Circular Trellis based Low Density Parity Check Codes

A thesis presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Irina Aniţei

November 2008

© 2008 Irina Aniţei. All Rights Reserved.
This thesis titled
Circular Trellis based Low Density Parity Check Codes

by
IRINA ANIȚEI

has been approved for
the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology by

Jeffrey C. Dill
Professor of Electrical Engineering and Computer Science

Dennis Irwin
Dean, Russ College of Engineering and Technology
ABSTRACT

ANIȚEI, IRINA, M.S., November 2008, Electrical Engineering

Circular Trellis based Low Density Parity Check Codes (75 pp.)

Director of Thesis: Jeffrey C. Dill

Tail biting circular trellis block codes (TBC)\(^2\) used along with iterative Maximum A-Posteriori (MAP) decoders achieve performance very close to the Shannon limit. A Low Density Parity Check (LDPC) code using a Sum Product Algorithm (SPA) decoder is also known to achieve comparable performance. In this work the performance of (TBC)\(^2\) encoder used with an SPA decoder is presented. The goal of this research is to compare the performance of (TBC)\(^2\) encoder with different iterative decoders.

In order to use the SPA for decoding, a parity check (H) matrix representation of the (TBC)\(^2\) is developed. It is shown that for small block lengths this H matrix achieves comparable performance. For larger block sizes the H matrix representation of the (TBC)\(^2\) encoder is found non-optimal for SPA decoding and the performance of the code is degraded.

Approved: _____________________________________________________________

Jeffrey C. Dill

Professor of Electrical Engineering and Computer Science
ACKNOWLEDGMENTS

It is my pleasure to thank the people who helped me through these years. Without their support and encouragement this work would not have been possible.

Firstly, I thank my academic advisor, Dr. Jeff Dill, who shared with me a lot of his expertise and research insight. With his enthusiasm and his talent to explain things clearly and simply, he helped me gain the knowledge and courage to complete this work.

I thank Dr. David Matolak, who set an example for me through his outstanding teaching style. His choice of course materials and homeworks has provided me with an in depth understanding of wireless communications concepts.

I would also like to thank Dr. Razvan Bunescu and Dr. Dinh van Huynh for their time, patience and their positive attitude.

I am very grateful to my colleague and friend Kamal Ganti for his constant help and good collaboration. I deeply appreciate his advice and availability for answering my questions at all times during the past three years.

I am thankful to the Department of Electrical Engineering for providing me with the financial assistance throughout my time as a graduate student.

In recognition to all their encouragements and help I thank my friends who always stood by my side: Cerasela and George Caia, Iulia Tomescu, Steven Huang, Sumit Bhattacharya, and Indranil Sen. I also thank Iulian Clapa for all his help.

Most importantly, I want to express my endless gratitude to people who helped me to do my best in all matters of life, my family: Teodora and Stefan Aniței, Magdalena, Maria, Smaranda and Costel Oprea. I dedicate this thesis to them.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>LIST OF FIGURES</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>CHAPTER 1: INTRODUCTION</td>
<td>10</td>
</tr>
<tr>
<td>1.1</td>
<td>1.1 Motivation</td>
<td>10</td>
</tr>
<tr>
<td>1.2</td>
<td>1.2 Background Information</td>
<td>11</td>
</tr>
<tr>
<td>1.2.1</td>
<td>1.2.1 Digital Communication Systems and Error Correction Codes</td>
<td>12</td>
</tr>
<tr>
<td>1.2.2</td>
<td>1.2.2 LDPC Codes</td>
<td>14</td>
</tr>
<tr>
<td>1.2.3</td>
<td>1.2.3 Turbo Codes</td>
<td>15</td>
</tr>
<tr>
<td>1.3</td>
<td>1.3 Organization of the Thesis</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>CHAPTER 2: LDPC CODES AND TURBO CODES OVERVIEW</td>
<td>17</td>
</tr>
<tr>
<td>2.1</td>
<td>2.1 LDPC Codes</td>
<td>17</td>
</tr>
<tr>
<td>2.1.1</td>
<td>2.1.1 Representation of LDPC Codes</td>
<td>17</td>
</tr>
<tr>
<td>2.1.2</td>
<td>2.1.2 Properties of LDPC Codes</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>2.2 LDPC Encoder</td>
<td>20</td>
</tr>
<tr>
<td>2.2.1</td>
<td>2.2.1 Encoding using the G Matrix</td>
<td>20</td>
</tr>
<tr>
<td>2.2.2</td>
<td>2.2.2 Encoding Using the H Matrix</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>2.3 LDPC Decoder</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>2.4 Performance of LDPC Codes</td>
<td>23</td>
</tr>
<tr>
<td>2.5</td>
<td>2.5 Turbo Codes Encoder</td>
<td>25</td>
</tr>
<tr>
<td>2.5.1</td>
<td>2.5.1 Description of Parallel Concatenated Encoder</td>
<td>25</td>
</tr>
<tr>
<td>2.5.2</td>
<td>2.5.2 Interleavers</td>
<td>26</td>
</tr>
<tr>
<td>2.6</td>
<td>2.6 Turbo Codes Decoder</td>
<td>27</td>
</tr>
<tr>
<td>2.6.1</td>
<td>2.6.1 Decoding Process</td>
<td>27</td>
</tr>
<tr>
<td>2.6.2</td>
<td>2.6.2 Decoding Algorithm</td>
<td>29</td>
</tr>
</tbody>
</table>
2.7 Performance of Turbo Codes ........................................................................ 30
2.8 Comparison of Performance of Turbo Codes and LDPC codes .............. 31

CHAPTER 3: CIRCULAR TRELLIS BASED CODES USING PARITY CHECK MATRIX .............................................................................................. 33
3.1 (TBC)² Encoder ........................................................................................................ 33
     3.1.1 Properties of (TBC)² ...................................................................... 33
     3.1.2 Advantages of (TBC)² ................................................................. 35
3.2 H Matrix Representation of (TBC)² ................................................................. 37
3.3 SP Algorithm ........................................................................................................ 42
3.4 Parallel Concatenated Encoder – SPA Decoder ........................................... 46
3.5 Turbo SPA Decoder .......................................................................................... 47
     3.5.1 Parallel SPA Decoders with Interleaving ...................................... 47
     3.5.2 Parallel Gallager SPA Decoders .................................................. 51

CHAPTER 4: RESULTS AND DISCUSSIONS ......................................................... 53
4.1 Results for Designing an H matrix for (TBC)² ................................................. 53
     4.1.1 H matrix for Systematic Codes ...................................................... 54
     4.1.2 H Matrix for Nonsystematic Codes ............................................... 63
     4.1.3 H Matrix Representation of two Parallel Concatenated (TBC)² .... 65
4.2 Turbo Encoder-Turbo SPA Decoder .............................................................. 67
     4.2.1 Results for Parallel SPA Decoders with Interleaving .................. 67
     4.2.2 Results for Parallel Gallager SPA Decoders ............................... 69

REFERENCES ........................................................................................................... 73
LIST OF TABLES

Table 1 - Example of State table for 16-state trellis [17] .................................................. 34
Table 2 - Number of 4-length loops of H matrix .............................................................. 41
Table 3 - Performance for the initial H matrix ................................................................. 54
Table 4 - Results for the improved H matrix ................................................................. 55
Table 5 - Results for the improved H matrix using different systematic table .......... 58
Table 6 - The results for our H matrix using different systematic table.................. 60
Table 7 - Results for our H matrix for a punctured code of rate 2/3 ......................... 61
Table 8 - Results for our H matrix for a punctured code of rate 4/5 ......................... 62
Table 9 - The results for H matrix using a non-systematic table ......................... 64
Table 10 - Results for H matrix representation of two PC-(TBC)$^2$ encoders .......... 65
Table 11 - Results for H matrix representation of two PC-(TBC)$^2$ using a symmetrical interleaver ................................................................. 66
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Digital communication system basic diagram</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2</td>
<td>a) H matrix; b) Tanner’s Graph</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Examples of 4-length cycle</td>
<td>19</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Performance of LDPC codes [21]</td>
<td>24</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Diagram of Turbo Codes Encoder</td>
<td>26</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Turbo Decoder diagram</td>
<td>28</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Performance of Turbo Codes [21]</td>
<td>31</td>
</tr>
<tr>
<td>Figure 8</td>
<td>The performance of LDPC and Turbo Codes [21]</td>
<td>32</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Butterfly structure [17]</td>
<td>35</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Performance of (TBC)$^2$ Encoder [17]</td>
<td>36</td>
</tr>
<tr>
<td>Figure 11</td>
<td>The path through the trellis of a minimum weight codeword</td>
<td>38</td>
</tr>
<tr>
<td>Figure 12</td>
<td>The matrix representation of 6-length loops</td>
<td>41</td>
</tr>
<tr>
<td>Figure 13</td>
<td>The graphical representation of 6-length loops</td>
<td>41</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Information exchange between v-nodes and c-nodes.</td>
<td>43</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Information exchange between c-nodes and v-nodes</td>
<td>43</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Flow chart of the Sum Product decoding algorithm</td>
<td>45</td>
</tr>
<tr>
<td>Figure 17</td>
<td>General flow chart of our Matlab code</td>
<td>46</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Diagram of an LDPC - Turbo system</td>
<td>47</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Turbo-SPA decoder flow chart</td>
<td>50</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Parallel Gallager codes [13]</td>
<td>51</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Performance comparison of LDPC and PCGC [22]</td>
<td>52</td>
</tr>
</tbody>
</table>
Figure 22 - Performance of our code using the initial H matrix ..................................... 54

Figure 23 - The performance of our code using the improved H matrix ...................... 56

Figure 24 – Comparative performance of initial H matrix and improved H matrix .... 58

Figure 25 - The performance of H matrix using different systematic symbol table 2 .... 59

Figure 26 - The performance of our code using different H matrix............................. 60

Figure 27 - The performance of H matrix for a puncture code of rate 2/3............... 61

Figure 28 - The performance of H matrix for a puncture code of rate 4/5............... 62

Figure 29 - The performance of our code using a non-systematic table ..................... 64

Figure 30 - The performance of our code using H matrix representation of two PC
(TBC)² encoders................................................................................................................ 65

Figure 31 - The comparative performance of different codes................................. 68

Figure 32 - The comparative performance of PG and LDPC codes........................ 69

Figure 33 - Comparative performances for our [16x32] H matrix............................ 70

Figure 34 - Performance of our H matrix having zero 4-length cycles..................... 71
CHAPTER 1: INTRODUCTION

One of the major challenges of contemporary times is to find a reliable way of communicating information. Shannon’s work in the 40's gave birth to the field of information theory and error control codes. An appreciable amount of work was done in error control codes for the next few decades. By the late 80's the research in this area had matured. However, the improvement in performance was limited by the ability of the decoder. With the discovery of Turbo codes and LDPC codes in the 90’s a powerful and efficient decoding algorithm was found [1]. This work invigorated the coding community and fuelled the current wave of research in error control codes and iterative decoding algorithms.

In the present chapter, we provide the motivation for this research in the context of the past work on this topic. Also, a discussion of basic concepts used in this research is included. This chapter ends with an overview of the thesis.

1.1 Motivation

LDPC codes and Turbo Codes both use iterative decoding [1]. There have been many independent studies on Turbo Codes and LDPC codes in the past. Also, a combination of the two is known as Parallel concatenated Gallager codes [13], Turbo like LDPC codes, or decoding Turbo Codes based on their parity check matrices [14].
The purpose of this research is to compare the performance of Turbo Codes and LDPC codes using the same encoder but different decoders. The performance of the Turbo Code which uses a Tail biting circular trellis block codes (TBC)\(^2\) Encoder has been investigated in previous research [3] - [6][17].

The research presented in this thesis focuses on two main ideas. Firstly, our aim is to study the performance of (TBC)\(^2\) encoder using a LDPC (Sum product Algorithm - SPA) decoder instead of the Turbo MAP decoder. This innovative idea implies finding good H matrix representations of the (TBC)\(^2\) encoder. Secondly, the goal is to verify that a Parallel Concatenated Encoder represented by an H matrix and a Turbo SPA decoder will give the same consistent performance as a Turbo Code performance, given the same code rate and block length.

1.2 Background Information

In the following section we will introduce the reader to some of the basic concepts necessary to understand the present work.

Initially, we introduce the digital communication system concept and the place of error correcting codes in this context. We continue our introduction by discussing the error correction codes, their use and importance, the block diagram, the types of forward error correction codes (FEC) and their performance. Further, for completion, the history of Turbo Codes and LDPC codes is presented.
Digital communication systems are communication systems which transmit the encoded information in digital form [1]. The need for the use of digital communication systems is justified by the data processing options and resilience attained against the use of analog transmission.

The simplified block diagram of a communication system contains a transmitter side, a channel and a receiver side [2] as shown in Figure 1. The transmitter system contains an Encoder, the place where the data is represented as a member of the same finite code or message set [1]. The data is transmitted through a channel where noise is added before feeding to the receiver. Here, the decoder has the role of decoding the encoded data in order for the original transmitted information to be retrieved.

![Digital communication system basic diagram](image)

*Figure 1 - Digital communication system basic diagram*

The primary function of an ideal digital communication system is to transmit the information to the receiver with as little degradation as possible. At the same time, the
digital communication system should be reliable from the transmitted energy and bandwidth point of view. The metric considered to ascertain the quality of a digital communication system is bit error rate (BER) or probability of bit error ($P_b$) [2]. Therefore, to achieve the characteristics described above, error correction codes are used to encode and decode the information.

The error correction codes are used in a digital communication system in order to improve the performance of the system. This is realized by adding redundancy which enables the transmitted signal to be resistant to different channel effects such as noise, interference or fading [2]. In a general diagram (Figure 1) of a communication system the error correcting codes are named channel coding.

The types of error correction codes are:

1. Block codes
2. Convolutional & Trellis codes [1]

When talking about Block codes the two important characteristics are the code rate and the block size. Each segment of data contains a fixed number of $m$ bits, and each segment is encoded and respectively decoded one block at a time. The encoder adds the parity bits, which are the redundancy so that the output segment is $n$ bits, with $n > m$. Furthermore, the code rate is calculated as the ratio of the input bits and output bits, in our case $R = m/n$ [1].

Depending on whether the original input sequence is found or not in the output of the encoder we have systematic codes or nonsystematic codes, respectively. In case of systematic codes the input sequence is not altered in the output of the encoder [2]. Another important metric in defining the capability of an error correcting code is the
minimum distance \( d \) between two codewords [1]. This is defined as the smallest value of the set of Hamming distances for the specific code. The error correcting capability \((t)\) is given by formula:

\[
t = \frac{d - 1}{2}
\]

As we can notice, minimum distance and error correcting capability are directly proportional to each other [1]. A coding scheme that has memory, such as a convolutional code, can be referred to as a trellis code. Such a trellis code when combined with modulation to achieve error-correction performance without increasing the bandwidth is referred to as trellis coded modulation [1].

LDPC codes, Hamming codes, Golay codes are some examples of Block codes [1].

### 1.2.2 LDPC Codes

Initially proposed by Robert Gallager [15], LDPC (low density parity check) codes were not studied much for more than 35 years. One reason is that Reed-Solomon (RS) codes were invented in the same period and they were more suitable codes for the applications developed at that time. Another reason was the high computational complexity required for LDPC codes compared to RS codes.

In 1998 Richardson and Urbanke resuscitated the interest in LDPC codes. Also in 1999 MacKay published his work on LDPC codes [16].

LDPC codes are essentially a type of block code, which means that the data is encoded and decoded in a block by block manner [16]. The encoder has the role to add parity bits
to the input sequence and the decoder detects and corrects the possible errors. The decoding algorithm is the same class of algorithms as the Turbo Codes decoder algorithm. It is named SPA (sum product algorithm) or MPA (message passing algorithm) [16].

It has been proved that the LDPC codes provide very high decoder throughput [16]. The LDPC codes can be represented using Tanner’s graph or using the parity-check matrix named the H matrix [17]. Also, from the point of view of the weight of the rows or columns of the H matrix the LDPC codes can be regular or irregular. An example of the performance of LDPC codes is given in following chapters.

### 1.2.3 Turbo Codes

Invented by Berrou et. al., in 1993 [1], Turbo Codes were the first error correcting codes which achieved increased data rate by not increasing the transmitted power. Used in satellite communications and wireless communications, Turbo Codes offer a high performance in terms of error correction and protection of data.

A Turbo Code Encoder contains two parallel concatenated encoders separated by an interleaver. At the decoder side, the turbo structure is kept by having two parallel iterative decoders separated by an interleaver and deinterleaver. This structure has the advantage of being capable of decoding much longer codes with a moderate degree of decoding algorithm complexity.

The basic block diagram is presented in Chapter 2, where we discuss in more detail Turbo Codes and also an example of Turbo Codes performance is presented.
1.3 Organization of the Thesis

This thesis is divided into five chapters. The first chapter is the Introduction chapter in which the motivation of our research is presented followed by background information and the outline of the thesis. In Chapter 2, an overview of Turbo Codes and LDPC codes is given. The Encoding and Decoding algorithms are presented for both types of FEC codes and the performance of each is illustrated. In Chapter 3 we describe our research base notions. The \((TBC)^2\) Encoder, the SPA algorithm, the H matrix representation of \((TBC)^2\) and the parallel concatenated encoder with one SPA decoder and with turbo SPA decoders are described. Chapter 4 is the results and discussions chapter. It contains the analytical and simulation results for the H representation of the \((TBC)^2\) encoder using different methods. The thesis ends with Chapter 5, which concludes this work and summarizes the results obtained. Some suggestions and new ideas for future work are also briefly discussed in this chapter.
CHAPTER 2: LDPC CODES AND TURBO CODES OVERVIEW

In Chapter 2 an overview of LDPC codes and Turbo codes is presented. The concepts described here are the fundamentals of our research.

Both Turbo Codes and LDPC codes achieve performance very close to the Shannon limit. The primary reason for this performance is their iterative decoding algorithms. In order to optimize the performance of these decoding algorithms specialized encoders were developed.

2.1 LDPC Codes

Low density parity check codes are a class of forward error correcting codes with the property that at higher rates, when using high-order modulations, they seem to have a distinct advantage over other forward error correction codes [16].

2.1.1 Representation of LDPC Codes

LDPC codes can be represented in two different ways. The two representations are interrelated and equivalent.

One way of representing LDPC codes is using the parity check matrix, which in literature is named the H-matrix [11]. The elements of the H-matrix are 0s and 1s. The density of 1s in the H-matrix should be very low in order for the decoder to give a good
performance. In this case the matrix is named sparse H-matrix. From this point of view, the LDPC codes can be classified into two categories: Regular LDPC codes and Irregular LDPC codes [12]. If the number of 1s in each row and in each column is constant than the LDPC code is regular, otherwise the LDPC code is irregular.

The other way of representing the LDPC codes is using Tanner’s Graphs [16]. The Tanner Graphs and H matrix are equivalent, and they can be derived one from the other [12]. Tanner Graphs are bipartite graphs and the containing nodes are named variable nodes (v-nodes) and check nodes (c-nodes)[16]. For a better understanding we can consider the example illustrated in Figure 2.

\[
H = \begin{pmatrix}
1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 \\
\end{pmatrix}
\]

*Figure 2 - a) H matrix; b) Tanner’s Graph*

If an element of the H matrix is 1, than there will be a corresponding edge in Tanner’s graph. In the above example, the first element of the H matrix is 1, so there is an edge between first v-node and first c-node. Considering the second element of H, which has the value 0, notice that there is no edge between first v-node and second c-node in Tanner’s Graph.
2.1.2 Properties of LDPC Codes

The first metric of interest is the density of the H matrix, which is given by the total number of 1’s on each row and column of the matrix. From the literature we know that the best performance for a regular LDPC code is given by a (3, 6) structure [8]. That signifies that all the check nodes are degree 3 ones and all variable nodes have degree 6. The randomness of the H matrix is an important factor. In order to obtain a good performance the rows of the parity check matrix need to be linearly independent. Also, large minimum distance between codewords is required [15]. That means that the number of 4-length loops should be minimized. A 4-length loop is a path in Tanner’s Graph that contains four edges with the property that the initial and the last node coincide [16]. Four-length loop examples are illustrated in Figure 3.

![Figure 3 - Examples of 4-length cycle](image)

The sparseness of the H matrix and the number of 4-length loops in a LDPC code will have a drastic influence in the performance of the code.
There are a few design techniques used for LDPC codes. The approach used in our research is constructing a low desity parity check matrix using the encoder for Turbo Codes. Obviously the goal is, using an efficient encoder and decoder, to obtain near-capacity performance and low error rate floors.

Different design approaches have different names for LDPC codes. A few examples are: Gallager codes, MacKay codes, Irregular LDPC codes, Array codes, Combinatorial LDPC codes [16].

2.2 LDPC Encoder

The goal of this subchapter is to describe the encoder for the LDPC codes. In order to do so, we present two ways of encoding. The first method to encode is using the generator matrix of a code (G matrix). Another way of encoding is using the parity check matrix of the code (H matrix).

2.2.1 Encoding using the G Matrix

Considering C a $[n,k]$ code having a generator matrix $G$ and the input sequence $x$, one can encode it by applying the formula: $xG=c$. In this equation $c$ is the obtained codeword and consists of a $k$ systematic bits and $n-k$ parity or redundant bits [20]. This is when the $G$ matrix is in the standard form $[I|P]$, where $I$ is the $[k\times k]$ size identity matrix and $P$ is a
$[kx(n-k)]$ size matrix named the parity matrix. Usually systematic codes have their $G$ matrices in standard form.

If, however, the $G$ matrix is not in the standard form, by row permutations and manipulations, an equivalent code with a $G$ matrix in standard form can be obtained [2].

Having the $G$ matrix in the standard form facilitates obtaining the parity check matrix by applying the formula: $H=[P^T|I]$.

### 2.2.2 Encoding Using the H Matrix

Another way of encoding an input sequence or message $x$ is using the $H$ matrix of the given code $C$. As mentioned before, if we consider the first $k$ bits of $x$ as being the systematic bits and the $c=k-n$ bits left as being the parity check bits, our codeword can be written as:

$$x=[k|c] \quad (2.1)$$

Also, denote:

$$H=[A|B] \quad (2.2)$$

where $A$ is a $m \times m$ Identity matrix and $B$ is a $m \times (n-m)$ matrix. In this scenario $x$ is a valid codeword if the following constraint is true:

$$x \ast H^T = 0 \quad (2.3)$$

From the equations (2.1), (2.2) and (2.3) we can conclude that:

$$Ac+Bx=0 \quad (2.4)$$

$$c = A^{-1} \ast Bx \quad (2.5)$$
Equation (2.5) gives us means to calculate the check bit with the condition that matrix A is non-singular. It is worth mentioning that for a systematic code A is the identity matrix.

2.3 LDPC Decoder

The decoding algorithm used in our research is named Message Passing Algorithm or Sum Product Algorithm.

Message Passing Algorithm is an algorithm that computes iteratively the distributions of variables in a graph-based model [16]. In our case, the Message Passing Algorithm is based on Tanner’s graph.

In Message Passing Algorithm, the a-posteriori probability (APP) that a given bit in the transmitted codeword $c$ equals 1 given the received codeword $y$, $\Pr(c_i=1|y)$, is computed. The formula for the APP ratio [16] is:

$$l(c_i) = \frac{\Pr(c_i = 0 \mid y)}{\Pr(c_i = 1 \mid y)}$$  \hspace{1cm} (2.6)

Also, the log likelihood ratio can be defined, as in the following formula [2]:

$$L(c_i) = \log \left( \frac{\Pr(c_i = 0 \mid y)}{\Pr(c_i = 1 \mid y)} \right)$$  \hspace{1cm} (2.7)

Initially the APP is computed from the received data and sent to v-nodes. The probabilities contained in the v-nodes are sent to the adjacent c-nodes. The c-nodes
contain so called check parity equations and the obtained result is named extrinsic information.

The messages or extrinsic information which are sent in \(l\) iterations will be denoted \(m'_{cv}\), respectively \(m'_{vc}\). The formula for computing the extrinsic information [18] is:

\[
m'_{cv}^{l} = \ln \frac{1 + \prod_{v \in V, c \in [c]} \tanh(m'_{vc}) / 2}{1 - \prod_{v \in V, c \in [c]} \tanh(m'_{vc}) / 2}
\]  \(2.8\)

In formula (2.8) \(m'_{cv}^{l}\) is the extrinsic information from the c-nodes to the v-nodes and \(m'_{vc}^{l}\) is the extrinsic information from the v-nodes to the c-nodes known from the previous iteration.

In the next half iteration, the information contained in the c-nodes is send to the v-nodes. The formula [18] used is:

\[
m_{vc}^{(l)} = \begin{cases} m_{v}, l = 0 \\ m_{v} + \sum_{c \in C, c \in [c]} m'_{vc}^{l-1}, l \geq 1 \end{cases}
\]  \(2.9\)

At the end of each iteration (at the v-node), hard decisions are taken. The last step is to check if the obtained word is a valid codeword \((x^{*H^T} = 0)\). If the codeword is a valid one than the iteration is stopped if not the process continues.

### 2.4 Performance of LDPC Codes
In Figure 4 the performance of the two types of LDPC codes is presented [21]. The block length of the code is $10^6$ bits, the code rate is $\frac{1}{2}$ and the Belief-propagation algorithm is used for decoding. Note that the distance from the Shannon limit for the irregular LDPC code is about 0.1 dB at $P_b=10^{-5}$. The irregular LDPC code performs better than the (3,6) regular LDPC code because of the randomness in the irregular code.

![Figure 4 - Performance of LDPC codes [21]](image-url)
2.5 Turbo Codes Encoder

Turbo Codes contain two key innovations: parallel concatenated encoder and iterative decoder [19]. These innovations are responsible for the superior performance of the Turbo Codes.

2.5.1 Description of Parallel Concatenated Encoder

The encoder of Turbo Codes is a parallel concatenated (PC) encoder. It is formed of two convolutional encoders separated by an interleaver. Typically recursive systematic codes (RSC) are used for convolutional encoding [7]. It is known that there exists an equivalent non-recursive convolutional code for every RSC. The reason for which the RSC codes are used for Turbo Codes encoding is because of the difference in the input sequences mapped into the output codewords. For RSC, low weight input sequences are mapped to high weight codewords, and it is this property that helps them perform well for Turbo Codes.

In the original Turbo Code a code rate 1/3 RSC code with generator matrices G1 [7] were used. In order to get a 1/2 code rate, puncturing was used.

Usually both the encoders used in the parallel concatenated structure are the same, but this may not necessarily be the case.

The following figure (Figure 5) illustrates the block diagram of a parallel concatenated encoder.
In the above figure Encoder 1 (ENC1) and the Encoder 2 (ENC2) are both recursive systematic codes.

2.5.2 Interleavers

The interleaver used in Turbo Codes is different from the channel interleaver. Channel interleaving is used to avoid burst errors. The interleaver used in Turbo Codes serves two main purposes:

- it ensures that the output of one decoder is uncorrelated with the output of the other one
- if an input sequence generates a low weight codeword with the first encoder then the interleaver is responsible to ensure that the interleaved input sequence doesn’t generate a low weight codeword from encoder two
The size of the interleaver determines the performance of the Turbo Code. As the length of the interleaver increases, the block length of the code increases and better performance can be achieved.

Commonly, a semi-random interleaver is used. The random properties of the interleaver provide decorrelation between the decoders. Careful design of semi-random interleavers is necessary in order to ensure that low weight codewords are avoided [17]. Note that a low weight input sequence is a sequence of small Hamming weight (4 or 5) and a low weight codeword is a codeword of small Hamming weight.

The design of the interleaver is accountable for the performance of the Turbo Code at high SNRs. Low weight codewords are responsible for the error floor seen in Turbo Codes. By using a good interleaver, perfectly designed, the minimum weight of the Turbo Code can be increased thus delaying the appearance of the error floor [1].

2.6 Turbo Codes Decoder

Iterative decoders are used in Turbo Codes and they can be used to efficiently decode complex codewordes. Before their invention, complex encoders existed but weren’t used because decoding was prohibitively difficult.

2.6.1 Decoding Process

In this section the decoding process applied in Turbo Codes is presented. Figure 6 illustrates the block diagram of the Turbo Code Decoder.
The input to Decoder 1 is the sequence of systematic bits, output of Encoder1, and extrinsic information from Decoder 2. Similarly, the input to Decoder2 is the sequence of systematic bits, output of Encoder 2, and extrinsic information from Decoder1. Note that the extrinsic information exchanged between the decoders has to be interleaved/de-interleaved [19].

Decoder1 (DEC1) and Decoder 2 (DEC2) are soft-input and soft-output Decoders. The inputs to the decoders are probabilities and so are the outputs.

After the first iteration the output of Decoder 1 consists of intrinsic and extrinsic information. The intrinsic information is the information known before decoding and the extrinsic information is the information gained due to the decoding process. This extrinsic information is exchanged between the two decoders. The iteration or exchange of information between the two decoders is continued until both the decoders converge to the same codeword and the maximum number of iterations is reached.

Typically the soft input, soft output maximum a-posteriori probability decoder is used for decoding [7].
2.6.2 Decoding Algorithm

The classical algorithm used in Turbo Code decoders is Maximum a-posteriori (MAP) algorithm also called BCJR algorithm [7].

If we consider, as described in section 2.3.1, that the log likelihood a-posteriori probability is given by the formula (2.10):

\[ L(c_i) = \log \left( \frac{\Pr(c_i = +1 | y)}{\Pr(c_i = -1 | y)} \right) \]  \hspace{1cm} (2.10)

where the decoder decides \( c_i = +1 \) if \( \Pr(c_i = +1 | y) > \Pr(c_i = -1 | y) \) and \( c_i = -1 \) otherwise.

For a trellis code, considering as starting state (or previous state) the state \( s_{k-1} = a \) and the end state (or current state) \( s_k = b \) the above formula can be written as:

\[ L(c_i) = \log \left( \frac{\sum_{s_{k-1} = a, s_k = b} \Pr(s_{k-1} = a, s_k = b, y) / \Pr(y)}{\sum_{s_{k-1} = a, s_k = b} \Pr(s_{k-1} = a, s_k = b, y) / \Pr(y)} \right) \]  \hspace{1cm} (2.11)

That means that given the probability of output \( y \) we can find the probability of input bit as being 1 or -1.

By looking at the previous formula notice that after cancelling the \( \Pr(y) \) the remaining part to calculate is [7]:

\[ \Pr(s_{k-1} = a, s_k = b, y) \]  \hspace{1cm} (2.12)

which can be written as in equation (2.13) [7]:

\[ \Pr(s_{k-1} = a, s_k = b, y) = \alpha_{k-1}(a) \cdot \gamma_k(a, b) \cdot \beta_k(b) \]  \hspace{1cm} (2.13)

Keeping in mind the above formula, the BCJR algorithm involves four basic steps:
1 - Calculate the forward metric \( \alpha_k(a) \). Defining \( \alpha_k(b) = \Pr(s_k, y_1^k) \) this is computed as:

\[
\alpha_k(b) = \sum \alpha_{k-1}(a) \gamma(a,b)
\]

with the conditions that \( \alpha_0(0) = 1 \) and \( \alpha_0(b \neq 0) = 0 \) [7].

2 - Calculate the intermediate metric (from state a to state b):

\[
\gamma_k(a,b) = \Pr(s_k = b, y_k \mid s_{k-1} = a)
\]

3 – Calculate the reverse metric \( \beta_k(b) \):

\[
\beta_k(b) = \sum \beta_k(a) \gamma(a,b)
\]

with the conditions that \( \beta_n(0) = 1 \) and \( \beta_n(b \neq 0) = 0 \) [7].

4 - Calculate the final bit probabilities \( \alpha_k(a) \cdot \gamma_k(a,b) \cdot \beta_k(b) \).

As explained previously in section 2.6.1, in a Turbo Code two iterative MAP based Decoders are used.

### 2.7 Performance of Turbo Codes

In *Figure 7* we have illustrated the performance of a Turbo Code [21] for a block length of \( 10^6 \) bits and code rate 1/2. The Turbo Code performance is about 0.3 dB at \( P_b = 10^{-5} \) away from the Shannon limit for the same block size as our LDPC example. The encoder used is the classical turbo code encoder and the decoding algorithm is a Belief-propagation algorithm.
2.8 Comparison of Performance of Turbo Codes and LDPC codes

In Figure 8 the comparative performance of (3,6) regular LDPC code, Irregular LDPC code and Turbo Code is presented. The performance of irregular LDPC is about 0.2 dB better than the Turbo Code performance for the same block length and code rate. From literature [21], it is known that the performance of the irregular LDPC code it is better than Turbo code performance for large block lengths.
Figure 8 - The performance of LDPC and Turbo Codes [21]
CHAPTER 3: CIRCULAR TRELLIS BASED CODES USING PARITY CHECK MATRIX

In Chapter 3 we present the basic concepts and the novel ideas of this research. The chapter starts with a description of the (TBC)\(^2\) encoder. It continues with a description of the H matrix representation of (TBC)\(^2\) encoder. Further, the SP algorithm used in our simulations is presented and a Parallel Concatenated encoder – SPA decoder scenario is described. In the last part of the chapter the Turbo-SPA idea is presented.

3.1 (TBC)\(^2\) Encoder

Tail-biting circular trellis block codes (TBC)\(^2\) are a novel error control scheme used in severe jamming environments. The block length and code rate for these codes can be chosen dynamically to provide flexibility and robustness to a communication system [5]. These trellis codes can be completely described using their state and symbol tables.

3.1.1 Properties of (TBC)\(^2\)

The properties of (TBC)\(^2\) encoder are given by its State Table and Symbol (Transmission) Table. In subsequent sections we discuss in detail the State Table and Symbol table for (TBC)\(^2\) encoder.

A state table indicates all possible state transitions in a trellis given the current state and input symbol. The state table for the (TBC)\(^2\) is a \(S \times n\) matrix, where \(S\) is the number of
states and $n$ is the input alphabet size. The state table for the $(TBC)^2$ has the following properties:

- **Tail-biting:**
  
  For the $(TBC)^2$ encoder, the trellis path for any given input sequence will have the same starting and ending state. This property of the table is known as tail-biting and makes the trellis circular. This tail-biting property of the state table alleviates the problem of the decoder knowing the starting and ending state of a received codeword [3].

- **Butterfly structure:**

  The state table used in this research has $S = 16$ states with an alphabet size $n = 4$, as shown in Table 1. For this table, the states can be grouped into sets of 4-flys, where an $n$-fly is a group of $n$ initial states that transit to the same $n$ next states as in Figure 9. This butterfly structure can be used to increase the free distance of the code [6].

<table>
<thead>
<tr>
<th>Current State</th>
<th>1 input</th>
<th>2 input</th>
<th>3 input</th>
<th>4 input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>10</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>16</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>12</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>15</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>11</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>9</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>7</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>13</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>8</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>5</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

*Table 1 - Example of State table for 16-state trellis [17]*
For a given input sequence, the state table is used to generate a unique path through the trellis. The symbol table is then used to map this trellis path onto a unique set of channel symbols. For the (TBC)$^2$, simplex symbol assignment is used to generate a symbol table that achieves larger than orthogonal minimum distance between codewords.

### 3.1.2 Advantages of (TBC)$^2$

The main advantages of (TBC)$^2$ [5][6][17] are:

a) Near-Shannon limit performance: (TBC)$^2$ encoder when used along with an iterative decoder achieve performance of approximately 1dB away from theoretical limits. The iterative decoder used had two soft-input soft-output (SISO) Maximum A-posteriori Probability (MAP) decoders separated by an interleaver/de-interleaver.
b) Low latency: This code uses blocks of length ranging from 32 to 1024 bits and has very low latency when compared to block lengths of the order of 10,000 bits.

c) High adaptivity: These codes can have a code rate within the range 1/12-4/5 for block lengths of 32-1024 bits.

d) Efficient decoding: The iterative decoder used for these codes can be made highly parallel thus further reducing the latency [5].

*Figure 10* shows the performance of the 16-state (TBC)$^2$ encoder with a turbo decoder for code rate 1/12 and different block lengths [17].
3.2 H Matrix Representation of \((TBC)^2\)

The main objective of this research is to generate a parity check (H) matrix for the \((TBC)^2\) that can be used in the sum product algorithm. In this subchapter the main steps to obtain a good H matrix representation of the encoder are presented.

- **Step 1: Generating the H matrix**

The first step is to generate a sparse H matrix representation of the \((TBC)^2\). This was accomplished by first forming a generator matrix \(G\), in standard form, for the code. The H matrix is then generated from the G matrix.

The generator matrix \(G\) was formed by sending an input sequence \(e_i = [0 \ 0 \ldots 0 \ 1 \ 0 \ldots 0]_{1 \times B}\) to the \((TBC)^2\) and collecting the corresponding output codeword \(C_i\). Each \(C_i\) forms a row of the G matrix, so that this is generated as:

\[
G = \begin{bmatrix}
C_1 \\
C_2 \\
\vdots \\
C_n
\end{bmatrix}
\]

This G matrix is transformed to the standard form \([I:P]\) using row manipulations, where \(I\) is the identity matrix (representing the systematic symbols) and \(P\) is the set of parity symbols.

\(H\) is then obtained as:

\[
H = [P^T : I]
\]

where \(P^T\) is the transposed matrix formed by the parity bits and \(I\) is the Identity matrix.
However, the H matrix obtained using the above method it is not sparse. As a consequence the number of 4-length cycles is large, and this translates to a poor performance at the decoder.

- **Step 2**: Making the H matrix sparse.

For the SPA to be executed in reasonable time, the number of branches connecting the check nodes and variable nodes should be small. This means that the H matrix has to be sparse. In order to make our H matrix sparse we use minimum weight input sequences. An input sequence that results in a minimum weight code word is known as a minimum weight input sequence. By weight we are referring to the Hamming weight of the sequence. So, a minimum weight codeword typically has an input sequence of weight less than 6 and the overall weight of the codeword less than 20.

It is known that for a trellis encoder there exist input sequences such that the trellis transitions deviate from the all 0 state and remerge with it after only a few transitions as shown in *Figure 11*.

![Figure 11 - The path through the trellis of a minimum weight codeword](image)

Such input sequences result in minimum weight codewords. An example to illustrate the above statement is described below. Consider the following rows of the H matrix:

- **First row**: \([110111|000100]\)
- Second row: [1 0 1 1 1 | 0 0 1 0 0 0]

In which the first six bits are the parity bits and the last six bits are the systematic bits.

Note that the Hamming weight of both rows is 6.

By adding these two rows (modulo-2) we obtain: [0 1 1 0 0 0 | 0 1 1 0 0 0]. The weight of this codeword is 4. This example shows that we can do row operations on the H matrix to make the matrix sparser.

The algorithm used to generate the G matrix has the following logic:

```plaintext
for row = i : k
  e = zeros(1, k)
  e[row] = 1
  codeword = TBC_encode(e[row])
  G[row] = codeword
end
```

In order to find a good H an algorithm which does a search for combination of rows that give minimum weight codeword was written. The logic of the code is as follows.

```plaintext
for row1 = 1 : k - 1
  for row2 = 2 : k
    combine = row1 + row2
    if weight (combine) < minweight
      minweight = weight (combine)
    end
  end
end
```

For a given H matrix we search through all possible weight-2, weight-3 and weight-4 input sequences. Once a minimum weight sequence was found, we performed row operations on the H matrix so that the systematic bits matched the bits of the minimum weight sequence. These row operations make the H matrix sparse by reducing the weight of the parity symbols.
For calculating the number of 4-length loops, the algorithm used is:

```matlab
for row1=1:k-1
    for row2=row1+1:k
        location=find(row2==1);
        new_row=mod(row1+row2,2);
        location_new=find(new_row(location)==1);
        if(length(location_new)<length(location))
            num_4=floor((length(location)-length(location_new))/2);
            num_4++
        end
    end
end
```

Another more organized way of reducing the density of the H matrix is to use a Greedy Algorithm. In this method, row manipulations are performed and the combination of rows that results in the lowest weight is chosen. For a $txn$ H matrix, the $i^{th}$ row is added to the remaining $t-i-1$ rows and the weight of each row is calculated. The combination of rows that results in the lowest weight is used to replace the $i^{th}$ row. This method, when applied to all the $t$ rows results in a sparser H matrix. We apply this technique multiple times until the density of the matrix remains constant. The resulting parity check matrix is the sparsest matrix possible.

In Table 2 we present the results that show the gain obtained by applying the above row combining methods.
<table>
<thead>
<tr>
<th>Block length</th>
<th>Code Rate</th>
<th>Number of 1’s</th>
<th>Number of 4 length loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial H matrix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>1/2</td>
<td>8704</td>
<td>139232</td>
</tr>
<tr>
<td>256</td>
<td>1/2</td>
<td>35072</td>
<td>1166912</td>
</tr>
<tr>
<td>512</td>
<td>1/2</td>
<td>140288</td>
<td>9476480</td>
</tr>
<tr>
<td>1024</td>
<td>1/2</td>
<td>560128</td>
<td>76012288</td>
</tr>
<tr>
<td>H matrix after row combining</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>1/2</td>
<td>1024</td>
<td>640</td>
</tr>
<tr>
<td>256</td>
<td>1/2</td>
<td>2048</td>
<td>1280</td>
</tr>
<tr>
<td>512</td>
<td>1/2</td>
<td>4096</td>
<td>2560</td>
</tr>
<tr>
<td>1024</td>
<td>1/2</td>
<td>8192</td>
<td>5120</td>
</tr>
</tbody>
</table>

*Table 2 - Number of 4-length loops of H matrix*

From the table, we note that row combining gives significant reduction in 4-length loops but is unable to eliminate all of them.

We also researched the influence of the 6-length loops on the performance of H matrix and tried to eliminate them. The pattern of a 6-length loop can be represented by matrices or corresponding Tanner graphs [12] as shown in the following figures.

\[
H_1 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}, \quad H_2 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \quad H_3 = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}
\]

*Figure 12 - The matrix representation of 6-length loops*

*Figure 13 - The graphical representation of 6-length loops*
We tried searching and eliminating the 6 length loops patterns but were not able to eliminate all of them. From literature [16], we found that even with some 6 length loops present in the matrices the performance is not badly degraded. So, even in our case we do not expect that a small number of 6 length loops to be detrimental to performance.

3.3 SP Algorithm

The decoding algorithm used in our research is the Sum Product Algorithm (SPA). As discussed in previous sections, Sum Product Algorithm is used to compute iteratively the distributions of variables in a graph-based model [16]. In our case, the SPA is based on Tanner’s graph. The algorithm used in our Matlab simulation is described below.

In SPA, the a-posteriori probability (APP) that a given bit in transmitted codeword $c$ equals 1 given the received word $y$: $\Pr(c_i=1|y)$ is computed. The formula for the APP ratio is:

$$l(c_i) = \frac{\Pr(c_i = 0 | y)}{\Pr(c_i = 1 | y)}$$

(3.1)

Also, the log likelihood ratio can be defined as:

$$l(c_i) = \log \left( \frac{\Pr(c_i = 0 | y)}{\Pr(c_i = 1 | y)} \right)$$

(3.2)

Initially the APPs are computed from the received data (for an AWGN channel) using the formula:

$$L(q_y) = L(c_i) = 2 * y_i / \sigma^2$$

(3.3)
where $\sigma^2$ is the noise variance.

The graphical representation of half iteration in SPA is illustrated in Figure 14.

As shown in the figure, the probabilities contained in the v-nodes are sent to the adjacent c-nodes. The c-nodes contain so-called check parity equations and the obtained result is the extrinsic information. The formula for computing the extrinsic information [7] is:

$$L(r_{ji}) = \tanh^{-1}\left( \prod_{i \in V \setminus i} \tanh\left( \frac{1}{2} L(q_{ij}) \right) \right)$$  \hspace{1cm} (3.4)

Where $L(q_{ij})$ are the log likelihood ratios of all the branches from the v-nodes to c-nodes excluding the branch $i^{th}$.

In the next half iteration, the information contained in the c-nodes is sent to the v-nodes [16] as shown in Figure 15.

The formula is:

$$L(q_{ji}) = L(c_i) + \sum_{j \in C \setminus i} L(r_{ji})$$  \hspace{1cm} (3.5)
Where $L(r_{ji})$ are the log likelihood ratios of all the branches from the c-nodes to v-nodes excluding the branch $j^{th}$.

Finally, for the next iteration LLRs are updated:

$$L(Q_i) = L(c_i) + \sum_{j \in c_j} L(r_{ji}) \quad (3.6)$$

At the end of each iteration (at the v-node), hard decisions are taken:

$$\hat{c}_i = 1 \text{ if } L(Q_i) < 0 \text{ or else } \hat{c}_i = 0 \quad (3.7)$$

The last step is to check if the obtained word is a valid codeword i.e. if $c^*H' = 0$.

If the codeword is a valid one, then the iteration is stopped, else the process continues until the maximum number of iterations is reached.

The SPA flow diagram used in our research is given in Figure 16.
START

Find Hij = 1

Initialize qij with channel data

if loop < max_iter

calculate probabilities from v nodes to c-nodes (qij)

compute probabilities from c-nodes to c-nodes (rji)

calculate new qij from old rji

calculate Qij

if cHT = 0

STOP

Figure 16 - Flow chart of the Sum Product decoding algorithm
3.4 Parallel Concatenated Encoder – SPA Decoder

In our work, we keep our code design as close as possible to the Turbo Codes design. In our simulation, we use an H matrix generated from a parallel concatenated encoder, similar to the one used in Turbo Codes, and one SPA decoder. Each constituent encoder of the parallel structure is a \((TBC)^2\) based encoder. The method to generate the H matrix from the Parallel Concatenated encoder is similar to the one described previously in Section 3.2. The motivation for using a parallel concatenated encoder to generate the H matrix is because of the use of the interleaver. This will provide more randomness in the H matrix which might improve performance.

A general flow diagram of our Matlab code is presented in Figure 17.

*Figure 17 - General flow chart of our Matlab code*
3.5 Turbo SPA Decoder

In this section the two kinds of Turbo decoders used in this research are presented. At the beginning of this section, the Parallel SPA turbo decoders are discussed. Further, the parallel Gallager SPA decoders are presented. The performance curves for both cases are presented in Chapter 4.

3.5.1 Parallel SPA Decoders with Interleaving

In the following figure the diagram of an LDPC-Turbo system is presented.

![Diagram of an LDPC Turbo system](image)

*Figure 18 - Diagram of an LDPC - Turbo system*

In the LDPC-Turbo decoder, each decoder- DEC1 and DEC2- uses SPA (sum product algorithm). As explained before, SPA (sum product algorithm) used in DEC1 and DEC2 contains the following steps:

1 - Calculate the initial probabilities, in other words, the information that comes from the channel into each decoder.
2 - Calculate the extrinsic information from the v-nodes to the c-nodes, denote it “q_{ij}”.

3 - Calculate the exchanged information from the c-nodes to the v-nodes, denote it “r_{ji}”.

4 - Calculate the output probabilities of the \textit{DEC1} and \textit{DEC2}, denoted by “Q_i”.

5 - Make hard decisions on the outputs $Q^1_i$ and $Q^2_i$ and obtain codewords as the outputs of \textit{DEC1} and \textit{DEC2}.

Compare the output codewords of \textit{DEC1} and \textit{DEC2} and if they are the same then the decoding process is done. If the output codewords do not match, then the Turbo process starts.

LDPC-Turbo decoder consists in the following steps:

I - Calculate the extrinsic information from each decoder.

Denote the extrinsic information from \textit{DEC1} as \textit{extr1}, then the formula for it is:

\[
\text{extr1} = e^1_i(0) = Q^1_i(0) - P^1_i(0)
\]

(3.8)

and

\[
e^1_i(1) = Q^1_i(1) - P^1_i(1)
\]

(3.9)

Similarly, for \textit{DEC2} we have:

\[
\text{extr2} = e^2_i(0) = Q^2_i(0) - P^2_i(0)
\]

(3.10)

and

\[
e^2_i(1) = Q^2_i(1) - P^2_i(1)
\]

(3.11)

II - Exchange the extrinsic information between \textit{DEC1} and \textit{DEC2}.

Looking at the figure, one can notice that after exchanging the extrinsic information at the input of the \textit{DEC1} we have the de-interleaved \textit{extr2}. The formula for this is:

\[
\text{extr2}(0) = e^2 \prod_{-i}^{0} (0) \quad \text{and} \quad \text{extr2}(1) = e^2 \prod_{-i}^{0} (1)
\]

(3.12)
Where $P_k^i(j)$ are the APPs for symbol $j$, decoder $k$, iteration $i$ and $\Pi$ denotes interleaving.

Similarly, the extrinsic information at the input of $DEC2$ we have the interleaved $extr1$.

Mathematically, we can write it as:

$$extr1(0) = e_{\Pi^{-1}(0)}^i(0) \text{ and } extr1(1) = e_{\Pi^{-1}(1)}^i(1)$$

(3.13)

III - Second iteration starts and the input of each decoder now is modified

At $DEC1$ we will have the deinterleaved extrinsic information from $DEC2$ and the initial information from $ENC1$. That is:

$$extr2 + ENC1 = Z_i$$

(3.14)

The intrinsic data was given by the $P_i$. So we can write:

$$Z_i^1(0) = e_{\Pi^{-1}(0)}^2(0) + P_i^1(0)$$

$$Z_i^1(1) = e_{\Pi^{-1}(1)}^2(1) + P_i^1(1)$$

(3.15)

At the $DEC2$ we will have the interleaved extrinsic information from $DEC1$ and the initial information from the $ENC2$. The mathematical expressions for this are given by the following formulae:

$$Z_i^2(0) = e_{\Pi^{-1}(0)}^1(0) + P_i^2(0)$$

$$Z_i^2(1) = e_{\Pi^{-1}(1)}^1(1) + P_i^2(1)$$

(3.16)

VI - Compare the output codewords and if they do not match begin the third iteration similarly to step III.

At this step make hard decisions after decoding using the SP algorithm explained in the beginning, and if in the obtained codewords are not the same, begin the third iteration
that is similar to repeating the Step III. Keep in mind that in this case the “P_i’s” will be replaced by the “Z_i’s”.

For our simulation the Turbo-SPA decoder flow chart is illustrated in Figure 19.
3.5.2 Parallel Gallager SPA Decoders

The difference between parallel Gallager decoders and the iterative decoder presented in section 3.5.1 is that two different H matrices are used for the two decoders and no interleaver/deinterleaver are used.

The diagram of the Parallel Concatenated Gallager Codes (PCGC) [13] is presented in Figure 20.

![Parallel Gallager Codes Diagram](image)

*Figure 20 - Parallel Gallager codes [13]*

The encoding for the parallel Gallager codes is slightly different from that of parallel concatenated encoding. The main difference is that there is no interleaver and in each encoder a different H matrix is used. In this encoder type the interleaver is avoided by using an appropriate H matrix for the second encoder.

At the decoder this translates also to no interleaver/deinterleaver and the two SP algorithms use two different H matrices. But, similar to Turbo SPA decoders only extrinsic information of systematic bits is exchanged between the two decoders. In the
results presented in Figure 21, the choice of the H matrices is such that the average minimum column weight should be 2.667 [13]. That is because low weight gives good performance at low SNRs and high weight gives good performance at high SNRs [22]. The performance curve for the Parallel Gallager code compared with an LDPC code is shown in the figure below [13].

![Performance comparison of LDPC and PCGC](image)

*Figure 21 - Performance comparison of LDPC and PCGC [22]*

*Figure 21* shows the comparison performance of a LDPC code and a PCGC for a block length of 1920 bits and code rate ½. Also, for the LDPC code, the minimum column weight of the H matrix is 2.67 as it is for the PCGC [22]. Notice that the coding gain for PCGC at $P_b = 10^{-2}$ is about 0.5 dB more than that of LDPC code for the same block size and code rate.
CHAPTER 4: RESULTS AND DISCUSSIONS

This chapter presents the results of our research. These results are obtained from simulations conducted using MATLAB. In these simulations at least 30 codeword errors were counted for every value of $E_b/N_0$.

4.1 Results for Designing an H matrix for $(TBC)^2$

The results we present in this chapter are the performance of the H matrix representations of a $(TBC)^2$ encoder. The method used to generate these H matrices was presented in previous chapters (section 3.2). In the following results different systematic tables were used in order to obtain more randomness in the H matrix. Once the H matrix was obtained, techniques to eliminate the 4 and 6 – length loops were used to make the H matrix sparse and improve performance.
4.1.1 H matrix for Systematic Codes

In Table 3 we present the results for the H matrix obtained as the representation of systematic (TBC)\(^2\) Encoder. Observe that the number of 4-length loops is large and it is directly proportional to the size of the H matrices.

<table>
<thead>
<tr>
<th>Systematic Symbol table: Txs_symbol_1 (size 64x4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size of H</strong></td>
</tr>
<tr>
<td>16x32</td>
</tr>
<tr>
<td>32x64</td>
</tr>
</tbody>
</table>

*Table 3 - Performance for the initial H matrix*

The performance curves are presented in Figure 22.

*Figure 22 - Performance of our code using the initial H matrix*
In comparison to BPSK performance we notice that there is no coding gain for block length of 16 bits and neither for block length of 32 bits.

Uncoded BPSK is better than LDPC as the FEC is performing very poorly. If energy per bit for uncoded BPSK is $E_b$ than for a rate $\frac{1}{2}$ FEC code bit the energy is $E_b/\sqrt{2}$. Because of this decrease in energy the poor performance of the decoder results in the LDPC code being worse than uncoded BPSK. Essentially, there is more noise in the FEC due to parity bits and a sub-par H matrix results in poor performance due to the bad decoder. We used a systematic symbol table, with code rate $\frac{1}{2}$, to generate the H matrix.

Next, we apply the Greedy algorithm to reduce the number of 4 length cycles in our H matrix representation of $(TBC)^2$ we found the results presented in Table 4.

<table>
<thead>
<tr>
<th>Systematic Symbol table: Txs_symbol_1 (size 64x4)</th>
<th>4 length loops for initial H</th>
<th>4 length loops for reduced H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16x32</td>
<td>156</td>
<td>29</td>
</tr>
<tr>
<td>32x64</td>
<td>1456</td>
<td>103</td>
</tr>
</tbody>
</table>

*Table 4 - Results for the improved H matrix*
The comparative plots for this case are illustrated in Figure 23.

Figure 23 - The performance of our code using the improved H matrix

From Table 4 note that as the block length increases, the number of 4 length cycles increases. This is because the number of ones in the parity bits increases with block length and these ones cannot be eliminated using row operations on the H matrix. Using row operations we can reduce the Hamming weight of half of the rows of the H matrix. It has been found that the Hamming weight of the other half of the rows cannot be reduced beyond a threshold and the ones in these rows are responsible for the 4-length cycles. Since the Hamming weight of half of the rows in the H matrix increases with block length so does the number of 4 length loops. This increase in the number of 4 length loops is responsible for degradation in performance with increase in block length.
The structure of the state table is responsible for the H matrix having half of the rows with small Hamming weight and the other half with an unacceptably high Hamming weight.

The \((TBC)^2\) encoder has the property that the input sequence \([1 \ 0 \ 2]\) results in a low weight code word (weight of codeword is \(~5\)) . This property is translated to the H matrix where the sequence \([1 \ 1 \ 0]\) results in a codeword of weight 4. In a similar fashion as the other input sequences result in codewords of high weight in the \((TBC)^2\) encoder they result in rows with high Hamming weight in the H matrix as well.

The property of the \((TBC)^2\) encoder where the weight of the codewords is high for at least half of the low weight input sequences helps it achieve good performance using the turbo decoder. But this property of the code reduces the sparseness of the H matrix and increases the number of 4 length loops, both of which are detrimental to the performance of the SPA decoder.

In order to verify that reducing the 4-length cycles has an effect on the performance of the H matrix we show the comparative performance for the same H matrix of size \([16 \times 32]\) used above.

The performance plots are shown in *Figure 24*. 
Using different a symbol table of size \([64 \times 4]\), we start with a different H matrix that has fewer 4-length loops. The results obtained using this H matrix are shown in the following table (Table 5).

<table>
<thead>
<tr>
<th>Systematic Symbol table: Txs_symbol_2 (size 64x4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size of H</strong></td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>16x32</td>
</tr>
<tr>
<td>32x64</td>
</tr>
</tbody>
</table>

Table 5 - Results for the improved H matrix using different systematic table
The comparative plots for the above case are presented in *Figure 25*.

*Figure 25* - The performance of H matrix using different systematic symbol table 2

Using a third systematic transmission table we obtain even sparser H matrices as shown in *Table 6*. The difference between the three transmission symbol tables used is that they have different distance properties. The description of the techniques used to generate these symbol tables is beyond the scope of this thesis.
Systematic Symbol table: Txs_symbol_3 (size 64x3)

<table>
<thead>
<tr>
<th>Size of H</th>
<th>4 length loops for initial H</th>
<th>4 length loops for modified H</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x32</td>
<td>196</td>
<td>16</td>
</tr>
<tr>
<td>32x64</td>
<td>2072</td>
<td>95</td>
</tr>
<tr>
<td>64x128</td>
<td>17328</td>
<td>295</td>
</tr>
<tr>
<td>128x256</td>
<td>139232</td>
<td>832</td>
</tr>
<tr>
<td>256x512</td>
<td>1166912</td>
<td>1586</td>
</tr>
</tbody>
</table>

*Table 6* - The results for our H matrix using different systematic table

The performance plots are shown in the following figure (*Figure 26*).

*Figure 26* - The performance of our code using different H matrix
From Figure 26 we note that using different symbol tables we can get different H matrices that result in different number of 4-length cycles. So with a fixed state table a symbol table can be designed that minimizes the number of four length loops. The properties of such a symbol table are still under investigation.

The performance of a punctured rate 2/3 is presented in Table 7 and Figure 27. In order to get a code rate of 2/3, half the parity bits were punctured. Txs_symbol_3 was used for this encoder.

<table>
<thead>
<tr>
<th>Block length</th>
<th>Size of H</th>
<th>4 length loops for initial H</th>
<th>4 length loops for modified H</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bits</td>
<td>8x24</td>
<td>52</td>
<td>14</td>
</tr>
<tr>
<td>32 bits</td>
<td>16x48</td>
<td>536</td>
<td>32</td>
</tr>
<tr>
<td>64 bits</td>
<td>32x96</td>
<td>4224</td>
<td>173</td>
</tr>
</tbody>
</table>

*Table 7 - Results for our H matrix for a punctured code of rate 2/3*

*Figure 27 - The performance of H matrix for a puncture code of rate 2/3*
From the results of testing the systematic $(TBC)^2$ it was found that the weight of the parity bits was responsible for the denseness of the $H$ matrix. One way of reducing the weight of the parity bits is by puncturing. In the above results note that by reducing the code rate to $2/3$ the number of 4-length loops were reduced.

The results for a punctured code of code rate $4/5$ and the performance plots for different block sizes are presented in Table 8 and Figure 28.

<table>
<thead>
<tr>
<th>Block size</th>
<th>Size of G</th>
<th>Size of H</th>
<th>4 length loops for initial H</th>
<th>4 length loops for modified H</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bits</td>
<td>16x20</td>
<td>4x20</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>32 bits</td>
<td>32x40</td>
<td>8x40</td>
<td>132</td>
<td>40</td>
</tr>
<tr>
<td>64 bits</td>
<td>64x80</td>
<td>16x80</td>
<td>992</td>
<td>80</td>
</tr>
<tr>
<td>256 bits</td>
<td>256x320</td>
<td>64x320</td>
<td>71456</td>
<td>1007</td>
</tr>
</tbody>
</table>

*Table 8* - Results for our $H$ matrix for a punctured code of rate $4/5$

*Figure 28* - The performance of $H$ matrix for a puncture code of rate $4/5$
By inspecting Figure 28 we can draw the following conclusions. As the block size increases it is expected that the minimum weight increases as well. This increase in the minimum weight should result in an improved performance.

From all the above plots, we notice that as the block length increases there is no significant change in performance. This is due to the fact that H matrices used have 4-length loops. Because of these 4-length cycles the minimum weight of the code remains the same despite of the increase in block length and hence results in same performance.

In the results, different symbol tables for the systematic (TBC)$^2$ encoder were used. The goal of this exercise was to study the impact of the symbol table on the 4-length loops. We note that by modifying the symbol table the number of 4-length loops can be reduced considerably and this reduction leads to performance improvements.

The properties of the symbol table that lead to zero 4-length loops are not well understood and is a topic of future investigation.

**4.1.2 H Matrix for Nonsystematic Codes**

In the next section, we test if using a non-systematic table in generating the H matrix helps in reducing the number of 4-length cycles. Here a rate 1/12 non-systematic symbol table along with puncturing is used to generate the H matrix. The results and performance for this case are presented in Table 9 and Figure 29.
### Table 9 - The results for H matrix using a non-systematic table

<table>
<thead>
<tr>
<th>Size of H</th>
<th>4 length loops for initial H</th>
<th>4 length loops for modified H</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x32</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>32x64</td>
<td>471</td>
<td>7</td>
</tr>
<tr>
<td>256x512</td>
<td>351712</td>
<td>371</td>
</tr>
</tbody>
</table>

The number of 4-length cycles obtained using non-systematic codes, rate ½, is less when compared to the systematic codes. This is due to the relative randomness of the non-systematic codes. In spite of the randomness the number of 4-length loops could not be reduced to zero.
4.1.3 H Matrix Representation of two Parallel Concatenated (TBC)\(^2\)

In order to find a better performance of our code we tried finding a H matrix representation of two parallel concatenated (PC) (TBC)\(^2\) encoders. The results for this scenario are shown in Table 10 and Figure 30.

<table>
<thead>
<tr>
<th>Size of H</th>
<th>4 length loops for initial H</th>
<th>4 length loops for modified H</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x32</td>
<td>161</td>
<td>14</td>
</tr>
<tr>
<td>32x64</td>
<td>1724</td>
<td>370</td>
</tr>
</tbody>
</table>

*Table 10* - Results for H matrix representation of two PC-(TBC)\(^2\) encoders

*Figure 30* - The performance of our code using H matrix representation of two PC (TBC)\(^2\) encoders
Here we attempt to reduce the number of 4-length loops in an H matrix by using a parallel concatenated encoder separated by an interleaver. The description of this encoder was presented in section 3.4.

Another technique used to achieve sparseness is to use a symmetric interleaver. A symmetric interleaver is one in which if the $i^{th}$ element is interleaved to $j^{th}$ location, then the element in the $j^{th}$ location is moved to $i^{th}$ location. The results of using a symmetrical interleaver are presented in Table 11.

<table>
<thead>
<tr>
<th>Size of H</th>
<th>4 length loops for initial H</th>
<th>4 length loops for modified H</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x32</td>
<td>134</td>
<td>12</td>
</tr>
<tr>
<td>32x64</td>
<td>1579</td>
<td>297</td>
</tr>
</tbody>
</table>

*Table 11 - Results for H matrix representation of two PC-(TBC)$^2$ using a symmetrical interleaver*

As can be noted in Table 11 by applying this method we are unable to improve the sparseness of the H matrix.

From the results of testing systematic and non-systematic (TBC)$^2$ encoder it was found that resultant H matrices were dense. To achieve sparseness, randomness was added to the H matrix in the form of a parallel concatenated encoder. In order to achieve maximum gains the structure of the interleaver has to be such that the resultant H matrix is sparse. This can be a topic of future research.
4.2 Turbo Encoder-Turbo SPA Decoder

For convenience, throughout this chapter, the Turbo encoder-Turbo SPA Decoder will be denoted as TLDPC. In this section we present the results for TLDPC. The theoretical framework for this section was presented in subchapters 3.5.

4.2.1 Results for Parallel SPA Decoders with Interleaving

For the TLDPC to perform better than the version having one SPA, the gain from the feedback has to be such that it outweighs the noise added due to increase in code rate. In order for the feedback to be effective we only need to feed back extrinsic information. If intrinsic information is exchanged the errors will get propagated from one decoder to the other and result in performance degradation. As systematic bits are the only common information sent between the two decoders only the extrinsic information for the systematic bits has to be exchanged.

An initial comparison is shown below in Figure 31. In this figure we show the performance of a Turbo Code for a block length on 128 bits and code rate \( \frac{1}{2} \). Also, the dotted line is the performance of a LDPC code, using one SPA decoder and a sparse H matrix with zero 4-length loops. The line with triangles on top, is showing the performance of TLDPC for 10 - turbo iterations and only one iteration inside the SPA.
decoders. From this plot it can be seen that the current implementation of TLDPC doesn’t give gains.

![Graph](image)

*Figure 31 - The comparative performance of different codes*

The reasons for which we do not see a significant gain in the above figure are that:

a) The gains from the iterative decoding are not significant enough. That means that exchanging the extrinsic information between the two decoders is ineffective.

b) The design of the encoder is not optimized. The design of the interleaver and the constituent encoder has to be such that the output codeword needs to have a large weight for any low weight input sequence.
4.2.2 Results for Parallel Gallager SPA Decoders

In the Figure 32 we compare the performance of a Parallel Gallager (PG) code with that of a LDPC code. The design of this Parallel Gallager code was described earlier in section 3.5.2.

![Figure 32 - The comparative performance of PG and LDPC codes](image)

The two H matrices used in the PG code are generated from the (TBC)² encoder. The difference between H1 and H2, the two parity check matrices used in the PG codes, is in the transmission symbol table used to generate them.

A rate ½ LDPC code using the parity check matrix H2 is used for comparison. From the plot we note that the PG code performs worse than the LDPC code. This performance degradation shows that the H matrices chosen for the PG decoding are not optimal.
Further research is required in investigating the design of H matrices that achieves performance for PG codes.

**Chapter 5: Conclusions and Future work**

Using the results presented in Chapter 4, a figure comparing the performances of H matrices, all of size [16x32], generated using a systematic, nonsystematic and a parallel concatenated, rate $\frac{1}{2}$ code is presented below:

![Figure 33 - Comparative performances for our [16x32] H matrix](image)

From *Figure 33* we note that reducing the number of 4-length loops in the ‘initial H matrix’ results in improvement in performance.
There is no difference in the performance of the other H matrices as all of them have non-zero number of four length loops. Though the number of 4-length loops in the individual matrices differ as long as an H matrix has at least one 4-length loop its performance will be degraded. The performance of the initial H matrix is worse because of the very large multiplicity of 4-length loops.

Also, the comparison plot of BER for an H matrix of size [8x16] having no 4-length cycles is shown below, in Figure 34.

In the above plot the H matrix was generated from the (TBC)^2 encoder and it has the property of having zero 4-length cycles. Note that with this H matrix we get consistent
coding gains for all values of $E_b/N_0$. In this case we show that for an H matrix from $(TBC)^2$ with zero four length cycles gains can be achieved over the range of SNRs. This clearly points to the fact that the main problem with the other H matrices is their non-zero four length cycles.

In [14] an H matrix representation of the parallel concatenated encoder with constituent RSC codes was presented. In this work it is shown that for the same encoder, the performance of the SPA decoder is worse than the iterative MAP decoder. This work corroborates the findings in the current research.

Another observation from this research is that using our H matrix representation of $(TBC)^2$ in a TLDPC scenario did not show any performance gain. This shows that optimization of the design of H matrix and the iterative TLDPC decoder is required in order to achieve performance.

Future work involves finding innovative ways of obtaining sparse H matrix representation of $(TBC)^2$ for large block lengths. Once such an H matrix is found, good performance can be achieved using the techniques presented in this work.
REFERENCES


