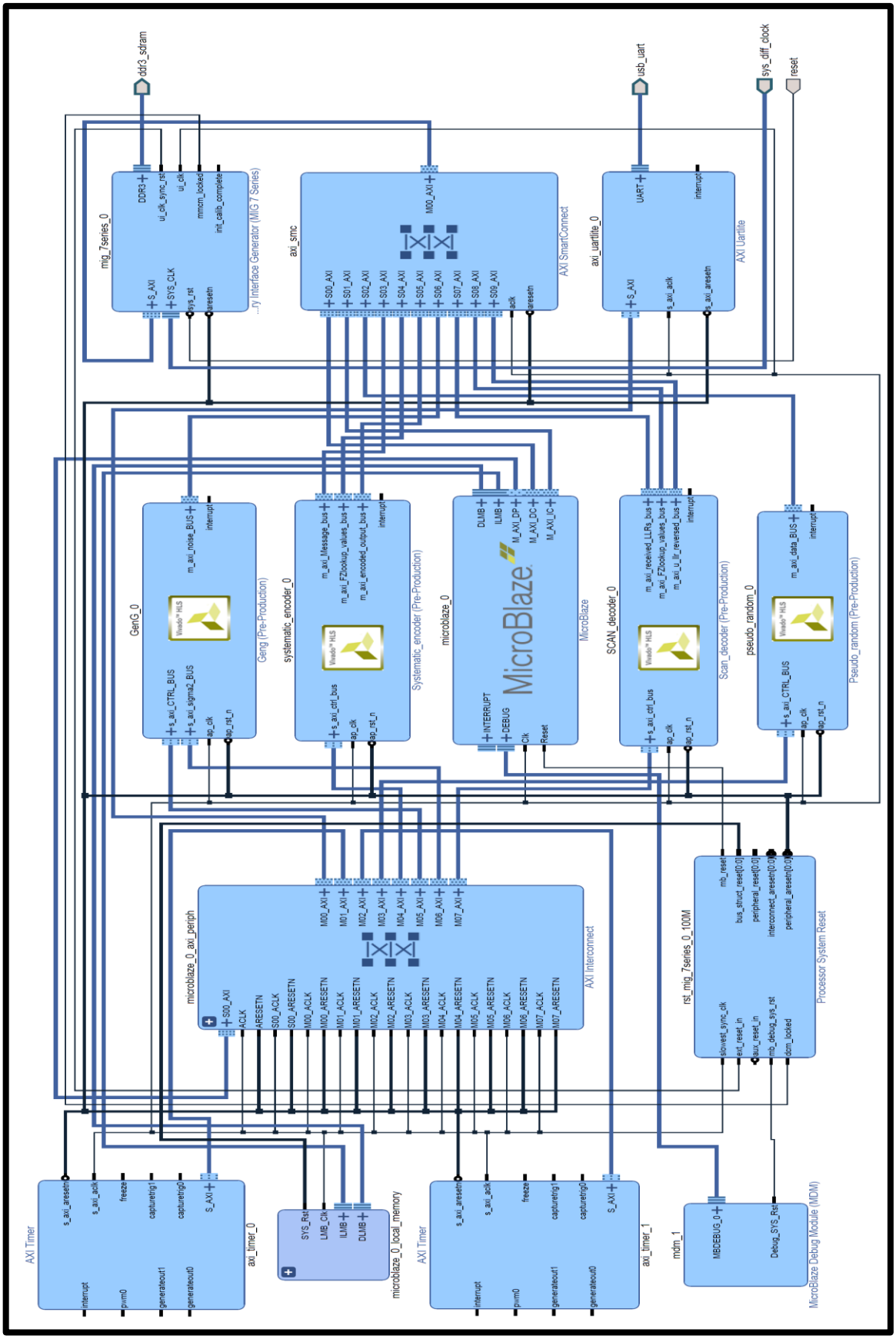


Figure (4.1): The FPGA Implementation of the Systematic Polar Code.

Table (4.1) presents the resource utilization of the SCAN decoder implementation with several code lengths and four solutions. The first solution represents the original implementation of the SCAN decoder without directives or an arbitrary precision data type, while the second solution is designed with a pipelining directive. The second solution is re-implemented in the third solution using both integer and fixed-point arbitrary precision data types. Whereas in the fourth solution, the second solution is re-implemented by using only the integer arbitrary precision. The integer arbitrary precision data type is defined in the third and fourth solutions with a width (W) ranging from one to twelve bits. While the fixed-point arbitrary precision data type is defined in the third solution as ap_fixed<16, 8, AP_RND_INF, AP_SAT>.

On the other side, Table (4.2) shows the resource utilization of the implementation of the polar encoders with different code lengths and three solutions. The first two solutions match the first two of the decoder, which are the original and the pipelining versions, while the third solution has been implemented with the pipeline directive and the arbitrary precision integer data types. Furthermore, as an indication of performance, Table (4.3) illustrates the throughputs of the polar encoders and the SCAN decoder for code length (N=256, K=128) as a case study.

Table (4.1) shows that the first and the fourth solutions achieve the best resource utilization, but given the performance in Table (4.3), the fourth solution outperforms the first. Table (4.3) shows that the fourth solution of the SCAN decoder achieves the best performance compared to the other solutions. Moreover, Figure (4.2) shows that all solutions have the same BER and FER performance. Therefore the fourth solution can be chosen as a golden solution. The resource utilization and throughput of the Systematic and Non-Systematic SCAN decoders are almost identical, so the presented

results in Tables (4.1) and (4.3) can be used to analyze both the Systematic and Non-Systematic SCAN decodes.

Table (4.1): The resource utilization of the SCAN decoder with several code lengths

Code Type	Solutions	BRAM18K	DSP48E	FF	LUT
SCAN (N=64, K=32)	D1	10	19	4260	7705
	D2	14	22	5068	8097
	D3	12	13	6778	22208
	D4	14	15	4557	7039
SCAN (N=128, K=64)	D1	10	15	4166	7122
	D2	14	18	4547	7646
	D3	12	13	5917	22205
	D4	14	15	4049	6900
SCAN (N=256, K=128)	D1	22	19	4262	7862
	D2	26	22	4648	8180
	D3	18	13	6207	22471
	D4	26	15	4140	7137
SCAN (N=512, K=256)	D1	38	19	4259	7634
	D2	42	22	4743	8192
	D3	26	13	6384	22352
	D4	42	15	4196	7189
SCAN (N=1024, K=512)	D1	70	28	4418	7230
	D2	74	31	4680	7577
	D3	42	26	6417	22198
	D4	74	28	4407	7039

Solution D1➔without pipelining + Standard variables types

Solution D2➔with pipelining + Standard variables types

Solution D3➔with pipelining + Integer&Fixed-point Arbitrary Precision types

Solution D4➔with pipelining + Integer Arbitrary Precision type

Table (4.2) shows that the resource utilizations of the Systematic polar Encoder (SPE) and Non-Systematic Polar Encoder (NSPE) in the first

solution are the perfect choices compared with other solutions. However, we can note from Table (4.3) that the performance of these encoders in the first solution is inferior to that of other solutions. Therefore, when considering the performance and resource utilization together, the third solution can be selected as the best solution compared to other solutions.

Table (4.2): The resource utilization of polar encoders with several code lengths.

Code Type	Solutions	BRAM18K	DSP48E	FF	LUT
SPE (N=256, K=128)	E1	10	0	2282	3006
	E2	12	0	5047	17017
	E3	12	0	2658	6657
SPE (N=512, K=256)	E1	40	1	2384	3182
	E2	42	128	17414	26441
	E3	42	0	3264	10759
SPE (N=128, K=85)	E1	10	0	2306	3038
	E2	12	0	4824	15130
	E3	12	0	2640	8335
SPE (N=256, K=85)	E1	10	0	2325	3051
	E2	12	0	4850	15324
	E3	12	0	2664	8525
NSPE (N=256, K=128)	E1	26	0	2191	2895
	E2	26	0	14823	19783
	E3	23	0	2795	6599
Modified NSPE (N=256, K=128)	E1	11	0	2166	2984
	E2	11	0	2199	3037
	E3	11	0	2079	2889
Solution E1 ➔ without pipelining + Standard variables types Solution E2 ➔ with pipelining + Standard variables types Solution E3 ➔ with pipelining + Integer Arbitrary Precision type					

Table (4.3): The Throughputs of SPC & NSPC Encoders and SCAN Decoder with Code Length(256,128).

Codes	C/S	Solution1	Solution2	Solution3	Solution4
SCAN (N=256, K=128)	Thr. (b/s)	88.888K	0.12138M	0.113926M	0.124102M
SPE (N=256,K=128)	Thr. (b/s)	0.1894M	0.73476M	0.73476M	0.73476M
NSPE (N=256,K=128)	Thr. (b/s)	32.3426K	77.42173K	77.42173K	77.42173K
Modified NSPE (N=256,K=128)	Thr. (b/s)	2.92569M	2.92853M	3.3551M	3.3551M
Solution1➔ SolutionD1+ SolutionE1 Solution2➔ SolutionD2+ SolutionE2				Solution3➔ SolutionD3+ SolutionE3 Solution4➔ SolutionD4+ SolutionE3	

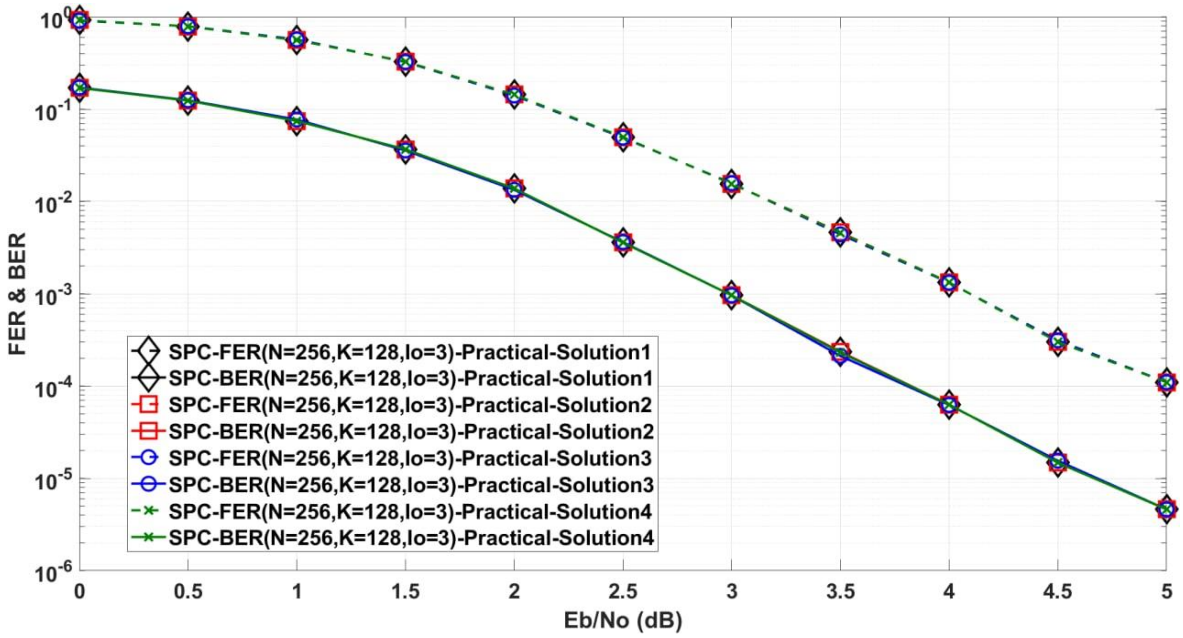
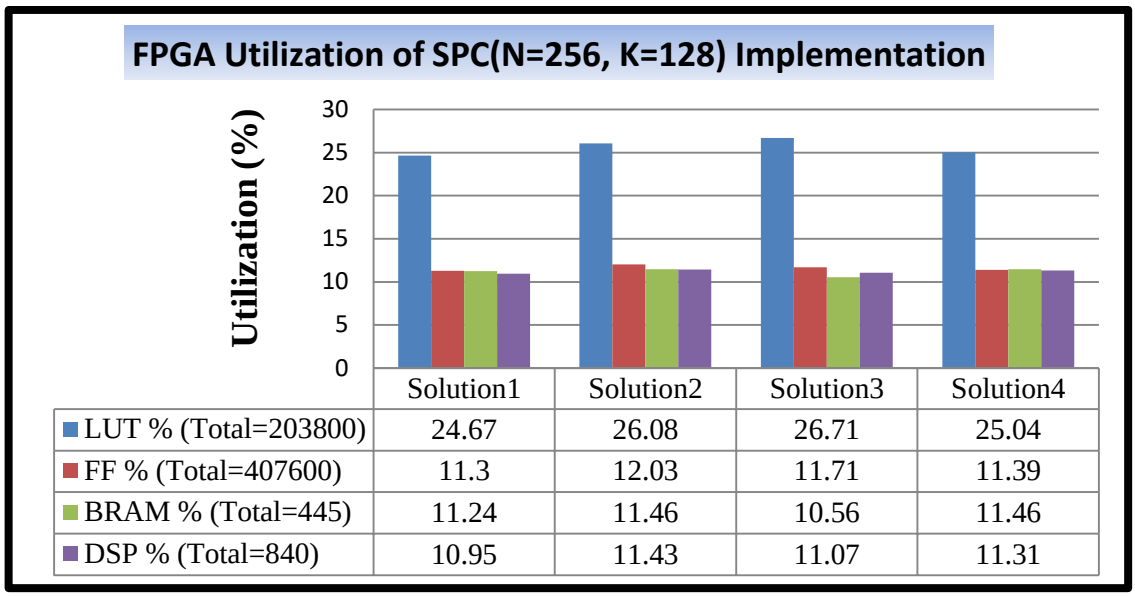


Figure (4.2): The Practical Performance of the SPC(256,128) with 4 Solutions

The FPGA implementation of the original non-systematic encoder depicted in algorithm-2 (Appendix-1) consumes a lot of resources and with lower throughput, as shown in Tables (4.2) and (4.3). Therefore, the modified Non-Systematic Encoder in algorithm-3 is suggested to enhance the resource utilization and performance of the original polar encoder. Tables (4.2) and (4.3) show that the modified polar encoder outperforms the original NSPE regarding resource utilization and throughput.

Figures (4.3) and (4.4) depict the resource utilization of the overall Systematic and Non-Systematic polar systems (i.e., including binary source, Encoder, AWGN channel, and Decoder) with the four solutions depicted in Table (4.3). These Figures (i.e., 4.3 and 4.4) show that resource utilizations of all solutions are close to each other, with little progression to the first and fourth solutions over the others. The differentiation between these solutions can be expanded by entering the performance in Table (4.3) in this issue. Table (4.3) shows that the fourth solution, which includes solution-E3 and Solution-D4 of encoder and decoder, outperforms the other solutions. In the fourth solution, the Pipelining and the arbitrary precision integer data type play a central role in reducing latency and resource utilization for the system as a whole.

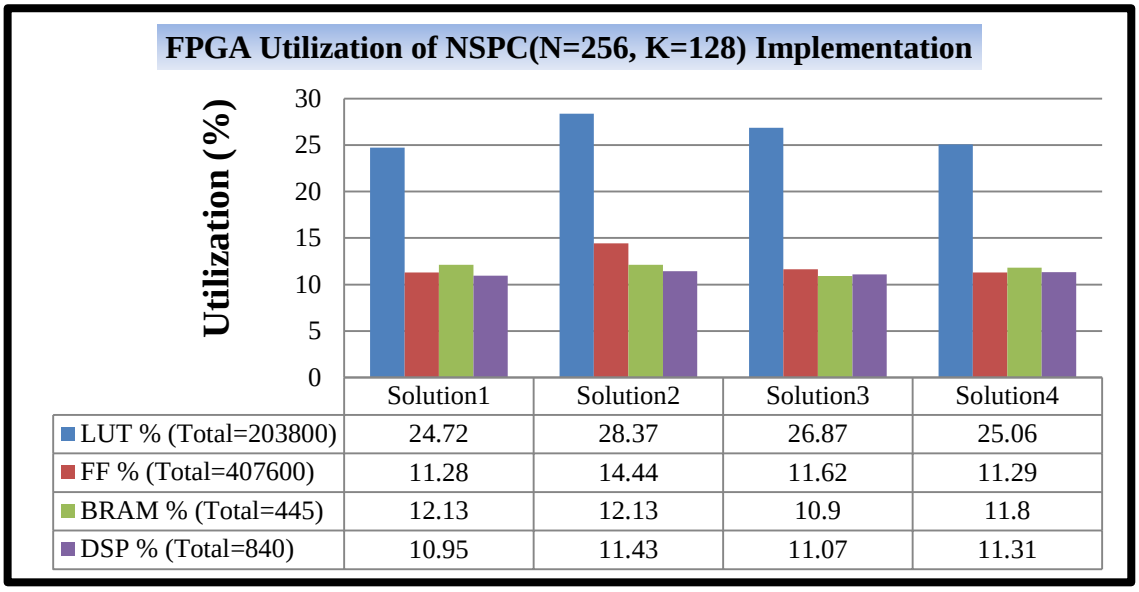


	Solution1	Solution2	Solution3	Solution4
LUT % (Total=203800)	24.67	26.08	26.71	25.04
FF % (Total=407600)	11.3	12.03	11.71	11.39
BRAM % (Total=445)	11.24	11.46	10.56	11.46
DSP % (Total=840)	10.95	11.43	11.07	11.31

Figure (4.3): The Resource Utilization of the FPGA Implementation of the SPC(N=256, K=128).

Finally, we can conclude that the original Non-Systematic Polar Encoder (NSPE) consumes more resources and processing time than the systematic Polar Encoder (SPE), as shown in Tables (4.2) and (4.3). The SPE consumes less time and resources because it requires fewer operations and deals with

relative small matrices (i.e., $G_{\mathcal{F}^c{}_{\lambda^c}}$ and $G_{\mathcal{F}^c{}_{\lambda}}$) compared to the Kronecker matrix ($F^{\otimes n}$) needed in the NSPE.



	Solution1	Solution2	Solution3	Solution4
LUT % (Total=203800)	24.72	28.37	26.87	25.06
FF % (Total=407600)	11.28	14.44	11.62	11.29
BRAM % (Total=445)	12.13	12.13	10.9	11.8
DSP % (Total=840)	10.95	11.43	11.07	11.31

Figure (4.4): The Resource Utilization of the FPGA Implementation of the NSPC(N=256, K=128).

4.2.2. Simulation Result of the SPC and NSPC

This section compares the performance of Non-Systematic and Systematic polar codes using simulation results from three different polar scenarios. The first case includes four polar codes with code rates 1/2 and lengths 128 and 1024, while in the second case, four polar codes are presented with lengths 128 and rates 1/3 and 2/3. Finally, in the third case, the simulation and practical results of the polar code (64,32) are compared. All codes in this sub-section are simulated with a three-iterations SCAN decoder and a maximum error limit of 1000 frames. The maximum error limit is defined as the maximum number of frames in which an error occurs during the performance estimation of a specific E_b/N_o.

Figure (4.5) compares the simulation results of four polar codes, namely SPC(N=128, K=64), SPC(N=1024, K=512), NSPC(N=128, K=64), and NSPC(N=1024, K=512). This Figure shows that the BER and FER performance of polar codes are improved as the code length increases, where

SPC(N=1024, K=512) has about 1.5dB gain over the SPC(N=128, K=64) at BER and FER of 10^{-4}. This enhancement is due to the improvement of channel polarization and the minimum Hamming distance by increasing the code length. Figure (4.5) also shows that the FER performance of the systematic and the Non-systematic polar codes is identical, while the BER performance of the systematic codes is better than that of the Non-systematic codes. Where the SPC(N=1024, K=512) and SPC(N=128, K=64) have about 0.25dB and 0.2dB gains over the NSPC(N=1024, K=512) and NSPC(N=128, K=64), respectively, at BER of 10^{-4}.

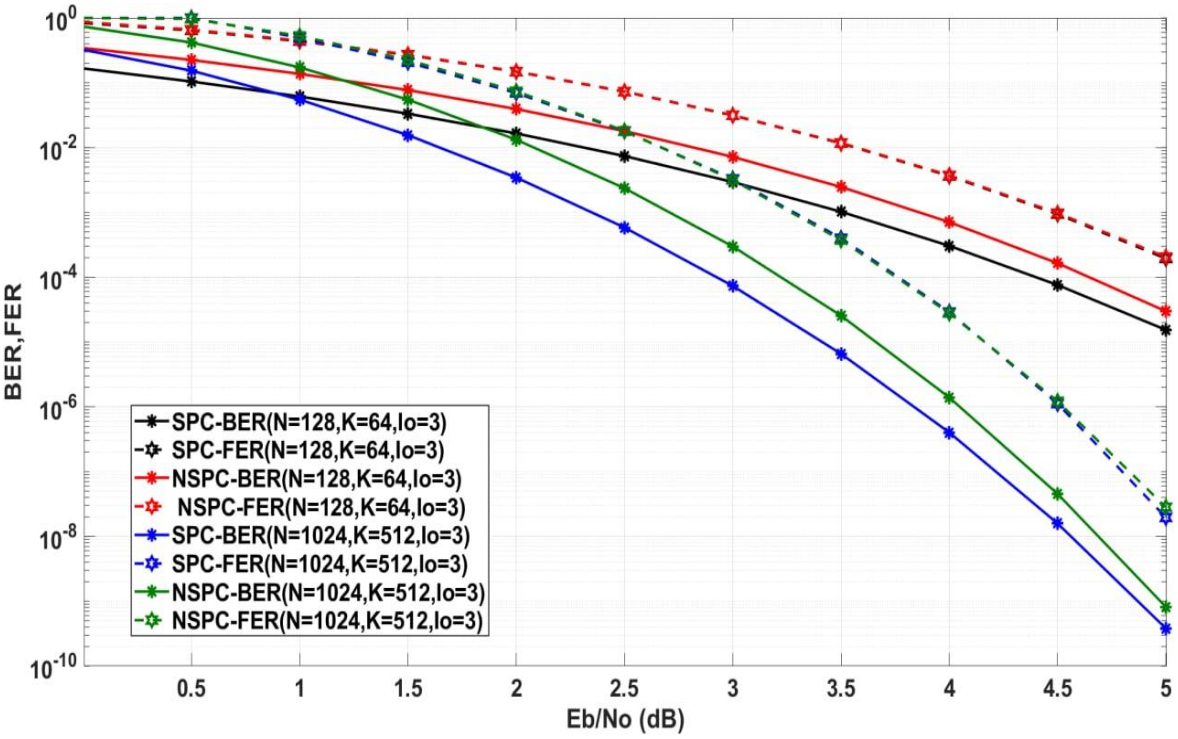


Figure (4.5): The Simulation Results of Polar Codes with a Rate of 1/2.

As shown earlier in the second chapter (i.e., Section 2.5), the channel-splitting process creates three groups of channels: good, bad, and oscillating. The number of good and bad channels is almost equal, while the number of oscillating channels is much less than that of the good and bad channels, especially with increasing the code length. Therefore, high-performance polar codes can be obtained using only good channels to transmit information.

Thus, we get polar codes with a rate of less than one-half. These codes perform better than those with a rate of one-half or more, as depicted in Figure (4.6). Figures (4.5) and (4.6) show that the polar codes with a rate of 1/3 have BER and FER performance better than those of polar codes with rates of 1/2 and 2/3. Compared to SPC(N=128, K=85) and NSPC(N=128, K=85), SPC(N=128, K=32) and NSPC(N=128, K=32) have about 0.5dB and 0.65dB performance gains, respectively, at BER of 10^{-4}.

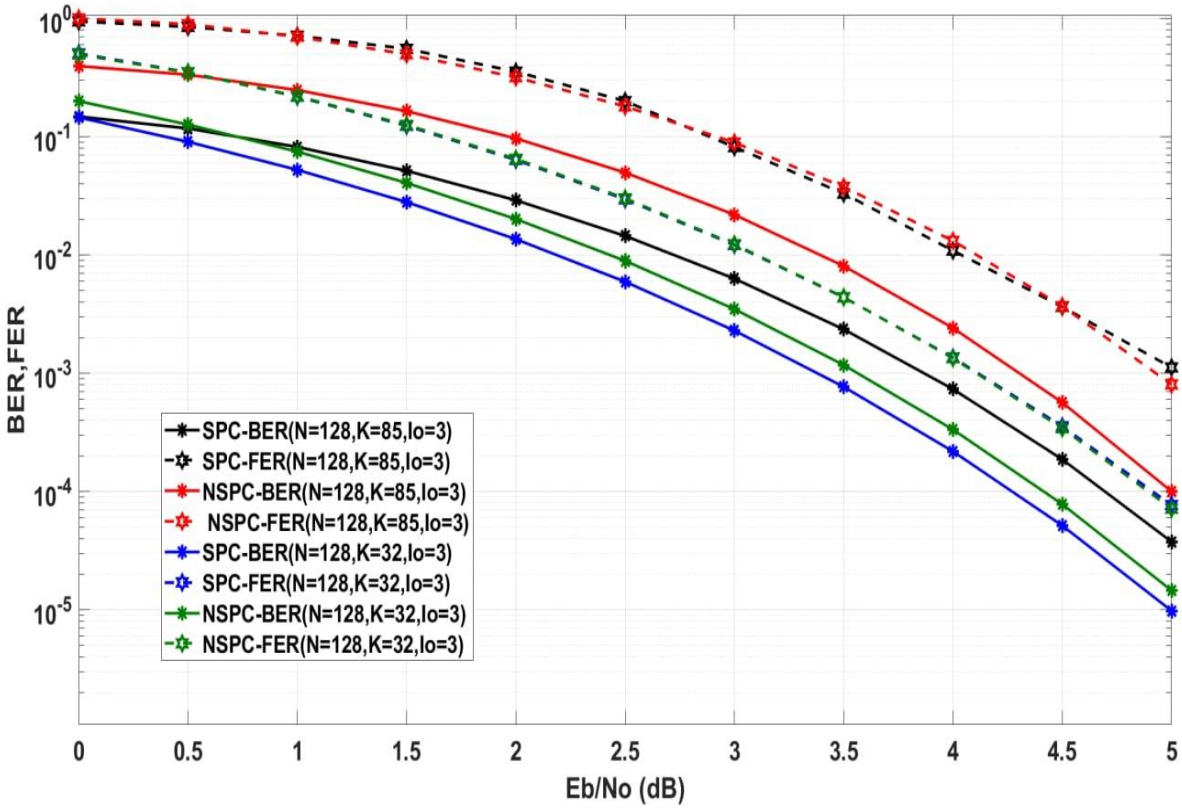


Figure (4.6): The Simulation Results of Polar Codes with Rates of 1/3 & 2/3.

Finally, Figure (4.7) compares the simulation and practical SDK results of SPC(N=64, K=32) and NSPC(N=64, K=32) in order to optimize the FPGA implementation of the systematic and non-systematic polar codes discussed in the previous subsection. This comparison demonstrates that the performance of these codes is identical. Therefore we can conclude that our FPGA implementation in subsection 4.2.1 is functionally equivalent to the simulation design

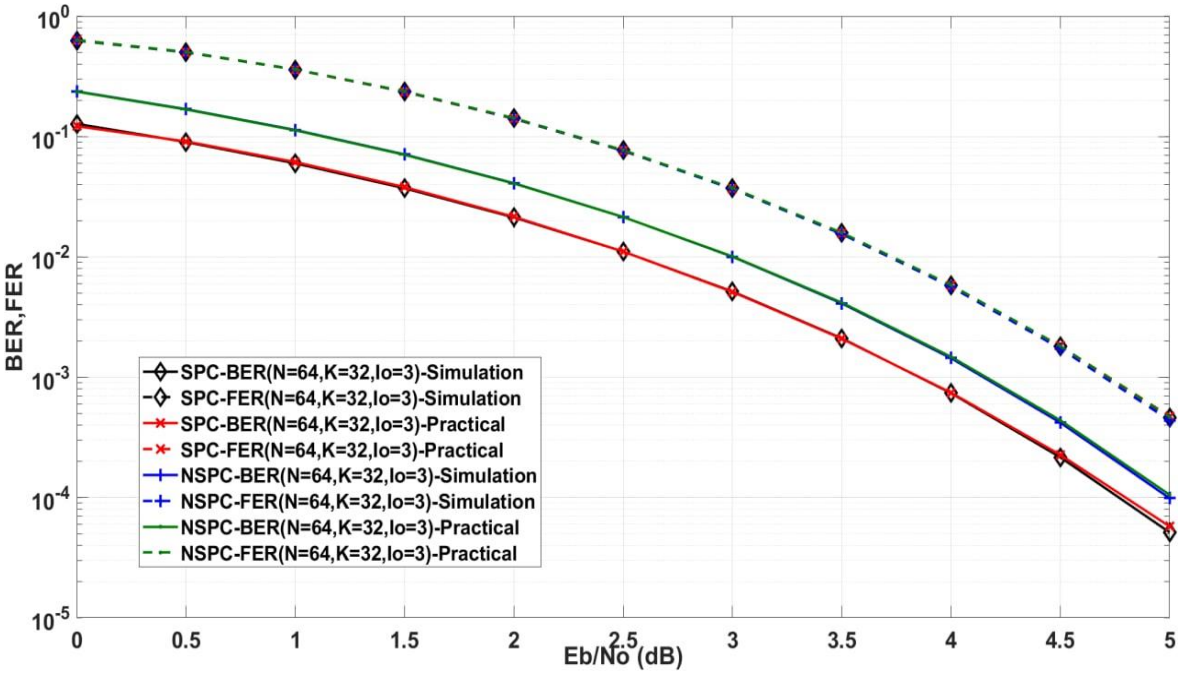


Figure (4.7): A Comparison Between the Simulation and Practical Results of SPC and NSPC.

4.3. Parallel Concatenated Turbo Polar Codes (PCTPCs)

The parallel concatenated turbo polar codes can be implemented based on the systematic polar codes implemented in the previous section (i.e., section 4.2). Figures (3.1) and (3.2) show the encoder and decoder of the proposed PCTPC. In this section, the iterative decoding algorithm of the presented PCTPC scheme is implemented with a single-iteration SCAN decoder, S-random interleaver, eight outer iterations, and also with fixed scaling factors for each examined SNR as follows:

SF_1 = [0.5,0.5,0.5,0.5,0.5, 0.67, 0.75, 0.55, 0.8, 0.87, 0.9].

SF_2 = [0.7,0.7,0.7,0.7,0.7, 0.75, 0.75, 0.75, 0.8, 0.87, 0.95].

These factors are chosen experimentally with SNRs from 0dB to 5dB and a step value of 0.5dB. In addition, this iterative decoding algorithm is implemented with an error frame limit of 1000 and frames number ranging from 50000 to 3000000.

4.3.1. FPGA Implementation of the PCTPCs

The FPGA implementation of the suggested PCTPC scheme is performed based on the construction of Vivado IPs for the turbo encoder and decoder illustrated in Figures (3.1) and (3.2), respectively. Algorithms 7 & 8 illustrate the implementation procedure of these IPs. The PCTPC encoder shown in algorithm-7 consists of two identical systematic polar encoders, while the iterative decoding algorithm (i.e., Algorithm-8) consists of two SISO-SCAN decoders. This subsection presents the FPGA implementation of the suggested PCTPC scheme using SPC (N=128, K=64), a single-iteration SCAN decoder, eight outer iterations, and a turbo rate of 1/3. Figure (4.8) shows the FPGA implementation of this scheme with five Vivado IPs for the encoder, decoder, demultiplexer, binary source, and AWGN channel. These IPs are designed and implemented using the Vivado HLS and Vivado IP integrator.

In the previous section, the performance and resource utilization analysis of the systematic encoder and SCAN decoder shows that the third encoder solution (E3) and the fourth decoder solution (D4) are the best. Therefore, these solutions are used in this section to implement the PCTPC scheme. Table (4.4) shows the FPGA resources required to implement the encoder and the iterative decoding algorithm of the suggested PCTPC with the solutions (E3) and (D4). The FPGA resources required to implement the systematic polar encoder are almost the same as those used to implement the encoder of the parallel concatenated turbo polar codes, with some differences. These resources are used twice to build the suggested turbo encoder because this encoder consists of two identical systematic polar encoders. Although the encoder of the suggested PCTPC scheme uses almost the same resources as SPE, its throughput is not more than half that of the single systematic polar encoder, as shown in Table (4.5).

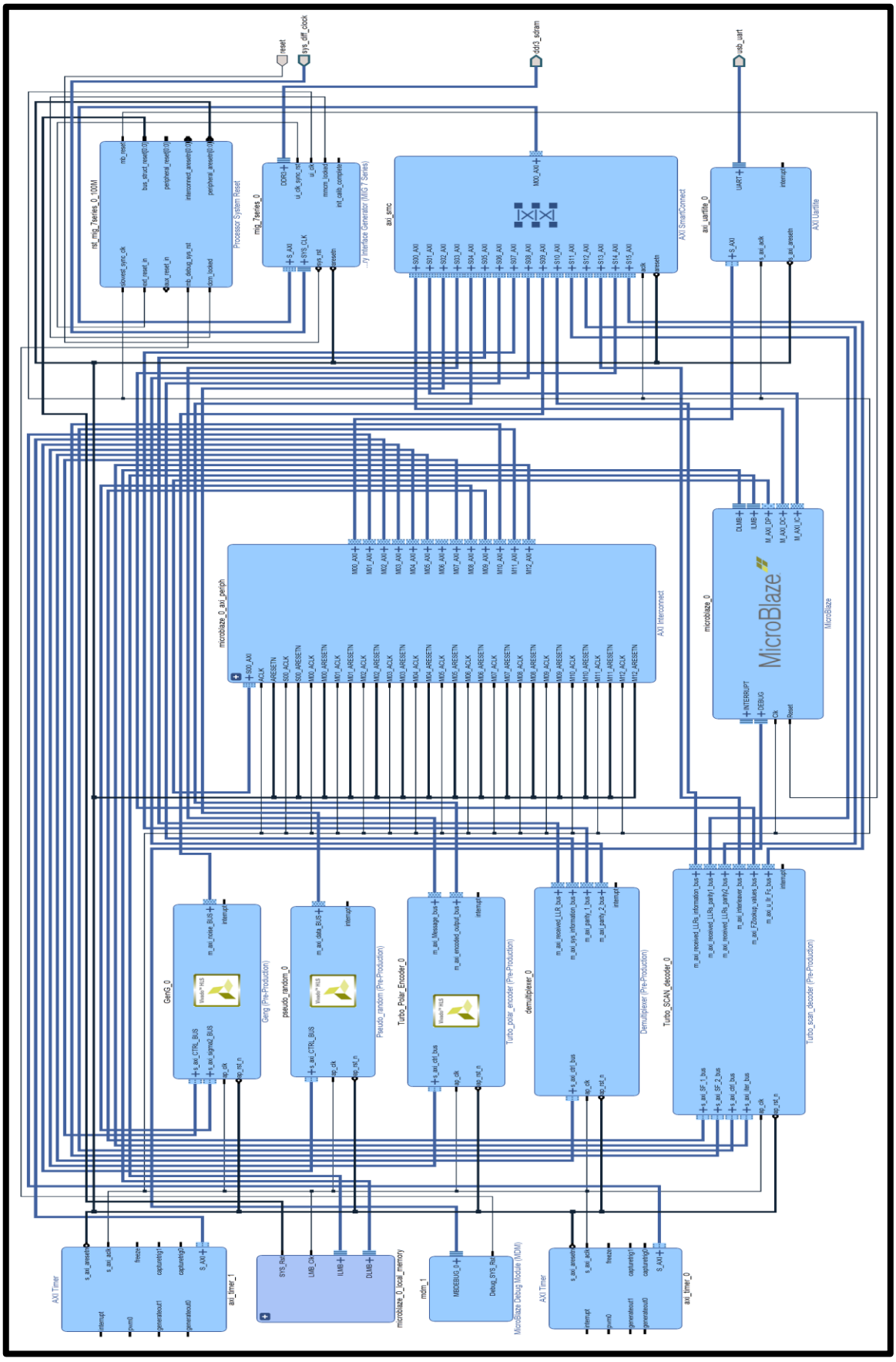


Figure (4.8): The FPGA Implementation of the Suggested PCTPC Scheme.

Table (4.4): The FPGA Resource Utilization of the Encoder and Decoder of the PCTPC Scheme.

Code Type	Solutions	BRAM18K	DSP48E	FF	LUT
PCTPC-E (N=192, K=64)	E3	9	0	2070	6614
PCTPC-E (N=384, K=128)	E3	11	0	2446	9927
PCTPC-SCAN (N=192, K=64)	D4	23	24	7350	14948
PCTPC-SCAN (N=384, K=128)	D4	35	28	7669	15848

Table (4.5): The Throughput of the PCTPC's Encoder and Decoder.

Codes	Throughput (b/s)
PCTPC-E (Nc=192, K=64)	0.73001M
PCTPC-SCAN (Nc=192, K=64)	24.0884K

The FPGA implementation of the iterative decoding process presented in Algorithm-8 (Appendix-1) can be achieved by two strategies. The first strategy involves implementing the entire decoding algorithm by Vivado and creating a Vivado IP to perform the all-decoding process (compact form), as shown in Figure (4.8). In the first strategy, the MicroBlaze is only responsible for the controlling and routing purposes, while the decoding process is performed by the Vivado IP, which is designed for the iterative decoding algorithm. On the other side, the second strategy uses the MicroBlaze to construct the outer framework of the iterative decoding algorithm. In this strategy, the SISO decoders (i.e., SCAN decoder) are implemented as Vivado IPs, while the remaining operations between these decoders include calculating a-prior and extrinsic information, interleaving and bit reversing operations, etc., are performed by the MicroBlaze (distributed form). The first strategy consumes more FPGA resources than the second, as shown in Figure (4.9), but at the expense of throughput.

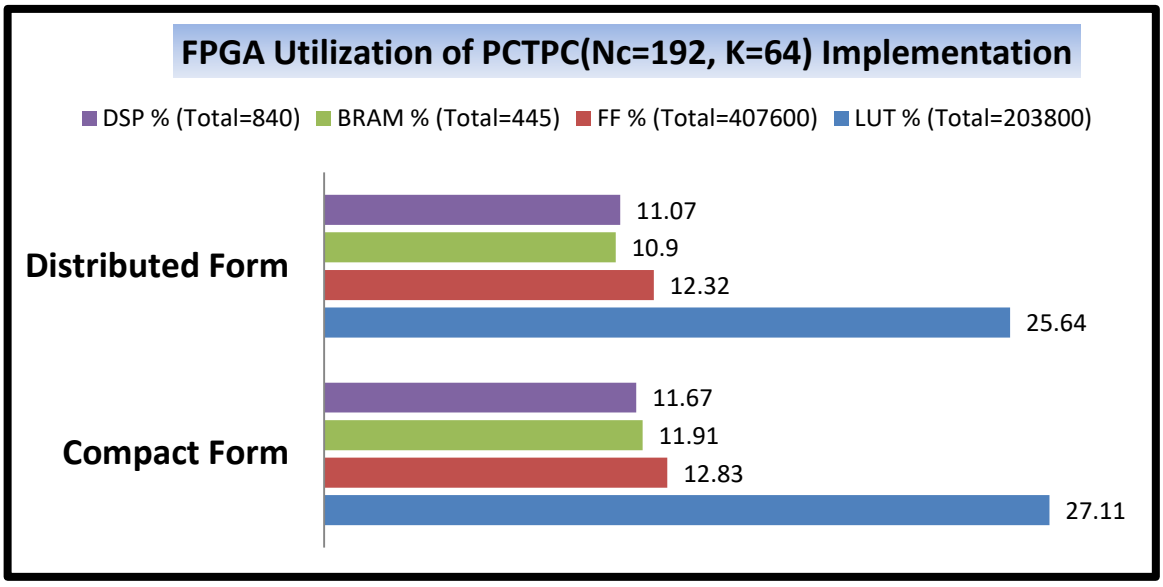


Figure (4.9): The Resource Utilization of the FPGA Implementation of the PCTPC with SPC(N=128, K=64) and Rate of 1/3.

4.3.2. Simulation Results of the PCTPCs

The simulation results of the suggested PCTPC scheme are presented with various code lengths, Signal-to-Noise ratios, and outer iterations to analyze and compare this scheme with the original polar and turbo codes. Therefore, this subsection presents the PCTPC's results in four scenarios. First, the behavior of the PCTPC at different code lengths is discussed and compared with the behavior of the original systematic polar codes. Second, the iterative decoding algorithm of the suggested scheme is tested with various numbers of outer-iteration. While in the third case, we compare the performance of the PCTPC with that of the original turbo convolutional code. Finally, the simulation and practical results of PCTPC are compared in the last scenario. The simulation results in this subsection are calculated in terms of average BER and FER. These average results are estimated by testing the suggested scheme several times with the same code length and with different input and noise sequences and taking the average of the runs.

Figure (4.10) shows the BER and FER performance of PCTPC schemes constructed using the constituent codes SPC(128,64) and SPC(256,128). This

Figure depicts that the performance of the suggested PCTPC scheme improves with increasing the code length, especially at high SNRs. As mentioned earlier, the performance of the constituent polar codes improves at large code lengths due to the perfect channel polarization, and thus the performance of PCTPC improves. Figure (4.10) illustrates that the suggested PCTPC outperforms the original systematic polar codes. At BER and FER of 10^{-4} and a code length of 128, the suggested PCTPC achieves BER and FER gains of 0.8dB and 1.2dB, respectively. Therefore, we can conclude from the previous results that the suggested scheme can be used to improve the performance of the polar codes at a finite code-length regime.

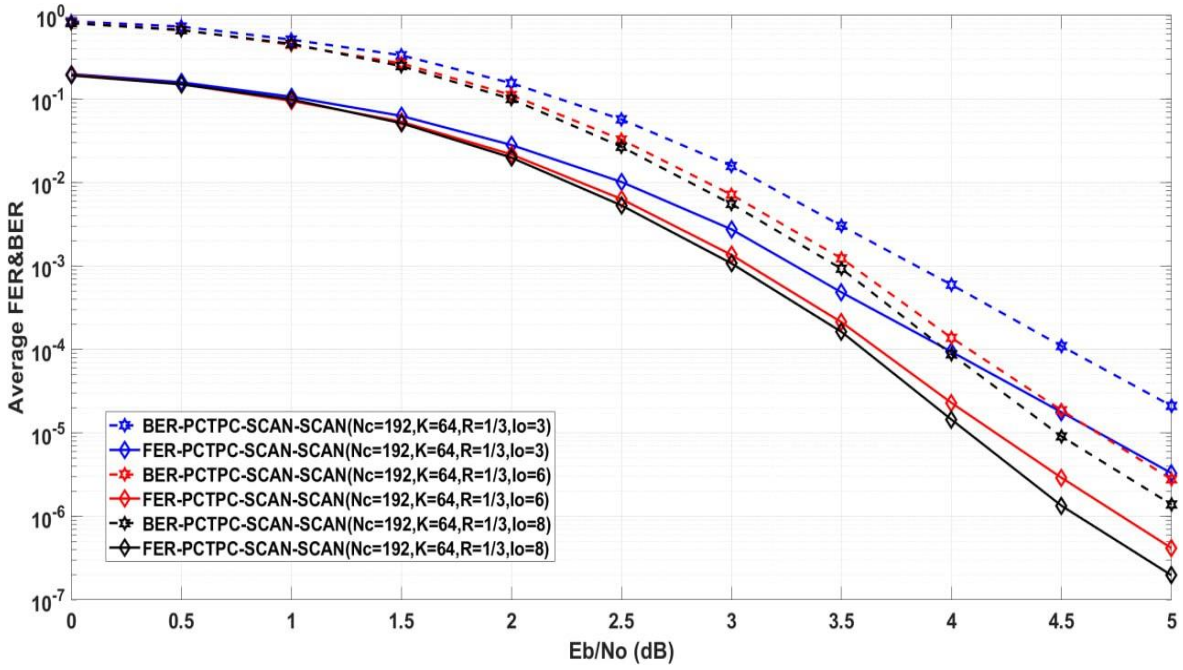


Figure (4.10): The BER and FER Performance of PCTPC scheme and SPC(N=128, K=64).

The second scenario shows the effect of the outer iterations on the BER and FER performance of the suggested PCTPC. Figures (4.11) and (4.12) illustrate the performance of the PCTPC(Nc=192, K=64) and PCTPC(Nc=384, K=128) with three, six, and eight iterations. At BER of 10^{-4}, PCTPC with eight iterations outperforms PCTPC with six and three iterations by 0.1dB and 0.4dB gains, respectively. While at FER of 10^{-4},

PCTPC with eight iterations outperforms PCTPC with six and three iterations by 0.1dB and 0.55dB gains, respectively. The performance of the suggested PCTPC at six iterations is close to that of eight, as shown in Figures (4.11) and (4.12). Therefore, the six iterations can be used to avoid excessive complexity and latency as the number of iterations increases.

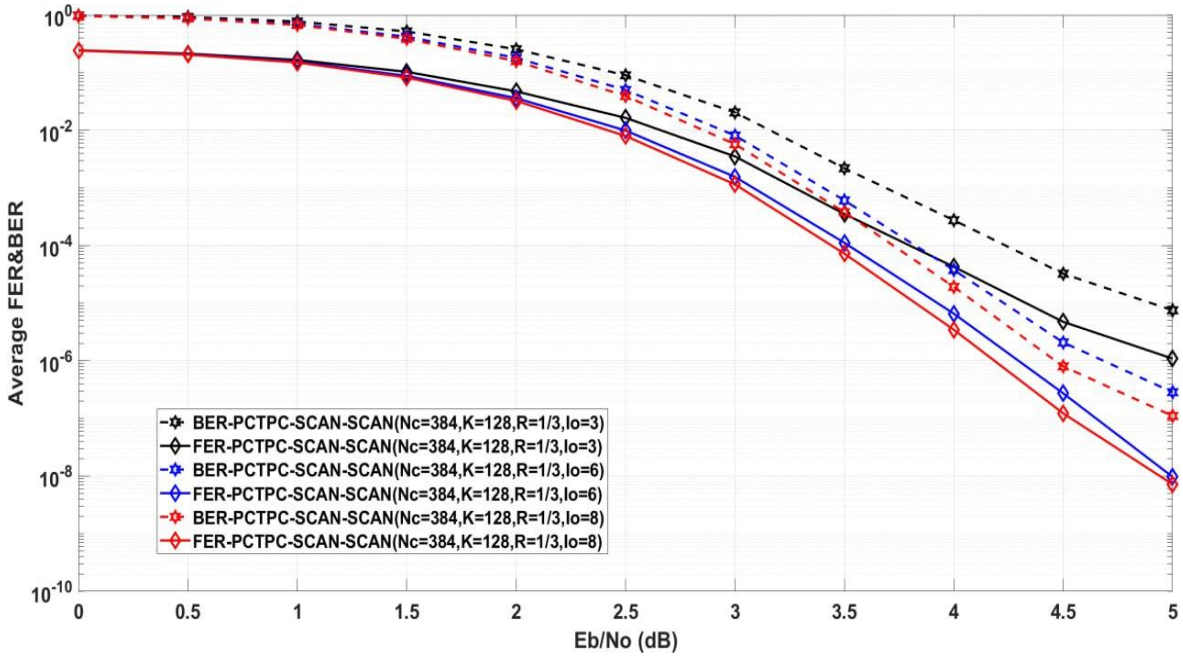


Figure (4.11): The Performance of the PCTPC(Nc=192, K=64) with 3, 6, and 8 Iterations

The error floor problem is the main issue facing traditional turbo convolutional codes. The suggested PCTPC scheme is introduced to reduce this problem in the turbo schemes. In the third scenario, Figure (4.13) shows the superiority of PCTPC over the original turbo convolutional code at high SNRs. The suggested PCTPC achieves gains of more than 0.5dB at FER & BER of 10^{-4} & 10^{-6}, respectively, as shown in Figure (4.13). Therefore, the suggested PCTPC can replace traditional turbo codes in applications that have challenges with error floor problems. Finally, the last scenario compares the simulation and practical results of the suggested PCTPC scheme. Figure (4.14) shows the practical results are very close to the simulation results, with some differences due to the noise generated during the processing operation.

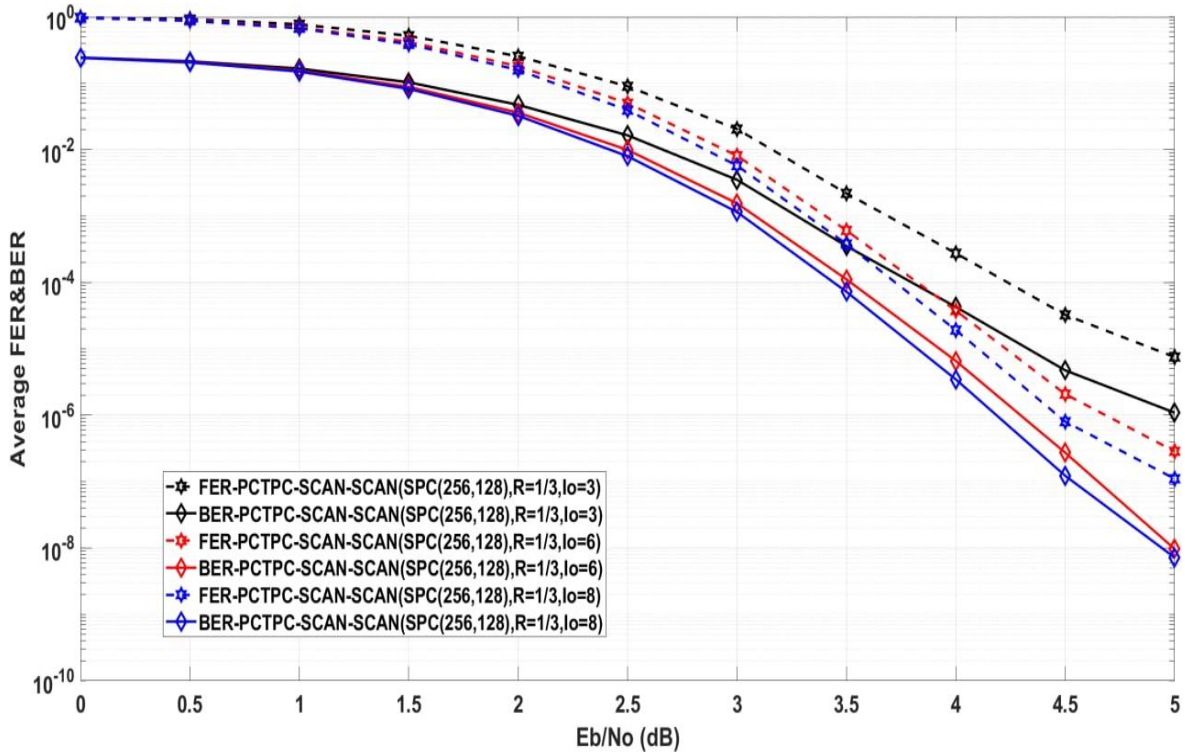


Figure (4.12): The Performance of the PCTPC(Nc=384, K=128) with 3, 6, and 8 Iterations.

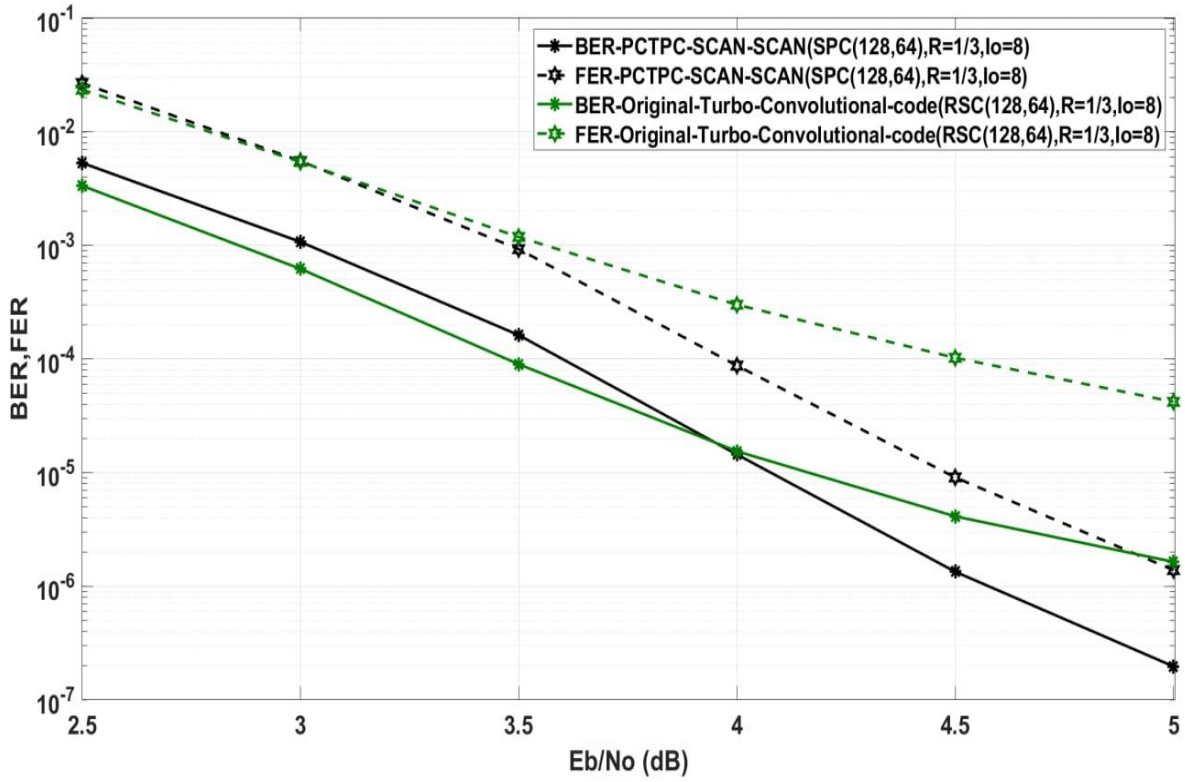


Figure (4.13): Performance Comparison Between the Suggested PCTPC and the Original Turbo Convolutional Code.

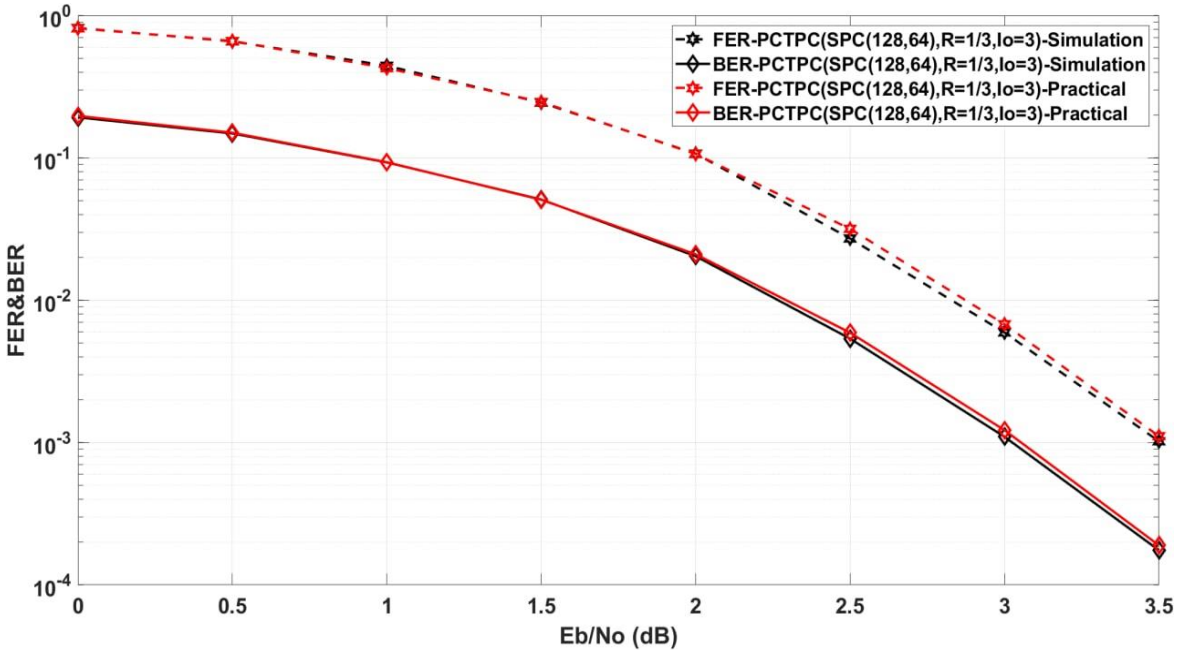


Figure (4.14): Performance Comparison Between the Simulation and Practical Results of the Suggested PCTPC scheme.

4.4. Parallel Concatenated Turbo Polar-Convolutional Codes (PCTPCCs)

The proposed PCTPCC scheme has been constructed by parallel concatenation of systematic polar and convolutional codes. Figures (3.3) and (3.4) illustrate the turbo encoder and iterative decoding algorithm of the proposed PCTPCC. Figure (3.3) shows that the turbo encoder consists of an RSC(13,15) with trellis termination as a recursive systematic convolutional encoder and an SPC(N, K) with different lengths as a systematic polar encoder. While the iterative decoder in Figure (3.4) consists of SOVA or LogMap as a convolutional decoding algorithm and SCAN as a polar decoding algorithm.

In this section, the proposed PCTPCC is implemented with the following parameters: LTE Interleaver, Decoding depth of 20 for SOVA decoder, Single-Iteration for SCAN decoder, 1-8 Outer iterations, Error limit of 1000 frames, 50000- 30000000 Frames, and SF1=[1, 1, 1, 1, 1, 1, 1, 0.9, 1, 1, 1] & SF2=[0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.4, 0.5, 0.4, 0.4, 0.3] as scaling

factors. These factors are chosen experimentally with SNRs from 0dB to 5dB with a step value of 0.5dB.

4.4.1. FPGA Implementation of the PCTPCCs

The FPGA implementation of the proposed PCTPCC scheme is achieved based on the creation of Vivado IPs for the turbo encoder and decoder illustrated in Figures (3.3) and (3.4). These Vivado IPs are constructed based on algorithms 10 and 13 in appendix-I. The PCTPCC encoder presented in algorithm-10 comprises systematic convolutional and polar encoders, whereas the iterative decoding algorithm (i.e., Algorithm-13) comprises SCAN and SOVA decoders. This subsection presents the FPGA implementation of the proposed PCTPCC (Nc=192, K=61) scheme, which is designed using RSC(13,15), SPC(128,64), a single-iteration SCAN decoder, twenty depth-SOVA decoder (or LogMap), eight outer iterations, convolutional trellis termination, and a turbo rate of 0.317. The FPGA implementation of this scheme is shown in Figure (4.15), which includes five Vivado IPs for the encoder, decoder, demultiplexer, binary source, and awgn channel. The Vivado HLS and Vivado IP integrator are used to develop and implement these IPs.

The proposed PCTPCC is implemented with the third encoder solution (E3) and the fourth decoder solution (D4), as the implemented PCTPC scheme in the previous section. Table (4.6) shows the FPGA resources required to implement the encoder and the iterative decoding algorithm of the suggested PCTPCC with the solutions (E3) and (D4). The iterative decoding algorithm is implemented with both SOVA and LogMap algorithms; thus, the resource utilization is presented in Table (4.6) with both algorithms. RSC and SPC encoders consume the FPGA resources required to implement the PCTPCC's encoder. The RSC encoder requires fewer resources than the SPC encoder because constructing the recursive systematic convolutional encoder

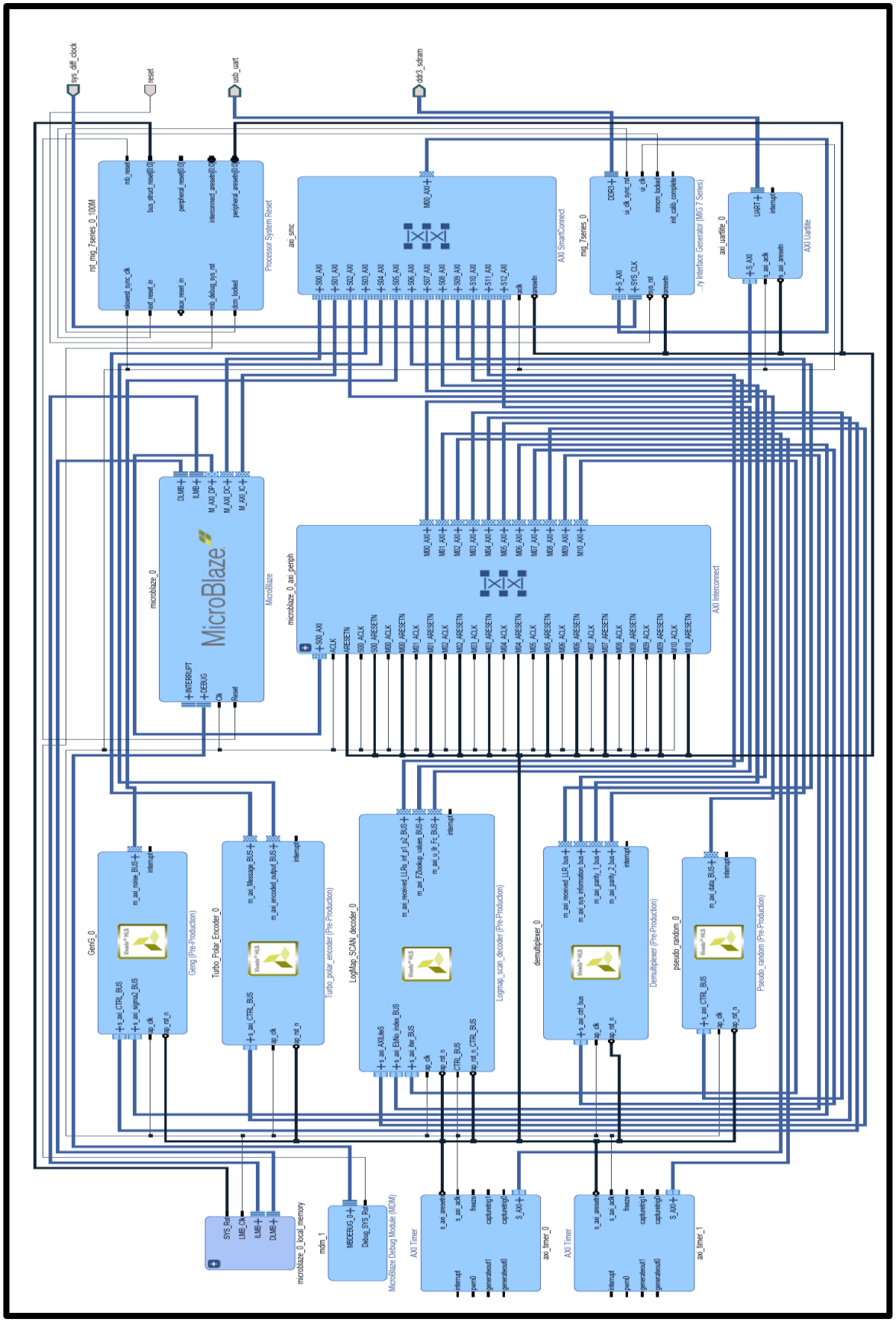


Figure (4.15): The FPGA Implementation of the Suggested PCTPCC Scheme.

is more straightforward than the systematic polar encoder. Therefore PCTPCC's encoder requires fewer resources than PCTPC's encoder, which consists of two systematic polar encoders, as shown in Tables (4.4) and (4.6).

The FPGA implementation of the iterative decoding process presented in Algorithm-13 (Appendix-1) is implemented with SOVA and LogMap decoders. As mentioned earlier in chapter three (Section 3.4), the SOVA decoder has less complexity than the LogMap decoder. Table (4.6) shows that the proposed iterative decoding algorithm of the PCTPCC scheme with SOVA decoder (i.e., SOVA-PCTPCC scheme) consumes fewer resources compared to that using LogMap (i.e., LogMap-PCTPCC scheme). Thus the first algorithm has better throughput than the second, as shown in Table (4.7). Moreover, the FPGA implementation of the SOVA iterative decoding algorithm consumes less power and generates a lower junction temperature than the LogMap iterative decoding algorithm. In this section, the entire decoding algorithm is implemented as Vivado IP, as shown in Figure (4.15), while the MicroBlaze is only employed for control and routing purposes to increase system throughput and reduce decoding latency.

Table (4.6): The FPGA Resource Utilization of the Encoder and Decoder of the PCTPCC Scheme.

Code Type	Solutions	BRAM18K	DSP48E	FF	LUT
PCTPCC-E (Nc=192, K=61)	E3	7	0	1858	3825
PCTPCC-E (Nc=384, K=125)	E3	9	0	1994	6268
PCTPCC-SOVA (Nc=192, K=61)	D4	22	24	8490	12920
PCTPCC-SOVA (Nc=384, K=125)	D4	35	26	8320	13320
PCTPCC-LogMap (Nc=192, K=61)	D4	22	47	12607	16741
PCTPCC-LogMap (Nc=384, K=125)	D4	34	47	12544	17615

Table (4.7): The Throughput of the PCTPCC's Encoder and Decoder.

Codes	Throughput (b/s)
PCTPCC-E (Nc=192, K=61)	1.33037 M
PCTPCC-SOVA (Nc=192, K=61)	16.4398K
PCTPCC-LogMap (Nc=192, K=61)	16.0000K

4.4.2. Simulation Results of the PCTPCCs

The proposed PCTPCC scheme has been tested with various values of code lengths, SNRs, and outer iterations. The simulation results are presented in this subsection to analyze the suggested PCTPCC and compare it with the original polar and turbo codes. These results can be classified into several cases; Firstly, the BER and FER performance of the PCTPCC is estimated with both SOVA and LogMap decoders and with a fixed number of outer iterations. Then, this performance is compared with the performance of the original convolutional turbo codes. Finally, the practical results of the PCTPCC scheme implemented in the previous subsection have been presented and compared with the simulation results. The simulation results are also calculated in terms of average BER and FER, as in the previous section.

Figure (4.16) shows the BER and FER performance of the original SPC(128,64) and the proposed SOVA-PCTPCC scheme with different code lengths and with almost 1/3 code rate. As stated earlier with the PCTPC scheme, this Figure depicts that the performance of the suggested SOVA-PCTPCC also improves with increasing the code length, especially at high SNRs. At BER and FER of 10^{-5}, Figure (4.16) shows that the SOVA-PCTPCC (SPC(512,256), $\Re$=0.33) outperforms the SOVA-PCTPCC (SPC(256, 128), $\Re$= 0.32) by gains of about 0.16dB and 0.3dB, while it outperforms the SOVA-PCTPCC (SPC(128,64), $\Re$ =0.317) by gains of about

0.85dB and 1.14dB. Moreover, at BER and FER of 10^{-4}, the proposed PCTPCC achieves BER and FER gains of 0.8dB and 1.1dB, respectively, over the original systematic polar code. Therefore, we can use the proposed PCTPCC to replace the original polar code at a finite code-length regime.

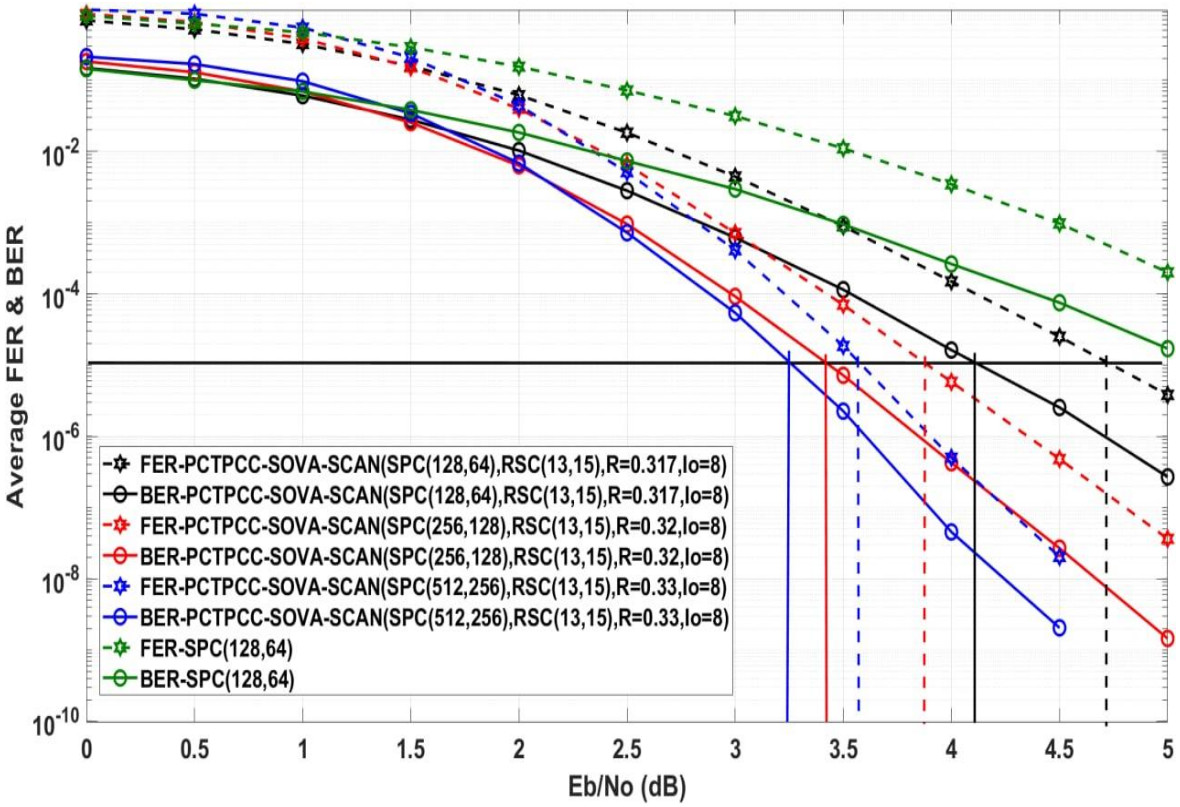


Figure (4.16): The BER and FER Performance of the SOVA-PCTPCC scheme and the SPC(N=128, K=64).

The proposed PCTPCC scheme can be further improved by using the LogMap decoder with the SCAN decoder to construct the iterative decoding algorithm instead of the SOVA decoder, as shown in Figure (4.17). At BER and FER of 10^{-6}, this Figure shows that the LogMap-PCTPCC (Nc=384, K=125) scheme outperforms the SOVA-PCTPCC (Nc=384, K=125) scheme by gains of about 0.125dB and 0.21dB, repectively.

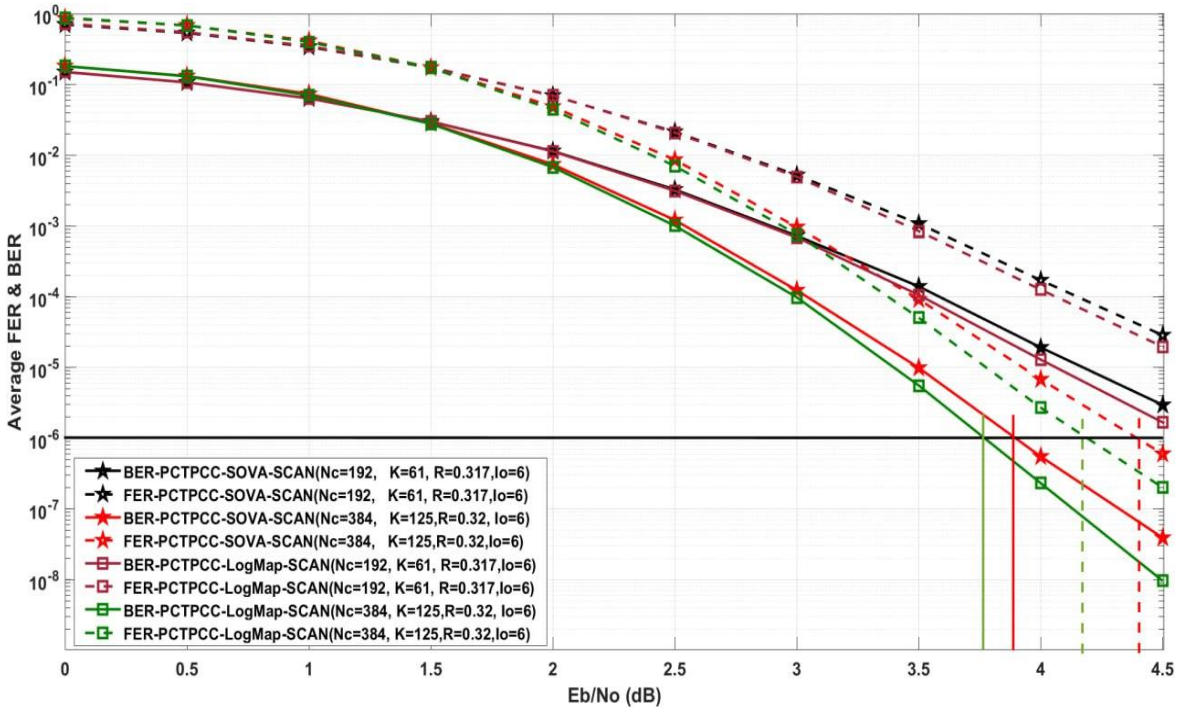
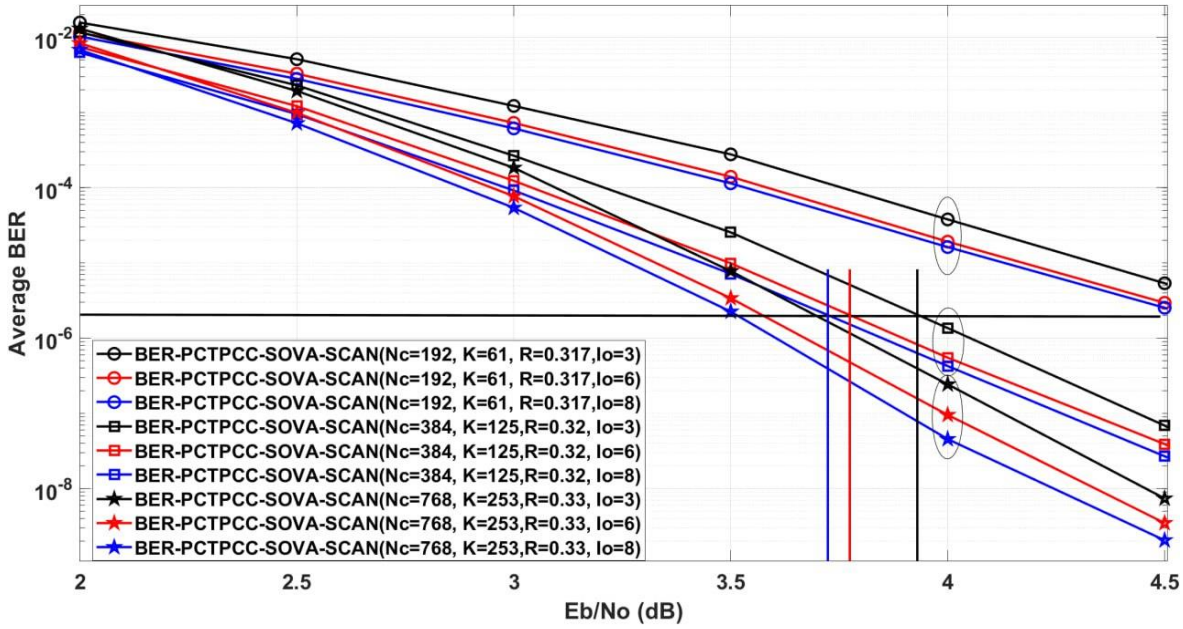
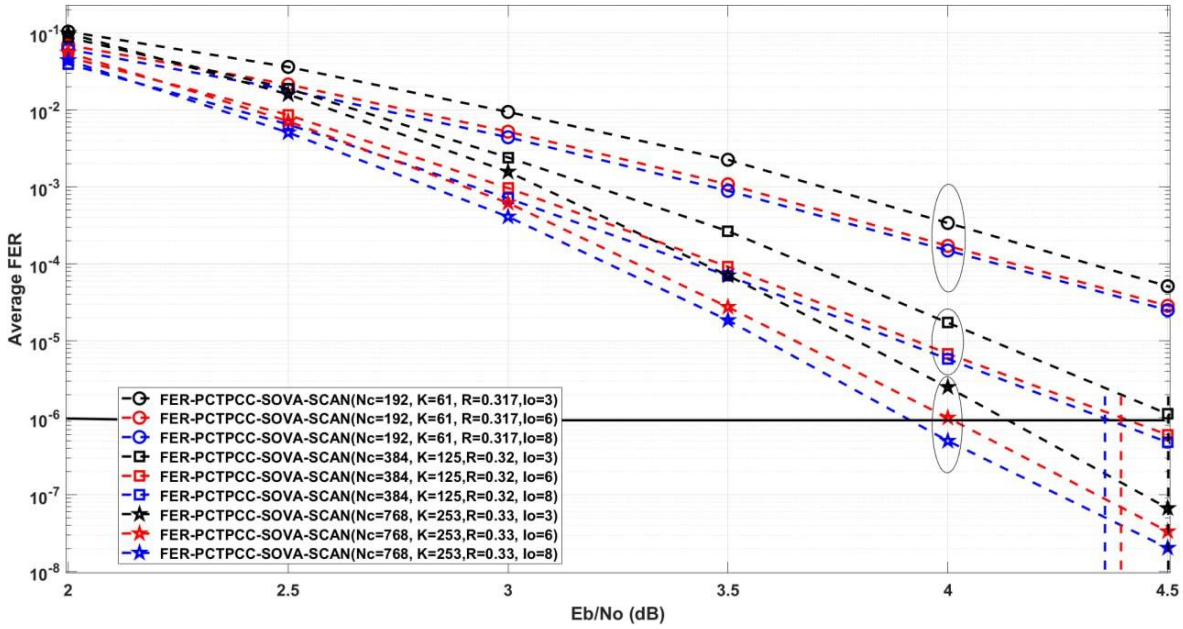


Figure (4.17): Performance Comparison Between the SOVA-PCTPCC and LogMap-PCTPCC schemes.

Figures (4.18) and (4.19) show the effect of the outer iterations on the BER and FER performance of the suggested PCTPCC scheme with both SOVA and LogMap decoders. These Figures illustrate the performance of the SOVA-PCTPCC and LogMap-PCTPCC schemes with three, six, and eight iterations. At BER and FER of 10^{-6}, Figure (4.18) shows that the SOVA-PCTPCC(Nc=384, K=125) scheme with eight iterations outperforms that scheme with three iterations by about 0.2dB and 0.165dB gains, respectively. At the same time, Figure (4.19) shows that the LogMap-PCTPCC(Nc=384, K=125) scheme with eight iterations outperforms that scheme with three iterations by about 0.225dB and 0.3dB gains, respectively, at BER and FER of 10^{-6}. As illustrated in the previous section with the PCTPC scheme, Figures (4.18) and (4.19) show that the performance of the suggested PCTPCC at six iterations is close to that of eight. Therefore the six iterations can be used to overcome the higher complexity and latency as the number of iterations increases.



(a)

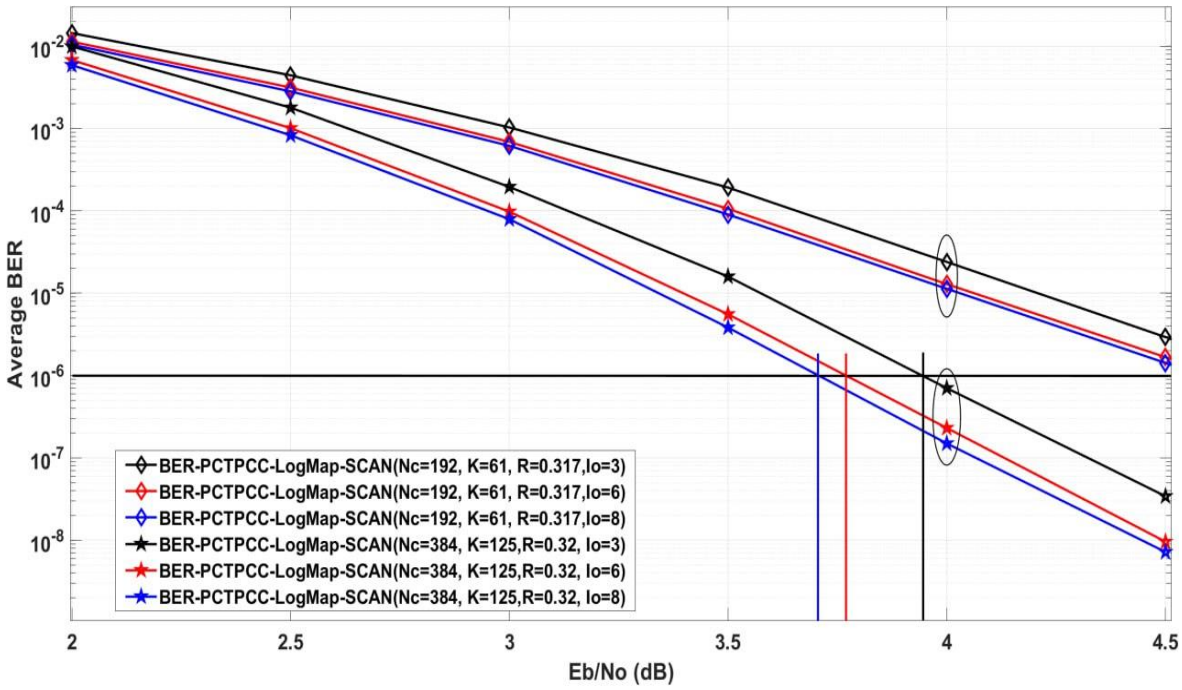


(b)

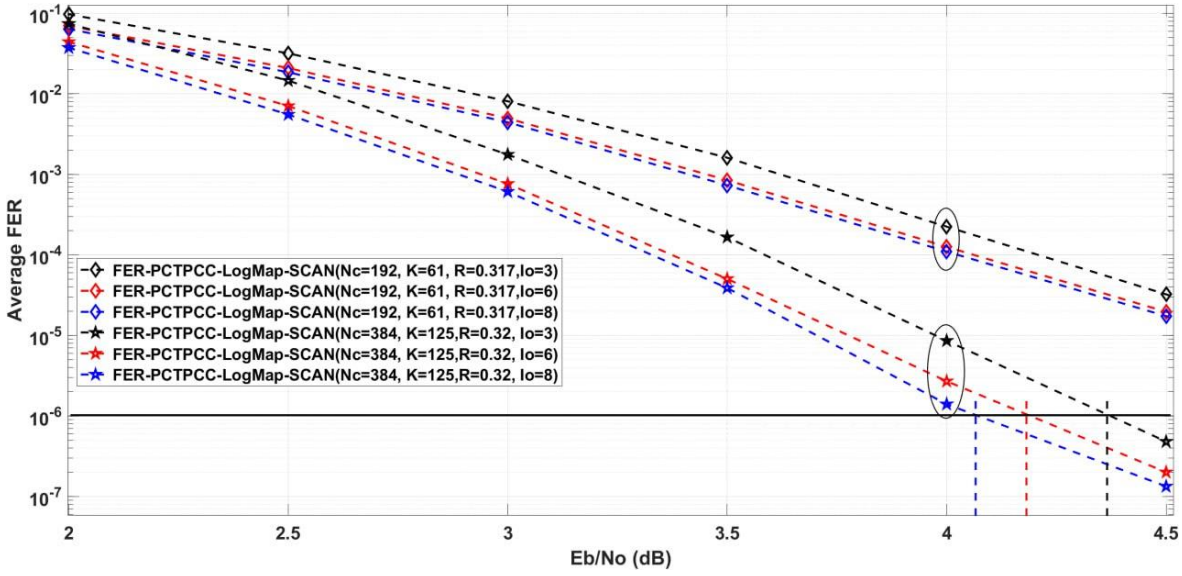
Figure (4.18): The Performance of the SOVA-PCTPCC with 3, 6, and 8 Iterations: (a) BER Performance, (b) FER Performance.

Figure (4.20) compares the original convolutional turbo code and the proposed PCTPCC with K=64 and I_{outer}= 8. This Figure shows the proposed PCTPCC scheme handles the error floor problem inherent in the traditional turbo convolutional codes. At FER of 10^{-4}, the proposed SOVA-PCTPCC outperforms the original convolutional turbo code by about 0.4dB gain, while

at BER of 10^{-6}, it exceeds the original convolutional turbo code by more than 0.35dB gain. Finally, the last case compares the simulation and practical results of the suggested SOVA-PCTPCC scheme. Figure (4.21) shows the practical results are very close to the simulation results. Therefore, we can conclude the FPGA implementation work well as the simulator.



(a)



(b)

Figure (4.19): The Performance of the LogMap-PCTPCC with 3, 6, and 8 Iterations: (a) BER Performance, (b) FER Performance.

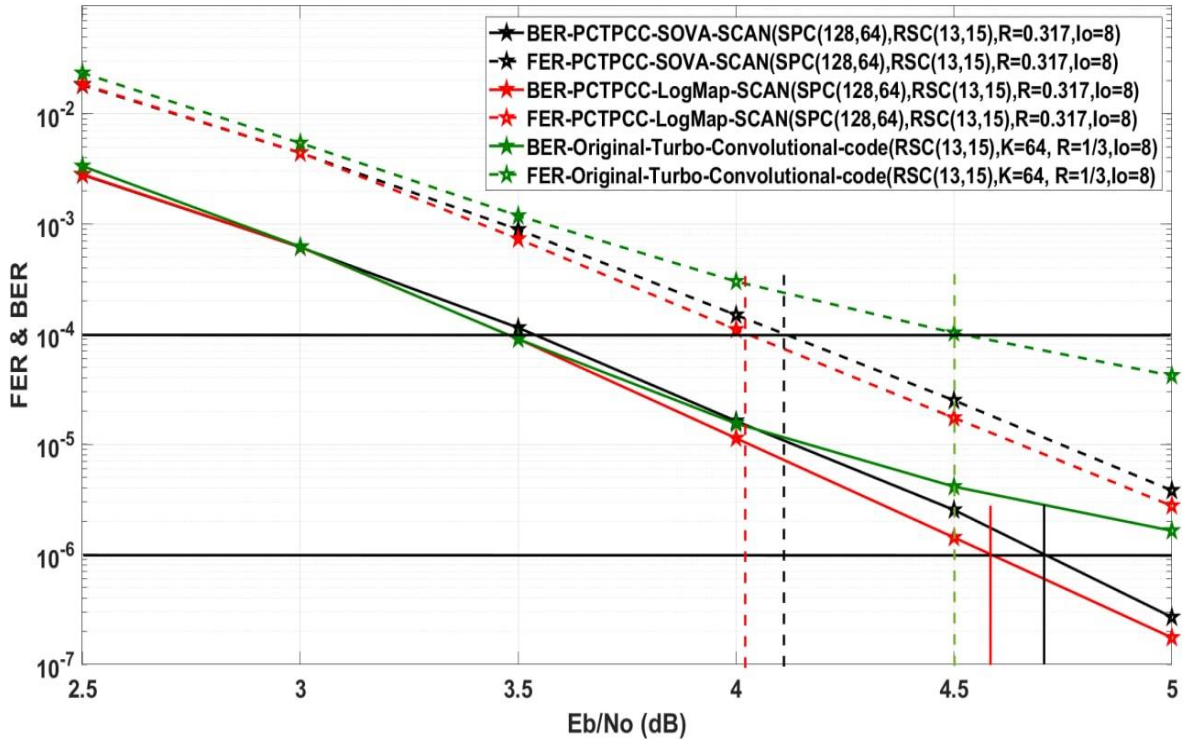


Figure (4.20): Performance Comparison Between the Proposed PCTPCC and the Original Turbo Convolutional Code.

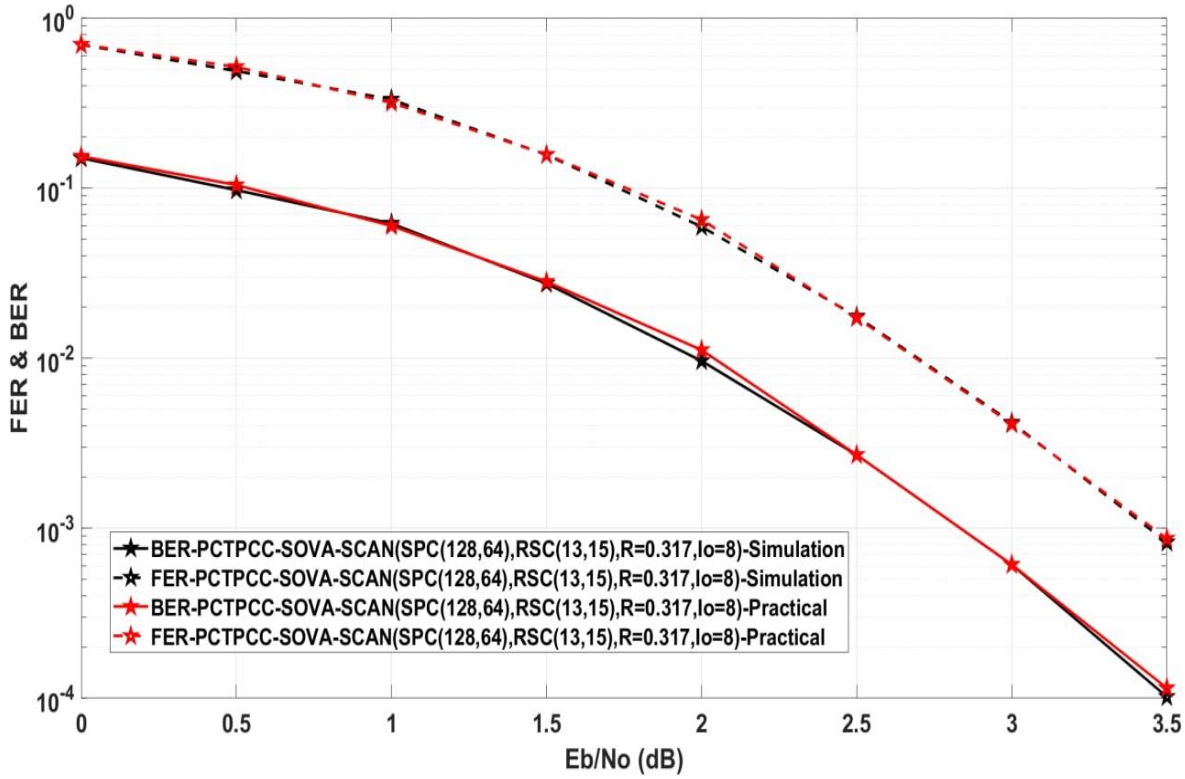


Figure (4.21): Performance Comparison Between the Simulation and Practical Results of the Propsed PCTPCC scheme.

4.5. Serial Concatenated Turbo Polar-Convolutional Codes (SCTPCCs)

The proposed SCTPCC scheme has been constructed by serially concatenating systematic polar and convolutional codes. Figures (4.22) and (3.5) illustrate the turbo encoder and iterative decoding algorithm of the proposed SCTPCC. Figure (4.22) shows that the proposed serial encoder consists of an SPC(N, K) with a rate of 2/3 as outer code and an RSC(13,15) with a rate of 1/2 as inner code; thus, the overall rate of this scheme is 1/3. While the iterative decoder in Figure (3.5) consists of SOVA or LogMap as a convolutional decoding algorithm and SCAN as a polar decoding algorithm.

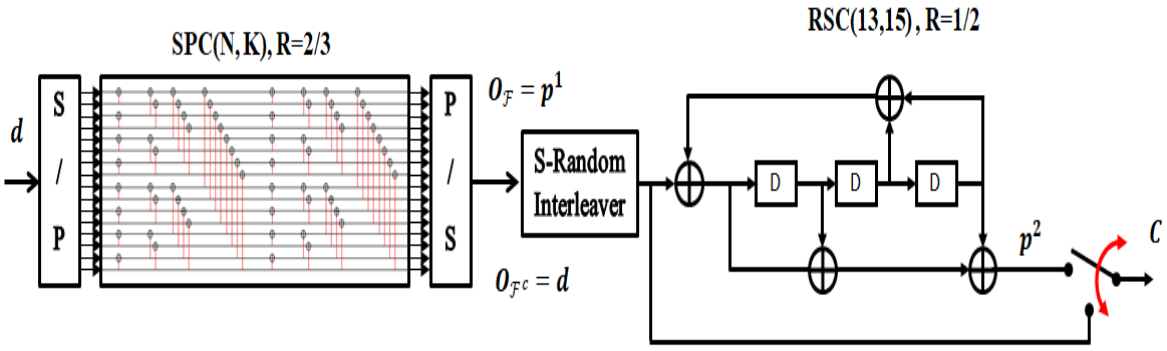


Figure (4.22): The Encoder of the Proposed Serial Concatenated Turbo Polar-Convolutional Code.

In this section, the proposed SCTPCC is implemented with the following parameters: S-Random Interleaver, decoding depth of 20 for SOVA decoder, single-iteration for SCAN decoder, 1-8 outer iterations, error limit of 1000 frames, 50000- 20000000 frames, SF1=0.5 & SF2=0.55 as fixed scaling factors. These factors are chosen experimentally with SNRs from 0dB to 4.5dB.

4.5.1. FPGA Implementation of the SCTPCCs

The FPGA implementation of the proposed SCTPCC scheme is performed based on the creation of Vivado IPs for the turbo encoder and decoder illustrated in Figures (4.22) and (3.5), respectively. These Vivado IPs

are constructed based on Algorithms 14 and 15 in appendix-I. The encoders and decoders that make up these algorithms are the same as those used in the PCTPCC algorithms (i.e., Algorithms 10 and 13), but with the SCTPCC scheme, they are linked in serial procedure rather than the parallel style. This subsection presents the FPGA implementation of the proposed SCTPCC, which is designed using RSC(13,15), SPC(128,85), a single-Iteration SCAN decoder, twenty depth-SOVA decoder (or LogMap), eight outer iterations, and a turbo rate of 0.33. The FPGA implementation of this scheme is shown in Figure (4.23), which includes four Vivado IPs for the encoder (Serial_Turbo_Encoder), decoder (Serial_SOVA_SCAN), binary source (Pseudo_random), and AWGN channel (GenG).

The third encoder solution (E3) and the fourth decoder solution (D4) are used in this section to implement the proposed SCTPCC scheme as used previously with PCTPC and PCTPCC schemes. The FPGA resources required to implement the encoder and the iterative decoding algorithm of the suggested SCTPC are presented in Table (4.8) with two cases. Firstly, the proposed system is implemented with a data length of 85 bits and then with a length of 170 bits. Moreover, Table (4.8) shows the FPGA resources of the proposed iterative decoding algorithm with both SOVA and LogMap decoder. The proposed SCTPCC scheme requires more resources than the PCTPCC scheme because the inner code of the SCTPCC scheme has a 1/2 code rate; thus inner code length is twice the outer code length. So, the proposed scheme uses an SPC($N_{outer} = 128$, $K_{outer} = 85$) as the outer code and an RSC($K_{inner} = 128$, $N_{inner} = 256$) as the inner code for a data length of 85. Tables (4.6) and (4.8) show that the PCTPCC with a data length of 64 requires fewer resources compared to the SCTPCC with a code length of 85 despite both using a systematic polar code with a length of 128.

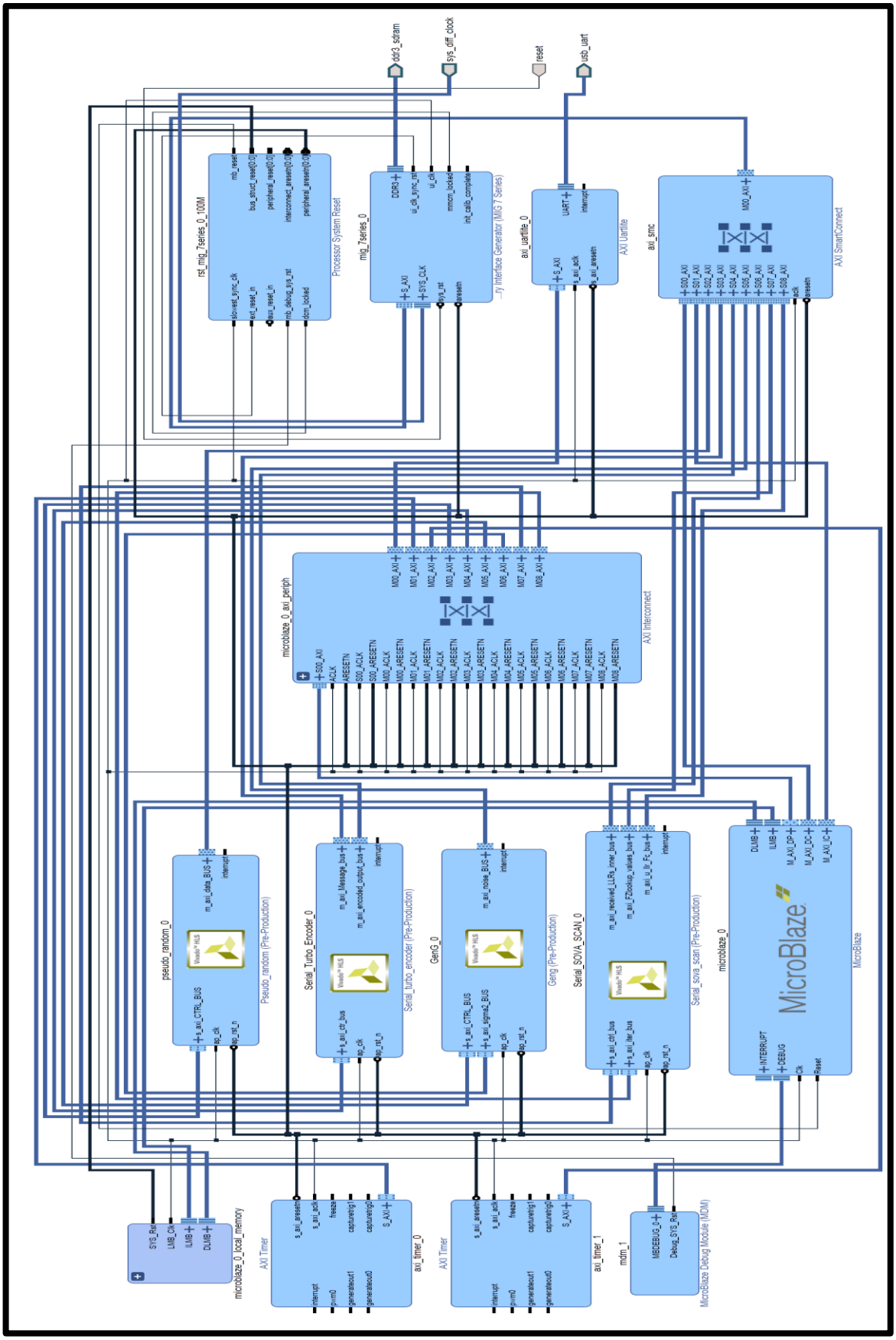


Figure (4.23): The FPGA Implementation of the Suggested SCTPCC scheme.

Table (4.8): The FPGA Resource Utilization of the Encoder and Decoder of the SCTPCC scheme.

Code Type	Solutions	BRAM-18K	DSP-48E	FF	LUT
SCTPCC-E (N_{inner}= 256, K_{outer}= 85)	E3	9	0	2017	7983
SCTPCC-E (N_{inner}= 512, K_{outer}= 170)	E3	17	0	2339	14317
SCTPCC–SOVA (N_{inner}= 256, K_{outer}= 85)	D4	22	35	8569	13088
SCTPCC–SOVA (N_{inner}= 512, K_{outer}= 170)	D4	36	37	8575	13727
SCTPCC-LogMap (N_{inner}= 256, K_{outer}= 85)	D4	24	56	12499	16786
SCTPCC-LogMap (N_{inner}= 512, K_{outer}= 170)	D4	35	58	12504	17547

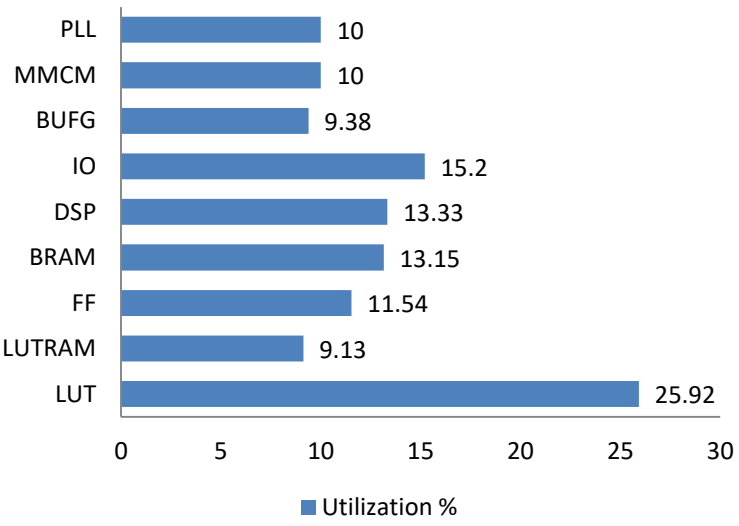
The proposed serial iterative decoding algorithm (i.e., Algorithm-15) is implemented using SOVA and LogMap decoders. Table (4.8) shows that this algorithm with SOVA decoder (i.e., SOVA-SCTPCC scheme) consumes fewer resources than that of LogMap (i.e., LogMap-SCTPCC scheme). Thus the SOVA-SCTPCC algorithm has better throughput than the LogMap-SCTPCC algorithm, as shown in Table (4.9). Moreover, the FPGA implementation of the SOVA iterative decoding algorithm consumes less power and generates a lower junction temperature than the LogMap iterative

Table (4.9): The Throughput of the SCTPCC's Encoder and Decoder.

Codes	Throughput (b/s)
SCTPCC-E (N_{inner}= 256, K_{outer}= 85)	1.319966 M
SCTPCC–SOVA (N_{inner}= 256, K_{outer}= 85)	12.956544K
SCTPCC-LogMap (N_{inner}= 256, K_{outer}= 85	7.4486645K

decoding algorithm, as shown in Figures (4.24) and (4.25). In this section, the entire decoding algorithm is implemented as Vivado IP, as shown in

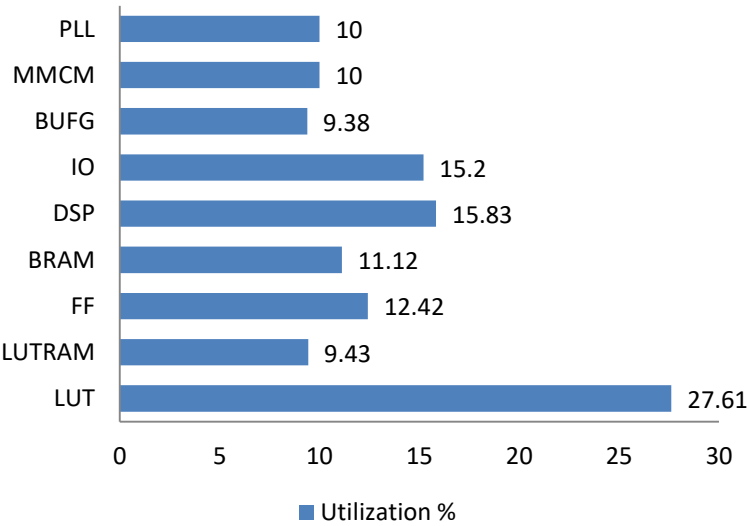
Figure (4.23), while the MicroBlaze is only employed for control and routing purposes to increase system throughput and reduce decoding latency.



power	
Total On-Chip Power	2.092W
Junction Temperature	28.7 °C
Thermal Margin	56.3 °C (30.9 W)
Effective Thermal resistance	1.8 °C/W

	LUT	LUTRAM	FF	BRAM	DSP	IO	BUFG	MMCM	PLL
Utilization	52819	5840	47028	58.5	112	76	3	1	1
Available	203800	64000	407600	445	840	500	32	10	10
Utilization %	25.92	9.13	11.54	13.15	13.33	15.2	9.38	10	10

Figure (4.24): The Resource and Power Utilization of the FPGA Implementation of the SOVA-SCTPCC scheme.



power	
Total On-Chip Power	2.19W
Junction Temperature	28.9 °C
Thermal Margin	56.1 °C (30.8 W)
Effective Thermal resistance	1.8 °C/W

	LUT	LUTRAM	FF	BRAM	DSP	IO	BUFG	MMCM	PLL
Utilization	56278	6033	50604	49.5	133	76	3	1	1
Available	203800	64000	407600	445	840	500	32	10	10
Utilization %	27.61	9.43	12.42	11.12	15.83	15.2	9.38	10	10

Figure (4.25): The Resource and Power Utilization of the FPGA Implementation of the LogMap-SCTPCC scheme.

4.5.2. Simulation Result of the SCTPCCs

The proposed SCTPCC scheme has been tested with various values of code lengths, SNRs, and outer iterations. In this subsection, the simulation results of this scheme are presented and discussed in detail with various scenarios. Firstly, the average BER and FER performance of the SCTPCC are estimated with both SOVA and LogMap decoders and with several outer iterations. Then, this performance is compared with the performance of the systematic polar codes and the practical FPGA implementation. The simulation results are also calculated in terms of average BER and FER, as in the previous sections.

Figure (4.26) shows the BER and FER performance of the original SPC(128,64) and the proposed SOVA-SCTPCC scheme with different code lengths and a 1/3 code rate. The BER and FER performance of the SOVA-SCTPCC scheme improves with increasing the code length. Therefore the SOVA-SCTPCC (SPC(256,170), RSC(13,15)) outperforms the SOVA-SCTPCC(SPC(128,85), RSC(13,15)) by gains of 0.65dB at BER and FER of 10^{-5}, as depicted in Figure (4.26). Furthermore, this Figure shows the superiority of the SOVA-SCTPCC over the original systematic polar code by about 0.94dB at BER of 10^{-4}. So, the SOVA-SCTPCC achieved its first goal by improving the polar code's performance in a finite code-length regime. The proposed SCTPCC scheme can be further enhanced by replacing the SOVA decoder with a LogMap decoder in the iterative decoding algorithm, as illustrated in Figure (4.27). At BER and FER of 10^{-5}, this Figure shows that the LogMap-SCTPCC(SPC(256,170), RSC(13,15)) outperforms the LogMap-SCTPCC(SPC(128,85), RSC(13,15)) by gains of about 0.475dB and 0.57dB. Moreover, Figure (4.28) show that the LogMap-SCTPCC (N_{inner}= 256, K_{outer}= 85) scheme outperforms the SOVA-SCTPCC (N_{inner} = 256, K_{outer}

= 85) scheme by gains of about 0.1dB and 0.11dB at BER & FER of 10^{-5}, respectively.

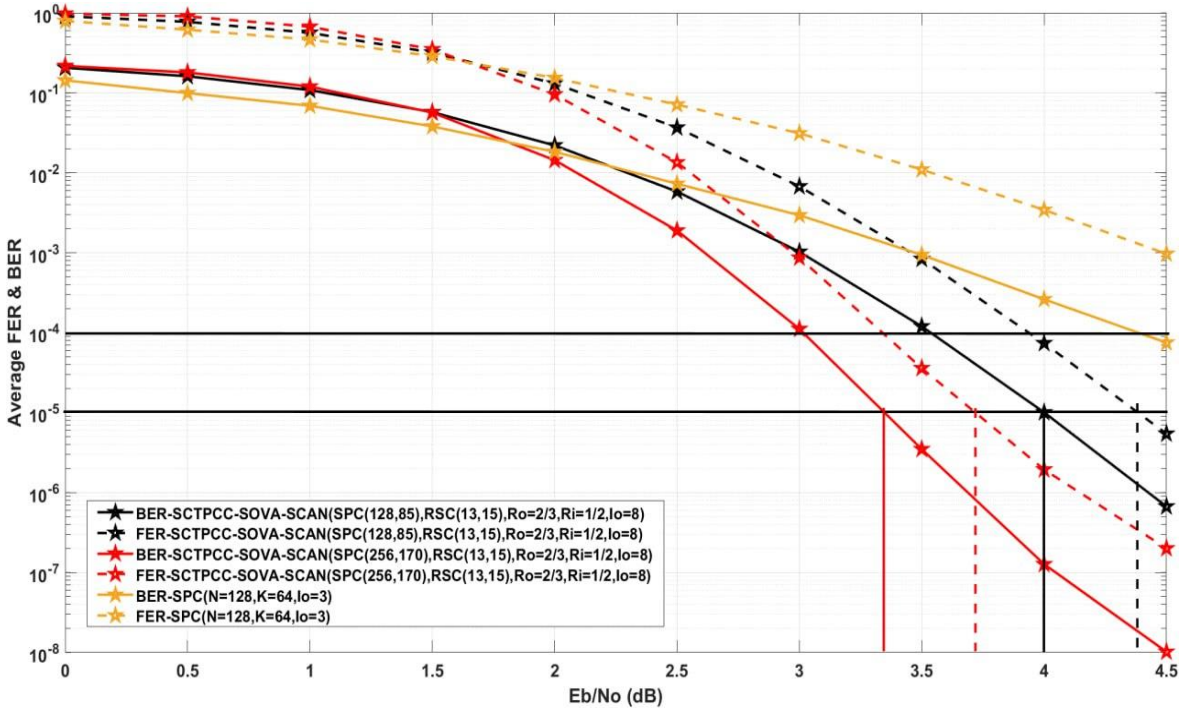


Figure (4.26): The BER and FER Performance of the SOVA-SCTPCC scheme and the SPC(N=128, K=64).

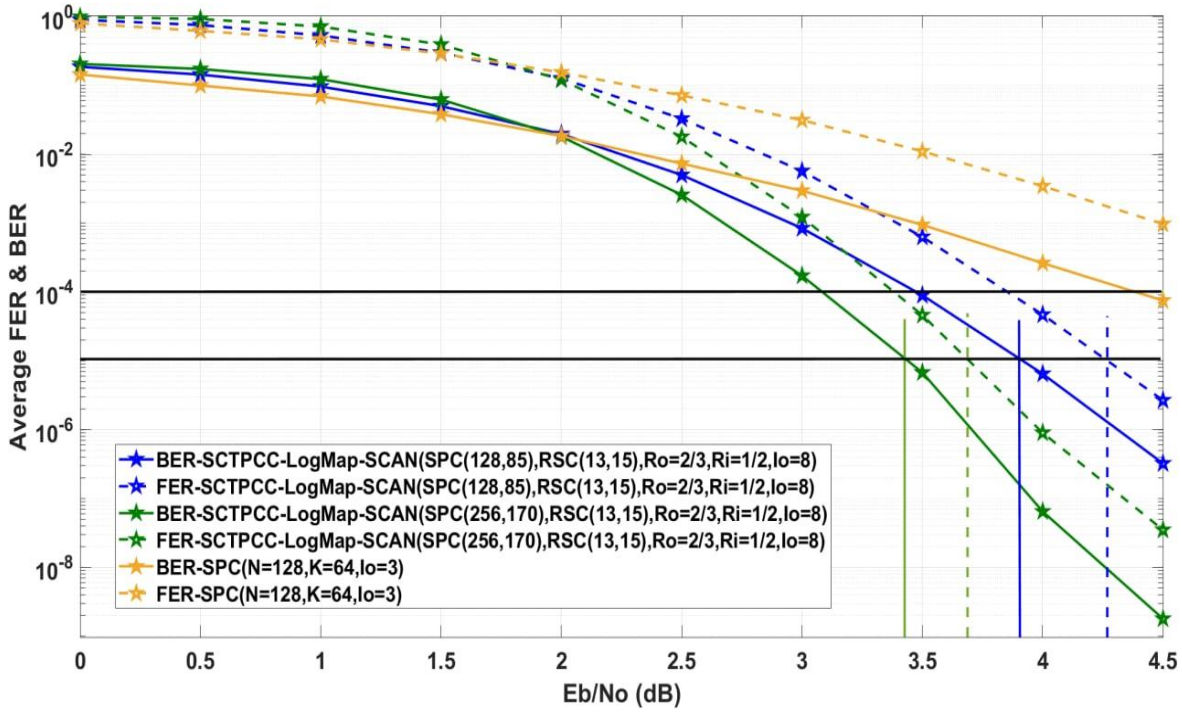


Figure (4.27): The BER and FER Performance of the LogMap-SCTPCC scheme and the SPC(128,64).

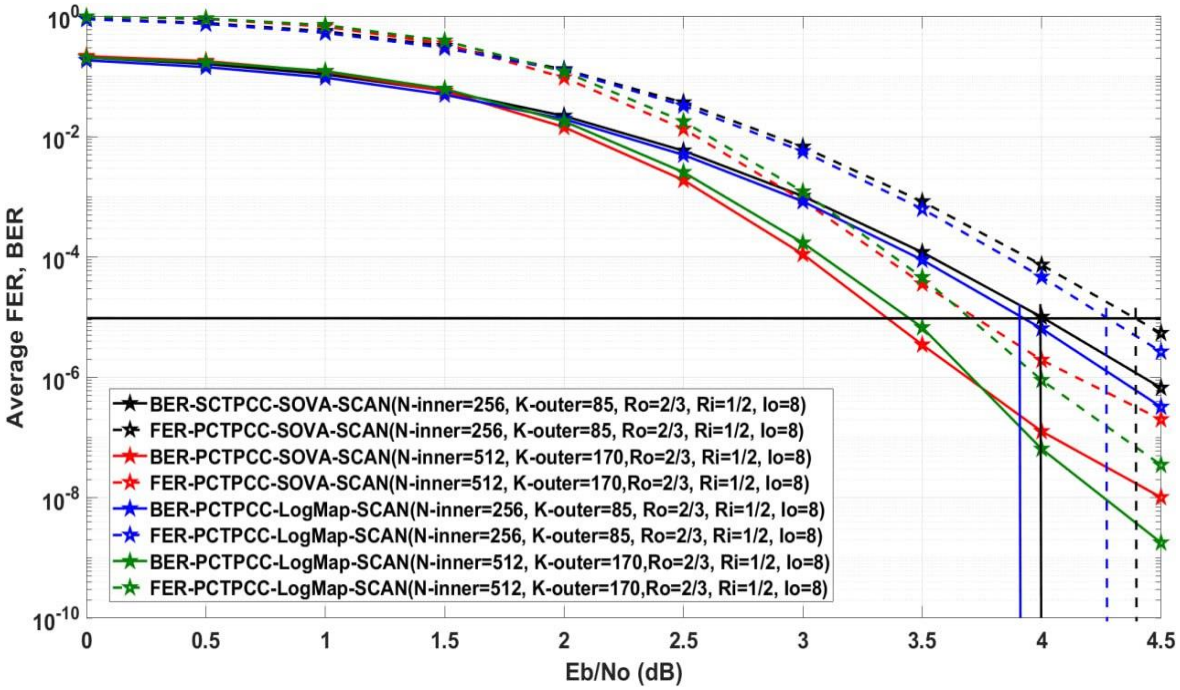
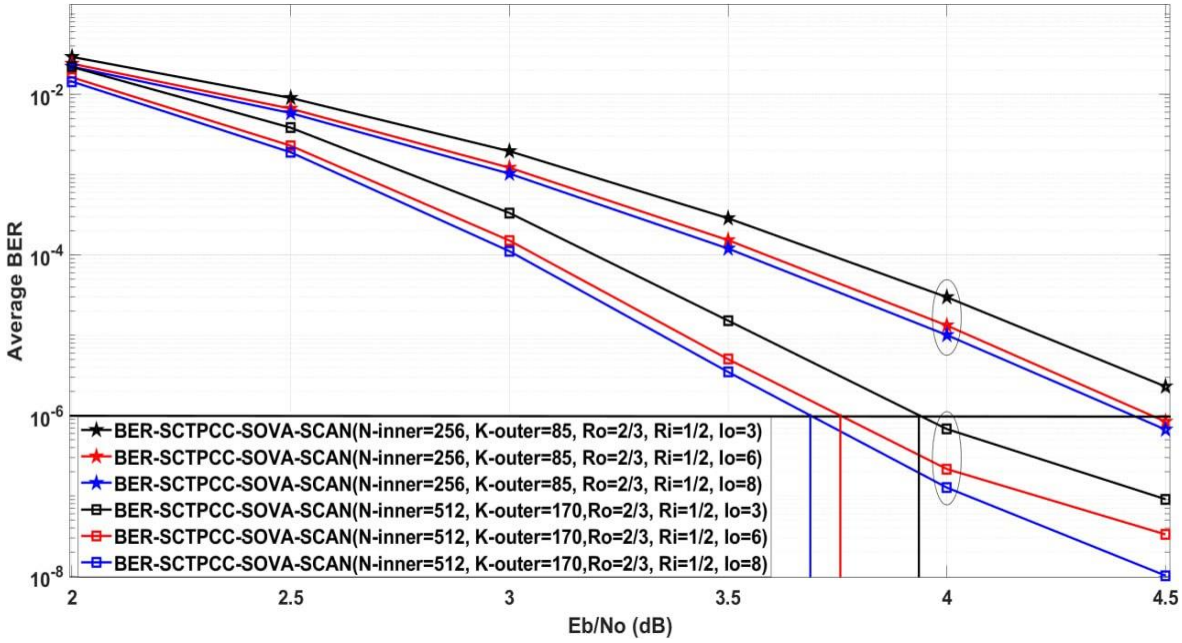


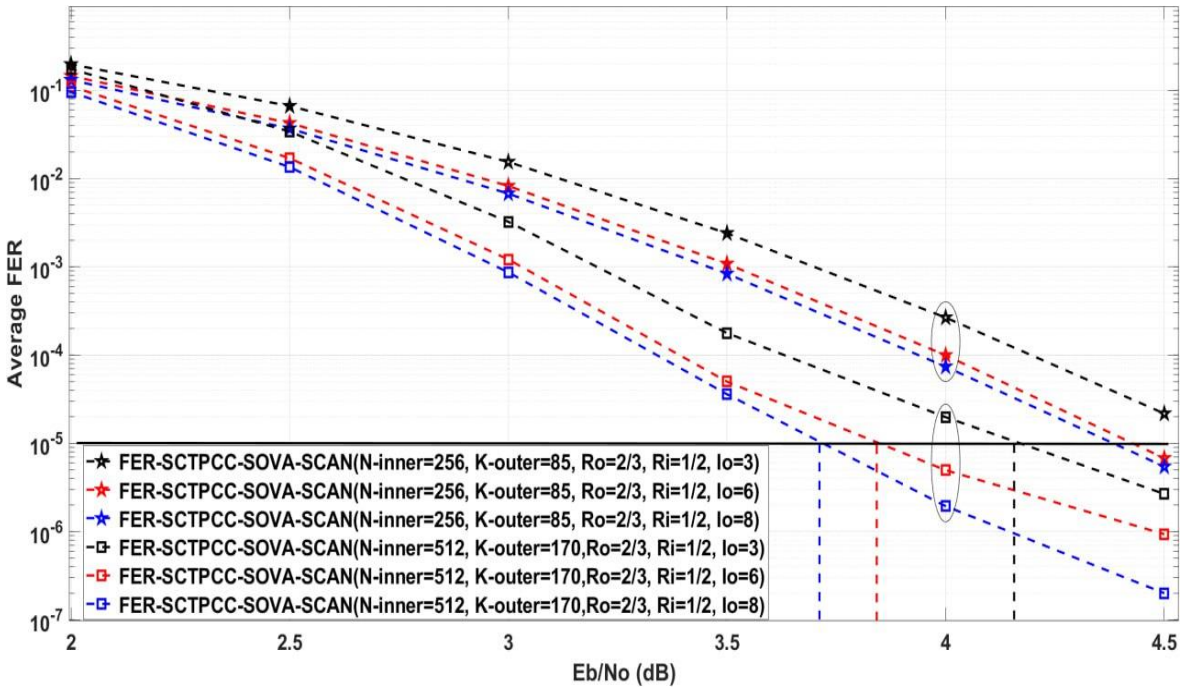
Figure (4.28): Performance Comparison Between the SOVA-SCTPCC and LogMap-SCTPCC Schemes.

Figures (4.29) and (4.30) show the effect of the outer iterations on the BER and FER performance of the suggested SCTPCC scheme with both SOVA and LogMap decoders. These Figures show that the performance can be improved by increasing the outer iterations from three to eight. At BER of 10^{-6} and FER of 10^{-5}, Figure (4.29) shows that the SOVA-SCTPCC(N_{inner}= 512, K_{outer}= 170) scheme with eight iterations outperforms the same scheme with three and six iterations by about (0.25dB & 0.45dB) and (0.07dB & 0.13dB) gains, respectively. At the same time, Figure (4.30) shows that the LogMap-SCTPCC(N_{inner}= 256, K_{outer}= 85) scheme with eight iterations outperforms the same scheme with three and six iterations by about (0.2dB & 0.2dB) and (0.05dB & 0.05dB) gains, respectively, at BER of 10^{-6} and FER of 10^{-5}. Figures (4.29) and (4.30) show that the performance of the suggested SCTPCC at six iterations is close to that of eight. Therefore the six iterations can be used to overcome the higher complexity and latency as the number of iterations increases. Finally,

Figure (4.31) shows that the practical results of the SOVA-SCTPCC scheme are almost identical to the simulation results. Therefore we can conclude the FPGA implementation work well as the simulator.

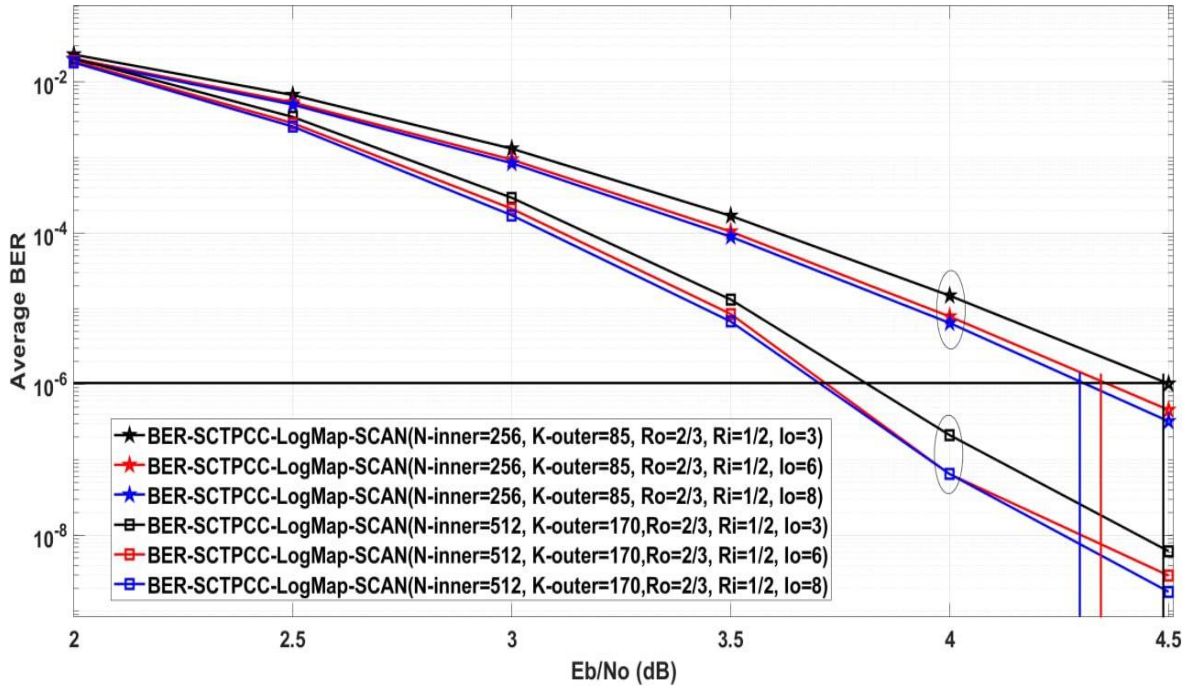


(a)

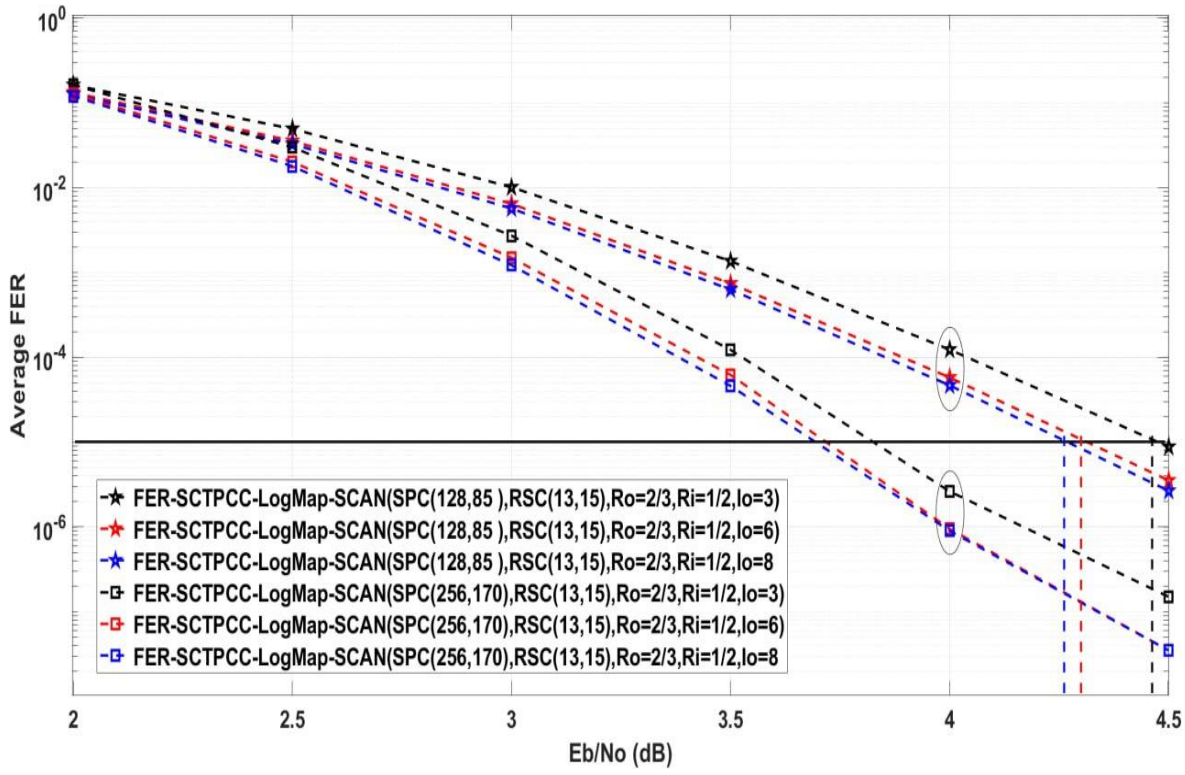


(b)

Figure (4.29): The Performance of the SOVA-SCTPCC with 3, 6, and 8 Iterations: (a) BER Performance, (b) FER Performance.



(a)



(b)

Figure (4.30): The Performance of the LogMap-SCTPCC with 3, 6, and 8 Iterations: (a) BER Performance, (b) FER Performance.

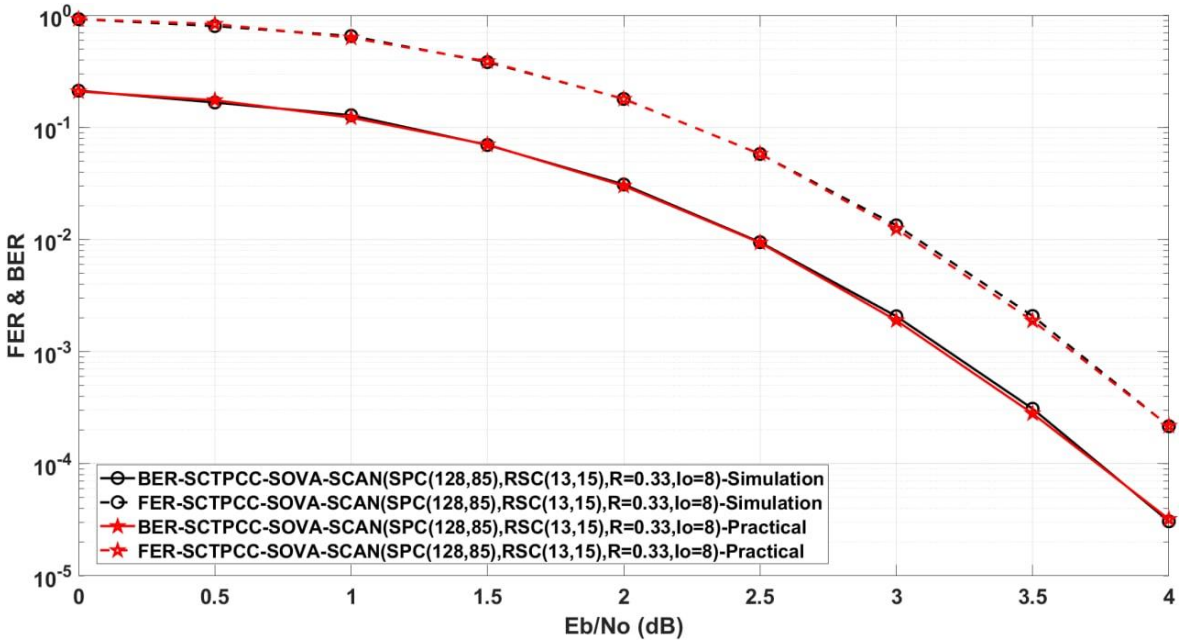


Figure (4.31): Performance Comparison Between the Simulation and Practical Results of the Suggested SCTPC scheme.

4.6. Comparison between the Proposed PCTPC and PCTPCC, and Other Schemes

It is very important to compare the various concatenated polar codes mentioned previously in this chapter with other similar codes. Therefore, in this section, the proposed parallel concatenated polar codes are compared with each other and with other codes in [33]–[36]. Table (4.10) summarizes the parameters of the proposed concatenated codes and the other codes in [33]–[36].

Figure (4.32) shows that the proposed SOVA-PCTPCC(Nc=384, K=125) outperforms the proposed PCTPC(Nc=384, K=128) scheme with gains of 0.4dB and 0.295dB at BER & FER of 10^{-6}. While the LogMap-PCTPCC(Nc=384, K=125) exceeds the PCTPC(Nc=384, K=128) by gains of about 0.525dB and 0.5dB. At BER of 10^{-4}, the SOVA and LogMap PCTPCC(Nc=192, K=61) schemes outperform the PCTPC(Nc=192, K=64) with gains of 0.075dB and 0.145dB, respectively, as shown in Figure (4.33). This Figure also shows that the PCTPC scheme roughly matches the Z.Liu

SSCL-PCTPC scheme of $I_{list} = 32$, although the complexity of the proposed PCTPC scheme is greater than that of the SSCL-PCTPC scheme. The SOVA-PCTPCC exceeds the D.Wu BP-PCTPC[33], Z.Liu SCANPCTPC[34], and Z.Liu BP-PCTPC[34] schemes by gains of 0.215dB, 0.43dB, 0.55dB, respectively, at BER of 10^{-4}. Furthermore, Figure (4.33) shows the superiority of the LogMap-PCTPCC over the D.Wu BP-PCTPC[33], Z.Liu SCAN-PCTPC[34], and Z.Liu BP-PCTPC[34] schemes by gains of 0.285dB, 0.5dB, 0.6dB, respectively.

Table (4.10): Parameters of the Studied Schemes

Code Name/ Author	Constituent Codes		Rate	Iterations		K/Nc
	Code-I	Code-II		Inner	Outer	
BP-PCTPC/ D.Wu [33]	SPC(128,64)		1/3	60	6	64/192
BP-PCTPC/ Z.Liu [34]	SPC(128,64)		1/3	60	6	64/192
SCAN-PCTPC/ Z.Liu [34]	SPC(128,64)		1/3	-	6	64/192
SSCL-PCTPC/ Z.Liu [34]	SPC(128,64)		1/3	I_{list} = 8-64	6	64/192
Optimized BP-PCTPC/ Z.Liu [35]	SPC(128, 64)		1/3	60	3	64/192
Optimized SCAN-PCTPC/ Z.Liu [35]	SPC(128, 64)		1/3	-	6	64/192
Punctured PCTPC/ Y.Qi [36]	SPC(128,64) SPC(512,256)		1/2 1/3	4	18	64/128 256/768
Proposed PCTPC	SPC(128,64) SPC(256,128)		1/3	1	3-8	64/192
Proposed SOVA-PCTPCC & LogMap-PCTPCC	RSC (13,15)	SPC(128,64) SPC(256,128) SPC(512,256)	≅1/3	1	3-8	61/192 125/384 255/768
Proposed SOVA-SCTPCC & LogMap-SCTPCC	SPC(128,85) SPC(256,170) SPC(512,341)	RSC (13,15)	≅1/3	1	3-8	85/256 170/512 341/1024

As a comparison, Figures (4.34) and (4.35) show the superiority of the SOVA and LogMap PCTPCC(Nc=192, K=61) schemes over the optimized Z.Liu-2018 SCAN-PCTPC(Nc=192, K=64) and BP-PCTPC(Nc=192, K=64) schemes. At a BER of 10^{-5}, the proposed SOVA-PCTPCC scheme exceeds the optimal-SF and 1s-SF SCAN-PCTPC(Nc=192, K=64) schemes by gains of 0.025dB and 0.25dB, respectively, as shown in Figure (4.34). At the same time, the proposed LogMap-PCTPCC(Nc=192, K=61) scheme exceeds the optimal-SF and 1s-SF SCAN-PCTPC(Nc=192, K=64) schemes by gains of 0.14dB and 0.375dB, respectively, at a BER of 10^{-5}. At the same BER, Figure (4.35) shows that the proposed SOVA and LogMap (Nc=192, K=61)schemes outperform the optimal-SF BP-PCTPC(Nc=192, K=64) scheme by gains of 0.245dB and 0.375dB, respectively.

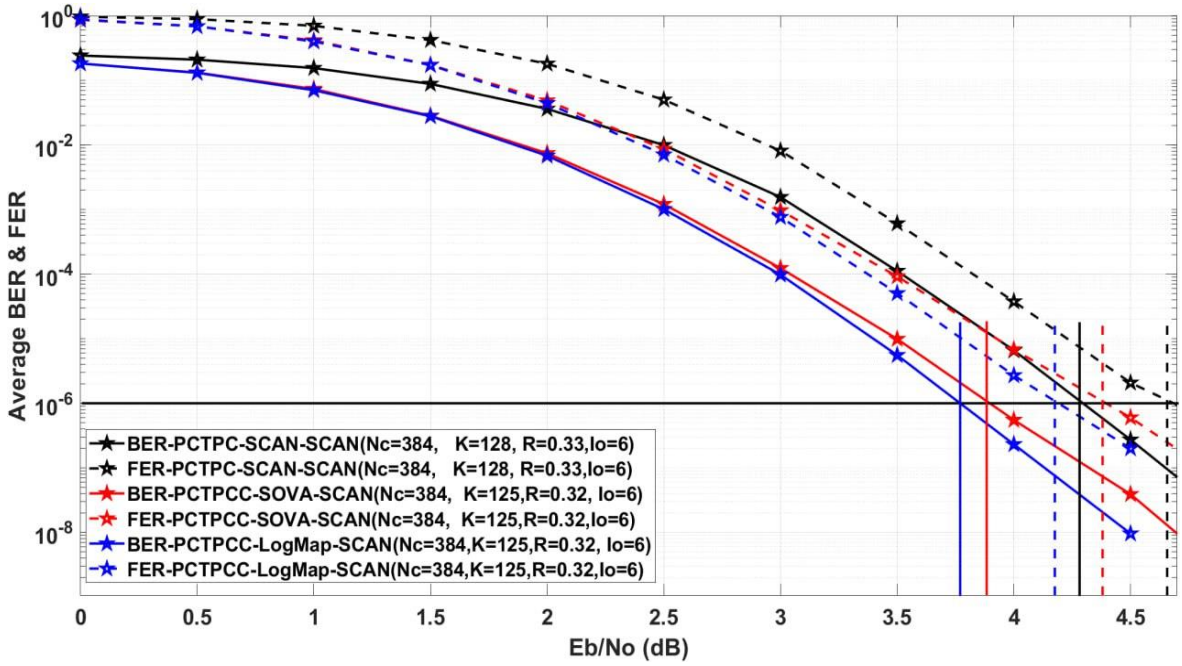


Figure (4.32): Performance Comparison Between the Suggested PCTPC and PCTPCC Schemes.

Eventually, Figure (3.36) shows that the proposed PCTPC and SOVA-PCTPCC(Nc=192, K=61) schemes outperform the punctured Y.Qi-2019 PCTPC(Nc=128, K=64) scheme by gains of 0.3dB and 0.5dB, respectively, at FER of 10^{-4}. While at FER of 10^{-5} and a rate of 0.33, the proposed SOVA-

PCTPCC(Nc=768, K=253) scheme exceeds the Y.Qi-2019 PCTPC(Nc=768, K=256) scheme by a gain of 0.4dB. Furthermore, due to the single-iteration SCAN algorithm, the proposed PCTPCC scheme has less complexity than the Y.Qi-2019 scheme that uses the fourth-iteration SCAN algorithm.

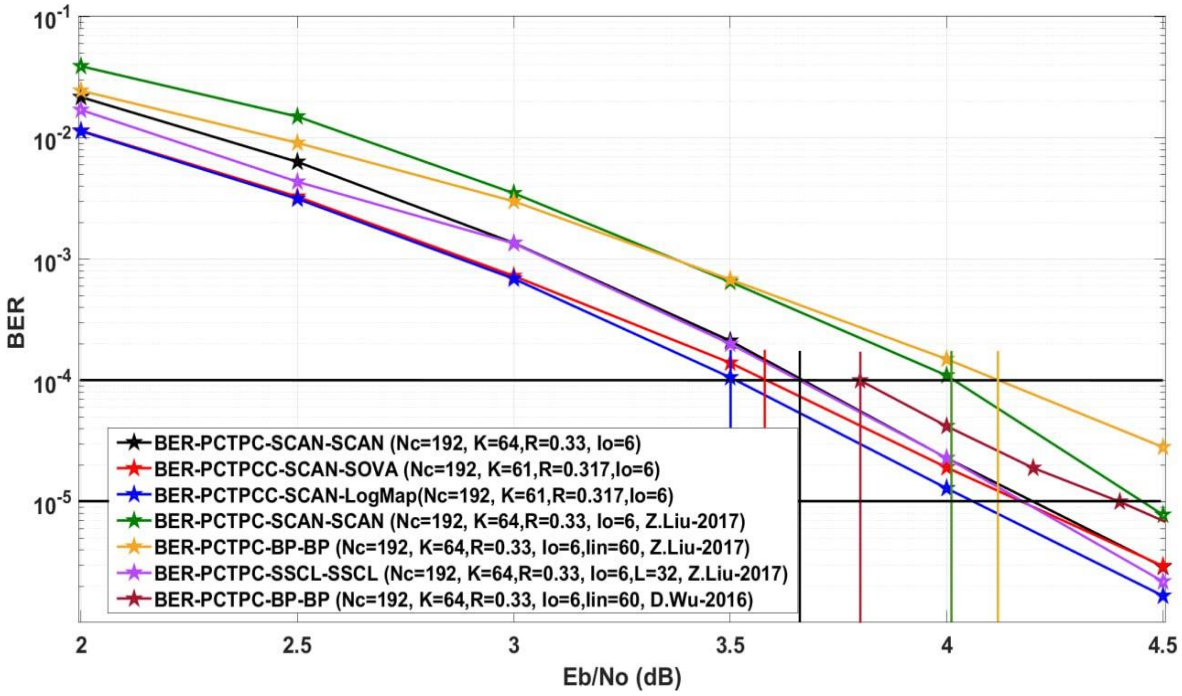


Figure (4.33): Performance Comparison Between the Proposed Parallel Concatenated Codes and other Codes in [33] (i.e., D.Wu-2016) and [34] (i.e., Z.Liu-2017).

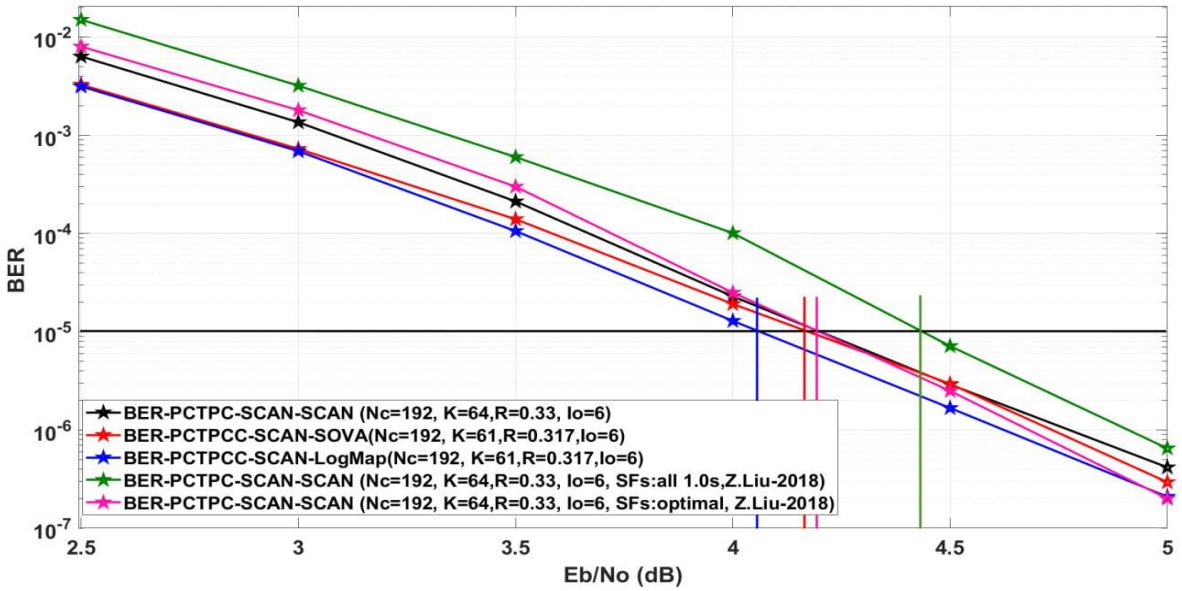


Figure (4.34): Performance Comparison Between the Proposed Parallel Concatenated Codes and SCAN-PCTPC in [35] (Z.Liu-2018).

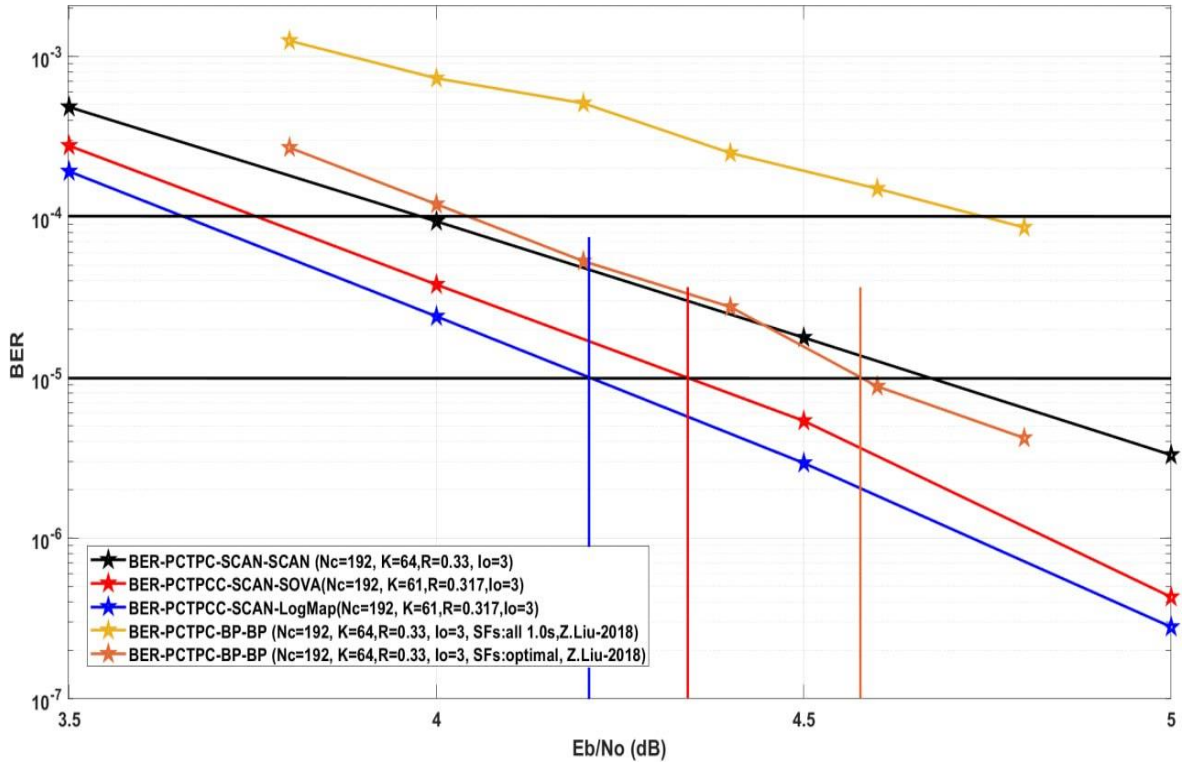


Figure (4.35): Performance Comparison Between the Proposed Parallel Concatenated Codes and BP-PCTPC in [35] (Z.Liu-2018).

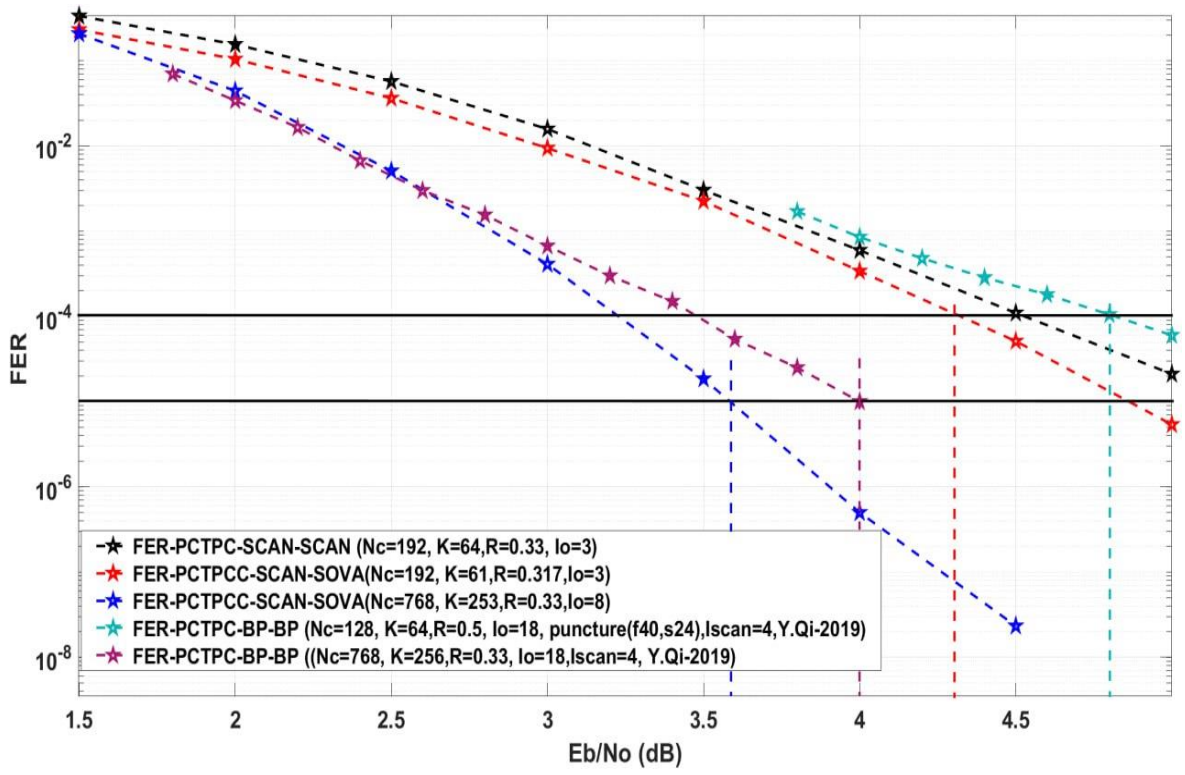


Figure (4.36): Performance Comparison Between the Proposed Parallel Concatenated Codes and SCAN-PCTPC in [36] (i.e., Y.Qi-2019).

4.7. Comparison between the Proposed Parallel and Serial Turbo Schemes, and Other Schemes

The previous section shows that the proposed parallel schemes outperform other schemes presented in [33]–[36]. In this section, the performance of the proposed SCTPCC scheme is compared with the proposed parallel schemes and the same schemes presented in the previous section. Figure (4.37) shows that the proposed PCTCCs outperform the proposed SCTPCCs in low SNR regions, while the SCTPCC superiors the PCTPCCs in high SNR regions. At BER & FER of 10^{-2}, the PCTCC(Nc=192, K=61) schemes exceed the SOVA and LogMap SCTPCC(N_{inner}=256, K_{outer}=85) schemes by gains of (0.3dB &0.19dB) and (0.225dB & 0.125dB), respectively. While at FER of 10^{-5}, the SOVA-SCTPCC(N_{inner}=256, K_{outer}=85) scheme outperforms the SOVA and LogMap PCTPCC(Nc=192, K=61) schemes with gains of 0.34dB & 0.25dB, respectively, and the LogMap-SCTPCC(N_{inner}=256, K_{outer}=85) scheme outperforms them with gains of 0.4dB and 0.36dB. At BER of 10^{-6}, the SOVA-SCTPCC(N_{inner}=256, K_{outer}=85) scheme superiors the SOVA and LogMap PCTPCC(Nc=192, K=61) schemes by gains of 0.26dB & 0.15dB, respectively, and the LogMap-SCTPCC(N_{inner}=256, K_{outer}=85) scheme superiors them by gains of 0.36dB and 0.255dB.

The second scenario in this section includes a comparison between the proposed SCTPCC and the polar turbo codes in [33]–[36], as shown in Figures (4.38) and (4.39). First, at a BER of 10^{-4}, the 6th-iteration LogMap-SCTPCC(N_{inner}=256, K_{outer}=85) scheme superiors the Z.Liu-2017 SCAN-PCTPC(Nc=192, K=64) and BP-PCTPC(Nc=192, K=64) schemes by gains of about 0.5dB and 0.6dB. Second, at a BER of 10^{-5}, the 6th-iteration LogMap-SCTPCC(N_{inner}=256, K_{outer}=85) outperforms the D.Wu-2016 BP-PCTPC(Nc=192, K=64) with gains of about 0.45dB, and the 3rd-iteration

LogMap-SCTPCC(N_{inner}=256, K_{outer}=85) outperforms the Z.Liu-2018 BP-PCTPC(Nc=192, K=64) with a gain of about 0.5dB. Third, at a BER of 10^{-6}, the 6th-iteration LogMap-SCTPCC(N_{inner}=256, K_{outer}=85) scheme outperforms the Z.Liu-2017 SSCL-PCTPC(I_{list}=64) and Z.Liu-2018 SCAN-PCTPC(Nc=192, K=64) schemes with gains of about 0.22dB and 0.31dB. Notably, the Z.Liu-2017 SSCL-PCTPC(Nc=192, K=64) scheme with a list size of 64 almost matches the proposed 8th-iteration LogMap-PCTPCC(Nc=192, K=61) scheme, as shown in Figure (4.38), despite the Z.Liu scheme being more sophisticated than the proposed PCTPCC one. Finally, Figure (4.39) shows the proposed 8th-iteration LogMap-SCTPCC(N_{inner}=256, K_{outer}=85) exceeds the Y.Qi-2019 punctured 18th-iteration SCAN-PCTPC(Nc=128, K=64) scheme by a gain of 0.95dB at FER of 10^{-4}. Moreover, at FER of 10^{-5}, the proposed SOVA-SCTPCC(N_{inner}=1024, K_{outer}=341) scheme outperforms the Y.Qi-2019 punctured and unpunctured schemes with gains of 0.11dB and 0.9dB, respectively.

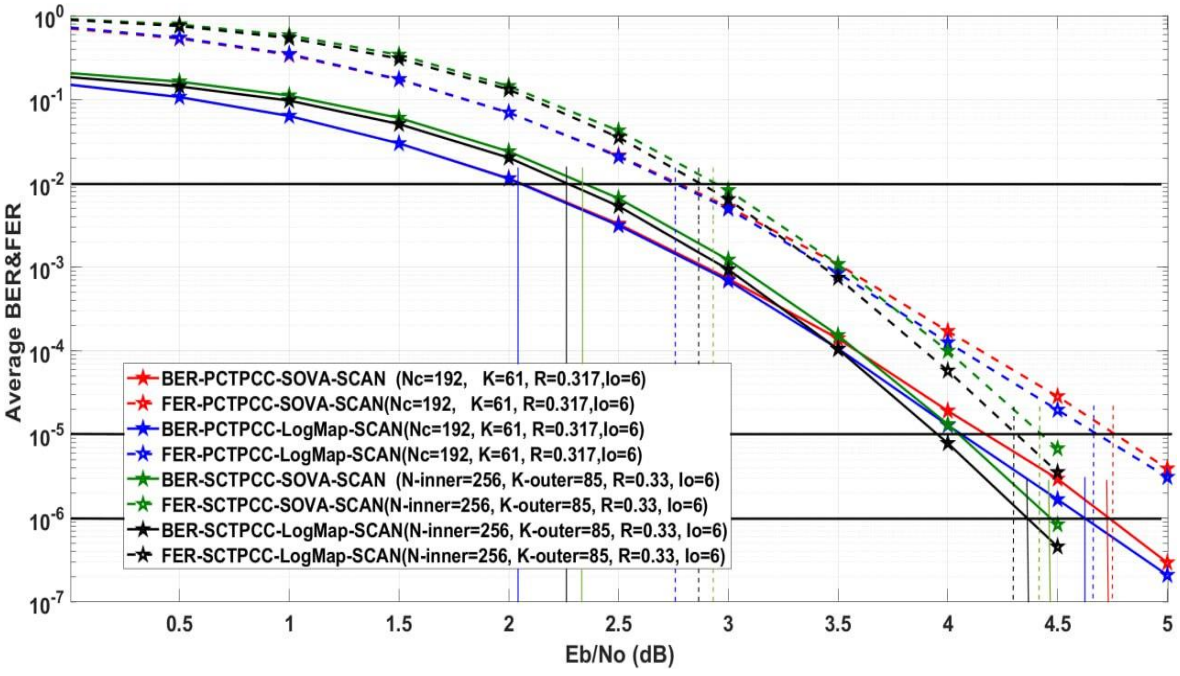


Figure (4.37): Performance Comparison Between the Proposed Parallel and Serial Concatenated Codes.

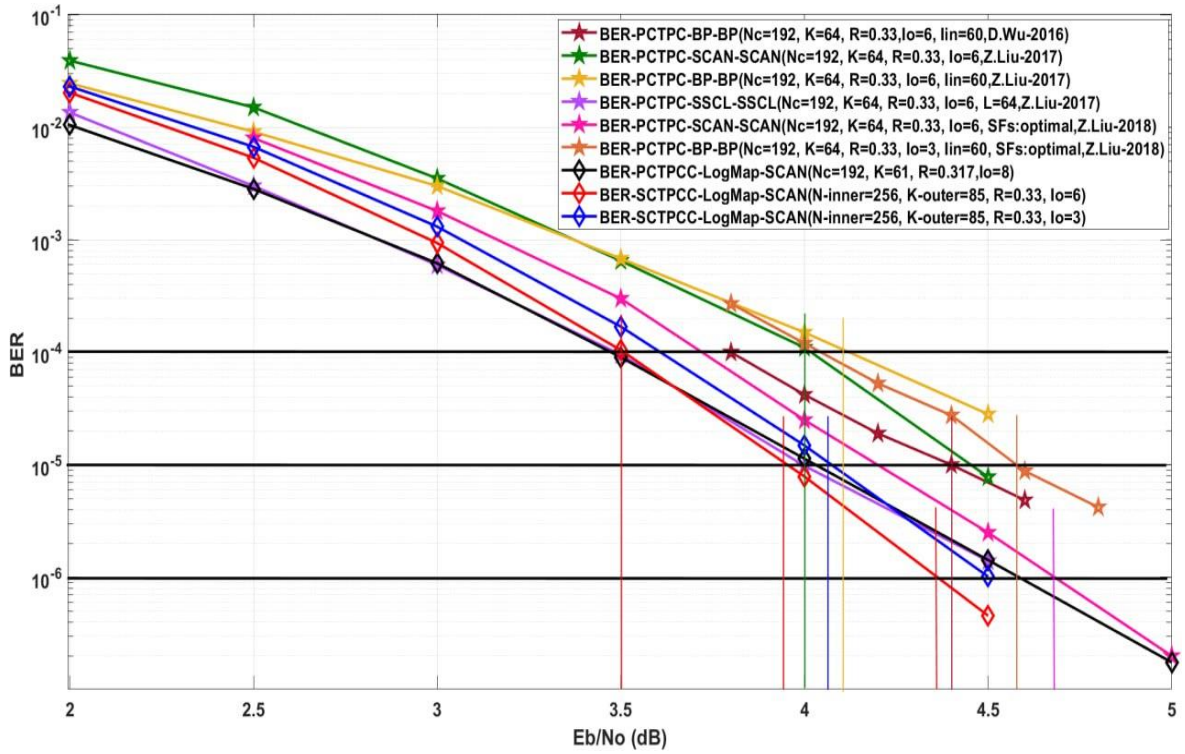


Figure (4.38): Performance Comparison Between the Proposed Serial Concatenated Codes and other Codes in [33]–[35](i.e., D.Wu-2016, Z.Liu-2017, and Z.Liu-2018).

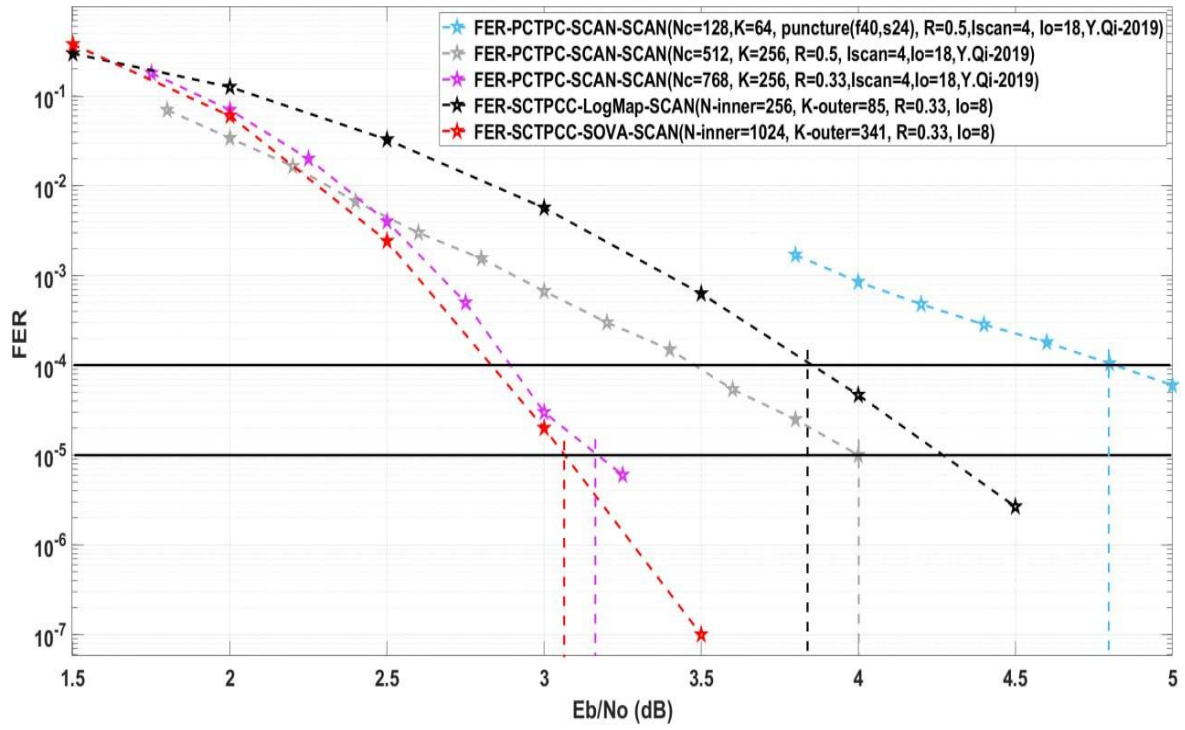


Figure 4.39: Performance Comparison Between the Proposed Serial Concatenated Codes and SCAN-PCTPC in [36] (i.e., Y.Qi-2019).

4.8. Discussion

The previous results show that the deteriorated performance of the finite length polar codes can be improved by using the concatenated turbo polar codes. The proposed concatenated codes are divided into Parallel and Serial Concatenated Turbo Polar Codes. The PCTPC scheme outperforms the original polar codes due to the proposed iterative SCAN decoding algorithm that exploits the prior information exchanged between the constituent SISO decoders to produce a robust decoding algorithm. Furthermore, the PCTPC can be further improved by replacing one of the SPC with an RSC to create the PCTPCC. The PCTPCC outperforms the PCTPC and reduces the error floor problem that typical parallel concatenation codes have. The superiority of PCTPCCs over the PCTPCs returns to the fact that some polar decoders produce invalid codewords in some decoding failures, in contrast to convolutional decoders, which produce valid codewords despite all decoding failures. These invalid codewords can deteriorate the code performance due to successive decoding errors occurring within the system. Therefore, the good error floor performance of polar codes and convolutional decoders' ability to produce only valid codewords make PCTPCCs perform very well and outperform the other PCTPCs. Moreover, the SCTPCC is presented in this dissertation to largely eliminate the error floor problem. The SCTPCC offers well-improved performance compared to other polar codes.

The low complexity of the proposed concatenated codes returns to the single iteration of the SCAN decoder. Complexity analysis shows that the proposed codes offered an excellent complexity level compared to other concatenated polar codes. Therefore, the FPGA implementation of the proposed codes requires a modest utilization of resources.

5

Chapter Five

Conclusions and Future Works

Chapter Five

Conclusions and Future Works

5.1. Conclusions

In this study, several turbo polar codes are presented in addition to the original systematic and Non-Systematic polar codes based on the Soft Cancellation (SCAN) algorithm. Moreover, these codes are implemented using the Xilinx version of Kintex-7™ Field Programmable Gate Array (FPGA) to overcome some of the hardware implementation challenges. The proposed turbo polar codes can be classified into Parallel and Serial Turbo Polar Codes. The parallel category includes Parallel Concatenated Turbo Polar Codes (PCTPCs) and Parallel Concatenated Turbo Polar-Convolutional Codes (PCTPCCs), while the serial category includes Serial Concatenated Turbo Polar-Convolutional Codes (SCTPCCs).

Several points were reached during the implementation of the proposed systems, which are summarized below:

1. The Proposed polar codes are introduced to improve the performance of polar codes in the finite code-length regime with a modest complexity level.
2. The single-Iteration SCAN decoding algorithm has played a central role in reducing the complexity level of the proposed concatenated turbo polar codes compared to current state-of-the-art codes.
3. The PCTPC schemes Outperforms the original SPC with an acceptable complexity level.
4. The introduction of the PCTPCCs can further aggrandize the enhancement achieved by PCTPCs.

5. The superiority of PCTPCCs over the TPCs returns to the fact that some polar decoders produce invalid codewords in some decoding failures, in contrast to convolutional decoders, which produce valid codewords despite all decoding failures. These invalid codewords can deteriorate the code performance due to successive decoding errors occurring within the system. Therefore, the good error floor performance of polar codes and convolutional decoders' ability to produce only valid codewords make PCTPCCs perform very well and outperform the other TPCs.
6. Although the proposed PCTPCC reduces the error floor problem inherent in the parallel turbo codes, this problem persists. Therefore, the SCTPCCs have been presented to eliminate this problem largely.
7. At High SNR, the SCTPCC scheme outperforms the PCTPCC schemes, whereas, the PCTPCCs are superior to SCTPCCs in low SNR regions.
8. The trade-off between minimizing resource utilization and maximizing throughput complicates choosing the perfect FPGA implementation of the proposed codes.
9. The pipeline directive and the arbitrary precision data types can be used to reach the appropriate solution that achieves acceptable resource utilization and good throughput.

5.2. Recommendations For Future Works

In this section, some future suggestions are listed to improve the proposed schemes and codes, as follows:

1. Other constituent SISO decoders like Soft Successive Cancellation list (SSCL) and BP could replace the SCAN decoder. On the other hand, The CRC code can be used to increase the reliability of the selected decoding path.

2. The performance of Turbo Polar Codes can be further enhanced by replacing the fixed scaling factors with adaptive ones depending on the relationship between intrinsic and extrinsic information. Furthermore, the average number of iterations can be reduced using an early termination mechanism.
3. Use the proposed systems with multiple-access techniques such Sigle-Carrier or Multi-Carriers FDMA-DS-CDMA to improve the BER performance by reducing the system PAPR (Peak to Average Power Ratio).
4. Improve the performance of an optical system by using the proposed polar codes with the all-optical OFDM system.
5. Reimplement the FPGA designs by two separate boards with wireless connection.

REFERENCES

References

[1] C. E. Shannon, “A Mathematical Theory of Communication,” *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x.

[2] Y. Polyanskiy, H. V. Poor, and S. Verdú, “Channel Coding Rate in the Finite Blocklength Regime,” *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2307–2359, 2010, doi: 10.1109/TIT.2010.2043769.

[3] H. VANGALA, “Efficient Algorithms for Polar Codes,” Monash University, 2017. doi: 10.4225/03/5a1de0e3407cb.

[4] Y. Polyanskiy, “Channel Coding: Non-Asymptotic Fundamental Limits,” Princeton University, 2010.

[5] T. J. Richardson and R. L. Urbanke, “Efficient Encoding of Low-Density Parity-Check Codes,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, 2001, doi: 10.1109/18.910579.

[6] B. K. Butler and P. H. Siegel, “Error Floor Approximation for LDPC Codes in the AWGN Channel,” *IEEE Trans. Inf. Theory*, vol. 60, no. 12, pp. 7416–7441, 2014, doi: 10.1109/TIT.2014.2363832.

[7] B. K. Butler, “Error Floors of LDPC Codes and Related Topics,” University of California, San Diego, 2013. [Online]. Available: https://escholarship.org/uc/item/1gr3m0wm

[8] R. Garello, F. Chiaraluce, P. Pierleoni, M. Scaloni, and S. Benedetto, “On Error Floor and Free Distance of Turbo Codes,” in *IEEE International Conference on Communications*, 2001, vol. 1, pp. 45–49. doi: 10.1109/ICC.2001.936270.

[9] R. Garello, P. Pierleoni, and S. Benedetto, “Computing the Free Distance of Turbo Codes and Serially Concatenated Codes with Interleavers: Algorithms and Applications,” *IEEE J. Sel. Areas Commun.*, vol. 19, no. 5, pp. 800–812, 2001, doi: 10.1109/49.924864.

[10] M. Awais, A. Singh, E. Boutillon, and G. Masera, “A Novel Architecture for Scalable, High Throughput, Multi-Standard LDPC Decoder,” in *Proceedings - 2011 14th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2011*, 2011, pp. 340–347. doi: 10.1109/DSD.2011.112.

[11] S. U. Rehman, A. Sani, P. Coussy, and C. Chavet, “On-Chip Implementation of Memory Mapping Algorithm to Support Flexible Decoder Architecture,” in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2013, pp.

2751–2755. doi: 10.1109/ICASSP.2013.6638157.

[12] E. Arikan, “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels,” *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, 2009, doi: 10.1109/TIT.2009.2021379.

[13] L. Liu and C. Zhang, “Circuits and Systems for 5G Network: Massive MIMO and Advanced Coding,” in *Proceedings - 2015 IEEE 11th International Conference on ASIC, ASICON 2015*, 2016, pp. 1–4. doi: 10.1109/ASICON.2015.7517047.

[14] E. Arikan, “A Performance Comparison of Polar Codes and Reed-Muller Codes,” *IEEE Commun. Lett.*, vol. 12, no. 6, pp. 447–449, 2008, doi: 10.1109/LCOMM.2008.080017.

[15] I. Tal and A. Vardy, “List Decoding of Polar Codes,” in *IEEE International Symposium on Information Theory - Proceedings*, 2011, pp. 1–5. doi: 10.1109/ISIT.2011.6033904.

[16] U. U. Fayyaz and J. R. Barry, “Polar Codes for Partial Response Channels,” in *IEEE International Conference on Communications*, 2013, pp. 4337–4341. doi: 10.1109/ICC.2013.6655247.

[17] U. U. Fayyaz and J. R. Barry, “Low-Complexity Soft-Output Decoding of Polar Codes,” in *GLOBECOM - IEEE Global Telecommunications Conference*, 2013, pp. 2692–2697. doi: 10.1109/GLOCOM.2013.6831481.

[18] V. Taranalli and P. H. Siegel, “Adaptive Linear Programming Decoding of Polar Codes,” in *IEEE International Symposium on Information Theory - Proceedings*, 2014, pp. 2982–2986. doi: 10.1109/ISIT.2014.6875381.

[19] H. Mahdavifar, M. El-Khamy, J. Lee, and I. Kang, “Fast Multi-Dimensional Polar Encoding and Decoding,” in *2014 Information Theory and Applications Workshop, ITA 2014 - Conference Proceedings*, 2014, pp. 1–5. doi: 10.1109/ITA.2014.6804236.

[20] D. Wu, Y. Li, X. Guo, and Y. Sun, “Ordered Statistic Decoding for Short Polar Codes,” *IEEE Commun. Lett.*, vol. 20, no. 6, pp. 1064–1067, 2016, doi: 10.1109/LCOMM.2016.2539170.

[21] O. Dizdar and E. Arikan, “A High-Throughput Energy-Efficient Implementation of Successive Cancellation Decoder for Polar Codes Using Combinational Logic,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 63, no. 3, pp. 436–447, 2016, doi: 10.1109/TCSI.2016.2525020.

[22] G. Sarkis, I. Tal, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross,

"Flexible and Low-Complexity Encoding and Decoding of Systematic Polar Codes," *IEEE Trans. Commun.*, vol. 64, no. 7, pp. 2732–2745, 2016, doi: 10.1109/TCOMM.2016.2574996.

[23] G. Berhault, C. Leroux, C. Jego, and D. Dallet, "Memory Requirement Reduction Method for Successive Cancellation Decoding of Polar Codes," in *Journal of Signal Processing Systems*, 2017, vol. 88, no. 3, pp. 425–438. doi: 10.1007/s11265-016-1179-5.

[24] P. Giard, G. Sarkis, C. Leroux, C. Thibeault, and W. J. Gross, "Low-Latency Software Polar Decoders," *J. Signal Process. Syst.*, vol. 90, no. 5, pp. 761–775, 2018, doi: 10.1007/s11265-016-1157-y.

[25] M. Bakshi, S. Jaggi, and M. Effros, "Concatenated Polar Codes," in *IEEE International Symposium on Information Theory - Proceedings*, 2010, pp. 918–922. doi: 10.1109/ISIT.2010.5513508.

[26] A. Eslami and H. Pishro-Nik, "On Finite-Length Performance of Polar Codes: Stopping Sets, Error Floor, and Concatenated Design," *IEEE Trans. Commun.*, vol. 61, no. 3, pp. 919–929, 2013, doi: 10.1109/TCOMM.2013.012313.110692.

[27] H. Mahdavifar, M. El-Khamy, J. Lee, and I. Kang, "Performance Limits and Practical Decoding of Interleaved Reed-Solomon Polar Concatenated Codes," *IEEE Trans. Commun.*, vol. 62, no. 5, pp. 1406–1417, 2014, doi: 10.1109/TCOMM.2014.050714.130602.

[28] Y. Wang and K. R. Narayanan, "Concatenations of Polar Codes with Outer BCH Codes and Convolutional Codes," *2014 52nd Annu. Allert. Conf. Commun. Control. Comput. Allert. 2014*, pp. 813–819, 2014, doi: 10.1109/ALLERTON.2014.7028538.

[29] J. Guo, M. Qin, A. Guillen I Fabregas, and P. H. Siegel, "Enhanced Belief Propagation Decoding of Polar Codes Through Concatenation," *IEEE Int. Symp. Inf. Theory - Proc.*, no. 1, pp. 2987–2991, 2014, doi: 10.1109/ISIT.2014.6875382.

[30] M. A. Hanif and S. Vafi, "A Modified Approach to Punctured Product Polar Codes," *J. Telecommun. Inf. Technol.*, vol. 3, no. 3, pp. 63–69, 2019, doi: 10.26636/jtit.2019.132219.

[31] X. Wang, J. Li, H. Chang, and J. He, "Optimization Design of Polar-LDPC Concatenated Scheme Based on Deep Learning," *Comput. Electr. Eng.*, vol. 84, p. 106636, 2020, doi: 10.1016/j.compeleceng.2020.106636.

[32] Q. Zhang, A. Liu, Y. Zhang, and X. Liang, "Practical Design and Decoding of Parallel Concatenated Structure for Systematic Polar

Codes," *IEEE Trans. Commun.*, vol. 64, no. 2, pp. 456–466, 2016, doi: 10.1109/TCOMM.2015.2502246.

[33] D. Wu, A. Liu, Y. Zhang, and Q. Zhang, "Parallel Concatenated Systematic Polar Codes," *Electron. Lett.*, vol. 52, no. 1, pp. 43–45, 2016, doi: 10.1049/el.2015.2592.

[34] Z. Liu, K. Niu, and J. Lin, "Parallel Concatenated Systematic Polar Code Based on Soft Successive Cancellation List Decoding," in *International Symposium on Wireless Personal Multimedia Communications, WPMC*, 2018, vol. 2017-Decem, pp. 181–184. doi: 10.1109/WPMC.2017.8301804.

[35] Z. Liu, K. Niu, J. Lin, J. Sun, and H. Guan, "Scaling Factor Optimization of Turbo-Polar Iterative Decoding," *China Commun.*, vol. 15, no. 6, pp. 169–177, 2018, doi: 10.1109/CC.2018.8398513.

[36] Y. Qi, D. Chen, and C. Zhang, "Punctured Turbo-Polar Codes," in *2019 IEEE 11th International Conference on Communication Software and Networks, ICCSN 2019*, 2019, pp. 736–741. doi: 10.1109/ICCSN.2019.8905352.

[37] Z. Liu, K. Niu, C. Dong, and J. Lin, "Adding a Rate-1 Third Dimension to Parallel Concatenated Systematic Polar Code: 3D Polar Code," *Wirel. Commun. Mob. Comput.*, vol. 2018, 2018, doi: 10.1155/2018/8928761.

[38] I. Tal and A. Vardy, "List Decoding of Polar Codes," Lund University, 2015. doi: 10.1109/TIT.2015.2410251.

[39] R. Pedarsani, "Polar Codes: Construction and Performance Analysis," School of Computer and Communication Sciences (IC), 2011. [Online]. Available: http://www.eecs.berkeley.edu/~ramtin/MSThesis.pdf

[40] B. Tahir, "Construction and Performance of Polar Codes for Transmission over the AWGN Channel," Vienna University of Technology, 2017. [Online]. Available: https://publik.tuwien.ac.at/files/publik_262980.pdf

[41] N. Kaur and A. P. S. Kalsi, "Implementation of Polar Codes Over AWGN and Binary Symmetric Channel," *Indian J. Sci. Technol.*, vol. 9, no. 19, 2016, doi: 10.17485/ijst/2016/v9i19/91383.

[42] O. Gazi, *Polar Codes: A Non-Trivial Approach to Channel Coding*, 1st ed., vol. 15. Springer Publishing Company, Incorporated, 2019. doi: 10.1007/978-981-13-0737-9_1.

[43] E. Arikan, "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, 2009, doi:

10.1109/TIT.2009.2021379.

[44] A. Hadi, “Optimisation and Analysis of Polar Codes in Communication Systems,” The University of Manchester (United Kingdom), 2018. [Online]. Available: https://www.research.manchester.ac.uk/portal/en/theses/optimisation-and-analysis-of-polar-codes-in-communication-systems(eb4cfc8a-86b5-465c-9231-4e02a3c2723c).html

[45] E. Arikan and E. Telatar, “On the Rate of Channel Polarization,” in *IEEE International Symposium on Information Theory - Proceedings*, 2009, pp. 1493–1495. doi: 10.1109/ISIT.2009.5205856.

[46] K. Niu, K. Chen, J. Lin, and Q. T. Zhang, “Polar Codes: Primary Concepts and Practical Decoding Algorithms,” *IEEE Commun. Mag.*, vol. 52, no. 7, pp. 192–203, 2014, doi: 10.1109/MCOM.2014.6852102.

[47] D. Wu, Y. Li, and Y. Sun, “Construction and Block Error Rate Analysis of Polar Codes Over AWGN Channel Based on Gaussian Approximation,” *IEEE Commun. Lett.*, vol. 18, no. 7, pp. 1099–1102, 2014, doi: 10.1109/LCOMM.2014.2325811.

[48] A. Sural, “An FPGA Implementation of Successive Cancellation List Decoding for Polar Codes Declaration of Authorship,” bilkent university, 2016. [Online]. Available: http://repository.bilkent.edu.tr/handle/11693/28951

[49] J. Sedin, “A Comparison of Polar Code Constructions and Puncturing Methods for AWGN and Fading Channels,” KTH, School of Electrical Engineering (EES), 2017. [Online]. Available: http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Akth%3Adiva-212307

[50] P. Trifonov, “Efficient Design and Decoding of Polar Codes,” *IEEE Trans. Commun.*, vol. 60, no. 11, pp. 3221–3227, 2012, doi: 10.1109/TCOMM.2012.081512.110872.

[51] E. Arikan, “Systematic Polar Coding,” *IEEE Commun. Lett.*, vol. 15, no. 8, pp. 860–862, 2011, doi: 10.1109/LCOMM.2011.061611.110862.

[52] U. U. Fayyaz, “Polar Code Design and Decoding for Magnetic Recording,” Georgia Institute of Technology, 2014. [Online]. Available: https://smartech.gatech.edu/bitstream/handle/1853/52994/FAYYAZ-DISSERTATION-2014.pdf

[53] C. Kestel, S. Weithoffer, and N. Wehn, “Polar Code Decoder Exploration Framework,” *Adv. Radio Sci.*, vol. 16, pp. 43–50, 2018, doi: 10.5194/ars-16-43-2018.

[54] S. Choi, Y. Lee, and H. Yoo, “Recent Successive Cancellation Decoding Methods for Polar Codes,” *J. Semicond. Eng.*, vol. 1, no. 2, pp. 74–80, 2020, doi: https://doi.org/10.22895/jse.2020.0005.

[55] A. C. Arli, A. Colak, and O. Gazi, “The Implementation of a Successive Cancellation Polar Decoder on Xilinx System Generator,” in *ICECS 2017 - 24th IEEE International Conference on Electronics, Circuits and Systems*, 2018, vol. 2018-Janua, pp. 372–376. doi: 10.1109/ICECS.2017.8291995.

[56] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross, and A. Burg, “Hardware Architecture for List Successive Cancellation Decoding of Polar Codes,” *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 61, no. 8, pp. 609–613, 2014, doi: 10.1109/TCSII.2014.2327336.

[57] U. U. Fayyaz and J. R. Barry, “Low-Complexity Soft-Output Decoding of Polar Codes,” *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 958–966, 2014, doi: 10.1109/JSAC.2014.140515.

[58] L. Zhang, Y. Sun, Y. Shen, W. Song, X. You, and C. Zhang, “Efficient Fast-SCAN Flip Decoder for Polar Codes,” in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2021, vol. 2021-May, pp. 1–5. doi: 10.1109/ISCAS51556.2021.9401398.

[59] C. Pillet, C. Condo, and V. Bioglio, “SCAN List Decoding of Polar Codes,” in *IEEE International Conference on Communications*, 2020, vol. 2020-June, pp. 1–6. doi: 10.1109/ICC40277.2020.9149202.

[60] G. Berhault, C. Leroux, C. Jego, and D. Dallet, “Hardware Implementation of a Soft Cancellation Decoder for Polar Codes,” in *Conference on Design and Architectures for Signal and Image Processing, DASIP*, 2015, vol. 2015-Decem, pp. 1–8. doi: 10.1109/DASIP.2015.7367252.

[61] C. Pillet, C. Condo, and V. Bioglio, “Fast-SCAN Decoding of Polar Codes,” in *2021 11th International Symposium on Topics in Coding, ISTC 2021*, 2021, pp. 1–5. doi: 10.1109/ISTC49272.2021.9594208.

[62] S. Lee, J. Park, I. M. Kim, and J. Heo, “Flexible Soft-Output Decoding of Polar Codes,” *Eurasip J. Wirel. Commun. Netw.*, vol. 2021, no. 1, p. 170, 2021, doi: 10.1186/s13638-021-02042-x.

[63] B. Le Gal, C. Leroux, and C. Jego, “High-Performance Software Implementations of SCAN Decoder for Polar Codes,” *Ann. des Telecommun. Telecommun.*, vol. 73, no. 5–6, pp. 401–412, 2018, doi: 10.1007/s12243-018-0634-7.

[64] J. Castineira and P. Guy, “Essentials of Error-Control Coding.” John

Wiley & Sons Ltd, England, 2006.

[65] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. John Wiley & Sons, 2005. doi: 10.1002/0471739219.

[66] G. C. Clark and J. B. Cain, *Error-Correction Coding for Digital Communications*. Springer Science & Business Media, 1981. doi: 10.1007/978-1-4899-2174-1.

[67] S. J. Johnson, *Iterative Error Correction: Turbo, Low-Density Parity-Check and Repeat-Accumulate Codes*. Cambridge university press, 2010.

[68] J. Hagenauer, "Source-Controlled Channel Decoding," *IEEE Trans. Commun.*, vol. 43, no. 9, pp. 2449–2457, 1995, doi: 10.1109/26.412719.

[69] J. Hagenauer and P. Hoeher, "Viterbi Algorithm with Soft-Decision Outputs and its Applications," in *IEEE Global Telecommunications Conference and Exhibition*, 1989, vol. 3, pp. 1680–1686. doi: 10.1109/glocom.1989.64230.

[70] A. A. Hamad, "Improved Turbo Coded Signals Using Pilot Insertion for Single Carrier and OFDM Transmission," University of Technology, 2007.

[71] J. P. Woodard and L. Hanzo, "Comparative Study of Turbo Decoding Techniques: An Overview," *IEEE Trans. Veh. Technol.*, vol. 49, no. 6, pp. 2208–2233, 2000, doi: 10.1109/25.901892.

[72] C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x.

[73] C. Berrou, R. Pyndiah, P. Adde, C. Douillard, and R. Le Bidan, "An Overview of Turbo Codes and Their Applications," in *Wireless Technology 2005, Conference Proceedings - 8th European Conference on Wireless Technology*, 2005, vol. 2005, pp. 1–10. doi: 10.1109/ecwt.2005.1617639.

[74] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Encoding: Turbo-Codes," in *IEEE International Conference on Communications*, 1993, vol. 2, pp. 1064–1070. doi: 10.1109/icc.1993.397441.

[75] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near Optimum Decoding of Product Codes," in *1994 IEEE GLOBECOM. Communications: The Global Bridge*, 2002, pp. 339–343. doi: 10.1109/glocom.1994.513494.

[76] G. D. Forney, "Concatenated Codes," Massachusetts Institute of

Technology, 1965. [Online]. Available: http://hdl.handle.net/1721.1/13449

[77] S. Benedetto, G. Montorsi, and D. Divsalar, "Concatenated Convolutional Codes with Interleavers," *IEEE Commun. Mag.*, vol. 41, no. 8, pp. 102–109, 2003, doi: 10.1109/MCOM.2003.1222725.

[78] A. Banerjee, F. Vatta, B. Scanavino, and D. J. Costello, "Nonsystematic Turbo Codes," *IEEE Trans. Commun.*, vol. 53, no. 11, pp. 1841–1849, 2005, doi: 10.1109/TCOMM.2005.858672.

[79] P. C. Massey and J. Costello, "Turbo Codes with Recursive Nonsystematic Quick-Look-In Constituent Codes," in *IEEE International Symposium on Information Theory - Proceedings*, 2001, p. 141. doi: 10.1109/isit.2001.936004.

[80] L. C. Perez, J. Seghers, and D. J. Costello, "A Distance Spectrum Interpretation of Turbo Codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1698–1709, 1996, doi: 10.1109/18.556666.

[81] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 909–926, 1998, doi: 10.1109/18.669119.

[82] D. Mishra, T. V. S. Ram, K. S. Dasgupta, and S. Jit, "Concatenated Convolutional Codes for Deep Space Mission," *Int. J. Inf. Commun. Technol. Res.*, vol. 2, no. 6, 2012.

[83] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, 1974, doi: 10.1109/TIT.1974.1055186.

[84] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding," *Eur. Trans. Telecommun.*, vol. 8, no. 2, pp. 119–125, 1997, doi: 10.1002/ett.4460080202.

[85] L. A. Perisoara and R. Stoian, "The Decision Reliability of MAP , Log-MAP , Max-Log-MAP and SOVA Algorithms for Turbo Codes," *Int. J. Commun.*, vol. 2, no. 1, pp. 65–74, 2008, [Online]. Available: https://www.naun.org/main/NAUN/communications/c-37.pdf

[86] J. Hagenauer, P. Robertson, and L. Papke, "Iterative Turbo Decoding of Systematic Convolutional Codes with the MAP and SOVA Algorithms," *Proc. {ITG} Conf.*, pp. 21–29, 1994.

[87] M. R. Soleymani, Y. Gao, and U. Vilaipornsawai, *Turbo Coding for*

Satellite and Wireless Communications, vol. 702. Springer Science & Business Media, 2002. doi: 10.1007/b117297.

[88] M. S. C. Ho, "Serial and Parallel Concatenated Turbo Codes," University of South Australia, 2002.

[89] D. Mishra, S. J. S.Jit, and K. S Dasgupta, "Hybrid Concatenated Convolutional Code for Deep Space Mission," *Int. J. Comput. Appl.*, vol. 47, no. 19, pp. 33–35, 2012, doi: 10.5120/7298-0539.

[90] E. N. ETSI, "300 421 V1. 1.2 (1997-08)," *Digit. Video Broadcast.*.

[91] E. N. ETSI, "301 210 V1. 1.1 (1999-03),"," *Digit. Video Broadcast.*.

[92] T. S. ETSI, "102 441 V1. 1.1 Digital Video Broadcasting (DVB)," *DVB-S2 Adapt. Coding Modul. Broadband Hybrid Satell. Dialup Appl.*, 2005.

[93] S. Lin, *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Inc, 2004. [Online]. Available: https://pg024ec.files.wordpress.com/2013/09/error-control-coding-by-shu-lin.pdf

[94] L. V. Ph.D and S. R. Robinson, *Principles and Practice of Information Security*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[95] E. Arikan, "Serially Concatenated Polar Codes," *IEEE Access*, vol. 6, pp. 64549–64555, 2018, doi: 10.1109/ACCESS.2018.2877720.

[96] K. Niu, K. Chen, and J. R. Lin, "Beyond Turbo Codes: Rate-Compatible Punctured Polar Codes," in *IEEE International Conference on Communications*, 2013, pp. 3423–3427. doi: 10.1109/ICC.2013.6655078.

[97] N. Chandran and M. C. Valenti, "Bridging the Gap Between Parallel and Serial Concatenated Codes," in *Proc. Virginia Tech Symp. Wireless Personal Commun.*, 2002, pp. 145–154.

[98] A. A. Hamad, "Performance Enhancement of SOVA Based Decoder in SCCC and PCCC Schemes," *Wirel. Eng. Technol.*, vol. 04, no. 01, pp. 40–45, 2013, doi: 10.4236/wet.2013.41006.

[99] C. M. Maxfield, *FPGAs: Instant Access*. Newnes, 2008. [Online]. Available:http://www.sciencedirect.com/science/article/pii/B9780750689748000016

[100] V. Pangracious, Z. Marrakchi, and H. Mehrez, *Three-Dimensional Design Methodologies for Tree-based FPGA Architecture*. Springer, 2015. doi: 10.1007/978-3-319-19174-4.

[101] B. Ronak and S. A. Fahmy, "Mapping for Maximum Performance on

FPGA DSP Blocks," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 35, no. 4, pp. 573–585, 2016, doi: 10.1109/TCAD.2015.2474363.

[102] S. Churiwala and I. Hyderabad, "Designing with Xilinx® FPGAs," in *Circuits &Systems*, Springer, 2017.

[103] O. V. Bisikalo *et al.*, "Implementation Complexity Analysis of the Turbo Decoding Algorithms on Digital Signal Processor," in *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018*, 2018, vol. 10808, p. 81. doi: 10.1117/12.2501504.

[104] Xilinx Inc., "Vivado Design Suite User Guide," *Ug902*, pp. 1–589, 2021, [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug902-vivado-high-level-synthesis

APPENDICES

Algorithm-1: Non-Systematic Polar Encoder (**NSPE** $[\boldsymbol{d}, \boldsymbol{G}, \boldsymbol{\mathcal{F}}, \boldsymbol{N}, \boldsymbol{C}]$)

Inputs: Message Bits(d), Generator Matrix(G), Frozen set$(\mathcal{F})$, and Code Length (N)

Outputs: Encoded vector (C)

For $i = 0 \rightarrow K - 1$ ▶Prepare Input Vector (U)
  $U^i_{\mathcal{F}^c} = d_i$
End
For $i = 0 \rightarrow N - K - 1$
  $U^i_{\mathcal{F}} = 0$ ▶ Frozen bits are zero values
End
Get the reverse indexed version of U ➔ U^r
$C = U * G$ Where $\mathrm{G} = \mathrm{F}^{\otimes \mathrm{n}}$ ▶ Multiplication in Binary Mode

Algorithm-2: Modified Non-Systematic Polar Encoder (**MNSPE**$[\boldsymbol{d}, \boldsymbol{G}, \boldsymbol{\mathcal{F}}, \boldsymbol{N}, \boldsymbol{C}]$)

Inputs:
Message Bits (d), Generator Matrix (G), Frozen Indices$(\mathcal{F})$, and Code Length (N)

Outputs: Encoded vector (C)

For $i = 0 \rightarrow K - 1$ ▶Prepare Input Vector (U)
  $U^i_{\mathcal{F}^c} = d_i$
End
For $i = 0 \rightarrow N - K - 1$
  $U^i_{\mathcal{F}} = 0$
End
Get the reverse indexed version of U ➔ U^r
For $k = \frac{N}{2} \xrightarrow{k/2} 0$
  For $j = 0 \xrightarrow{j=j+2*k} N - 1$
    For $i = 0 \xrightarrow{i++} k - 1$
      $C[j+i] = U[j+i] \oplus U[k+j+i]$
    End
  End
End

Algorithm-3: Systematic Polar Encoder (**SPE** $[\boldsymbol{d}, \boldsymbol{G_{\mathcal{F}^c\mathcal{F}^c}}, \boldsymbol{G_{\mathcal{F}^c\mathcal{F}}}, \boldsymbol{\mathcal{F}}, \boldsymbol{N}, \boldsymbol{C}]$)

Inputs: Message Bits (d), $G_{\mathcal{F}^c\mathcal{F}^c}$, and $G_{\mathcal{F}^c\mathcal{F}}$, Frozen set $(\mathcal{F})$, Code Length (N).

Outputs: Encoded vector (C)

For $i = 0 \rightarrow K - 1$ ▶ Allocate the user bits to encoded inf. location
  $C^i_{\lambda^c} = d_i$ Where $\lambda^c = \mathcal{F}^c$ & $\lambda = \mathcal{F}$
End
For $i = 0 \rightarrow N - K - 1$ ▶ Prepare the input vector of SPE (S)
  $S^i_{\mathcal{F}} = 0$ ▶ Frozen bits are zero values
End
▶Estimate the information bits of the input vector of the SPE

$$S_{\mathcal{F}^c} = (C_{\lambda^c} - S_{\mathcal{F}} * G_{\mathcal{F}\,\lambda^c}) * (G_{\mathcal{F}^c\,\lambda^c})^{-1} \quad \text{Eq.(2.19)}$$

▶Estimate the parity bits of the systematic encoded vector

$$C_{\lambda} = [(C_{\lambda^c}) * (G_{\mathcal{F}^c\,\lambda^c})^{-1}] * G_{\mathcal{F}^c\,\lambda} \quad \text{Eq.(2.20)}$$

Algorithm-4: SCAN Decoder (**SCAN**[**R**, **N**, **I**, $\mathcal{F}$, $\hat{d}$])

Inputs: Code Length (N), Received LLRs (R); Iterations Number (I); Frozen Set ($\mathcal{F}$).

Outputs: Decoded user bits ($\hat{d}$)

Get the reverse indexed version of R ➔ R^r

$n = \log_2 N$

▶ Initialization

$\alpha\ (i, 0) \leftarrow R_i^r$ where $0 \le i \le 2^n - 1$

$\beta(i, n) \leftarrow \begin{cases} \infty & if\ i \in \mathcal{F} \\ 0 & otherwise \end{cases}$

$\beta(i, j) \leftarrow 0,\ where\ 0 \le i \le N - 1\ and\ 0 \le j \le n - 1$

For $i = 0 \rightarrow I$

  For $\emptyset = 0 \rightarrow (N - 1)$

    UpdateAlphaMap(n, $\emptyset$)

    If $\emptyset$ is *odd* then UpdateBetaMap(n, $\emptyset$)

  End

End

If NSPE is used

  For $i = 0 \rightarrow (N - 1)$

    $O_i^{SCAN} = \alpha\ (i, n) + \beta(i, n)$

  End

  For $i = 0 \rightarrow (K - 1)$

    $\boldsymbol{if}\ \left(O_{\mathcal{F}^c}^{SCAN}\right)_i \ge 0\ then\ \hat{d}_i \leftarrow 0$

    Else $\hat{d}_i \leftarrow 1$

  End

End

If SPE is used

  For $i = 0 \rightarrow (N - 1)$

    $O_i^{SCAN} = \alpha\ (i, 0) + \beta(i, 0)$

  End

  Get the reverse indexed version of O^{SCAN} ➔ $O^{SCAN,r}$

  For $i = 0 \rightarrow (K - 1)$

    $\boldsymbol{if}\ \left(O_{\mathcal{F}^c}^{SCAN,r}\right)_i \ge 0\ then\ \hat{d}_i \leftarrow 0$

    Else $\hat{d}_i \leftarrow 1$

  End

End

Algorithm-5: UpdateAlphaMap (p, $\emptyset$) Function

Inputs: p, $\emptyset$

Outputs: updated Alpha Map

if $p = 0$ then return

$\psi \leftarrow \left\lfloor \frac{\emptyset}{2} \right\rfloor$

if $\emptyset$ is even then UpdateAlphaMap ($p - 1$, ψ)

For $\omega = 0 \rightarrow (2^{n-p} - 1)$

  if $\emptyset$ is even

    $\alpha_{p-1}\ (\emptyset, \omega) \leftarrow \alpha_p(\psi, 2\omega) \boxplus [\alpha_p(\psi, 2\omega + 1) + \beta_{p-1}(\varphi + 1, \omega)]$

  End

Continue to Next Page

Else
  $\alpha_{p-1}(\emptyset, \omega) \leftarrow \alpha_p(\psi, 2\omega + 1) + [\alpha_p(\psi, 2\omega) \boxplus \beta_{p-1}(\varphi - 1, \omega)]$
End
End

Algorithm-6: UpdateBetaMap (p, $\emptyset$) Function

Inputs: p, $\emptyset$
Outputs: Updated Beta Map

$\psi \leftarrow \left\lfloor \frac{\emptyset}{2} \right\rfloor$
if $\emptyset$ is odd
  For $\omega = 0 \rightarrow (2^{n-p} - 1)$
    $\beta_p(\psi, 2\omega) \leftarrow \beta_{p-1}(\emptyset - 1, \omega) \boxplus [\beta_{p-1}(\emptyset, \omega) + \alpha_p(\psi, 2\omega + 1)]$
    $\beta_p(\psi, 2\omega + 1) \leftarrow \beta_{p-1}(\emptyset, \omega) + [\beta_{p-1}(\emptyset - 1, \omega) \boxplus \alpha_p(\psi, 2\omega)]$
  End
  if ψ is odd then UpdateBetaMap ($p - 1$, ψ)
End

Algorithm-7: Turbo Polar Encoder (**TPE[**$\boldsymbol{d}, \boldsymbol{G_{\mathcal{F}^c\mathcal{F}^c}}, \boldsymbol{G_{\mathcal{F}^c\mathcal{F}}}, \boldsymbol{\pi}, \boldsymbol{N}, \boldsymbol{E}$**]**)

Inputs: Message Bits(d), $G_{\mathcal{F}^c\mathcal{F}^c}$, $G_{\mathcal{F}^c\mathcal{F}}$, Interleaver($\pi$), Frozen set($\mathcal{F}$), Code Length(N)
Outputs: Turbo Encoded Vector (E)

► Estimate parity bits of the first encoder

$$p^1 = C_\lambda^1 = SPE\ [\boldsymbol{d}, \boldsymbol{G_{\mathcal{F}^c\mathcal{F}^c}}, \boldsymbol{G_{\mathcal{F}^c\mathcal{F}}}, \boldsymbol{\mathcal{F}}, \boldsymbol{N}]$$

► Estimate the interleaved message bits

$$d^\pi = \pi(d)$$

► Estimate parity bits of the second encoder

$$p^2 = C_\lambda^2 = SPE(d^\pi, G_{\mathcal{F}^c\mathcal{F}^c}, G_{\mathcal{F}^c\mathcal{F}})$$

► Multiplexing the Bits of Message, First Parity, and Second Parity to Form the Turbo Encoded vector (E) .

If puncturing is required
  For index $= 0 \overset{+3}{\rightarrow}$ 3K where K= length of Message Bits
    $E_{index} = d_i$ where $i = 0,1, \ldots, K$
    $E_{index+1} = even\ indexed\ bits\ of\ p^1$
    $E_{index+2} = odd\ indexed\ bits\ of\ p^2$
  End
End
Else
  For index $= 0 \overset{+3}{\rightarrow}$ 3K
    $E_{index} = d_i$ where $i = 0,1, \ldots, K$
    $E_{index+1} = p_j^1$ where $j = 0,1, \ldots, N - K$
    $E_{index+2} = p_j^2$ where $j = 0,1, \ldots, N - K$
  End
End

Algorithm-8: Polar Iterative Decoding Algorithm

Inputs: Received LLRs (R), Interleaver (π), puncture type, Outer Iterations Number (I_o), SCAN Iterations Number (I_{si}), Frozen Set ($\mathcal{F}$), Scaling factors (SF_1 & SF_2) and Code Length (N).

Outputs: Estimated decoded bits ($\hat{d}$)

Demultiplexing the received LLRs (R) into Three vectors: R_d, R_{p1}, and R_{p2}

For $i = 0 \rightarrow I_o$ ►Outer iterations loop

- $\varphi_1^s = SF_1 * (\varphi_1),\ \ Where\ \varphi_1 = \pi^{-1}(\varepsilon_2)$ ►First a-prior information
- $I_{\mathcal{F}c}^{SCAN1} = R_d + \varphi_1^s\ \ \&\ I_{\mathcal{F}}^{SCAN1} = R_{p1}$ ►Input vector of SCAN-1
- $O_{\mathcal{F}c}^1 = SCAN1(I^{SCAN1}, N, I_{si}, \mathcal{F})$ ► LLRs of the input sequence
- $\varepsilon_1 = O_{\mathcal{F}c}^1 - \frac{2*R_d}{\sigma^2} - \varphi_1^s\ , Where\ L_c = \frac{2}{\sigma^2}$ ► Extrinsic information of SCAN-1
- $\varphi_2^s = SF_2 * (\varphi_2)\ \ Where\ \varphi_2 = \pi(\varepsilon_1)$ ► Second a-prior information
- $I_{\mathcal{F}c}^{SCAN2} = \pi(R_d) + \varphi_2^s\ \ \&\ \ I_{\mathcal{F}}^{SCAN2} = R_{p2}$ ►Input vector of SCAN-2
- $O_{\mathcal{F}c}^2 = SCAN2(I^{SCAN2}, N, I_{si}, \mathcal{F})$ ► LLRs of the interleaved input sequence
- $\varepsilon_2 = O_{\mathcal{F}c}^2 - \frac{2*\pi(R_d)}{\sigma^2} - \varphi_2^s\ ,\ \ Where\ L_c = \frac{2}{\sigma^2}$ ►Extrinsic information of SCAN-2

End

►Deinterleaving the decoded LLRs of the Second SCAN decoder:

$$O^d = \pi^{-1}(O_{\mathcal{F}c}^2)$$

For $i = 0 \rightarrow (K-1)$

- **If** $O_i^d \geq 0$ then $\hat{d}_i \leftarrow 0$
- **Else** $\hat{d}_i \leftarrow 1$

End

Algorithm-9: The Recursive Systematic Convolutional Encoder Algorithm

RSCE ($d, [G_{FF}, G_{FB}]_2, T_f, [C^p), C^d]$)

Inputs: $Message\ bits\ (d)$, Generator polynomials $[G_{FF}, G_{FB}]_2$, Termination Factor (T_f)

Outputs: Parity bits (C^p) and modified Message bits C^d

If (T_f==1) ►The Trellis Termination is required

- $K_{New} = K - m$ Where : m=No. of D-elements & K= length of Message

End

Else $K_{New} = K$

For i $= 0 \rightarrow m-1$

- $PS[i] = 0$ ►Initialize the Present State Vector

End

For j $= 0 \rightarrow K_{New} - 1$

- ***For*** $i = 0 \rightarrow m-1$
  - $FB = FB \oplus (G_{FB}[i+1] * PS[i])$ ►Feedback Effect
- **End**

Continue to Next Page

For $i = 0 \rightarrow m-1$
  $FF = FF \oplus (G_{FF}[i+1] * PS[i])$ ▶Feedforward Effect
End
$Parity[j] = Message[i] \oplus FF \oplus FB$
For i $= (m-1) \rightarrow 1$
  $PS[i] = PS[i-1]$
End
$PS[0] = Message[i] \oplus FB$
} Update Present State
End
If (T_f==0) i.e. The Trellis Termination is not Required
For i $= 0 \rightarrow m-1$
  $Parity[K_{New} + j] = 0$
End
End
Else
For $i = 0 \rightarrow m-1$
  For $i = 0 \rightarrow m-1$
    $FB = FB \oplus (G_{FB}[i+1] * PS[i])$ ▶Feedback Effect
  End
  For $i = 0 \rightarrow m-1$
    $FF = FF \oplus (G_{FF}[i+1] * PS[i])$ ▶Feedforward Effect
  End
  Message$[K_{New} + i] = FB$
  Parity$[K_{New} + i] = FF$
  For i $= (m-1) \rightarrow 1$
    $PS[i] = PS[i-1]$
  End
  $PS[0] = FB \oplus FB$
} Update Present State
End
End

Algorithm-10: PCTPCC Encoder (**PCTPCCE[**$d, G_{\mathcal{F}^c\mathcal{F}^c}, G_{\mathcal{F}^c\mathcal{F}}, \pi$, N, $[G_{FF}, G_{FB}]_2, E$**]**)

Inputs: Message Bits (d), $G_{\mathcal{F}^c\mathcal{F}^c}$, $G_{\mathcal{F}^c\mathcal{F}}$, Interleaver ($\pi$), and Code Length (N), Frozen Set ($\boldsymbol{\mathcal{F}}$) Convolutional Generator polynomials $[G_{FF}, G_{FB}]_2$.

Outputs: Turbo Encoded Vector (E)

▶Estimate parity bits of the first encoder

$$[p^1, \bar{d}] = \text{RSCE}(\, d, [G_{FF}, G_{FB}]_2, 1)$$

▶Estimate the interleaved message bits

$$d^{\pi} = \pi(\bar{d})$$

▶Estimate parity bits of the second encoder

$$p^2 = C_{\lambda} = SPE(d^{\pi}, G_{\mathcal{F}^c\mathcal{F}^c}, G_{\mathcal{F}^c\mathcal{F}}, \mathcal{F}, N)$$

▶Multiplexing the Bits of Modified Message, First Parity, and Second Parity to Form the Turbo Encoded vector (E) .

If puncturing is required
  For index $= 0 \overset{+3}{\rightarrow} 3K$ Where K= length of message bits
    $E_{index} = \bar{d}_i$ Where $i = 0, 1, \ldots, K$

Continue to Next Page

$E_{index+1} = even\ indexed\ bits\ of\ p^1$
$E_{index+2} = odd\ indexed\ bits\ of\ p^2$
End
End
Else
For index $= 0 \overset{+3}{\rightarrow} 3K$
$E_{index} = \bar{d}_i$ where $i = 0,1,\dots,K$
$E_{index+1} = p_j^1$ where $j = 0,1,\dots,N-K$
$E_{index+2} = p_j^2$ where $j = 0,1,\dots,N-K$
End
End

Algorithm-11: The Soft-Output Viterbi Algorithm (SOVA)

SOVA(R, φ, T_f, Trellis information, $UserLLR$)

Inputs: Received LLRs(R), a-prior information(φ),Termination Factor(T_f),Trellis information ($LastOutput[2^m][2^m], LastState[2^m][2], NextOutput[2^m][2^m],\ NextState[2^m][2]$)

Outputs: LLRs of user sequence ($UserLLR$)

Initialize the path metric of all trellis states by infinity value ▶ $Metrics[2^m][k] = \infty$
For $t = 0 \rightarrow k-1$ ▶ Path Metrics Calculation
For $state = 0 \rightarrow 2^m - 1$
$P_{M0} = R[2t+0] * LastOutput[state][0] + R[2t+1] * LastOutput[state][1] - \frac{\varphi[t]}{2} + Metrics\left[LastState[state][0]\right]\left[LastState[state][1]\right]$ ▶ Zeros Paths(Eq.(2.51))
$P_{M1} = R[2t+0] * LastOutput[state][2] + R[2t+1] * LastOutput[state][3] + \frac{\varphi[t]}{2} + Metrics\left[LastState[state][2]\right]\left[LastState[state][3]\right]$ ▶ Ones Paths(Eq.(2.51))
If $P_{M0} > P_{M1}$
$Metrics[state][t+1] = P_{M0}$
$Mdiff[state][t+1] = P_{M0} - P_{M1}$ ▶ Eq.(2.52)
$DecodedBits[state][t+1] = 0$
End
Else
$Metrics[state][t+1] = P_{M1}$
$Mdiff[state][t+1] = P_{M1} - P_{M0}$ ▶ Eq.(2.52)
$DecodedBits[state][t+1] = 1$
End
End
End

Continue to Next Page

```
18    If (T_f == 1)
          ML(T = k) = 0                                  ▶ state S_0
          EstimatedBits[k] = DecodedBits[k][0]
20    End
21    Else
22        ML(T = k) = j|_{max Metrics[j][k]}             ▶ state S_j
                  Where j = 0 → 2^m − 1
          EstimatedBits[k] = DecodedBits[k][j]
23    End
24    For i = k − 1 → 0
25        ML(T = i) = j|_{max Metrics[j][k]}             ▶ state S_j
                  Where j = 0 → 2^m − 1
          EstimatedBits[i] = DecodedBits[i][j]
26    End
27    For t = 0 → k − 1
28        LLR = ∞
29        For i = 0 → δ                                  ▶ Decoding Depth
30            If (t + i < k)
31                Bit = 1 − EstimatedBits[t + i]
32                Ignored_state = LastState[ML[t + i + 1]][Bit]
33                For j = i − 1 → 0
34                    Bit= DecodedBits[Ignored_state][t + j + 1]
35                    Ignored_state = LastState[Ignored_state]][Bit]
36                End
37                If(Bit ≠ EstimatedBits[t])
38                    If(LLR > Mdiff[ML[t + i + 1]][t + i + 1])
39                        LLR = Mdiff[ML[t + i + 1]][t + i + 1]
40                    End
41                End
42            End
43        End
44        UserLLR = (2 ∗ EstimatedBits[t + i] − 1) ∗ LLR
45    End
```

ML path Estimation

output LLR Calculation eq. (2.42)

Algorithm-12: LogMap Algorithm **(LogMap**[R, φ, T_f, Trellis information, $UserLLR$])

Inputs: Received LLRs(R), a-prior information(φ),Termination Factor(T_f),Trellis information ($LastOutput[2^m][2^m], LastState[2^m][2], NextOutput[2^m][2^m],\ NextState[2^m][2]$)

Outputs: LLRs of user sequence ($UserLLR$)

► Initialization

Alpha[0][0]=0

$Alpha[i][0] = -\infty\}_{i=1}^{(Nstate-1)}$ where Nstate=2^m

if $(T_f == 1)$

  $Beta[0][K-1] = 0$ Where K=length of message bits

  $Beta[i][K-1] = -\infty\}_{i=1}^{(Nstate-1)}$

End

Else $Beta[i][K-1] = 0\}_{i=0}^{(Nstate-1)}$

► Forward Recursion

For $l = 1 \rightarrow K$

  For $state = 0 \rightarrow Nstate - 1$

    $\Gamma[i] = -\infty\}_{i=0}^{(Nstate-1)}$

    $\Gamma\big[LastState[state][0]\big] =$

    $-1 * R[2l-2] + R[2l-1] * LastOutput[state][1] - \log\left(\frac{1}{1+e^{\varphi[l-1]}}\right)$

    $\Gamma\big[LastState[state][1]\big] = +1 * R[2l-2] +$

    $R[2l-1] * LastOutput[state][3] + \varphi[l-1] - \log\left(\frac{1}{1+e^{\varphi[l-1]}}\right)$

    $\boldsymbol{Temp}[i] = (\Gamma[i] + Alpha[i][l-1])\}_{i=0}^{(Nstate-1)}$

    $S = sum\ elements\ of\ vector\ \boldsymbol{Temp}$

    if $(S < e^{-300})$ $Alpha[state][l] = -\infty$

    Else $Alpha[state][l] = \log(S)$

  End

  $TempMax[l] = largest\ value\ of\ Alpha[i][l]\}_{i=0}^{(Nstate-1)}$

  $Alpha[i][l] -= TempMax[l]\ \}_{i=0}^{(Nstate-1)}$

End

► Backward Recursion

For $l = K-2 \rightarrow 0$

  For $state = 0 \rightarrow Nstate - 1$

    $\Gamma[i] = -\infty\}_{i=0}^{(Nstate-1)}$

    $\Gamma\big[NextState[state][0]\big] =$

    $-1 * R[2l+2] + R[2l+3] * NextOutput[state][1] - \log\left(\frac{1}{1+e^{\varphi[l+1]}}\right)$

    $\Gamma\big[NextState[state][1]\big] = +1 * R[2l+2] +$

    $R[2l+3] * NextOutput[state][3] + \varphi[l+1] - \log\left(\frac{1}{1+e^{\varphi[l+1]}}\right)$

    $\boldsymbol{Temp}[i] = (\Gamma[i] + Beta[i][l+1])\}_{i=0}^{(Nstate-1)}$

    $S = sum\ elements\ of\ vector\ \boldsymbol{Temp}$

    if $(S < e^{-300})$ $Beta[state][l] = -\infty$

    Else $Beta[state][l] = \log(S)$

  End

Continue to Next Page

$Beta[i][l] -= TempMax[l+1]\ \}_{i=0}^{(Nstate-1)}$
End
► Compute the soft-output
For $l = 0 \rightarrow K-1$
  For $state = 0 \rightarrow Nstate - 1$
    $\Gamma_0 = -1 * R[2l] + R[2l+1] * LastOutput[state][1] \quad - \log\left(\frac{1}{1+e^{\varphi[l]}}\right)$
    $\Gamma_1 = +1 * R[2l] + R[2l+1] * LastOutput[state][3] + \varphi[l] - \log\left(\frac{1}{1+e^{\varphi[l]}}\right)$
    $Temp_0[state] = \exp\left(\Gamma_0 + Alpha\left[LastState[state][[0]][l] + Beta[state][l]\right)\right.$
    $Temp_1[state] = \exp\left(\Gamma_1 + Alpha\left[LastState[state][[1]][l] + Beta[state][l]\right)\right.$
  End
  $S_0 = sum\ elements\ of\ vector\ Temp_0 \quad \& \quad S_1 = sum\ elements\ of\ vector\ Temp_1$
  $UserLLR[l] = S_0 + S_1$
End

Algorithm-13: Convolutional-Polar Iterative Decoding Algorithm

Inputs: Received LLRs (R), Interleaver (π), puncture type, Outer Iterations Number (I_o), SCAN Iterations Number (I_{si}), Frozen Set ($\mathcal{F}$), Scaling factors (SF_1 & SF_2) and Code Length (N).

Outputs: Estimated decoded bits ($\hat{d}$)

Demultiplexing the received LLRs (R) into Three vectors: R_d, R_{p1}, and R_{p2}
For $i = 0 \rightarrow I_o$ ►Outer iterations loop
  $\varphi_1^s = SF_1 * (\varphi_1) \qquad Where\ \varphi_1 = \pi^{-1}(\varepsilon_2)$ ►SOVA a-prior information
  $I_{even}^{SOVA} = R_d \& I_{odd}^{SOVA} = R_{p1}$ ►Input vector of SOVA (or LogMap)
  ►Decoded LLRs of the input sequence
  $O_{Inf}^1 = \boldsymbol{SOVA}(I^{SOVA}, \varphi_1^s, T_f = 1, Trellis\ information)$
  $\varepsilon_1 = O_{Inf}^1 - \frac{2*R_d}{\sigma^2} - \varphi_1^s, \quad Where\ L_c = \frac{2}{\sigma^2}$ ►SOVA Extrinsic information
  $\varphi_2^s = SF_2 * (\varphi_2) \quad Where\ \varphi_2 = \pi(\varepsilon_1)$ ►SCAN a-prior information
  $I_{\mathcal{F}c}^{SCAN} = \pi(R_d) + \varphi_2^s \ \& \ I_{\mathcal{F}}^{SCAN} = R_{p2}$ ►Input vector of SCAN
  $O_{\mathcal{F}c}^2 = \boldsymbol{SCAN}(I^{SCAN}, N, I_{si}, \mathcal{F})$ ►Decoded LLRs of the interleaved input
  $\varepsilon_2 = O_{\mathcal{F}c}^2 - \frac{2*\pi(R_d)}{\sigma^2} - \varphi_2^s, \quad Where\ L_c = \frac{2}{\sigma^2}$ ►SCAN Extrinsic information
End
►Deinterleaving the decoded LLRs of the Second SCAN decoder:
$$O^d = \pi^{-1}(O_{\mathcal{F}c}^2)$$
For $i = 0 \rightarrow (K-1)$
  If $O_i^d \geq 0$ then $\hat{d}_i \leftarrow 0$
  Else $\hat{d}_i \leftarrow 1$
End

Algorithm-14: SCTPCC Encoder

Inputs: Message Bits (d), $G_{\mathcal{F}^c\mathcal{F}^c}$, $G_{\mathcal{F}^c\mathcal{F}}$, Interleaver ($\pi$), and Code Length (N), Frozen Set($\boldsymbol{\mathcal{F}}$), Convolutional Generator polynomials $[G_{FF}, G_{FB}]_2$.

Outputs: Turbo Encoded Vector (E)

1 ►Estimate the output of the outer systematic polar encoder

$$C^1 = SPE(d, G_{\mathcal{F}^c\mathcal{F}^c}, G_{\mathcal{F}^c\mathcal{F}}, \mathcal{F}, N)$$

2 ►Estimate the interleaved version of polar encoded sequence:

$$C_1^{\pi} = \pi(C^1)$$

3 ►Estimate parity bits of the inner recursive systematic polar encoder

$$[p^2, C_1^{\pi}] = \text{RSCE}(\, C_1^{\pi}, [G_{FF}, G_{FB}]_2, 0)$$

4 **For** index $= 0 \xrightarrow{+2}$ 2N where N= length of polar encoded sequence

5 $\quad E_{index} = (C_1^{\pi})_i$ where $i = 0,1, \ldots, N$

6 $\quad E_{index+1} = p_i^2$

7 **End**

Algorithm-15: Serial Convolutional-Polar Iterative Decoding Algorithm

Inputs: Received LLRs (R), Interleaver (π), Outer Iterations Number (I_o), SCAN Iterations Number (I_{scan}), Frozen Set ($\mathcal{F}$), Scaling factors (SF_1 & $SF_2, SF_{damping}$) and Code Length (N).

Outputs: Estimated decoded bits ($\hat{d}$)

1 **For** $i = 0 \rightarrow I_o$ ►Outer iterations loop

2 $\quad \varphi_{inner}^s = SF_1 * (\varphi_{inner}), Where\ \varphi_{inner} = \pi(\varepsilon_{outer})$ ►Inner a-prior information

3 $\quad I^{SOVA} = R$ ► Input vector of inner SOVA (or LogMap)

4 $\quad O_{Inf}^{inner} = \boldsymbol{SOVA}(I^{SOVA}, \varphi_{inner}^s, T_f = 0, Trellis\ information)$

5 $\quad \varepsilon_{inner} = O_{Inf}^{inner} - \varphi_{inner}^s$ ► Inner Extrinsic information

►Outer a-prior information

6 $\quad \varphi_{outer}^s = SF_2 * (\varphi_{outer}), Where\ \varphi_{outer} = \pi^{-1}(\varepsilon_{inner})$

7 $\quad I^{SCAN} = \varphi_{outer}^s$ ► Input vector of outer SCAN

► LLRs of the input sequence

8 $\quad O^{outer} = [O_{\mathcal{F}c}^{outer}, O_{\mathcal{F}}^{outer}] = \boldsymbol{SCAN}(I^{SCAN}, N, I_{si}, \mathcal{F})$

9 $\quad \varepsilon_{outer} = SF_{damping} * O^{outer} - \varphi_{outer}^s$ ► Outer Extrinsic information

10 **End**

11 **For** $i = 0 \rightarrow (K-1)$ Where K= length of message bits

12 $\quad$ **If** $(O_{\mathcal{F}c}^{outer})_i \geq 0$ then $\hat{d}_i \leftarrow 0$

13 $\quad$ **Else** $\hat{d}_i \leftarrow 1$

14 **End**

الخلاصة

منذ أول كود قطبي لاريكان، كان مجال البحث الخاص بالرموز القطبية نشطًا بشكل مستمر. يركز الجوهر الرئيسي لهذا المجال على تحسين أداء الرموز في أنظمة طول الشفرة المحدودة وتحسين تنفيذ الأجهزة لهذه الرموز بالإضافة إلى تحسين تنفيذ الرموز. يتم تحسين تنفيذ الأجهزة من خلال التغلب على عنق الزجاجة الناتج عن مشكلة المفاضلة بين استخدام الموارد والإنتاجية. في هذه الأطروحة ، تم اقتراح العديد من النماذج القطبية التوربينية وحلول التنفيذ لتحسين الرموز القطبية باستخدام مفكك ترميز الإلغاء الناعم (SCAN) كخوارزمية لفك الترميز. يتم عرض هذه النماذج في هذه الأطروحة من خلال اربعة سيناريوهات.

أولاً ، يتم تنفيذ الرموز القطبية الأصلية والمعدلة باستخدام مصفوفة البوابات المنطقية القابلة للبرمجة في المجال من شركة Xilinx وبالاعتماد على أنواع البيانات التعسفية الدقيقة (Integer Arbitrary-Precision Data types) و توجيه خطوط الأنابيب (Pipelining Directive). ثانيًا، تم تقديم نموذج الرمز القطبي المتسلسل المتوازي (PCTPC) من خلال استخدام الرموز القطبية المنهجية (SPCs) كعناصر ترميز مكونة. على جانب مفكك الترميز، يتم استخدام اثنين من مفكك ترميز الإلغاء الناعم (SCAN) ذو التكرار الاحادي كخوارزميات ذات مدخلات ومخرجات ناعمة (SISO) لإنشاء خوارزمية فك الترميز التكراري لنموذج PCTPC المقترح. يقلل هذا النموذج من مشكلة أرضية الخطأ التي تواجه أكواد التوربو التلافيفية التقليدية. فعند معدل الخطأ في البتات (BER) البالغ 10^{-6}، يحقق نموذج PCTPC المقترح مكاسب تزيد عن 0.5 ديسيبل على الرمز التلافيفي التوربيني الأصلي.

ثالثًا، تم تقديم نموذج الرمز التلافيفي القطبي المتسلسل المتوازي (PCTPCC) باستخدام الرمز القطبي المنهجي (SPC) و الرمز التلافيفي المنهجي التكراري (RSC) كعناصر مكونة. تم إنشاء خوارزمية فك الترميز التكرارية التلافيفية القطبية المقترحة بواسطة مفكك ترميز الإلغاء الناعم (SCAN) ذو التكرار الأحادي للرمز SPC ومفككات الترميز SOVA & LogMap للرمز RSC. تم اقتراح النموذج PCTPCC لزيادة تحسين النموذجPCTPC عن طريق استبدال أحد الرموز القطبية المكونة للنموذج PCTPC برمز RSC. عند معدل خطأ البت (BER) و معدل خطأ الإطار (FER) البالغان 10^{-6}، يتفوق النموذج LogMap-PCTPCC(Nc=384, K=128) على النموذج PCTPC (Nc=384, K=128) بمكاسب تبلغ حوالي 0.525 ديسيبل و 0.5 ديسيبل على التوالي.

رابعًا، تم اقتراح الرمز التلافيفي القطبي التوربيني التسلسلي المتسلسل (SCTPCC) لزيادة المساهمة في القضاء على مشكلات أرضية الخطأ. يتكون نموذج SCTPCC من رمز SPC خارجي ورمز RSC داخلي بمعدلات 2/3 و 1/2 على التوالي. يتفوق هذا النموذج على النماذج القطبية التوربينية المتوازية ولكنه يتمتع بمستوى تعقيد متزايد نظرًا لأن طول الرمز الداخلي يبلغ ضعف طول الرمز الخارجي. عند معدل خطأ الإطار (FER) البالغ 10^{-5}، يتفوق طرازا SOVA-SCTPCC و LogMap-SCTPCC على طرازي SOVA-PCTPCC و LogMap-PCTPCC بمكاسب 0.34 ديسيبل و 0.36 ديسيبل على التوالي. على الرغم من أن نموذج SCTPCC يتفوق على نماذج PCTPCC في مناطق نسبة الإشارة إلى الضوضاء العالية، تُظهر نتائج المحاكاة أن نماذج PCTCC المقترحة تتفوق على نماذج SCTPCC المقترحة في المناطق نسبة الإشارة إلى الضوضاء المنخفضة.

جمهوريـــة العراق
وزارة التعليم العالي والبحث العلمي
جامعـــــة بابـــل
كليـــة الهندســـة / قسم الهندســـة
الكهربائية

تنفيذ الترميز القطبي التوربيني بالاعتماد على مصفوفة البوابات المنطقية القابلة للبرمجة في المجال والتوليف العالي المستوى

الاطروحة

مقدمـــة إلى كليـــة الهندســـة في جامعـــــة بابـــل

كجزء من متطلبات نيل درجـــــة الدكتوراه في

الهندسة الكهربائية/الالكترونيك والاتصالات

من قبل

ولاء ياسين يونس العبادي

بإشراف

الأستاذ المساعد

د. أحمد عبد الكاظم حمد

2023 م **1444 هـ**