HARDWARE IMPLEMENTATION OF FANO DECODER FOR POLARIZATION-ADJUSTED CONVOLUTIONAL (PAC) CODES

A DISSERTATION SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Amir Mozammel Hokmabadi

June 2022

Hardware Implementation of Fano Decoder for Polarization-Adjusted Convolutional (PAC) Codes

By Amir Mozammel Hokmabadi

June 2022

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Erdal Arıkan(Advisor)

Orhan Arıkan

Ali Ziya Alkar

Tolga Mete Duman

Barış Nakiboğlu

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

HARDWARE IMPLEMENTATION OF FANO DECODER FOR POLARIZATION-ADJUSTED CONVOLUTIONAL (PAC) CODES

Amir Mozammel Hokmabadi
Ph.D. in Electrical and Electronics Engineering
Advisor: Erdal Arıkan
June 2022

Polarization-adjusted convolutional (PAC) codes are a new class of error-correcting codes that have been shown to achieve near-optimum performance. By combining ideas from channel polarization and convolutional coding, PAC codes create an overall encoding transform that achieves a performance near the information-theoretic limits at short block lengths.

In this thesis we propose a hardware implementation architecture for Fano decoding of PAC codes. First, we introduce a new variant of Fano algorithm for decoding PAC codes which is suitable for hardware implementation. Then we provide the hardware diagrams of the sub-blocks of the proposed PAC Fano decoder and an estimate of their hardware complexity and propagation delay. We also introduce a novel branch metric unit for sequential decoding of PAC codes which is capable of calculating the current and previous branch metric values online, without requiring any storage element or comparator. We evaluate the error-correction performance of the proposed decoder on FPGA and its hardware characteristics on ASIC with TSMC 28 nm 0.72 V library. We show that, for a block length of 128 and a message length of 64, the proposed decoder can be clocked at 500 MHz and achieve approximately 38.1 Mb/s information throughput at 3.5 dB signal-to-noise ratio with a power consumption of 3.85 mW.

Keywords: PAC codes, sequential decoding, Fano decoding, polar coding, VLSI.

ÖZET

KUTUPSAL VE POLARİZAYSON AYARLI EVRİŞİMLİ (PAC) KODLAR İÇİN FANO ÇÖZÜCÜSÜNÜN DONANIM UYGULAMASI

Amir Mozammel Hokmabadi
Elektrik ve Elektronik Mühendisliği, Doktora
Tez Danışmanı: Erdal Arıkan
Haziran 2022

Kutupsal ve polarizasyon ayarlı evrişimli (PAC) kodlar, optimuma yakın performans elde ettiği gösterilmiş olan hata düzeltme kodlarının yeni bir sınıfıdır. PAC kodları, kanal polarizasyonu ve evrişimli kodlamadan gelen fikirleri birleştirerek, kısa blok uzunluklarında bilgi teorik sınırlarına yakın bir performans elde eden genel bir kodlama dönüşümü oluşturur.

Bu tezde, PAC kodlarının Fano algoritması ile çözülmesi için bir donanım uygulama mimarisi öneriyoruz. İlk olarak, PAC kodlarını çözmek için Fano algoritmasının donanım uygulamasına uygun olan yeni bir çeşidini tanıtıyoruz. Ardından, önerilen PAC Fano kod çözücünün alt bloklarının donanım şemalarını sağlamakla birlikte donanım karmaşıklığı ve yayılma gecikmesinin tahminini de sunuyoruz. Ayrıca, herhangi bir depolama öğesi veya karşılaştırıcı gerektirmeden mevcut ve önceki dal metrik değerlerini çevrimiçi olarak hesaplayabilen PAC kodlarının sıralı kodunun çözülmesi için yeni bir dal metrik birimi tanıtıyoruz. Önerilen kod çözücünün FPGA üzerinde hata düzeltme performansını ve ASIC donanım özelliklerini TSMC 28 nm 0.72 V kitaplığı ile değerlendiriyoruz. Blok uzunluğu 128 bit ve mesaj uzunluğu 64 bit olan bir kod için, önerilen kod çözücünün 500 MHz saat hızı ve 3.5 dB sinyal-gürültü oranında 3.85 mW güç tüketimi ile yaklaşık 38.1 Mb/s bilgi çıkışı elde edebileceğini gösteriyoruz.

Anahtar sözcükler: APAC kodları, sıralı kod çözme, Fano kod çözücü, polar kodlama, VLSI.

Acknowledgement

First, I would like to thank my advisor, Prof. Erdal Arıkan, whose office door was always open whenever I faced a problem or I had a question about my research. Without his guidance and advice, it would not have been possible to finish this thesis.

I would like to express my sincere gratitude to my thesis monitoring committee members, Prof. Orhan Arıkan and Prof. Ali Ziya Alkar, for their insightful and constructive suggestions throughout this work.

I would also like to thank Prof. Tolga Mete Duman and Asst. Prof. Barış Nakiboğlu for their willingness to serve as examiners for my thesis defense.

I would like to thank my wife, Maryam, for her love and unwavering support, for all the late nights and early mornings, and for keeping me sane during the course of this work. Thank you for being my muse, editor, and proofreader. Most importantly, thank you for being my best friend.

I would like to take this opportunity to thank my parents and siblings for their emotional and financial support throughout my years of study. Without their support, this journey would have been very difficult to get through.

Finally, I would like to thank my friend, colleague, and research partner, Mohsen Moradi, for our valuable discussions and the cherished time we spent together in the office and social settings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Channel coding is the process of improving the reliability of data transmission over a communication channel by adding redundancy to the source word. Figure 1.1 shows a general framework for channel coding that comprises a channel encoder, a communication channel, and a channel decoder. The channel encoder's primary function is to provide the receiver with the capability of detecting transmission errors and, in typical situation, correcting them. The process of error detection and correction is performed by the channel decoder located at the receiver side.

Figure 1.1: A general framework for channel coding.

Suppose that we want to transmit a source word $\mathbf{d}$ of K bits, $d_0, \ldots, d_{K-1}$. The encoder maps the information bits of the source word $\mathbf{d}$ to a codeword $\mathbf{x}$ of N bits, $x_0, \ldots, x_{N-1}$, resulting in a transmission rate of $R = K/N$. The encoded codeword is transmitted through a communication channel. At the receiver side, a noisy version of the codeword, $\mathbf{y} = (y_0, \ldots, y_{N-1})$, is received and passed to the channel decoder. The task of the decoder is to recover the source word bits d_i from the received noisy codeword $\mathbf{y}$.

According to Shannon's channel capacity theorem [3], by proper design of encoder and decoder, reliable transmission over a noisy channel is possible at any given rate R if $R < C$, where C (known as the channel capacity) is the largest rate at which information can be transmitted with a probability of error that approaches zero exponentially as the block length N increases.

In this regard, to allow the reliable transmission of information at rates close to the channel capacity, several high-performance channel codes have been developed. In particular, turbo codes [4], low-density parity-check (LDPC) codes [5–9], and polar codes [10] are three high-performing well-known classes of codes that have been used in many modern communication standards, such as, 3rd Generation (3G), 4th Generation (4G), and 5th Generation (5G) of mobile communication, and WiFi and satellite standards.

Although experimental results show that turbo codes and LDPC codes can achieve the channel capacity, they have no mathematical proof for their capacity achieving property. Polar codes, on the other hand, are the first class of codes that can provably achieve the channel capacity. These codes utilize the channel polarization that takes N independent copies of a given discrete memoryless channel (DMC) W and create a second set of N synthetic bit-channels $\{W_i : 0 \leq i \leq N-1\}$ that shows a polarization effect. Asymptotically, a $I(W)$ fraction of the bit-channel capacities $I(W_i)$ approach 1, and the rest approach 0, where $I(W)$ is the capacity of symmetric DMC. This class of codes benefits from low complexity encoding and decoding algorithms and have explicit code construction.

Originally proposed algorithm for decoding polar codes is the successive cancellation (SC) decoding [10]. Although SC algorithm benefits from low-complexity decoding, it suffers from throughput bottleneck and poor error-correction performance. To improve the performance of SC decoder, SC list (SCL) decoding has been proposed [11]. This algorithm was shown to achieve better error-correction performance compared to SC decoding algorithm, but still lacking the error-correction performance of the state-of-the art turbo and LDPC codes for short and moderate block lengths [11]. To address this issue, SCL decoding with cyclic

redundancy check (CRC) has been proposed in [12]. Another well-known decoder adapted to polar codes is belief propagation (BP) decoder which has been used in [13].

However, even under SCL-CRC decoding, there is a gap between the error-correction performance of polar codes and the dispersion approximation. The dispersion approximation [14] is an approximation of the lowest achievable probability of error for finite block length codes. Recently in [15], Arıkan proposed a new class of codes which combines the ideas from polar coding and convolutional coding and have been shown to perform near the dispersion approximation in certain cases [15] for short block lengths. These codes utilize a convolutional pretransform on polar codes and are called polarization-adjusted convolutional (PAC) codes. It has been shown that PAC codes can outperform short polar codes and convolutional codes for certain code rates [15–18].

The first algorithm used for decoding PAC codes is sequential decoding [19] algorithm in [15]. PAC codes can be decoded using any tree search algorithm such as depth-first, breadth-first, and beam search (constrained breadth-first search) algorithms. Sequential decoding is a depth-first tree search algorithm which was originally developed for convolutional codes. Despite their near-optimal performance, PAC codes under sequential decoding exhibit variable time complexity, resulting in variable decoding latency. Although the depth-first search algorithms have variable search complexity, their average search complexity is low in high signal-to-noise ratio (SNR) regime. On the other hand, breadth-first search algorithms have fixed but higher average search complexity. List decoding [17, 18] and list Viterbi decoding [20] of PAC codes are examples of beam search decoding. However, to achieve the error-correction performance of the PAC sequential decoder, these list decoders require a very large list size.

Two well-known sequential decoders are Fano [21] and stack algorithms [22,23]. Both algorithms use a path metric to search for the correct path through the code tree. The stack algorithm performs its search operation by storing the partial paths in a sorted stack in accordance with their path metrics and extends the most promising path at each step of decoding. On the other hand, the Fano

algorithm stores only the most promising path and requires a smaller memory than the stack algorithm. However, unlike the stack algorithm, the Fano decoder may visit a node more than once. In comparison to the stack algorithm, the Fano decoder is better suited for hardware implementations due to its lower memory usage and lack of sorting operations.

1.1 Motivation and Contributions

Various hardware implementation architectures have been reported in the literature for decoding of polar codes and Fano decoding of convolutional codes [2, 24–39], but to the best of our knowledge, the suitability of sequential decoding for PAC codes has never been studied from a hardware implementation perspective. Motivated by this, we implement a Fano decoder for PAC codes by introducing a new hardware-friendly variant of Fano algorithm for PAC codes. Moreover, we obtain a simplified branch metric function for sequential decoding of PAC codes and design a novel branch metric unit that is capable of calculating the current and previous branch metrics without requiring any storage element or comparator.

We provide analytical estimates for the hardware complexity and combinational delay of the proposed PAC Fano decoder. We also provide ASIC and FPGA implementation results of the proposed decoder and show that the decoder achieves an information throughput of approximately 38 Mb/s at 3.5 dB SNR with a power consumption of 3.85 mW and an area of 0.059 mm^2 with 28 nm 0.72 V technology, block length of $N = 128$, and a code rate of 1/2.

We compare the error-correction performance and ASIC implementation results of the PAC Fano decoder with those of the state-of-the-art polar decoders. The results show that the PAC Fano decoder achieves significantly better error correction performance compared to the SC and SCL decoders of polar codes with a block length of $N = 128$. The results also show that the PAC Fano decoder has similar error-correction performance with the SCL decoder (with a list size

of 2) of polar codes with a block length of $N = 1024$ and code rate of $1/2$. However, in terms of decoding throughput, the decoders of polar codes show superior performance compared to the proposed PAC Fano decoder.

1.2 Publications

P1 A. Mozammel, "Hardware Implementation of Fano Decoder for Polarization-adjusted Convolutional (PAC) Codes," *IEEE Transactions on Circuits and Systems II: Express Briefs, 2021.* [40].

P2 M. Moradi, A. Mozammel, K. Qin, and E. Arıkan, "Performance and Complexity of Sequential Decoding of PAC Codes," *arXiv preprint arXiv:2012.04990, 2020.* [16].

P3 M. Moradi and A. Mozammel, "A Monte-Carlo Based Construction of Polarization-adjusted Convolutional (PAC) Codes," *arXiv preprint arXiv:2106.08118, 2021.* [41].

P4 M. Moradi and A. Mozammel, "Concatenated Reed-Solomon and Polarization-Adjusted Convolutional (PAC) Codes," *arXiv preprint arXiv:2106.08822, 2021.* [42].

Paper P1 contains the fundamental results of this thesis. Paper P2 compares the error-correction performance and computational complexity of PAC codes with those of the conventional convolutional codes and polar codes with block length of $N = 128$. The results show that PAC codes under sequential decoding have superior error-correction performance compared to convolutional codes and have the potential to improve the error-correction performance of 5G polar codes. Paper P3 proposes a Monte-Carlo based rate-profile construction method for PAC codes that employs a trade-off between the error-correction performance and the decoding complexity of PAC codes. The simulation results show that, compared to RM-Polar rate profile construction method, the proposed construction method results in a coding gain of 0.5 dB at FER $= 10^{-3}$ for PAC codes with block

length of $N = 256$ and message length of $K = 128$. Paper P4 investigates the performance of PAC codes concatenated with Reed-Solomon (RS) codes. The simulation results show that concatenation scheme of RS and PAC codes has 0.25 dB coding gain at bit error rate of 10^{-5} compared to the concatenation scheme of RS and convolutional codes.

1.3 Organization of the Thesis

In Chapter 2, we give background information on polar codes, convolutional codes, and PAC codes. We explain the channel polarization phenomenon and common methods for rate profiling the polar codes. We provide a brief overview of SC polar code decoding. Then, we go over the convolutional codes, explain the Fano decoding of convolutional codes, and discuss the Fano algorithm's variable computational complexity. After that, we explain the encoding process of PAC codes and their Fano decoding algorithm.

In Chapter 3, we discuss the Fano decoding algorithm of PAC codes in more detail and obtain a hardware-friendly version of the algorithm which we use in the design of PAC Fano decoder. Then, we introduce a hardware architecture for Fano decoding of PAC codes. We provide a detailed discussion of hardware implementation of each block and its corresponding circuitry. We also analyze the hardware complexity and combinational delay of each individual block.

Chapter 4 presents the performance results for FPGA and ASIC implementations of our proposed PAC Fano decoder and compares them with the ASIC implementations of polar codes.

Chapter 5 concludes the thesis and provides potential use case applications for the PAC Fano decoder. This chapter also provides suggestions for new research directions that are relevant to the thesis's topics.

We investigate the impact of the number of quantization bits and the threshold

spacing on the error correction performance and time complexity of our PAC Fano decoder in Appendix A and B.

1.4 Notations

Throughout this thesis, we denote vectors and matrices by boldface letters. All matrix and vector operations are over vector spaces over the binary field $\mathbb{F}_2$. We represented addition over $\mathbb{F}_2$ by the $\oplus$ operator. Unless otherwise stated, all logarithmic functions log are in base 2. For any set $\mathcal{A} \subseteq \{0, 1, ..., N-1\}$, we denote its complement by $\mathcal{A}^c = \{i : i \notin \mathcal{A}\}$. For any vector $\mathbf{y}$ and set $\mathcal{A}$, $\mathbf{y}_\mathcal{A}$ denotes the sub-vector $(y_i : i \in \mathcal{A})$. For any vector $\mathbf{y}$, $\mathbf{y}^j = (y_0, y_1, ..., y_j)$ and $\mathbf{y}_i^j = (y_i, y_{i+1}, ..., y_j)$ for $i < j$. For any vector $\mathbf{y}$, $\mathbf{y}_e$ and $\mathbf{y}_o$ denotes sub-vectors with the elements of $\mathbf{y}$ whose indices are even and odd, respectively.

Chapter 2

Review of Codes

In this chapter, we provide background knowledge on the basics of polar codes, convolutional codes, and PAC codes.

2.1 Polar Codes

Polar codes [10] are the first family of codes to be proven to achieve Shannon channel capacity [3]. Polar codes have a low decoding and encoding complexity: both the decoding and encoding complexities are $\mathcal{O}(N \log N)$ for a code of length N. We investigate the system shown in Figure 2.1 in this section, which employs a polar code for channel coding. A polar code's block length is denoted by $N = 2^n$, where n is an integer number greater than 0. In the first block, the bits of the information vector $\mathbf{d} = (d_0, d_1, \cdots, d_{K-1})$ are inserted into the bits of a data-carrier vector $\mathbf{u} = (u_0, u_1, \cdots, u_{N-1})$. Then the uncoded bit vector $\mathbf{u}$, which contains the both information and redundant bits, is sent into the polar encoder. The encoder output codeword is $\mathbf{x}$, which is communicated across the channel W. The channel W in this system is a memoryless channel with a binary input alphabet $\mathcal{X}$, continuous output alphabet $\mathcal{Y}$, and the transition probabilities $W(y|x)$, where $y \in \mathcal{Y}$ and $x \in \mathcal{X}$. Each time the system is used, a codeword $\mathbf{x}$ is

communicated and a channel output vector $\mathbf{y}$ is received. On the receiver side, the decoder obtains an estimate of the data-carrier vector $\hat{\mathbf{u}}$ and then, using the frozen-bit vector as input, extracts the estimates of the information bits $\hat{\mathbf{d}}$ from the data-carrier vector $\hat{\mathbf{u}}$.

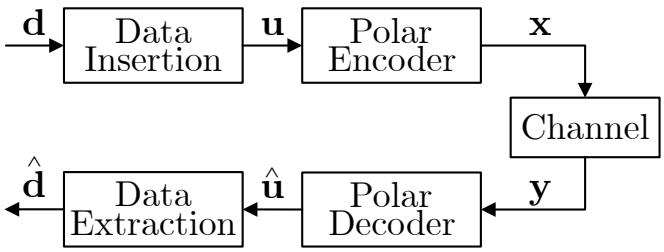

Figure 2.1: Polar coding scheme.

The mutual information of a channel W with uniform inputs (known as symmetric capacity) denoted by

$$I(W) \triangleq \sum_{y\in\mathcal{Y}} \sum_{x\in\mathcal{X}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)} \tag{2.1}$$

and the Bhattacharyya parameter of the channel

$$Z(W) \triangleq \sum_{y\in\mathcal{Y}} \sqrt{W(y|0)W(y|1)} \tag{2.2}$$

are the two important parameters. $I(W)$ reflects the maximum rate of the channel, and $Z(W)$ indicates the reliability of the channel (it is an upper bound for maximum-likelihood (ML) rule). Both parameters have values between 0 and 1 and are inversely related to one another.

2.1.1 Channel Polarization

Polar codes work by converting N independent and identical copies of a channel W into N synthetic (virtual) channels that are either better (less noisy) or worse (noisier) than the original channel W through a polarization transformation. Channel polarization is comprised of two processes: the channel combining process and the channel splitting process. We first explain the polarization effect using the $N = 2$ case. In this case, the channel combining process phase is depicted in Figure 2.2.

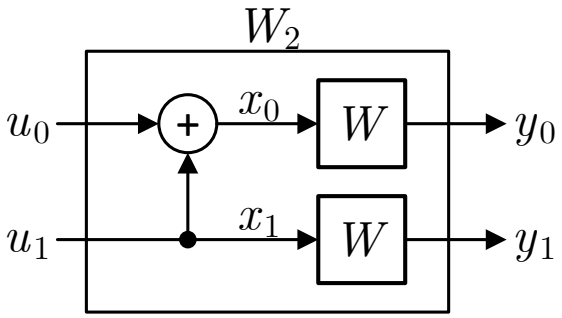

Figure 2.2: Channel combining for $N = 2$.

For a uniform input vector $\mathbf{u} = (u_0, u_1)$, channel combining produces the combined channel vector $W_2 : \{0,1\} \longrightarrow \mathcal{Y} \times \mathcal{Y}$, whose transition probabilities are denoted by

$$W_2(\mathbf{y}|\mathbf{u}) = W(y_0|x_0)W(y_0|x_0) = W(y_0|u_0 \oplus u_1)W(y_1|u_1). \tag{2.3}$$

We may express this figure transformation as matrix multiplication

$$W_2(\mathbf{y}|\mathbf{u}) = W^2(\mathbf{y}|\mathbf{u}G_2), \tag{2.4}$$

where

$$\mathbf{u}G_2 = (u_0, u_1)\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = (x_0, x_1). \tag{2.5}$$

The following step is channel splitting, which results in the creation of two synthetic channels $W_2^{(0)}$ and $W_2^{(1)}$. For the first synthetic channel $W_2^{(0)}$, the transition probability can be expressed as

$$\begin{aligned} W_2^{(0)}(\mathbf{y}|u_0) &= \sum_{u_1 \in \mathcal{X}} W_2(\mathbf{y}, u_1|u_0) = \sum_{u_1 \in \mathcal{X}} \frac{1}{2} W_2(\mathbf{y}|u_0, u_1) \\ &= \sum_{u_1 \in \mathcal{X}} \frac{1}{2} W(y_0|u_0 \oplus u_1)W(y_1|u_1) \\ &= \frac{1}{2} W(y_0|u_0)W(y_1|0) + \frac{1}{2} W(y_0|u_0 \oplus 1)W(y_1|1), \end{aligned} \tag{2.6}$$

where the third equality is by (2.3). If $W_2^{(0)}(\mathbf{y}|0) > W_2^{(0)}(\mathbf{y}|1)$, we can estimate the first bit as $\hat{u}_0 = 0$, and otherwise as $\hat{u}_0 = 1$.

Next assume that the genie-aided decoder provides the correct value of the previous bit u_0 to the second synthetic channel $W_2^{(1)}$. In this case, for $W_2^{(1)}$, we

can express the transition probability as

$$W_2^{(1)}(\mathbf{y}, u_0|u_1) = \frac{1}{2}W_2(\mathbf{y}|u_0, u_1) = \frac{1}{2}W(y_0|u_0 \oplus u_1)W(y_1|u_1), \tag{2.7}$$

where the first equality is by the Bayes' rule and the second equality is by (2.3).

The capacity of the original W and synthetic channels $(W_2^{(0)}, W_2^{(1)})$ are as

$$\begin{aligned} I(W_2^{(0)}) &\leq I(W) \leq I(W_2^{(1)}), \\ I(W_2^{(0)}) &+ I(W_2^{(1)}) = 2I(W). \end{aligned} \tag{2.8}$$

These formulas demonstrate that when channel polarization occurs, the overall mutual information is retained, and that one synthetic channel has a larger capacity than the initial channel while the other has a lower capacity [10].

A similar relationship is obtained in terms of the channels' Bhattacharyya parameters as

$$\begin{aligned} Z(W_2^{(0)}) &\geq Z(W) \geq Z(W_2^{(1)}), \\ Z(W_2^{(0)}) &+ Z(W_2^{(1)}) = 2Z(W), \end{aligned} \tag{2.9}$$

with the second equation satisfying the equality if and only if the channel W is a binary erasure channel (BEC).

Using the aforementioned polarization method, if one wishes to communicate a single bit of information, the information is carried onto u_1 bit and sent across the more reliable synthetic channel $W_2^{(1)}$. Other bit, u_0, is selected as frozen bit and given a value that is known to both the encoder and decoder. A common practice is to set the frozen bits to zero. The fixed value of frozen bit is utilized in the decoding to retrieve the information that has been encoded.

This technique of channel transformation can be expanded recursively using the formulae

$$\begin{aligned} W_{2N}^{(2i)}(\mathbf{y}^{2N-1}, \mathbf{u}^{2i-1}|u_{2i}) = \sum_{u_{2i+1}} \frac{1}{2} &W_N^{(i)}(\mathbf{y}^{N-1}, \mathbf{u}_o^{2i-1} \oplus \mathbf{u}_e^{2i-1}|u_{2i} \oplus u_{2i+1}) \\ &\cdot W_N^{(i)}(\mathbf{y}_N^{2N-1}, \mathbf{u}_e^{2i-1}|u_{2i+1}) \\ W_{2N}^{(2i+1)}(\mathbf{y}^{2N-1}, \mathbf{u}^{2i}|u_{2i+1}) = \frac{1}{2} &W_N^{(i)}(\mathbf{y}^{N-1}, \mathbf{u}_o^{2i-1} \oplus \mathbf{u}_e^{2i-1}|u_{2i} \oplus u_{2i+1}), \\ &\cdot W_N^{(i)}(\mathbf{y}_N^{2N-1}, \mathbf{u}_e^{2i-1}|u_{2i+1}) \end{aligned} \tag{2.10}$$

for $0 \leq i \leq N-1$, so that we get the $2N$ synthetic channels $(W_{2N}^{(2i)}, W_{2N}^{(2i+1)})$ in $\log N + 1$ recursions. The relations for the capacities and the Bhattacharyya parameters of the resulting bit-channels are as

$$\begin{aligned} I(W_{2N}^{(2i)}) &\leq I(W_N^{(i)}) \leq I(W_{2N}^{(2i+1)}) \\ I(W_{2N}^{(2i)}) &+ I(W_{2N}^{(2i+1)}) = 2I(W_N^{(i)}), \end{aligned} \tag{2.11}$$

and

$$\begin{aligned} Z(W_{2N}^{(2i)}) &\geq Z(W_N^{(i)}) \geq Z(W_{2N}^{(2i+1)}) \\ Z(W_{2N}^{(2i)}) &+ Z(W_{2N}^{(2i+1)}) \leq 2Z(W_N^{(i)}), \end{aligned} \tag{2.12}$$

where the last equation is an equality if and only if the channel $W_N^{(i)}$ is a BEC. Arıkan [10] proved that for each binary-input discrete memoryless channel (B-DMC) W, the synthetic channels $W_N^{(i)}$ polarize, i.e., asymptotically (as N goes to infinity) the fraction of synthetic channels for which $I(W_N^{(i)})$ is close to 1 (reliable channels) goes to $I(W)$ and the fraction of synthetic channels for which $I(W_N^{(i)})$ is close to 0 (noisy channels) goes to $1 - I(W)$.

2.1.2 Code Construction

An issue in the context of polar coding is the determination of the synthesized channels reliabilities, which is known as the polar code construction. While the code construction in polar codes is explicit in principle, estimating the reliability of synthesized channels is challenging in practice. Specifically, when the underlying channel of the synthesized bit channels or the SNR value change, the indices of reliable bit channels change as well. As a result, in an (N, K) polar code, in order to pick the K highest reliable channels from N synthesized channels, a rate-profiling algorithm must be developed for calculating the bit-channel reliabilities and selecting the K indices with highest reliabilities.

There is an efficient construction technique for a BEC [10]. If the underlying BEC's Bhattacharyya parameter is ϵ, from (2.12) the two obtained synthesized channels are also BECs with Bhattacharyya values of $2\epsilon - \epsilon^2$ and ϵ^2 [13]. The Bhattacharyya parameters is determined recursively, and eventually, the data

transmission is performed using the K channels with the least Bhattacharyya parameters.

Numerous different estimation approaches are given for constructing polar codes for generic channels. Arıkan suggested that for a given channel with capacity C, the Bhattacharyya parameters of the synthesized channels can be calculated in the same way as calculating the Bhattacharyya parameters of a BEC channel with capacity C [13]. As another technique, density evaluation [43] is used to construct polar codes in [44]. Moreover, it was proposed in [45] (Gaussian approximation method) that intermediate log-likelihood ratios (LLRs) can be treated as Gaussian random variables for the AWGN channel in order to decrease the complexity of density evaluation.

In the Gaussian approximation approach, the update rule for mutual information in a single level transform is as

$$\begin{aligned} I^- &= 1 - J\left[\sqrt{2}J^{-1}(1-I)\right], \\ I^+ &= J\left[\sqrt{2}J^{-1}(I)\right], \end{aligned} \tag{2.13}$$

where the initial value I is the mutual information of the given binary input AWGN channel and I^+ and I^- are the approximated mutual information values after one level evolution [46,47]. The approximations of the function $J(t)$ and its inverse $J^-(t)$ are as [48]

$$\begin{aligned} J(t) &= \left[1 - 2^{-0.3073t^{2\times 0.8935}}\right]^{1.1064}, \\ J^{-1}(t) &= \left[-\frac{1}{0.3073}\log_2\left(1 - t^{\frac{1}{1.1064}}\right)\right]^{\frac{1}{2\times 0.8935}}. \end{aligned} \tag{2.14}$$

After $\log N$ evolution levels, estimates for $I(W_N^{(i)})$ are obtained, and the K channels with the highest mutual information are employed to transmit the data in an (N, K) polar code.

2.1.3 Rate Profiling

In Figure 2.1, the first block is the data insertion block. This block accepts a source word $\mathbf{d}$ of length K and inserts it into a data carrier word $\mathbf{v}$ of length N in accordance with a data index set $\mathcal{A}$, where $\mathbf{v}_{\mathcal{A}} = \mathbf{d}$ and $\mathbf{v}_{\mathcal{A}} = \mathbf{0}$. The selection of index set $\mathcal{A}$ along with the data insertion is referred to as rate profiling. Two known rate-profiling techniques for the polar codes are the polar rate profiling [10] and Reed-Muller-polar (RM-polar) [49] rate profiling. As described in the preceding section, polar rate profiling selects the set $\mathcal{A}$ depending on the bit-channel reliabilities (corresponding to the highest $I(W_N^{(i)})$ values or equivalently with the smallest $Z(W_N^{(i)})$ values).

To understand the RM-polar rate profiling procedure, consider the generator matrix of a rate one polar code. The generator matrix of a rate-one polar code with a code length $N = 2^n$ is $\mathbf{F}^{\otimes n}$, which is the n-th Kronecker product of the kernel matrix $\mathbf{F} = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$. The matrix $\mathbf{F}^{\otimes n}$ has 1 row with weight N, $\binom{n}{1}$ rows with wight $N/2$, $\binom{n}{2}$ rows with wight $N/4$, and so on. To construct an (N, K) polar code with RM-polar rate profiling, we first obtain the smallest $k = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots \binom{n}{r}$, where $0 \leq r \leq n$, such that $k \geq K$. The corresponding (N, k) RM code has a minimum Hamming distance equal to $N/2^r$ [50]. In an (N, K) polar code, the set $\mathcal{A}$ is obtained from the corresponding row indices of (N, k) RM code with the K most reliable ones.

It is important to realize that polar codes constructed on the basis of the Bhattacharyya parameters or mutual information of the bit channels are optimized for SC decoders. As a result, they are not always optimal when used for other decoders such as the sequential or SCL decoders. To the best of our knowledge, there is no explicit polar code construction procedure optimised for sequential or SCL decoding, and so the nature of sequential or list decoding is often ignored when designing polar codes. As a result constructing polar codes for sequential or SCL decoding is still an open problem. However, there are several heuristic approaches for code construction that converge to a satisfactory result [41,51,52].

2.1.4 Encoding of Polar Codes

Polar encoder block of Figure 2.1 maps the output of the data insertion block $\mathbf{u}$ of length N into a codeword vector $\mathbf{x} = \mathbf{u}\mathbf{F}^{\otimes n}$. Polar encoder is a one-to-one mapping and $\mathbf{F}^{\otimes n}$ is the n-th Kronecker product of the kernel matrix $\mathbf{F} = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$.

2.1.5 SC Decoding of Polar Codes

Polar codes with the SC decoding were presented in [10]. Because of the channel combining in polar codes, which has been explained in the previous sections, a correlation between the data bits is introduced. As a consequence, each coded bit associated with a particular index i is dependent on the previous data bits associated with lower indices than i. When utilized, this type of correlation may be conceptualized like interference in the data-bit domain, which results in significantly improved decoding performance. As a result, in the SC decoding bits are decoded successively; $\hat{u}_{i-1}$ is estimated before $\hat{u}_i$ and all the previously determined bits $\hat{\mathbf{u}}^{i-1}$ impact the decision of current bit $\hat{u}_i$.

We explain the SC decoding for $N = 4$, and it would be straightforward to generalize it. Figure 2.3 demonstrates the factor graph of SC decoder for $N = 4$. As shown in this figure, in SC decoding, the data bits are estimated via hard decision on the final LLRs λ_i^0. If the ith bit is a frozen bit ($i \in \mathcal{A}^c$), regardless of the value of final LLR λ_i^0, decoder assigns $\hat{u}_i = 0$. Otherwise, the value of $\hat{u}_i$ is determined by a local maximum likelihood (ML) rule expressed as the equation (2.15), which is based on the output $(\mathbf{y}, \hat{\mathbf{u}}^{i-1})$ of the ith bit-channel $W_N^{(i)}$.

$$\hat{u}_i = h(\lambda_i^0) = \begin{cases} 0, & \text{if } \lambda_i^0 = \log \frac{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | \hat{u}_i = 0)}{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | \hat{u}_i = 1)} > 0, \\ 1, & \text{otherwise.} \end{cases} \tag{2.15}$$

The receiver first calculates the channel LLR vector $\boldsymbol{\lambda} = \lambda_0^{n-1}, \ldots \lambda_{N-1}^{n-1}$ with

$$\lambda_i^{n-1} = \ln \frac{P(y_i | x_i = 0)}{P(y_i | x_i = 1)}, \tag{2.16}$$

for each element of the channel output vector. Then the bits are estimated successively by updating the intermediate LLR values. each intermediate LLR λ_i^j, where i is the bit index and j is the stage index, is computed by

$$\lambda_i^j = \begin{cases} f(\lambda_i^{j+1}, \lambda_{i+2^j}^{j+1}), & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 2 = 0 \\ g(\lambda_{i-2^j}^{j+1}, \lambda_i^{j+1}, \hat{\beta}_{i-2^j}^j), & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 2 = 1, \end{cases} \tag{2.17}$$

where $0 \leq j \leq n$, $0 \leq i \leq N-1$, and $\hat{\beta}_i^j$ is used to represent the partial sum, which is the left to right propagation of the estimated bits $\hat{u}_i$ in the factor graph. The propagation of $\hat{\beta}_i^j$ is show in the Figure 2.3.

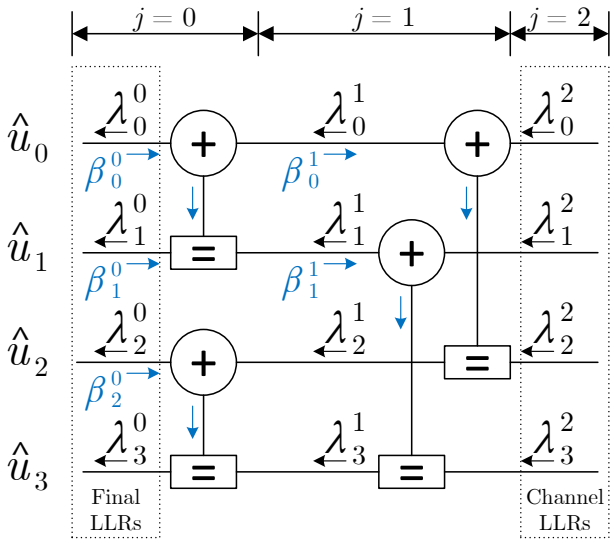

Figure 2.3: SC decoding factor graph.

Figure 2.4 depicts the f and g functions used in (2.17). These functions are well approximated as

$$f(\lambda_a, \lambda_b) \approx \operatorname{sgn}(\lambda_a) \cdot \operatorname{sgn}(\lambda_b) \cdot \min\left(|\lambda_a|, |\lambda_b|\right) \tag{2.18}$$

$$g(\lambda_a, \lambda_b, \hat{\beta}) = (-1)^{\hat{\beta}} \lambda_a + \lambda_b, \tag{2.19}$$

where λ_a and λ_b denote the entering LLRs to a node, respectively, and $\hat{\beta}$ denotes the partial sum of previously estimated bits.

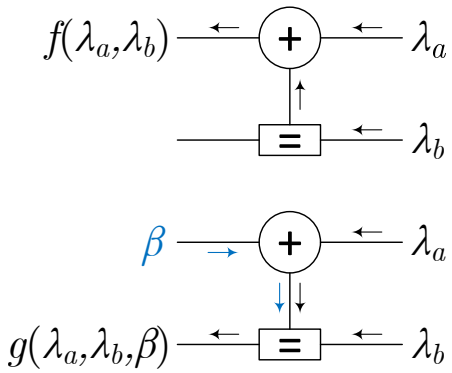

Figure 2.4: Internal LLR (λ) calculations.

2.2 Convolutional Codes

Convolutional codes [53] are a class of linear codes with a particular structure in the generator matrix so that the encoding operation can be viewed as a convolution operation. The encoding process of convolutional codes can be expressed by $\mathbf{x} = \mathbf{dT}$, where $\mathbf{d}$ and $\mathbf{x}$ are the input and output of the encoder, respectively, and $\mathbf{T}$ is the generator matrix. For time-invariant convolutional codes, $\mathbf{T}$ is a Toeplitz matrix in a row-reduced echelon form. In this case, the generator matrix is often characterized by a generator polynomial $\mathbf{c} = (c_0, \cdots, c_m)$, where $c_0 \neq 0$ and $c_m \neq 0$. The parameters m and $m+1$ are called the memory order and constraint length of the convolution, respectively. Then the convolution operation can be expressed as $x_i = \sum_{j=0}^{m} c_j d_{i-j}$ with $d_{i-j} = 0$ for $i < j$. This corresponds to combining m previous bits of the input $\mathbf{d}$ and the current input bit d_i in accordance with the generator polynomial $\mathbf{c}$. As a result, a rate $r = k/n$ convolutional encoder with memory order m can be implemented as a k-input, n-output circuit with shift register of length m.

Figure 2.5 shows an example circuit for a convolutional code with $k = 1$, $n = 2$, $m = 2$, $\mathbf{c}_0 = (1, 0, 1)$, and $\mathbf{c}_1 = (1, 1, 1)$, where $\mathbf{c}_0 = (c_{0,0}, c_{0,1}, c_{0,2})$ and $\mathbf{c}_1 = (c_{1,0}, c_{1,1}, c_{1,2})$ are the polynomials for generating the first and second output

of convolutional encoder, respectively. The generator matrix of this example is

$$\mathbf{T} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

It is also possible to represent a convolutional code's encoding in the form of a tree through which each source word $\mathbf{d} = (d_0, ..., d_{K-1})$ of length K defines a path. Figure 2.6 shows the first four levels of the tree corresponding to the convolutional code of Figure 2.5. For a node at level i of this tree, the upper-branch corresponds to $d_i = 0$ and the lower-branch corresponds to $d_i = 1$. Each branch is labeled by the corresponding convolution output pair $(x_{2i}x_{2i+1})$.

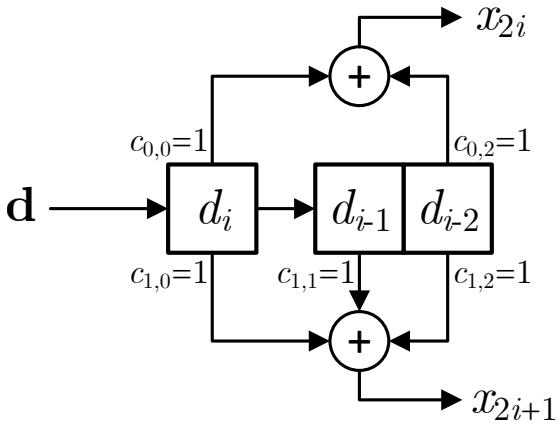

Figure 2.5: An example of convolutional encoding using a shift-register.

2.2.1 Fano Decoding of Convolutional Codes

The optimal algorithm for decoding convolutional codes is the Viterbi algorithm [54], which is a maximum likelihood sequence estimator. The Viterbi algorithm operates on the trellis diagram of the convolutional codes in which the nodes represent the encoder state [55]. At each step, the algorithm examines the encoder's entire state space $\mathcal{S} = \{s_1, \dots, s_{2^m}\}$, where m is the memory order

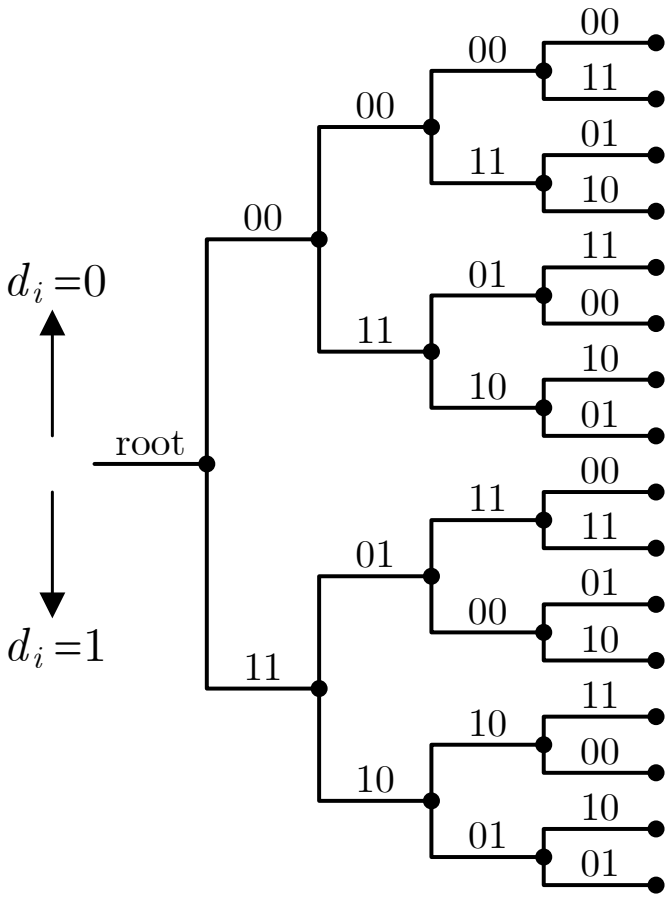

Figure 2.6: Tree representation of a convolutional code.

of convolutional code. For this reason, the computational complexity of Viterbi decoder grows exponentially with the memory order of convolutional code.

For practical implementations, especially when the memory order of the convolutional code is large, sub-optimal decoding algorithms are also of interest. These algorithms achieve the majority of the performance of the Viterbi algorithm while having a lower computational complexity. Two of the well known sub-optimal decoding algorithms are the two versions of sequential decoding [19] known as Fano algorithm [21] and stack algorithm [22, 23].

In stack decoding, the partial explored paths of Figure 2.6 are stored in a stack and sorted with respect to their path metric values. At each step, the algorithm pops the top element of the stack and extends its path. Then, the children paths of the top path are pushed to the stack and a sorting is performed on the stack. For this reason, the stack algorithm is a memory intensive algorithm and requires sorting operation at each step of decoding.

The Fano algorithm is a depth-first search algorithm that only keeps track of the most promising path. As a result, the memory requirement of the Fano decoder is smaller compared to the stack algorithm, but unlike the stack algorithm, the Fano algorithm may visit each node more than once. The task of Fano algorithm is to traverse through the convolution tree and find the correct path which is an estimate $\hat{\mathbf{d}}$ of the transmitted message $\mathbf{d}$. To identify the correct path, the Fano algorithm uses a path metric Γ and a metric threshold T.

The function used by the Fano decoder for calculating the path metric is called the Fano metric and is defined as

$$\Gamma_i = \sum_{j=0}^{i} \gamma_j = \sum_{j=0}^{i} \left[\log \frac{P(y_j|x_j)}{P(y_j)} - b_j \right], \tag{2.20}$$

where γ_j is the branch metric, $P(y_j|x_j)$ is the channel transition probability, $P(y_j)$ is the channel output probability, and b_j is a bias term. The Fano metric is the optimum metric for comparing paths of different lengths [56]. If the Fano metric grows along a given path, the algorithm considers it as a correct path and continues to search further along it. But if the metric drops significantly,

the algorithm backtracks and searches other paths. This is accomplished by comparing the path metric with a threshold T in a way that whenever the path metric grows significantly on a forward search, the threshold is tightened (raised by threshold spacing Δ), and whenever the algorithm backtracks, the threshold is relaxed (lowered by Δ). In this way, the algorithm ensures that no node will ever be searched twice using the same threshold [1, p 372-373].

To explain the Fano algorithm, we examine the flowchart of Figure 2.7. At the beginning of decoding, the threshold T and a search pointer i are initialized to 0. In the look forward block, the algorithm computes both metrics of the two branches $d_i = 0$ and $d_i = 1$, and if the algorithm is looking forward to most likely node, it adds the greater of the two branch metrics to the current path metric. If the look forward block is entered from $\boxed{\text{A}}$, it means that the most likely node has already been searched and the threshold T has been violated. Hence, the algorithm must look to the least likely node (the branch with the smaller branch metric). In either case, the Fano decoder compares the new path metric Γ_i with the threshold T, and if $\Gamma_i \geq T$ (T satisfied), the algorithm moves forward by incrementing the search pointer i. After a forward move, if the node is visited for the first time, the threshold is tightened by increasing the threshold by integer multiples of Δ such that the new threshold satisfies $\Gamma_i - \Delta < T \leq \Gamma_i$.

In the look forward block if $\Gamma_i < T$ (T violated), the algorithm looks back and compares the previous path metric Γ_{i-1} with the threshold, and if $\Gamma_{i-1} \geq T$ (T satisfied) it moves back by decrementing the search pointer i. If the backward move was made on the better of the two branches stemming from the node just reached (the most likely node), the least likely node has to be searched next. But, if it was made on the least likely branch, there is no branch left to be explored, and the algorithm must continue the backward search. If, during a backward look, the threshold T is violated, the algorithm cannot move back, and eventually, all paths from here have been searched and violated the current threshold. Thus, the algorithm decreases the threshold by Δ and continues the forward search using the new threshold. The Fano algorithm keeps repeating these steps until it reaches the last level of the convolution tree, i.e., $i = N - 1$, or a predefined early termination criterion like maximum search limit is satisfied. An extensive

study of the Fano algorithm may be found in [57].

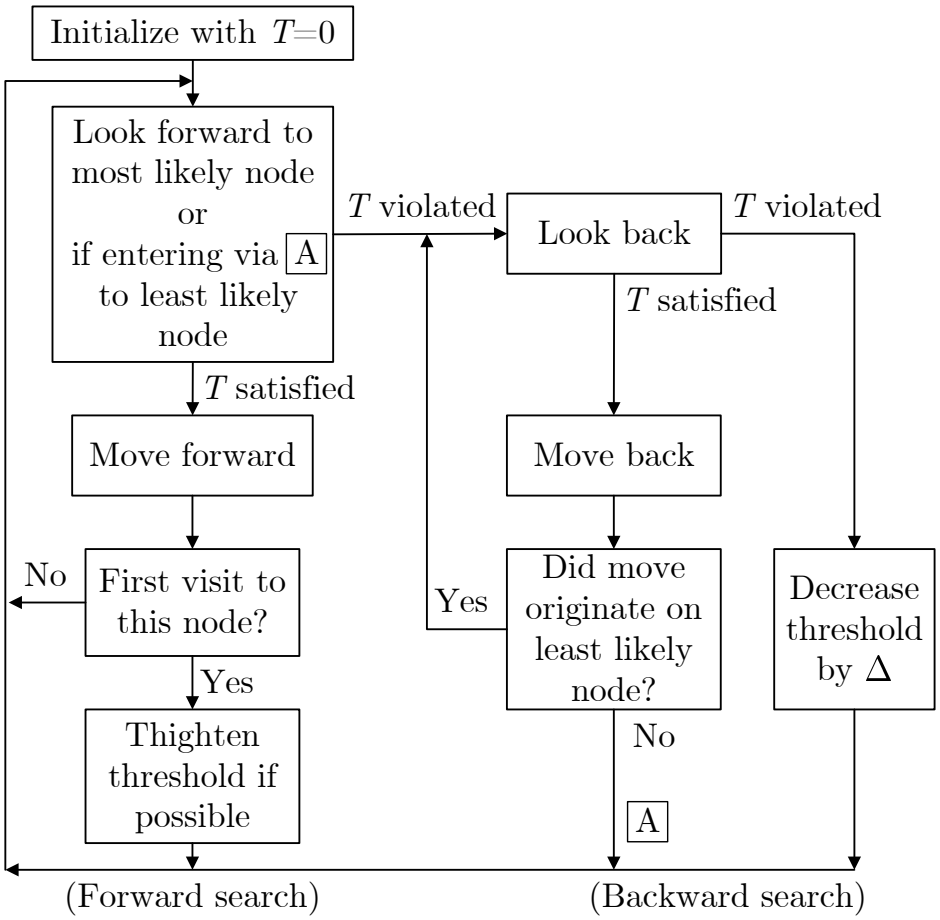

Figure 2.7: Fano decoding algorithm for binary tree, modified after [1, p 373]

2.2.2 Computational Complexity of Sequential Decoding

Sequential decoding has the benefit of having a computational complexity that does not depend to the convolutional encoder constraint length. When it comes to the Viterbi decoding, on the other hand, the computational complexity increases exponentially with respect to the encoder constraint length. This limits the use of Viterbi decoding to very small constraint lengths, which makes reaching arbitrarily low error probability in practice unfeasible Additionally, even when just a few or no errors are available in the received sequence (high SNR values), the Viterbi algorithm executes a constant amount of calculations per decoding

bit. On the other hand, sequential decoding is capable of adjusting the amount of computations per decoding bit in response to the intensity of background noise. However, the downside of sequential decoding is that the amount of computations needed is random and may become too large in conditions of extreme noise, resulting in decoding failures or erasure.

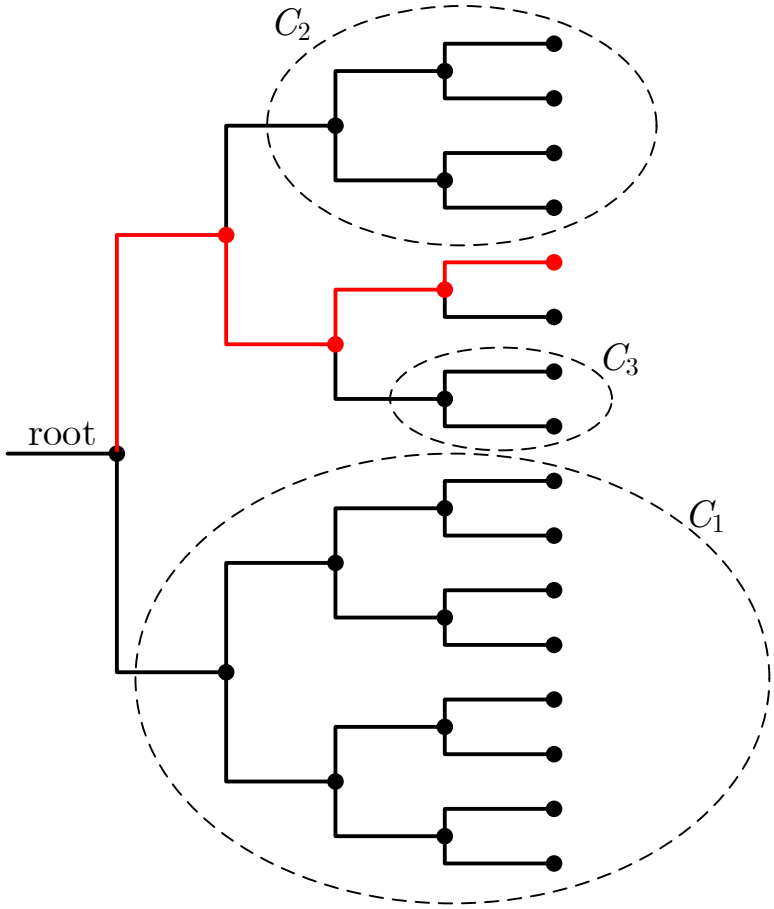

Figure 2.8: Incorrect subsets for the first three nodes.

The amount of computations performed throughout the decoding process is the primary assessment tool in sequential decoding. The ith wrong subset of the code tree is a notion that is used to theoretically measure the computation of sequential decoding and is defined as the set of all the nodes that branch out from the ith node of the correct path, $1 \leq i \leq N$. Figure 2.8 shows incorrect subsets for the first three nodes of a convolutional code whose correct path is highlighted by red. It is proved that the amount of computations C_i done in the

ith wrong subset follows a Pareto distribution, which is given by

$$P(C_i \geq x) \approx Ax^{-\rho}, \quad 1 \leq i \leq N-1, \tag{2.21}$$

for $0 < \rho < \infty$, where A is a constant that differ based on the sequential decoding technique in use.

The parameter ρ is referred to as the Pareto exponent, and it is associated to the coding rate R using the equation

$$R = \frac{E_0(\rho)}{\rho}, \quad 0 < R < C, \tag{2.22}$$

where C is the channel capacity (in bits per channel usage). The function $E_0(\rho)$ is referred to as the Gallager function, and for a symmetric B-DMC it is expressed by

$$E_0(\rho) = \rho - \log_2 \frac{1}{2} \sum_j \left[\sum_i W(j|i)^{\frac{1}{1+\rho}} \right]^{1+\rho}. \tag{2.23}$$

The average amount of computations per decoded branch is infinite if ρ is less than or equal to one. For a finite average amount of computations ($\rho > 1$) we have

$$R = \frac{E_0(\rho)}{\rho} < E_0(1) = R_0, \tag{2.24}$$

where R_0 is referred to as the computational cutoff rate of the channel ($R_0 < C$). This means that for sequential decoding to have a finite average computational complexity, the code rate R has to be less than the channel cutoff rate R_0.

R_0 is referred to as the computational cutoff rate and is often represented as R_{comp}. This is a feasible upper limit on the largest code rate R at which a sequential decoder may run, because a Pareto distribution with $\rho = 1$ has an infinite mean. This implies that any sequential decoder working at a rate greater than R_0 would have significant computational issues, including frequent buffer overflows. R_0 is often used to anticipate the minimal practical value of E_b/N_0 with which a sequential decoder would work [58, p. 303]. For this reason, R_0 is a critical performance parameter of sequential decoding. As an example, we get $R_0 = 1/2$ at 2.46 dB SNR for a convolutional code operating at the rate 1/2.

2.3 PAC Codes

Figure 2.9 shows a block diagram of PAC coding scheme. The data insertion block receives a source word $\mathbf{d}$ of length K and inserts it into a data carrier word $\mathbf{v}$ of length N in accordance with a data index set $\mathcal{A}$ such that $\mathbf{v}_{\mathcal{A}} = \mathbf{d}$ and $\mathbf{v}_{\mathcal{A}^c} = \mathbf{0}$. The bits fixed to zero are called frozen, whereas all the other bits are called non-frozen. As a result, the coding rate can be expressed as $R = K/N$. The data carrier word $\mathbf{v}$ goes through a convolution block with generator matrix $\mathbf{T}$ which is a Toeplitz matrix whose first row is the generator polynomial $\mathbf{c} = (c_0, \cdots, c_m)$, where $c_0 \neq 0$ and $c_m \neq 0$. Similar to the convolutional codes, the parameters m and $m+1$ are called the memory order and constraint length of the convolution, respectively. The resulting word $\mathbf{u}$ goes through a polar mapper and the overall encoding process of PAC codes can be expressed by $\mathbf{x} = \mathbf{vTF}^{\otimes n}$, where $\mathbf{F}^{\otimes n}$ is the generator matrix of polar codes with $\mathbf{F}^{\otimes n}$ being the n-th Kronecker product of the kernel matrix $\mathbf{F} = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$.

At the receiver side, the PAC decoder receives the channel output $\mathbf{y}$ and generates an estimate $\hat{\mathbf{v}}$ of $\mathbf{v}$. Then, a data extractor extracts an estimate $\hat{\mathbf{d}}$ of $\mathbf{d}$ from $\hat{\mathbf{v}}$ using $\hat{\mathbf{d}} = \hat{\mathbf{v}}_{\mathcal{A}}$. The performance of the system is measured by the probability of frame error $P_e = P(\hat{\mathbf{d}} \neq \mathbf{d})$.

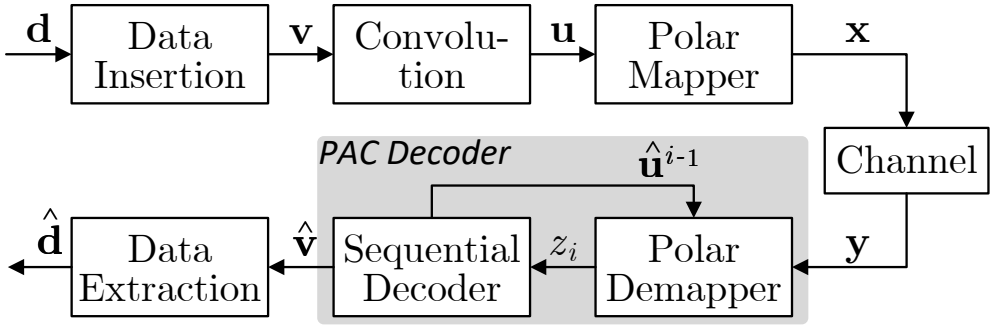

Figure 2.9: PAC coding scheme.

Consider an example of PAC codes with $N = 8$, $K = 4$, $\mathcal{A} = \{3, 5, 6, 7\}$, and $\mathbf{c} = (1, 0, 1)$. The data insertion block inserts the source word $\mathbf{d} = (d_0, \ldots, d_3)$ into the carried word $\mathbf{v} = (v_0, \ldots, v_7)$ so that

$$\mathbf{v} = (0, 0, 0, d_0, 0, d_1, d_2, d_3).$$

The encoded codeword $\mathbf{x}$ for this example can be obtained by

$$\mathbf{x} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ d_0 \\ 0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}^T}_{\mathbf{v}} \underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}}_{\mathbf{F}^{\otimes 3}} \tag{2.25}$$

$$= \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ d_0 \\ 0 \\ d_0 + d_1 \\ d_2 \\ d_1 + d_3 \end{bmatrix}^T}_{\mathbf{vT}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}}_{\mathbf{F}^{\otimes 3}} \tag{2.26}$$

Just like conventional convolutional codes, it is also possible to represent the convolution operation of PAC codes in the form of a tree through which each carrier word $\mathbf{v} = (v_0, \ldots, v_{K-1})$ defines a path. Due to the existence of frozen bits, the tree generated by the convolution operation of PAC codes is an irregular tree that branches only at indices provided by the data index set $\mathcal{A}$. In other words, for a node at level i, the tree branches if and only if $i \in \mathcal{A}$.

Figure 2.10 shows the convolution tree of a PAC code with $N = 8$, $K = 4$, $\mathcal{A} = \{3, 5, 6, 7\}$, and $\mathbf{c} = (1, 0, 1)$. For a node at level i of this tree, the upper-branch corresponds to $v_i = 0$ and the lower-branch corresponds to $v_i = 1$. Each branch is labeled by the corresponding convolution output u_i.

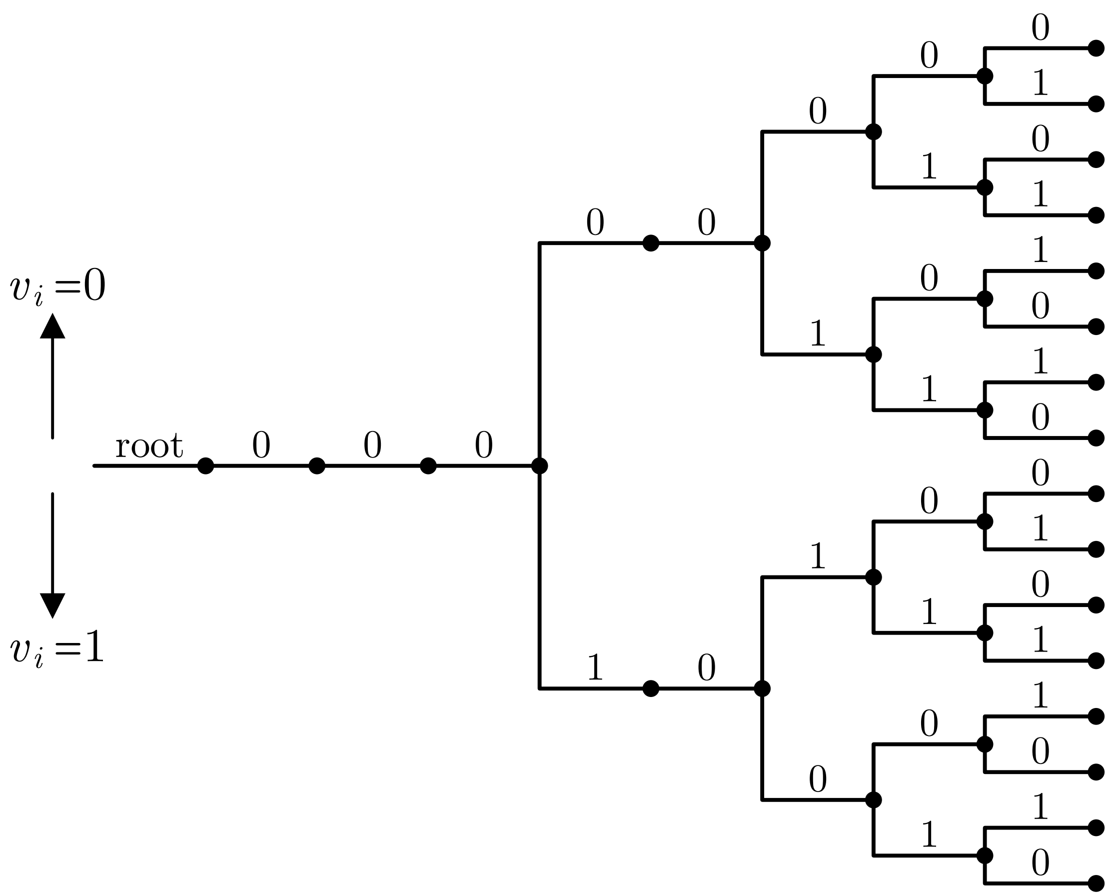

Figure 2.10: Tree representation of convolutional operation of a PAC code.

2.3.1 Fano Decoding of PAC Codes

As mentioned in Chapter 1, the Fano variant of sequential decoding requires a smaller memory size and is more suitable for hardware implementations compared to stack algorithm. For this reason, in this section, we focus on the Fano decoding of PAC codes. A PAC Fano decoder consists of two blocks: polar demapper and the Fano decoder. The polar demapper receives the channel output $\mathbf{y}$ and calculates a log-likelihood ratio (LLR) vector $\boldsymbol{\lambda}$ according to (2.16). Similar to the SC decoder of polar codes, the polar demapper also operates on the SC decoding factor graph. But unlike the SC decoder, the polar demapper does not make any hard decision on the final LLRs, i.e., it does not generate any bit-estimate output $\hat{u}_i$. Instead, it receives the prior bit-estimates $\hat{\mathbf{u}}^{i-1}$ from the Fano decoder and passes the soft LLR value of the ith bit back to the Fano decoder. In short, a polar demapper can be considered and implemented as a soft-in soft-out SC decoder. To distinguish the final LLRs from the intermediate LLRs, we denote the final LLRs (the output of polar demapper) by $\boldsymbol{z} = (z_0, \cdots, z_{N-1})$. Hence, we can express the output of polar demapper by

$$z_i \triangleq \ln \frac{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | \hat{u}_i = 0)}{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | \hat{u}_i = 1)}. \tag{2.27}$$

The Fano decoder uses the z_i value to calculate a path metric which helps the decoder to generate an estimate $\hat{\mathbf{v}}$ of $\mathbf{v}$.

The Fano decoding of PAC codes is similar to the Fano decoding of conventional convolutional codes but with a difference that the Fano decoder of PAC codes performs its search for the correct path on an irregular tree. Figure 2.11 demonstrates the flowchart of Fano decoding for PAC codes. The flowchart of PAC Fano decoder is similar to the flowchart of the Fano decoder of convolutional codes shown in Figure 2.7 with a single difference that upon a backward move, the PAC Fano decoder checks whether the currently reached node is frozen or non-frozen; if the node is frozen, there is no more branches to be explored, and the algorithm must continue the backward search; if the node is non-frozen the operation of algorithm is the same as decoding convolutional codes. This block is indicated with a dashed line in Figure 2.11.

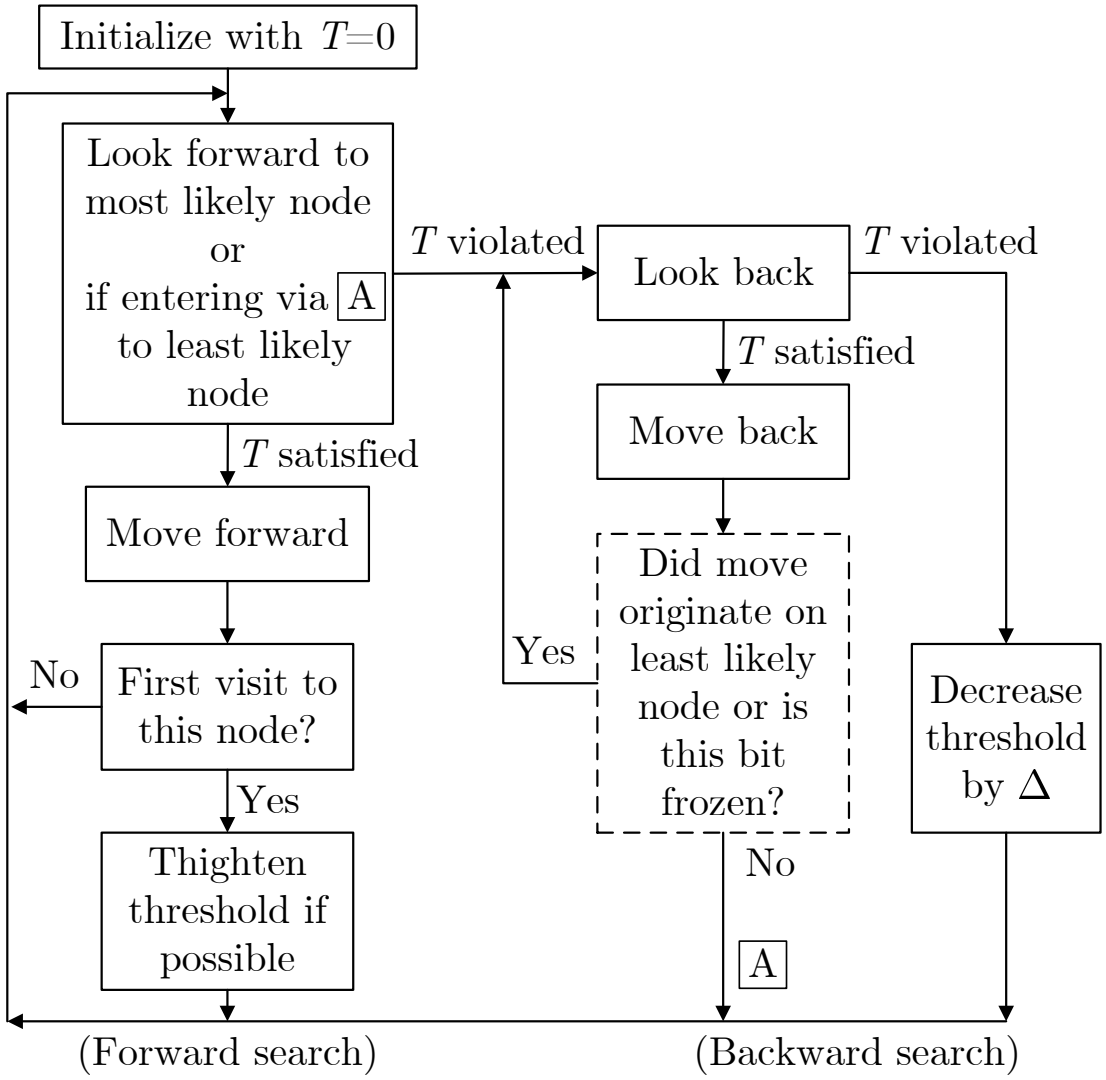

Figure 2.11: Fano decoding algorithm for PAC codes.

The Fano algorithm uses the output z_i of the polar demapper to calculate the path metric Γ_i of node i based on the hypothesis $\hat{u}_i = 0$ and $\hat{u}_i = 1$, which are also provided by the Fano decoder. Since the channel seen by the convolution operation of PAC codes is a polarized channel, the Fano metric of (2.20) is not suitable for decoding of PAC codes. Since the rate-one convolution operation and polar mapping are one-to-one transforms, the path metric for PAC codes can expressed as [47]

$$\Gamma_i = \log \frac{P(\mathbf{y}|\hat{\mathbf{u}}^i)}{P(\mathbf{y})} - \sum_{j=0}^{i} b_j, \tag{2.28}$$

where $\mathbf{y} = (y_0, \cdots, y_{N-1})$ are the channel output values, $\hat{\mathbf{u}}^i = (\hat{u}_0, \cdots, \hat{u}_i)$ are the prior bit-estimates, and b_j is a bias term. Since the Fano decoder always moves from a current node either to its predecessor or to one of its immediate successors, it is more convenient to obtain a branch metric function and calculate the path metric as the sum of branch metrics. The branch metric can be calculated using (2.28) as

$$\begin{aligned} \gamma_i(\hat{u}_i) &\triangleq \Gamma_i - \Gamma_{i-1} \\ &= \log \frac{P(\mathbf{y}|\hat{\mathbf{u}}^i)}{P(\mathbf{y}|\hat{\mathbf{u}}^{i-1})} - b_i \\ &= \log \frac{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1}|\hat{u}_i)}{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1})} - b_i, \end{aligned} \tag{2.29}$$

where $P(\mathbf{y}, \hat{\mathbf{u}}^{i-1}|\hat{u}_i)$ is the bit-channel transition probability, $P(\mathbf{y}, \hat{\mathbf{u}}^{i-1})$ is the bit-channel output probability, and b_i is a bias term. Consequently, the path metric can be calculated as

$$\Gamma_i = \sum_{j=0}^{i} \gamma_j(\hat{u}_j). \tag{2.30}$$

For a binary input channel with equiprobable inputs, $\hat{u}_i \in \{0, 1\}$ with $P(\hat{u}_i = 0) = P(\hat{u}_i = 1) = 1/2$. Assume $\hat{u}_i = 0$. Then, the branch metric function of

(2.29) can be written as

$$\begin{aligned}
\gamma_i(\hat{u}_i = 0) &= \log \frac{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1}|\hat{u}_i = 0)}{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1})} - b_i \\
&= \log \frac{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1}|\hat{u}_i = 0)}{\frac{1}{2}[P(\mathbf{y}, \hat{\mathbf{u}}^{i-1}|\hat{u}_i = 0) + P(\mathbf{y}, \hat{\mathbf{u}}^{i-1}|\hat{u}_i = 1)]} - b_i \\
&= 1 - \log\left(1 + \frac{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1}|\hat{u}_i = 1)}{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1}|\hat{u}_i = 0)}\right) - b_i \\
&= 1 - \log\left(1 + e^{-z_i}\right) - b_i,
\end{aligned}$$

where z_i is the output of polar demapper which is defined in (2.27). Similarly, we can obtain the branch metric for the condition $\hat{u}_i = 1$ as

$$\gamma_i(\hat{u}_i = 1) = 1 - \log\left(1 + e^{z_i}\right) - b_i.$$

We can express the Fano branch metric for decoding PAC codes in a compact form of

$$\gamma_i(\hat{u}_i) = 1 - \log(1 + e^{-(1-2\hat{u}_i)z_i}) - b_i. \tag{2.31}$$

Note that the metric function of (2.31) is a function of polar demapper output z_i and the estimate convolution output $\hat{u}_i$. To generate the estimate $\hat{u}_i$, the Fano decoder uses a convolutional encoder replica with the same generator polynomial $\mathbf{c}$ used in the encoder of PAC codes. As shown in Figure 2.10, for a non-frozen node at level i of the convolution tree, there are always two possible values of $\hat{u}_i$ each corresponding to one of the hypothesis $\hat{v}_i = 0$ (upper-branch) and $\hat{v}_i = 1$ (lower-branch). Let $\hat{u}_{i,0}$ and $\hat{u}_{i,1}$ be the convolution outputs for the hypothesis $\hat{v}_i = 0$ and $\hat{v}_i = 1$, respectively. After generating $\hat{u}_{i,0}$ and $\hat{u}_{i,1}$, the Fano decoder uses the output of polar demapper z_i and calculates $\gamma_i(\hat{u}_{i,0})$ and $\gamma_i(\hat{u}_{i,1})$. Now by comparing $\gamma_i(\hat{u}_{i,0})$ and $\gamma_i(\hat{u}_{i,1})$ the algorithm can distinguish the most likely branch from the least likely branch; if $\gamma_i(\hat{u}_{i,0}) \geq \gamma_i(\hat{u}_{i,1})$ holds, then the most likely branch is $\hat{v}_i = 0$; otherwise, then the most likely branch is $\hat{v}_i = 1$.

In the next chapter, we study the Fano decoding of PAC codes in more details and provide a detailed flowchart for Fano decoding of PAC codes. We then modify the Fano algorithm to obtain a hardware-friendly PAC Fano decoder.

Chapter 3

Hardware Implementation of PAC Fano Decoder

In section 2.3.1, we showed an abstract flowchart for Fano decoding of PAC codes (Figure 2.11). In this chapter, we provide a detailed version of the flowchart of Figure 2.11 and obtain a hardware-friendly variant of Fano algorithm for decoding PAC codes. After that, we introduce a hardware architecture for the obtained Fano decoder of PAC codes.

3.1 A Hardware-Friendly Fano Algorithm for PAC Codes

Consider the local node diagram of the Fano decoding tree shown in Figure 3.1. We denote the time unit (node pointer) by i. Assume that N1 is the current node, N2 is the most likely node (with larger metric), N3 is the least likely node (with smaller metric), and N4 is the previous node. M2 and M3 denote the metrics of branches from the current node to N2 and N3, respectively, and M1 is the metric of the branch from the previous node to the current node. We define N23 as a node that corresponds to N2 when the most likely node is being examined or N3

when the least likely node is being examined. Similarly, M23 corresponds to M2 when N2 is being examined or M3 when N3 is being examined.

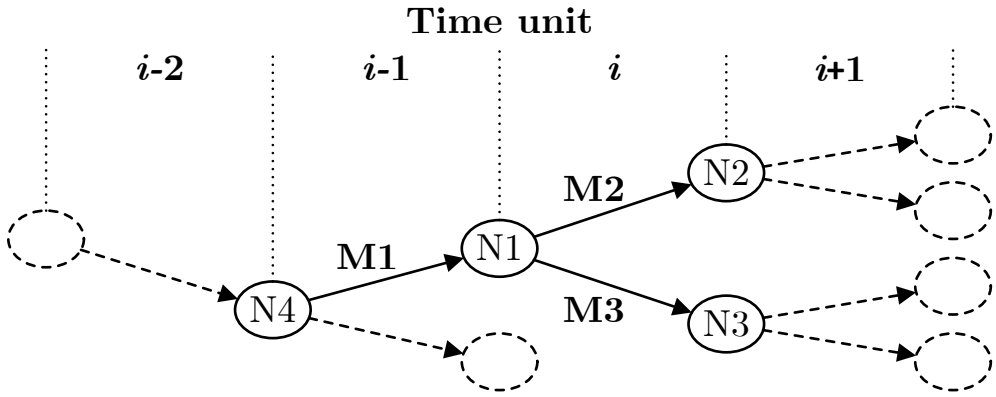

Figure 3.1: Fano decoding tree, modified after [2].

The path metric of (2.30) is expressed as the sum of branch metric values. This may result in a metric overflow especially at high SNR regime. To avoid this, we compute relative metric values rather than absolute path metric values [2]. In this regard, whenever the Fano decoder performs a forward move, the value of M23 is subtracted from the threshold T, and whenever it performs a backward move the value of M1 is added to the threshold T. Using this method lets the current node's path metric be zero and all other path metrics relative to the current node's metric. Moreover, using the relative metric values makes it possible for the Fano algorithm to perform its search operation by using the branch metric values (M1 and M23), and eliminates the need for calculating the path metric values. Let us define a variable Ψ such that $\Psi = 1$ when the Fano decoder backtracks to a frozen node or to a node whose both children are examined; otherwise, $\Psi = 0$. In other words, the Fano algorithm performs forward search when $\Psi = 0$; otherwise it performs backward search. We consider a node as a new node if it is being visited for the first time. Using the introduced parameters and concepts, we modify the flowchart of Figure 2.11 and present an extended flowchart in Figure 3.2.

By careful observation of the flowchart of Figure 3.2 we realize that there are five independent set of actions from which the Fano algorithm performs one at each step of decoding. These five sets of actions are labeled by A0-A4. Action A0

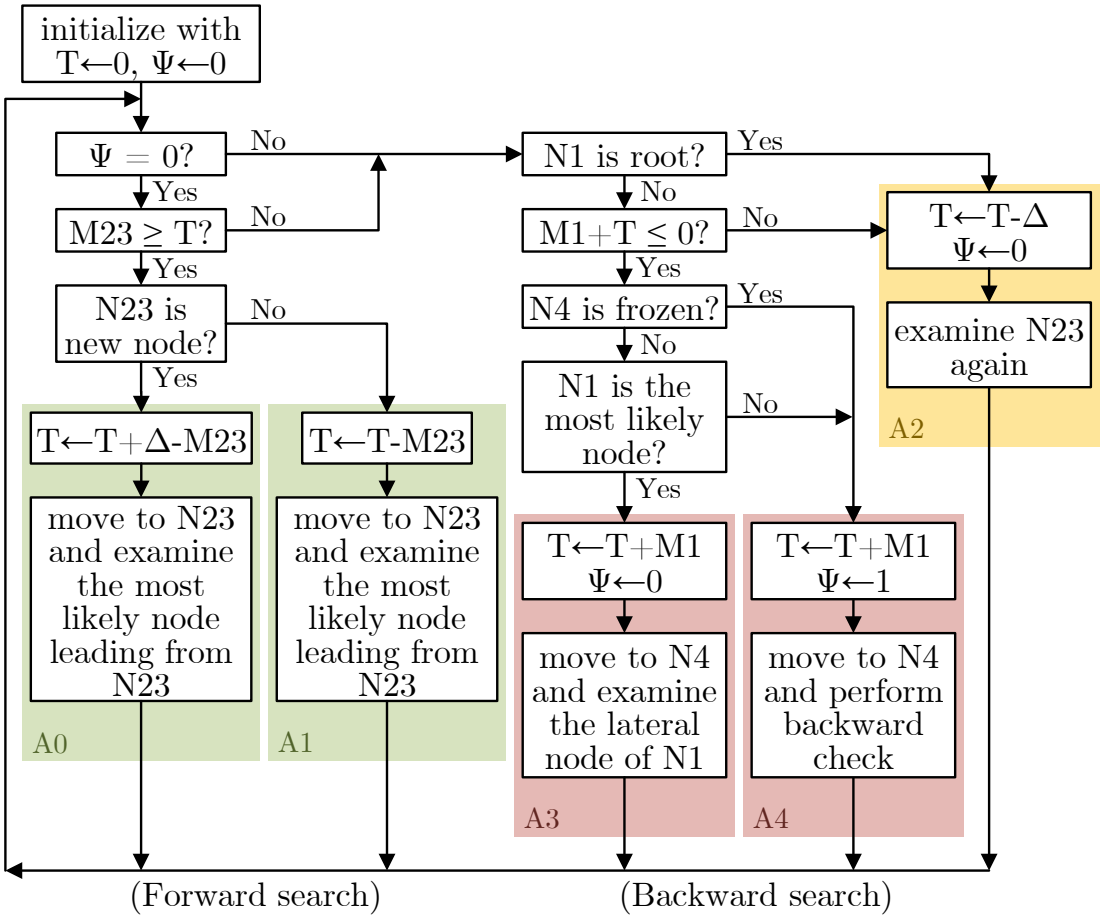

Figure 3.2: Extended Fano algorithm flowchart for PAC codes.

corresponds to moving forward to a new node; A1 corresponds to moving forward to an old node; A2 corresponds to the case when the Fano algorithm can move neither forward nor backward and needs to drop the threshold; A3 corresponds to moving backward and checking the lateral node of N1; A4 corresponds to moving backward and performing backward search again. Hence, we may restate the Fano algorithm as the following set of rules.

• *Rule 0*

Conditions: $\Psi = 0$, M23 $\geq T$, N23 is new node.

Actions: Move to N23, update T to $T + \Delta -$ M23, examine the most likely node leading from N23 at the next step.

• *Rule 1*

Conditions: $\Psi = 0$, M23 $\geq T$, N23 is old node.

Actions: Move to N23, update T to $T -$ M23, examine the most likely node leading from N23 at the next step.

• *Rule 2*

Conditions: $\Psi = 0$, M23 $< T$, N1 is root node; or $\Psi = 0$, M23 $< T$, N1 is not root node, M1 $+ T > 0$; or $\Psi = 1$, N1 is root node; or $\Psi = 1$, N1 is not root node, M1 $+ T > 0$

Actions: Make no move, update T to $T - \Delta$, assign $\Psi = 0$, examine N2 again at the next step

• *Rule 3*

Conditions: $\Psi = 0$, M23 $< T$, N1 is not root node, M1 $+ T \leq 0$, N4 is not frozen, N1 is the most likely node leading from N4; or $\Psi = 1$, N1 is not root node, N4 is not frozen, N1 is the most likely node leading from N4.

Actions: Move to N4, update T to $T +$ M1, assign $\Psi = 0$, examine the lateral node of N1 at the next step.

• *Rule 4*

Conditions: $\Psi = 0$, M23 $< T$, N1 is not root node, M1 $+ T \leq 0$, N4 is frozen or N1 is the least likely node leading from N4; or $\Psi = 1$, N1 is not root node,

$\mathrm{M1} + T \leq 0$, N4 is frozen or N1 is the least likely node leading from N4;
Actions: Move to N4, update T to $T + \mathrm{M1}$, assign $\Psi = 1$, perform backward check at the next step.

It is possible to detect if a node is new using M23 as follow [59, p. 436]:

- If $0 < T + \Delta$ holds, the node N1 cannot have been visited with a threshold higher than T. Hence, N1 was a new node and, since N23 is a successor of N1, N23 is also a new node.
- If $0 \geq T + \Delta$ and $\mathrm{M23} < T + \Delta$ hold, then N23 cannot have been visited before, but N1 has. Since $\mathrm{M23} < T + \Delta$, the threshold is already tight.
- If $0 \geq T + \Delta$ and $\mathrm{M23} \geq T + \Delta$ hold, both N1 and N23 have been visited before and the threshold should bot be increased.

Hence, node N23 is considered as a new node if

$$0 < T + \Delta \leq \mathrm{M23}, \tag{3.1}$$

where the first inequality indicates N23 is a new node and the second inequality implies that the threshold is not already tight. To detect whether a node is frozen or not, we define a binary vector $\boldsymbol{a}$ such that $a_i = 0$ for frozen nodes ($i \in \mathcal{A}^c$) and $a_i = 1$ for non-frozen nodes ($i \in \mathcal{A}$) for $i = 0, 1, \ldots, N-1$. Additionally, we define another binary vector $\mathbf{t}$ such that when $t_i = 0$, the Fano algorithm examines the most likely branch, and when $t_i = 1$, it examines the least likely branch for $i = 0, 1, \ldots, N-1$.

The branch metrics M1 and M23 are calculated using the metric function of (2.31) which is a function of polar demapper output. Hence, the polar demapper is required to provide the Fano algorithm with the values of z_{i-1} and z_i at every step of decoding. However, when the Fano algorithm drops the threshold (rule 2), the calculated value of z_i can be reused in the next step of decoding. Moreover, when the Fano decoder moves backward from a least-likely node or to a frozen node (rule 4), it does not need the value of M23 during the next step of decoding since it performs backward search.

Hence, if the polar demapper stores its previous output values, the activation of the polar demapper is not required after the Fano decoder executes rule 2 or rule 4. In this regard, we introduce a "polar demapper enable" (PDE) parameter such that the polar demapper is activated if PDE $= 1$. This parameter is set to logical one when any of rules 0, 1, or 3 is executed and to logical zero otherwise. This method prevents unnecessary activation of the polar demapper and significantly reduces the decoder's latency, especially at low SNR regimes where backtracking happens more frequently. Using these definitions and the condition for detecting a new node, we summarize the Fano rules and their corresponding conditions and actions in Table 3.1.

Table 3.1: Fano rules for decoding PAC codes.

Rule 0	Cond.	$(\Psi = 0)$ &$(\text{M23} \geq T)$ & $(0 < T + \Delta \leq \text{M23})$
	Actions	store $\hat{v}_i$ and $\hat{u}_i$, $T \leftarrow T + \Delta - \text{M23}$, $i \leftarrow i + 1$, $t_i \leftarrow 0$, PDE $\leftarrow 1$
Rule 1	Cond.	$(\Psi = 0)$ & $(\text{M23} \geq T)$ & $(T + \Delta \leq 0 \mid T + \Delta > \text{M23})$
	Actions	store $\hat{v}_i$ and $\hat{u}_i$, $T \leftarrow T - \text{M23}$, $i \leftarrow i + 1$, $t_i \leftarrow 0$, PDE $\leftarrow 1$
Rule 2	Cond.	$(\text{M23} < T)$ & $(\text{M1} + T > 0)$ $(\text{M23} < T)$ & $(i = 0)$ $(\Psi = 1)$ & $(\text{M1} + T > 0)$ $(\Psi = 1)$ & $(i = 0)$
	Actions	$T \leftarrow T - \Delta$, $\Psi \leftarrow 0$, $t_i \leftarrow 0$, PDE $\leftarrow 0$
Rule 3	Cond.	$(\text{M23} < T)$ & $(i \neq 0)$ & $(\text{M1} + T \leq 0)$ & $(a_{i-1} = 0)$ & $(t_{i-1} = 0)$ $(\Psi = 1)$ & $(i \neq 0)$ & $(\text{M1} + T \leq 0)$ & $(a_{i-1} = 0)$ & $(t_{i-1} = 0)$
	Actions	$T \leftarrow T + \text{M1}$, $\Psi \leftarrow 0$, $i \leftarrow i - 1$, $t_i \leftarrow 1$, PDE $\leftarrow 1$
Rule 4	Cond.	$(\text{M23} < T)$ & $(i \neq 0)$ & $(\text{M1} + T \leq 0)$ & $(a_{i-1} = 1 \mid t_{i-1} = 1)$ $(\Psi = 1)$ & $(i \neq 0)$ & $(\text{M1} + T \leq 0)$ & $(a_{i-1} = 1 \mid t_{i-1} = 1)$
	Actions	$T \leftarrow T + \text{M1}$, $\Psi \leftarrow 1$, $i \leftarrow i - 1$, PDE $\leftarrow 0$

Figure 3.3 shows the corresponding flowchart of the Fano decoder. The "Determine Rule and Perform Corresponding Actions" block executes the Fano rules of Table 3.1. A decoding session ends when either the Fano algorithm reaches the end of the frame (i.e. $i = N - 1$) or a predetermined termination criterion (such as maximum iteration bound) is satisfied.

As an example, consider a PAC code with $N = 8$, $K = 5$, $\mathcal{A} = \{1, 3, 5, 6, 7\}$, and $\mathbf{c} = (1, 0, 1)$. For a source word $\mathbf{d} = (0, 1, 1, 1, 1, 1)$, the data carrier word is constructed as $\mathbf{v} = (0, 0, 0, 1, 0, 1, 1, 1)$ and encoded as $\mathbf{x} =$

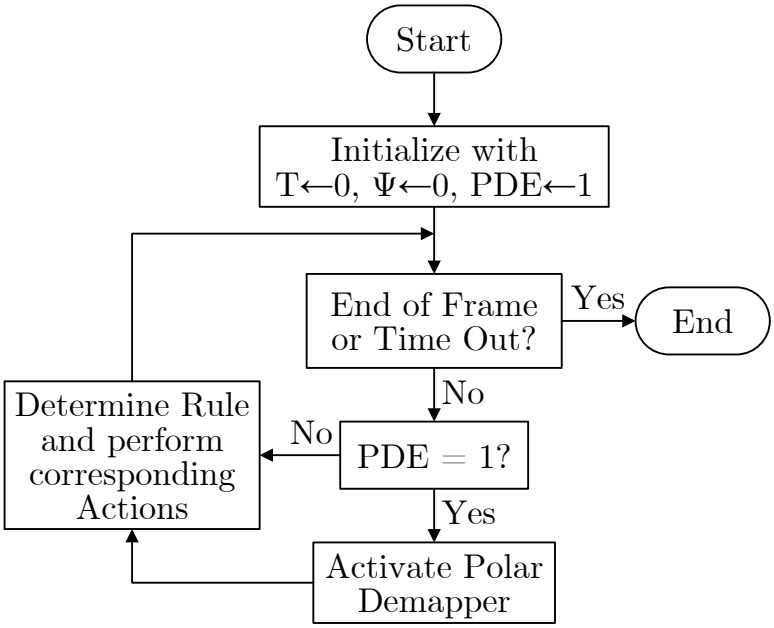

Figure 3.3: Flowchart of PAC Fano decoder.

$(0, 1, 0, 1, 1, 0, 1, 0)$. Assume a BPSK signalling over an AWGN channel with a noise variance corresponding to an SNR of 1 dB and a channel output of $\mathbf{y} = (-1.674, 1.180, 0.101, 1.109, 3.022, 0.218, 0.022, -2.880)$. The receiver calculates the channel LLR vector $\boldsymbol{\lambda}$ from the channel output $\mathbf{y}$ and passes it to the PAC Fano decoder. Table 3.2 traces the execution of the proposed PAC Fano decoder when the branch metric function of (2.31) is used with bias vector $\mathbf{b} = (0, 0, 0, 1, 0, 1, 1, 1)$ and $\Delta = 2$. The PAC Fano decoder obtains a correct estimate of data carrier word in 15 iterations. In this table, $\text{MF} = (\gamma_i(\hat{u}_{i,0}), \gamma_i(\hat{u}_{i,1}))$ denotes the forward branch metrics, where $\hat{u}_{i,0}$ and $\hat{u}_{i,1}$ correspond to the convolution output estimates for the branches $\hat{v}_i = 0$ and $\hat{v}_i = 1$, respectively.

Table 3.2: An example of PAC Fano decoder.

Step	i	Ψ	a_i	a_{i-1}	t_i	t_{i-1}	T	$\hat{\mathbf{v}}$	$\hat{\mathbf{u}}$	z_i	M1	MF	M23	Rule
1	0	0	0	—	0	—	0	()	()	0.056	—	$(0.040, -)$	0.040	1
2	1	0	1	0	0	0	-0.040	(0)	(0)	-0.606	0.040	$(-0.502, 0.372)$	0.372	1
3	2	0	0	1	0	0	-0.412	(0,1)	(0,1)	-3.342	0.372	$(-3.872, -)$	-3.872	3
4	1	0	1	0	1	0	-0.040	(0)	(0)	-0.606	0.040	$(-0.502, 0.372)$	-0.502	4
5	0	1	0	—	0	—	0	()	()	0.056	—	$(0.040, -)$	0.040	2
6	0	0	0	—	0	—	-2	()	()	0.056	—	$(0.040, -)$	0.040	1
7	1	0	1	0	0	0	-2.040	(0)	(0)	-0.606	0.040	$(-0.502, 0.372)$	0.372	1
8	2	0	0	1	0	0	-2.412	(0,1)	(0,1)	-3.342	0.372	$(-3.872, -)$	-3.872	3
9	1	0	1	0	1	0	-2.040	(0)	(0)	-0.606	0.040	$(-0.502, 0.372)$	-0.502	1
10	2	0	0	1	0	1	-1.534	(0,0)	(0,0)	2.242	-0.502	$(0.854, -)$	0.854	0
11	3	0	1	0	0	0	-0.392	(0,0,0)	(0,0,0)	-6.401	0.854	$(-9.237, -0.002)$	-0.002	1
12	4	0	0	4	0	0	-0.390	(0,0,0,1)	(0,0,0,1)	-0.197	-0.002	$(-0.149, -)$	-0.149	1
13	5	0	1	0	0	0	-0.241	(0,0,0,1,0)	(0,0,0,1,0)	2.222	-0.149	$(-3.354, -0.148)$	-0.148	1
14	6	0	1	1	0	0	-0.0915	(0,0,0,1,0,1)	(0,0,0,1,0,0)	-11.627	-0.149	$(-16.775, 0)$	0	1
15	7	0	1	1	0	0	-0.0915	(0,0,0,1,0,1,1)	(0,0,0,1,0,0,1)	—	—	—	—	—

3.2 Hardware Implementation

In this section, we introduce a hardware architecture for the Fano decoder of PAC codes which have been obtained in the previous section. Figure 3.4 shows the hardware architecture of the proposed PAC Fano decoder. This architecture mainly comprises a polar demapper (PD), a branch metric unit (BMU), a Fano control unit, a convolution output register (Ureg), and a convolution input register (Vreg). The Fano control unit implements the flowchart of Figure 3.3. The branch metric unit (BMU) provides the Fano control unit with the current branch metric M23 and previous branch metric M1. Vreg is a bidirectional shift register used to store the prior convolution input estimates $\hat{\mathbf{v}}$. Whenever the Fano decoder moves forward, the current convolution input estimate $\hat{v}_i$ is stored in Vreg. To allow a maximum backtracking depth of N, the size of Vreg is set to N and the first m part of Vreg provides the BMU with convolution state (CS), where m is the memory order of the convolution. The Ureg register is used to store the prior convolution output estimates $\hat{\mathbf{u}}$. When the Fano decoder moves forward, depending on the proceeding branch, the corresponding $\hat{u}_i$ is stored in the Ureg. The input buffer stores the channel output LLR values, and the output buffer stores the final estimate $\hat{\mathbf{v}}$ of $\mathbf{v}$. A clock cycle (CC) counter counts the number of clock cycles consumed to decode a single codeword. The decoding of a codeword is terminated whenever the value of the CC counter exceeds a predefined maximum cycle (MC). In this case, a timeout (TO) signal is generated, and a new LLR vector $\boldsymbol{\lambda}$ is loaded into the input buffer. The input $\boldsymbol{a}$ determines the frozen and non-frozen nodes such that $a_i = 0$ for frozen nodes ($i \in \mathcal{A}^c$) and $a_i = 1$ for non-frozen nodes ($i \in \mathcal{A}$). In the following subsections, we show how to design the hardware for three of the main sub-blocks of our proposed decoder: the polar demapper, the branch metric unit, and the Fano control unit.

3.2.1 Polar Demapper

A polar demapper can be implemented by modifying the SC decoder of polar codes as they both operate on the polar decoding factor graph of Figure 2.3. In

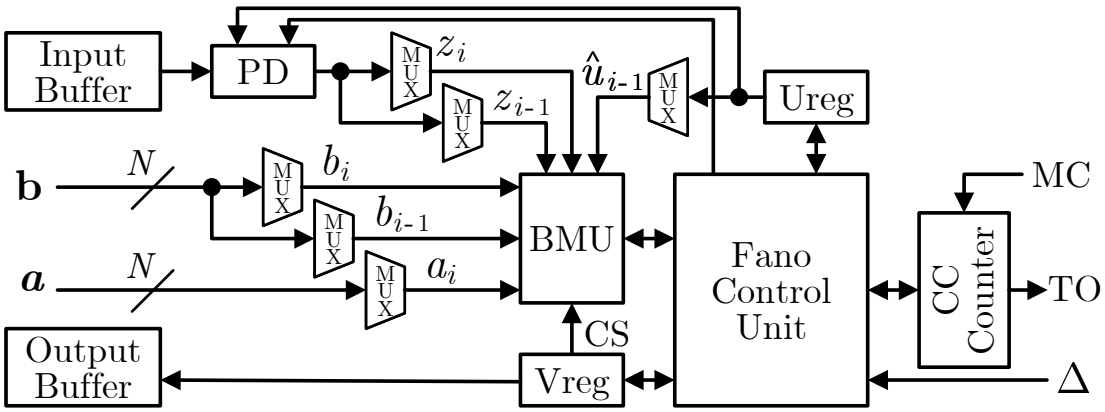

Figure 3.4: PAC Fano decoder.

this regard, we adopt the fully parallel FFT-like architecture of [60] and apply the following modifications to make it operate as a polar demapper.

1. The polar demapper is required to generate the soft values of final LLRs. For this reason, we remove the decision unit of the SC decoder which was used to generate the bit-decisions from the final LLRs.

2. The bit estimates $\hat{\mathbf{u}}$ are provided by the Fano decoder; hence, we remove the partial sum update unit and its corresponding registers. Instead, we implement the partial sum update network using a combinational circuit that receives the bit estimates $\hat{\mathbf{u}}$ from Ureg and updates the partial sum values. The reason behind using a combinational circuit is to reduce the latency of the polar demapper in terms of the number of clock cycles.

3. Due to the sequential nature of the Fano decoder, the z_i values are required to be generated in a natural order. We modify the bit-reversal architecture of [60] to generate the z_i values in a natural order.

Figure 3.5 shows the factor graph of polar demapper for $N = 8$. The channel LLR values λ_i are assumed to be presented to the right hand side of the graph and the final LLR values are generated on the left side. This polar demapper is composed of $n = \log N$ stages, each of which containing N process elements (f or g). The f and g blocks implement the functions of (2.18) and (2.18), respectively.

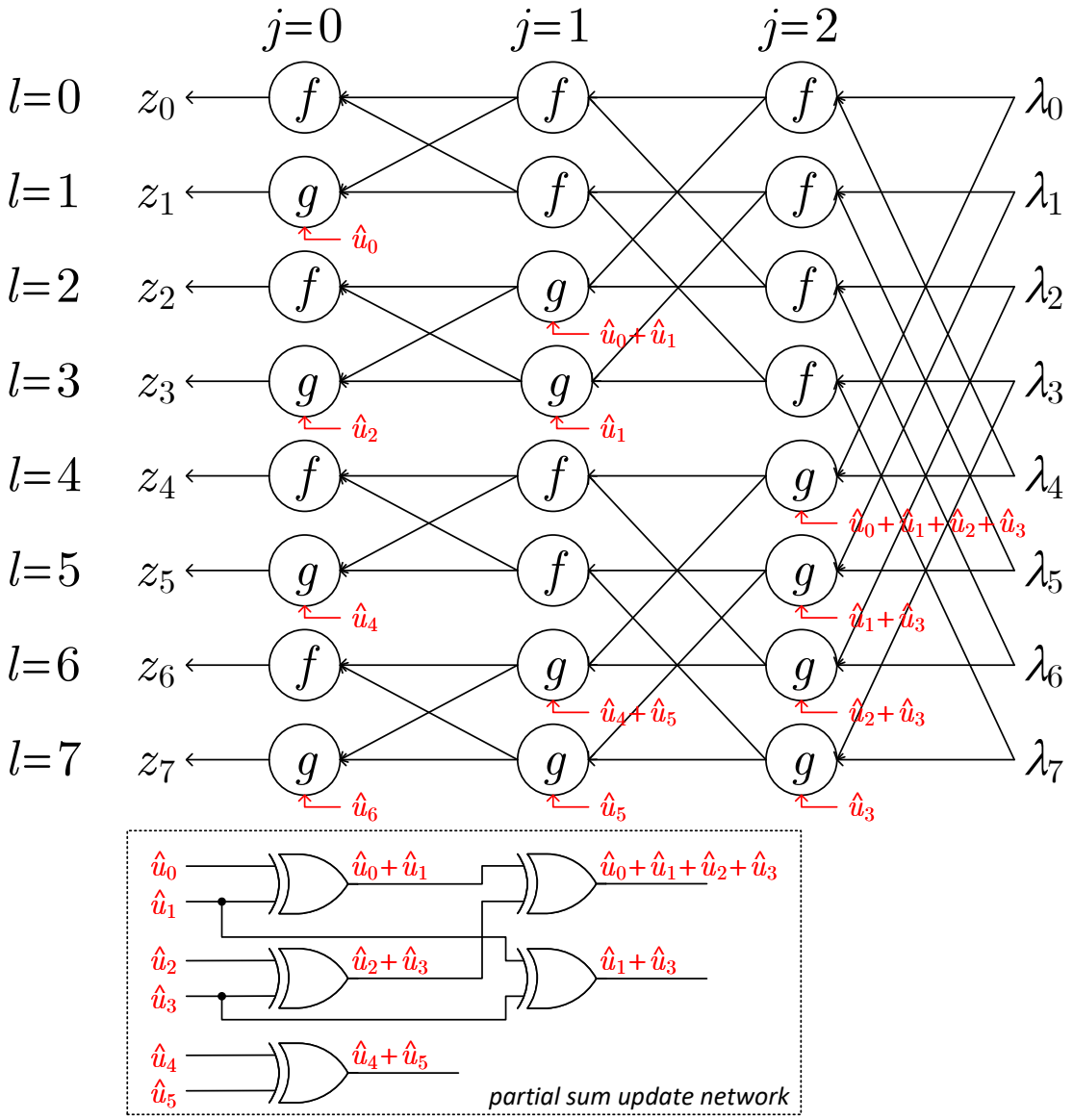

Figure 3.5: The FFT-like polar demapper architecture for $N = 8$.

We refer to a specific process element as $X_{j,l}$, where X represents the type of the process element (either f or g), and j and l denote the stage index and node index within a stage, respectively, for $(0 \leq j < n)$ and $(0 \leq l < N)$. To implement this polar demapper, $n \times N$ process elements are required. The number of XOR gates required to implement the combinational partial sum update network can be obtained as

$$\begin{aligned}
&(\frac{N}{2} - 1) + 2(\frac{N}{4} - 1) + 4(\frac{N}{8} - 1) + \cdots + \frac{N}{2}(\frac{N}{N} - 1) \\
&= \sum_{i=1}^{n-1} \frac{N}{2} - \sum_{i=0}^{n-2} 2^i = (n-1)\frac{N}{2} - \frac{N}{2} + 1 \\
&= (n-2)\frac{N}{2} + 1.
\end{aligned}$$

Despite the similarity in architecture, the timing schedule of polar demapper is different from that of the SC decoder in the sense that the polar demapper generates the values z_i one at a time. That is, once the LLR value of a node z_i has been generated, it remains idle until another node LLR value is requested. It should be noted that the next LLR value request may be for the immediate forward node or any other backward node. Hence, the polar demapper must be able to follow the Fano algorithm whenever it backtracks. To fulfill this requirement, all the intermediate LLR values must be stored and retained until the end of each decoding session. As a result, every time a new z_i value is requested, the polar demapper does not have to start from scratch. The FFT-like architecture of [60] uses distributed registers to stores the intermediate LLR values. Any SC decoder capable of storing the intermediate LLR values can be used as a polar demapper.

To understand the timing schedule of polar demapper, consider the simplified architecture of the polar demapper of Figure 3.6 which is obtained by grouping the parallel operations of Figure 3.5. To generate the LLR value z_i for the ith node, the block $X_{j,l}$ is activated, where X denotes the type of operation (either f or g), j denotes the stage index which is determined by $\text{ffs}^*(i)$ (find first set operation), and l denotes the ID number of the block determined by $l = \frac{i}{2^j}$. The

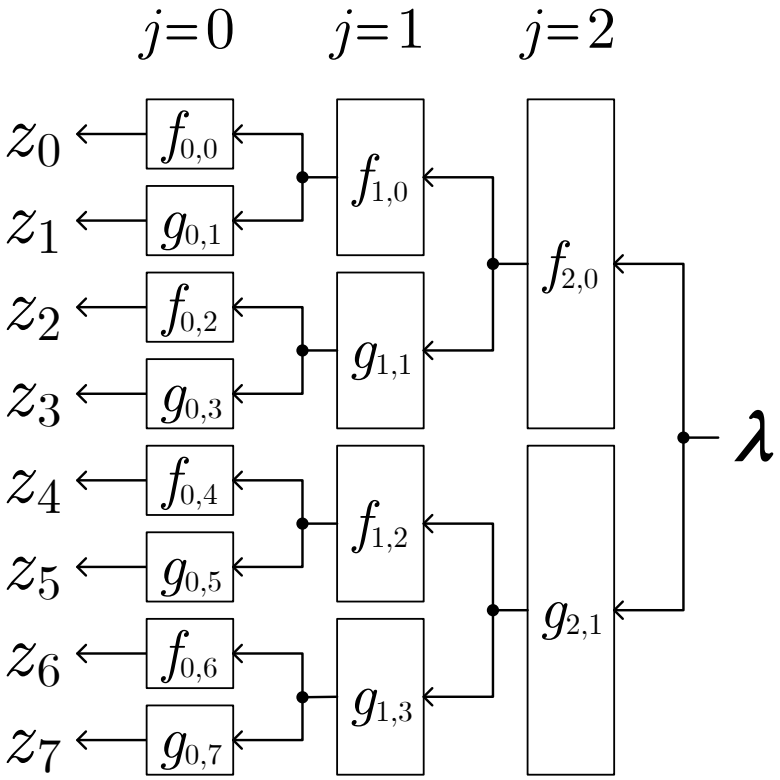

Figure 3.6: The FFT-like polar demapper simplified architecture for $N = 8$.

ffs* operation is defined as

$$\text{ffs}^*(i_{k-1}\cdots i_1 i_0) = \begin{cases} \min(j) : i_j = 1, & \text{if } i > 0, \\ k-1, & \text{if } i = 0, \end{cases} \tag{3.2}$$

where $i_{k-1}\cdots i_1 i_0$ is the binary representation of i. The control signals of blocks are connected in a way that once the output of a block is ready, it immediately activates its following block until reaching the stage $j = 0$ (the final stage). Upon reaching the final stage, the polar demapper stops and waits for the next request.

The hardware complexity of polar demapper can be expressed in terms of total number of comparators, adders, and subtractors used in the implementation. First, we estimate the number of comparators. Comparators are used in implementing the function f in (2.18). We define c_N as the total number of comparators used in a polar demapper of size N. From Figure 3.5 we realize that the number of f functions used at each stage of decoding is $N/2$ and there are $\log N$ stages in total. As a result, the total number of comparators used for implementing the polar demapper of size N is $c_N = (N/2)\log N$. Next, we estimate the number of adders and subtractors. Let a_N denote the total number of adders and subtractors used in a polar demapper of size N. Adders and subtractors are used in implementing the function g in (2.19). Since the total number of g functions used is the same as the total number of f functions and each g block uses one adder and one subtractor, the total number of adders and subtractors can be expressed as $a_N = N\log N$. Thus, the total number of basic logic blocks with similar complexities used for implementing the polar demapper is given by

$$c_N + a_N = \frac{3}{2} N \log N \tag{3.3}$$

which shows that the complexity of the polar demapper is $\mathcal{O}(N\log N)$.

We approximately calculate the delay of polar demapper using combinational logic delays of its components. First we calculate the propagation delay of partial sum update network. From Figure 3.5, the critical path of the partial sum update network is the path between either of the inputs $\hat{u}_i$ for $i = 0, 2, 4, \ldots, (N/2)-2$ and the process element $g_{n-1,N/2}$, which is activated when the Fano decoder request $z_{N/2}$. Let δ_x be the propagation delay of an XOR gate. Then, the worst-case

propagation delay of partial sum update network can be expressed as $(n-2)\delta_x$ or $(\log N - 2)\delta_x$ If we denote the propagation delays of f and g block by δ_f, δ_g, respectively, we can express the worst-case combinational delay of the polar demapper by $\max\{\delta_f, \delta_g, (\log N - 2)\delta_x\}$.

3.2.2 Branch Metric Unit

The BMU block is a fundamental block that distinguishes the Fano decoding of PAC codes from that of convolutional codes. This block uses the Fano metric of (2.31) to provide the Fano control unit with the current branch metric M23 and previous branch metric M1. The branch metric function of (2.31) is too complex to be implemented using simple logical gates as it contains log and exponential operations. To implement the exact branch metric functions, it is required to store the metric values of (2.31) for every possible input value of z_i. If we denote the number of quantization bits for the LLR values by Q, a memory of size $2^Q \times Q$ is required.

To obtain a hardware-friendly version of (2.31), we use the following approximation [61]

$$\log(1 + e^{-(1-2\hat{u}_i)z_i}) \approx \begin{cases} 0, & \text{if } \hat{u}_i = s(z_i), \\ |z_i|, & \text{otherwise,} \end{cases} \tag{3.4}$$

where $s(z)$ is a sign function such that

$$s(z) = \begin{cases} 0, & \text{if } z \geq 0, \\ 1, & \text{otherwise.} \end{cases} \tag{3.5}$$

By applying the approximation of (3.4) to the metric function of (2.31) we can obtain

$$\gamma_i(\hat{u}_i) = \begin{cases} 1 - b_i, & \text{if } \hat{u}_i = s(z_i), \\ 1 - |z_i| - b_i, & \text{otherwise.} \end{cases} \tag{3.6}$$

To simplify (3.6) further, we assume the bias term b_i can take only binary values, i.e. $b_i \in \{0, 1\}$. As a result, we can tabulate $\gamma_i(0)$ and $\gamma_i(1)$ for all the possible combinations of b_i and $s(z_i)$ in Table 3.3.

Table 3.3: $\gamma_i(\hat{u}_i)$ for Different Values of $s(z_i)$ and b_i.

$s(z_i)$	b_i	$\gamma_i(0)$	$\gamma_i(1)$
0	0	1	$1 - \lvert z_i \rvert$
0	1	0	$-\lvert z_i \rvert$
1	0	$1 - \lvert z_i \rvert$	1
1	1	$-\lvert z_i \rvert$	0

If we represent the LLR values z_i in the form of sign-magnitude, the term $s(z_i)$ corresponds to the sign bit of z_i. As a result, we can implement this table using two 4-to-1 multiplexers and one adder. Fig. 3.7 shows the hardware implementation of Table 3.3 (metric calculator). The number of quantization bits for LLRs is denoted by Q. The metric calculator receives z_i and b_i and generates the branch metric $\gamma_i(\hat{u}_i)$ for the two possible values of $\hat{u}_i = 0$ and $\hat{u}_i = 1$. The constant 0 and 1 inputs to the adder and multiplexers are padded with zeros to have Q-bit width (not shown in the figure for clarity).

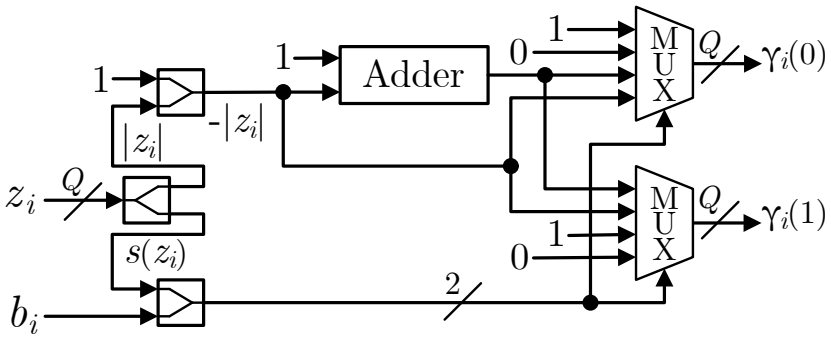

Figure 3.7: Hardware implementation of Table 3.3 (metric calculator).

Note that $\gamma_i(0)$ and $\gamma_i(1)$ are the two tentative branch metric values from which only one must be selected depending on the value of convolution output estimate $\hat{u}_i$ and whether the Fano decoder is examining the most likely node or least likely node. Let $\hat{u}_{i,0}$ and $\hat{u}_{i,1}$ be the convolution output estimates for the two possible successor branches $\hat{v}_i = 0$ and $\hat{v}_i = 1$, respectively. Due to the fact that rate-one convolution is a one-to-one operation, if $\hat{u}_{i,0}$ is 0, then $\hat{u}_{i,1}$ must be

1; and if $\hat{u}_{i,0}$ is 1, then $\hat{u}_{i,1}$ must be 0. In other words, the value of $\hat{u}_{i,1}$ is always the complement $\hat{u}_{i,0}$ and vice versa. As a result, a single convolution encoder is enough for obtaining $\hat{u}_{i,0}$ and $\hat{u}_{i,1}$.

With a careful observation of Table 3.3 we realize that $\gamma_i(0) \geq \gamma_i(1)$ when $s(z_i) = 0$ and $\gamma_i(0) \leq \gamma_i(1)$ when $s(z_i) = 1$. Hence, the most likely branch can be distinguished from the least likely branch without using an actual comparator. Recalling the set of rules listed in Table 3.1, we use a parameter t_i such that the Fano algorithm examines the most likely branch when $t_i = 0$ and the least likely branch when $t_i = 1$. This can be accomplished by passing the most likely branch metric when $t_i = 0$ or the least likely branch metric when $t_i = 1$ to the Fano control unit. Table 3.4 lists the current branch metric (M23) for all the possible combinations of $s(z_i)$ and t_i which can be implemented using an XOR gate and a multiplexer that selects $\gamma_i(0)$ when $s(z_i) = t_i$ and selects $\gamma_i(1)$ when $s(z_i) \neq t_i$.

Table 3.4: M23 for Different Values of $s(z_i)$ and t_i.

$s(z_i)$	t_i	M23
0	0	$\gamma_i(0)$
0	1	$\gamma_i(1)$
1	0	$\gamma_i(1)$
1	1	$\gamma_i(0)$

Figure3.8 shows the hardware diagram of BMU, which uses two metric calculator blocks to generate the current and previous branch metrics M1 and M23, respectively. We use a convolutional encoder replica to generate $\hat{u}_{i,0}$ which is the convolution output for the assumption $\hat{v}_i = 0$. The input t_i is provided by FCU and is used to request the most likely branch metric (M2) when $t_i = 0$ or the least likely branch metric (M3) when $t_i = 1$ from BMU. As explained before, this signal is XORed with $s(z_i)$ to select the metric corresponding to the branch which is currently being explored by the the Fano algorithm. Additionally, when the current node N1 is frozen (i.e. $a_i = 0$) the BMU is forced to output the branch metric which corresponds to $\hat{v}_i = 0$. We implement this using a multiplexer that assigns $\hat{u}_{i,0}$ to $\hat{u}_i$ when $a_i = 0$ and forces the BMU to output $\gamma_i(0)$. Otherwise, it assign the output of $(s(z_i) \oplus t_i)$ to $\hat{u}_i$. In addition to M1 and M23 metrics, the

BMU block provides FCU with the selected branch $\hat{v}_i$ and its corresponding convolution output $\hat{u}_i$. To generate $\hat{v}_i$, we compare $\hat{u}_i$ with $\hat{u}_{i,0}$ using an XOR gate. Recalling that $\hat{u}_{i,0}$ corresponds to the the convolution output for the assumption $\hat{v}_i = 0$, if the compared two signals are equal, then the decision $\hat{u}_i$ corresponds to $\hat{v}_i = 0$; otherwise, $\hat{v}_i = 1$. To generate the previous branch metrics (M1) the $\hat{u}_{i-1}$ is provided to the BMU block by the Ureg register. This signal is used to select the corresponding branch metric from the output of metric calculator block. The z_{i-1} and z_i input LLRs are provided by the polar demapper and the convolution state (CS) is provided by the Vreg register.

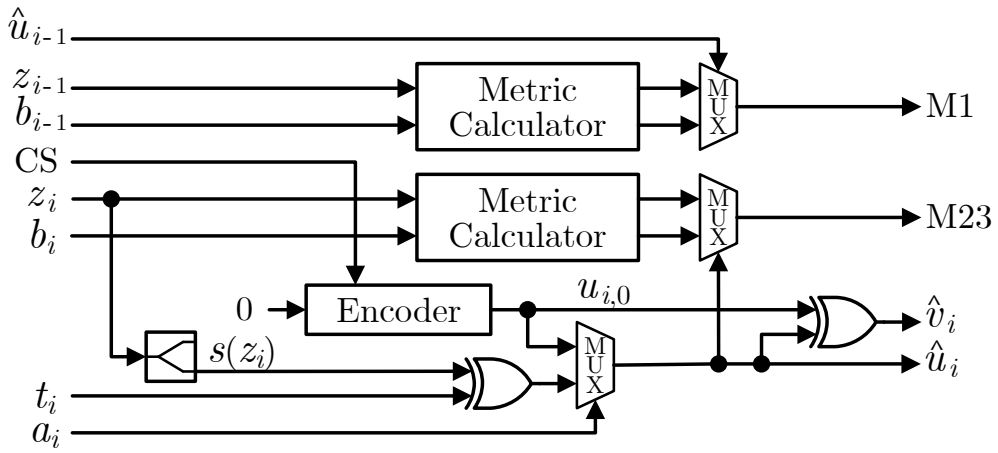

Figure 3.8: Hardware diagram of branch metric unit (BMU).

By observing Figure 3.7 and Figure 3.8, we can conclude that the hardware complexity of the BMU is independent of block-length N. Hence, the hardware complexity of BMU is $\mathcal{O}(1)$. The critical path of the proposed BMU is the path from either of input LLRs z_i/z_{i-1} to their corresponding output metric values M1/M23. Let us express the propagation delays of multiplexer and adder by δ_m and δ_a, respectively. One multiplexer is used to select z_i/z_{i-1} from the output of polar demapper, one adder and one multiplexer is used inside the metric calculator block, and one multiplexer is used to select the desired branch metric from the output of metric calculator. Hence, we can express the combinational delay of BMU by $3\delta_m + \delta_a$.

3.2.3 Fano Control Unit

The Fano control unit is the core block of the proposed PAC Fano decoder. This block is responsible for controlling the flow of Fano algorithm by checking the conditions of Table 3.1 and executing the corresponding actions. This block generates the control signals to activate the polar demapper and provides the necessary inputs to the BMU. By examining the rule sets in Table 3.1, we discover that the Fano control unit must check a total of six cases to determine which rule's conditions are satisfied. We denote these conditions by C0-C5 and list them in Table 3.5.

Table 3.5: Logical expressions used in the Fano rule set.

Case	Expression
C0	$\Psi = 1$
C1	$\mathrm{M23} \geq T$
C2	$0 < T + \Delta \leq \mathrm{M23}$
C3	$i = 0$
C4	$\mathrm{M1} + T \leq 0$
C5	$a_{i-1} = 1 \mid t_{i-1} = 1$

Figure 3.9 shows the circuit diagram for generating the signals of Table 3.5. Since the parameter Ψ is a binary variable, the signal C0 corresponds to this parameter and does not require any extra circuit to be generated. On the other hand, the signal i is a vector signal of width $n = \log N$ and to check the condition $i = 0$, it is required to check if all the bits of i are zero. We implement this condition using an n-input NOR gate.

Using these binary signals, we can implement the circuitry for detecting the rule whose conditions are being satisfied. Figure 3.10 demonstrates the circuitry for generating the rule signals of the PAC Fano decoder of Table 3.1. Here, the output signal Rx corresponds to the rule x for $x = 0, 1, ..., 4$. The input signals to these circuits are generated by the circuits of Figure 3.9.

The architecture of the Fano control unit is shown in Figure 3.11. The key part of Fano control unit is the control logic (CL) that governs which of the Fano rules is executed at the next step. This block also performs the threshold T and node

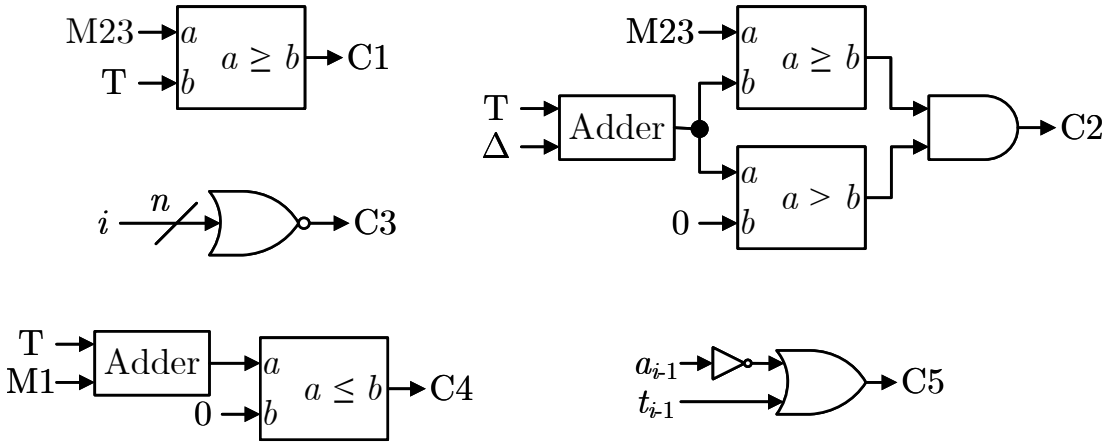

Figure 3.9: The circuit for generating the condition signals of Fano algorithm.

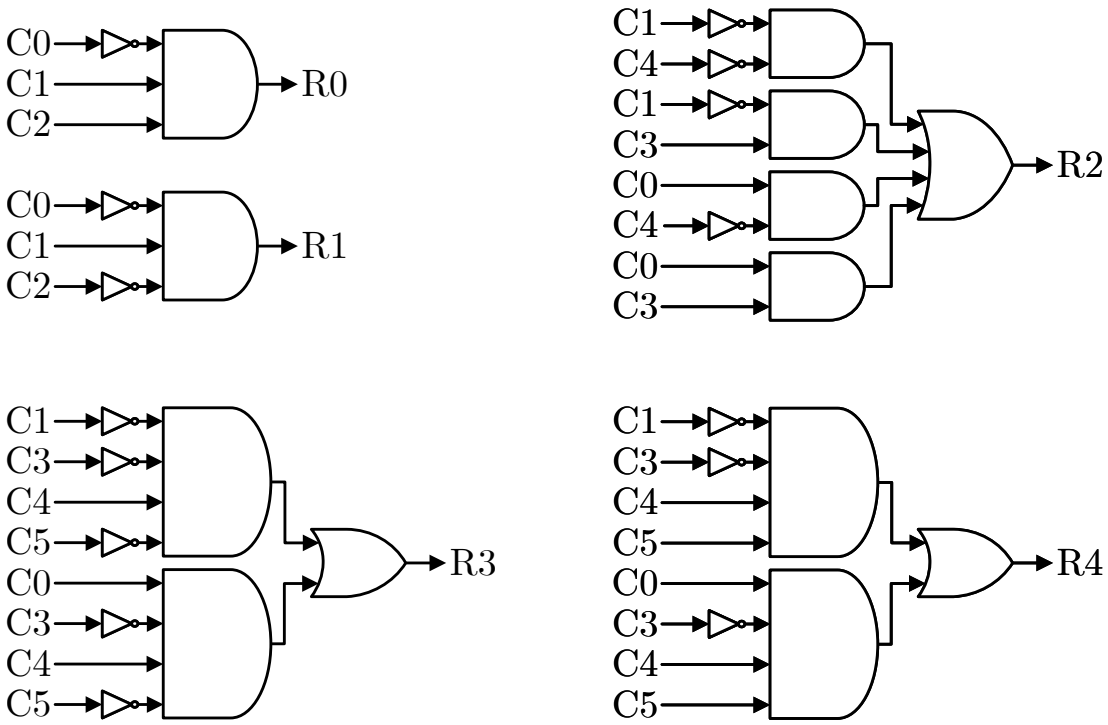

Figure 3.10: The circuit for generating the rule signals of Fano algorithm.

index i updates. The Vreg and Ureg registers are also updated by this module. Every time a forward move is performed, the Vreg is shifted to right, the estimate $\hat{v}_i$ is stored in the MSB of Vreg, and the estimate $\hat{u}_i$ is stored in the ith index of Ureg. On the other hand, whenever a backward move is performed, the Vreg is shifted to left. Activating the polar demapper when PDE = 1 is also the task of CL block.

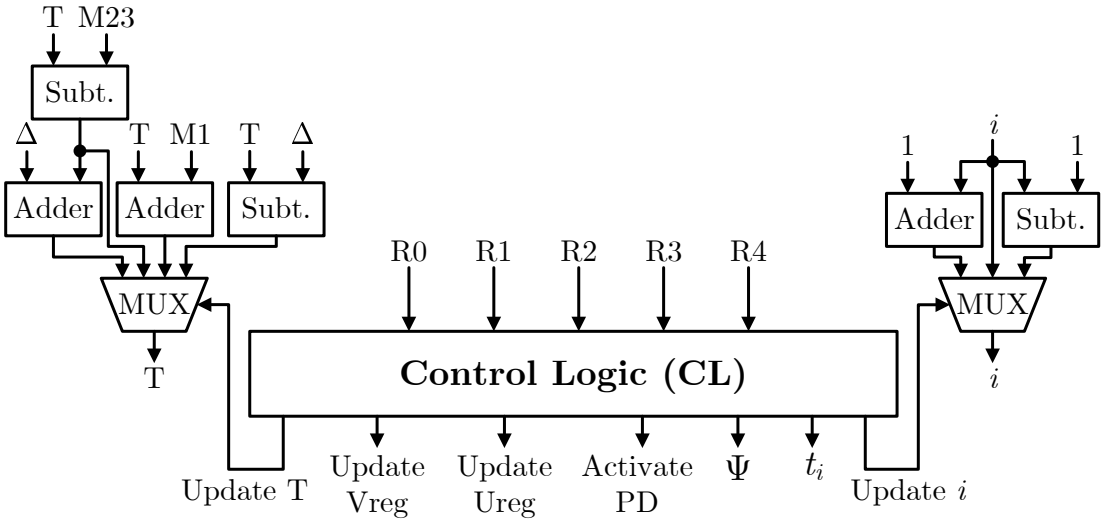

Figure 3.11: Fano control unit (FCU).

By observing Figure 3.9, Figure 3.10, and Figure 3.11, we can conclude that the hardware complexity of the Fano control unit is independent of block-length N. Hence, the hardware complexity of Fano control unit is $\mathcal{O}(1)$. The critical path of the Fano control unit is the path between the T input of C2 generating circuit and R1 output of the rule generating circuit. This path includes one adder, one comparator, one NOT gate, and one OR gate. If we denote the propagation delays of adder, comparator, NOT gate, and OR gate by δ_a, δ_c, δ_n, and δ_o, respectively, the combinational delay of Fano control unit can be expressed as $(\delta_a + \delta_c + \delta_n + \delta_o)$.

Chapter 4

Implementation Results

In this chapter, we present the FPGA and ASIC implementation results of the proposed PAC Fano decoder for block length $N = 128$ and message length $K = 64$. For both FPGA and ASIC implementation we use the following parameters. For the connection polynomial of convolution operation we use $\mathbf{c} = (1, 0, 1, 1, 0, 1, 1)$ or $\mathbf{c} = 131$ in octet representation. We choose the data index set $\mathcal{A}$ according to the Reed-Muller scoring rule as explained in [15]. As for the bias vector $\mathbf{b}$, we use the hard quantized (1-bit quantization) values of bit-channel capacities [47]. For number quantization bits Q and threshold spacing Δ we use $Q = 7$, $\Delta = 2$, respectively. The impact of Q and Δ on error-correction performance and computational complexity of the proposed PAC Fano decoder is investigated in Appendix A and Appendix B, respectively. We calculate the channel LLR values $\boldsymbol{\lambda}$ using a noise variance that corresponds to an SNR of $E_b/N_0 = 3.5$ dB.

4.1 FPGA Implementation Results

The proposed PAC Fano decoder is successfully implemented onto Xilinx Nexys 4 Artix®-7 (28 nm) FPGA. At 100 MHz clock frequency, the place-and-route results

show that the decoder uses 16443 lookup tables (LUTs) and 8306 registers. To evaluate the FER performance and measure the search complexity of the PAC Fano decoder, we use the test setup shown in Figure 4.1. At each iteration, a pseudorandom message $\mathbf{d}$ of length K is generated using MATLAB® software. Then the message $\mathbf{d}$ is encoded into a codeword $\mathbf{x}$ using a software implemented PAC encoder. After that, the codeword $\mathbf{x}$ is modulated using a BPSK modulator. To mimic an AWGN channel, the modulated signal is added with additive white Gaussian noise whose variance corresponds to the SNR point for which the test is being performed. The channel output $\mathbf{y}$ is then sent to the FPGA through the serial communication port. The PAC Fano decoder on the FPGA decodes the received data and transmits back the estimate of carrier word $\hat{\mathbf{v}}$ to the computer through the serial communication port. Also transmitted by the FPGA is the number of clock cycles consumed to decode each codeword which is measured by the CC counter. At the computer side, the $\hat{\mathbf{d}}$ is extracted from $\hat{\mathbf{v}}$ using $\hat{\mathbf{d}} = \hat{\mathbf{v}}_{\mathcal{A}}$, and compared with the actual transmitted message $\mathbf{d}$. The error correction performance of the decoder is measured by the probability of frame error $P_e = P(\hat{\mathbf{d}} \neq \mathbf{d})$.

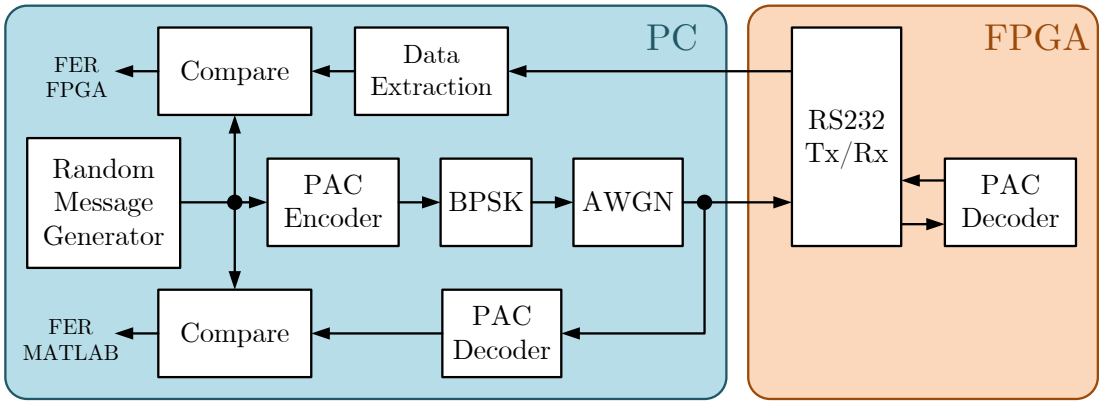

Figure 4.1: FPGA test setup.

Figure 4.2 plots the FER performance of the proposed PAC Fano decoder for different values of maximum clock cycles (MC) and compares them with the FER performance of software simulation of original PAC codes reported in [15]. We also plot the dispersion approximation with $N = 128$ and $K = 64$ to compare the FER performance of our decoder with the theoretical limit. As expected, increasing the value of MC allows the Fano algorithm to perform more searches

and maintain better FER performance. With MC = 2^{18}, the proposed PAC Fano decoder obtains a FER performance close to the FER performance of software implementation at high SNR regime; At $E_b/N_0 = 3.5$ dB the decoder achieves a FER performance of FER = 1.6×10^{-5}. It is worth mentioning that for MC values greater than 2^{18} the decoder maintains same FER performance. The performance loss is majorly due to the quantization of LLRs and approximation of the Fano metric. In low SNR regime, the metric approximation error is large since the term $\log(1 + e^{-(1-2\hat{u}_i)z_i})$ diverges from $|z_i|$ as SNR decreases. But, as SNR increases, this error becomes negligible.

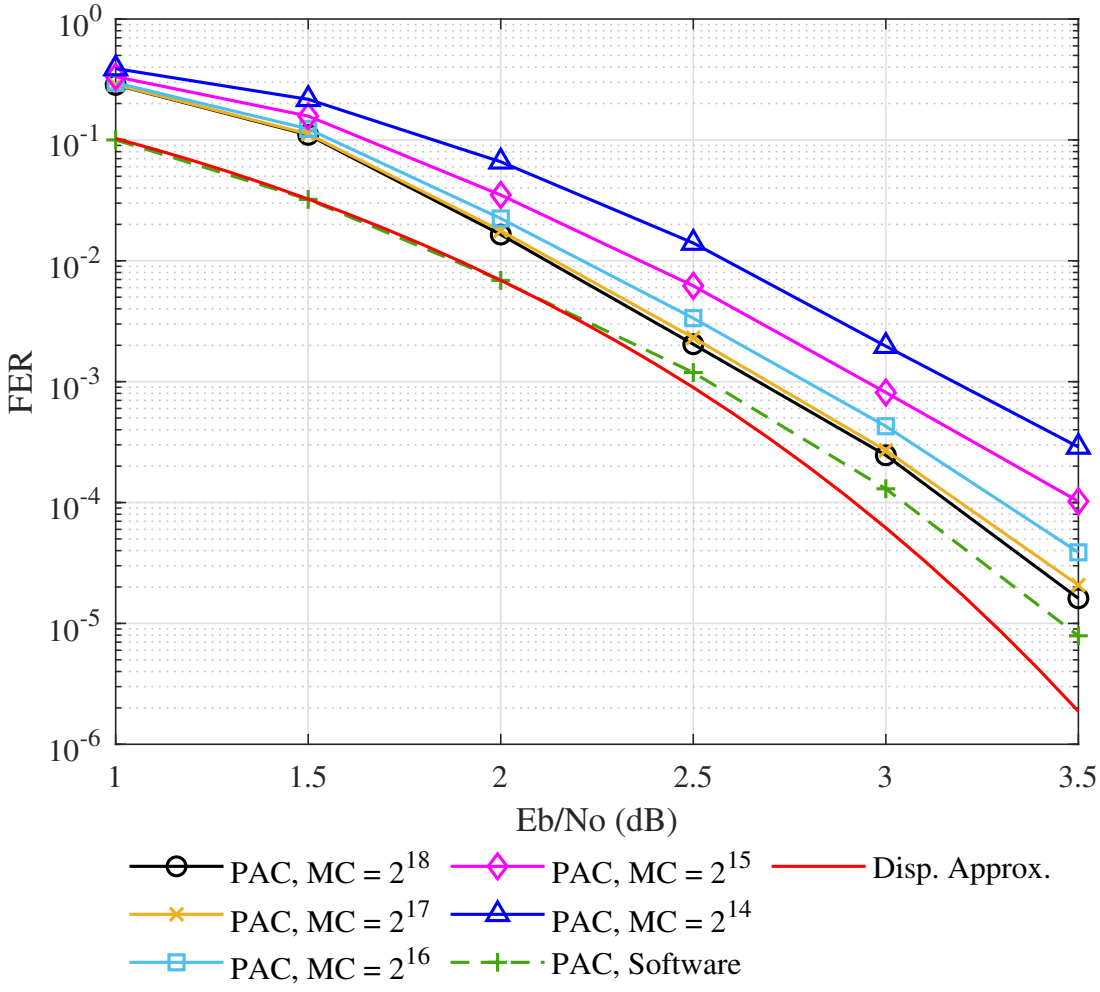

Figure 4.2: FER performance of FPGA implementation of PAC Fano decoder.

Figure 4.3 shows the average number of clock cycles consumed by the proposed PAC Fano decoder for decoding a single codeword with different values of MC.

From this figure we realize that the effect of MC value on average number of clock cycles required for decoding a codeword is significant at low SNR regime but as SNR increases, this effect fades out. This is due to the Pareto distribution of Fano decoder's search complexity such that for high SNR values, only a small fraction of codewords require a very large search complexity [16]. At $E_b/N_0 = 1$ dB, using MC $= 2^{14}$ reduces the average number of clock cycles per codeword by 51% compared to using MC $= 2^{18}$. Obviously, as shown in Figure 4.2, this complexity reduction is achieved at a cost of FER performance drop, especially at high SNR values (significant at $E_b/N_0 = 3.5$ dB).

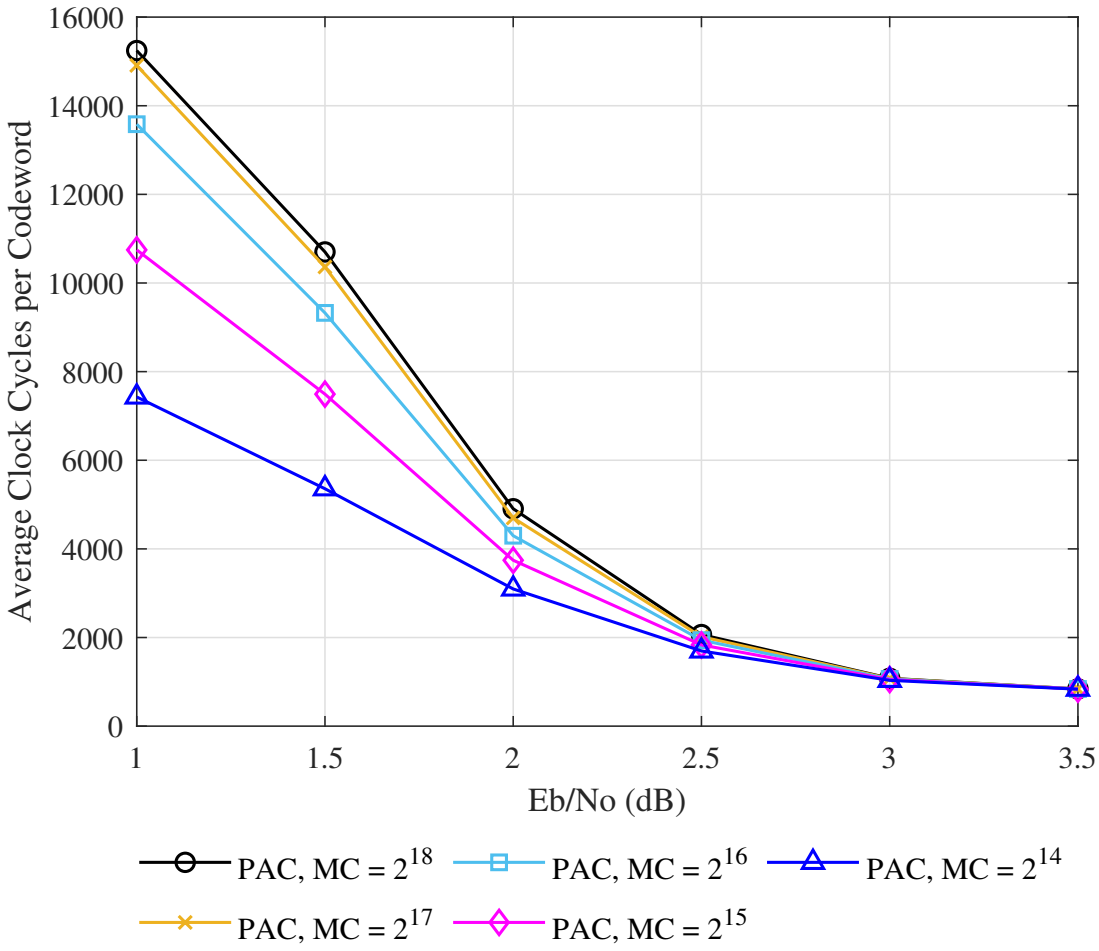

Figure 4.3: Time complexity of FPGA implementation of PAC Fano decoder.

It is worth mentioning that the PAC Fano decoder does not reach the iteration bound for MC values of 2^{18} and higher. At $E_b/N_0 = 3.5$ dB, regardless of the value of MC, the proposed decoder consumes an average of approximately 840

clock cycles to decode a single codeword. Operating at 100 MHz frequency, the decoder reaches an average information throughput of 7.6 Mb/s with an average latency of 8.39 μs at $E_b/N_0 = 3.5$ dB.

It is worth to investigate the average time complexity of the polar demapper. Figure 4.4 demonstrates the relative complexity of the polar demapper in terms of the portion of total average clock cycles that is consumed by the polar demapper. At $E_b/N_0 = 1$ dB, approximately 25% of the total clock cycles is consumed by the polar demapper; as SNR increases, the polar demapper's share increases as well, to the extent that at $E_b/N_0 = 3.5$ dB, the polar demapper accounts for more than one-third of the total time complexity. This is due to the fact, for higher SNR values, the Fano algorithm mostly performs forward search and, as a result, the polar demapper is activated more frequently.

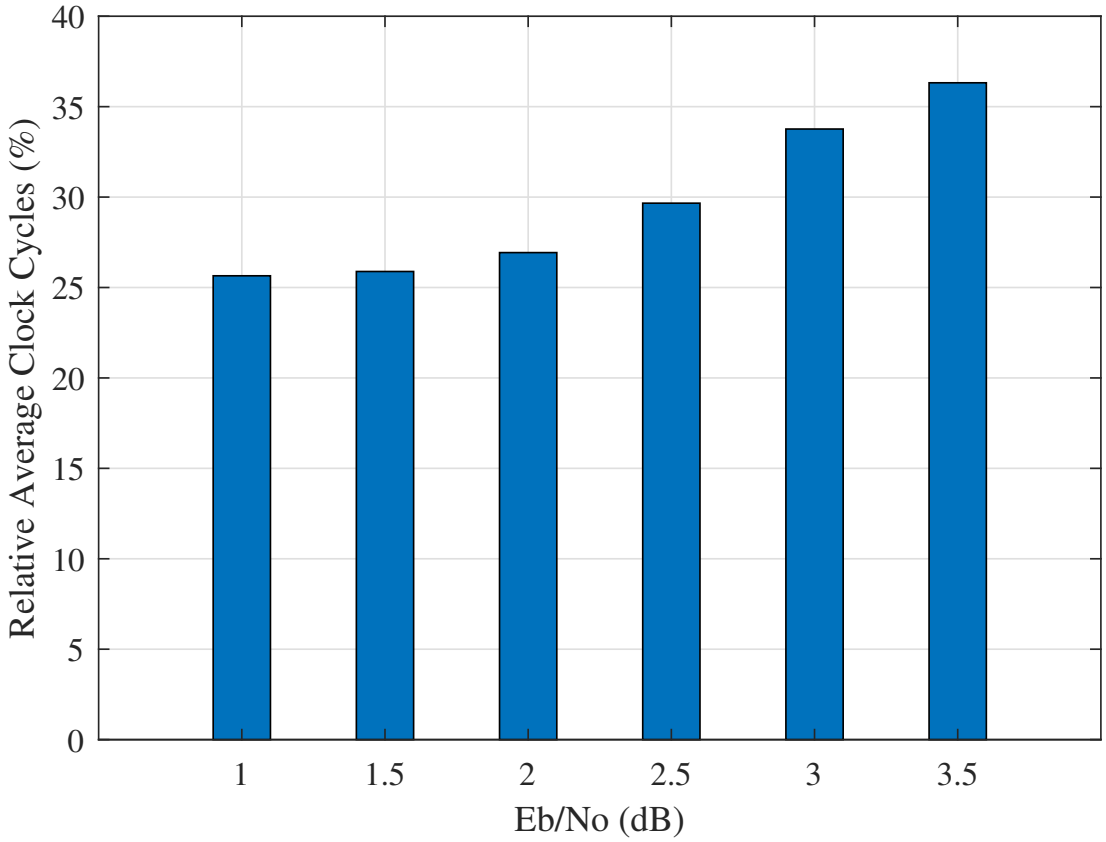

Figure 4.4: Relative time complexity of the polar demapper.

To justify the latter statement, we refer to Figure 4.5, which plots the average

relative frequency of the Fano rules executed by the Fano algorithm for various SNR values. At $E_b/N_0 = 1$ dB, roughly half of the executed actions correspond to forward moves (Rule 0 and Rule 1), and nearly all of the remaining actions correspond to backtracking (Rule 3 and Rule 4). However, at $E_b/N_0 = 3.5$ dB, approximately 85% of the executed actions belong to forward moves and only less than 13% of the total actions correspond to backtracking. As SNR decreases, the severity of the noise increases and the probability of the Fano algorithm identifying the correct path decreases which results in many incorrect branch selections and long backtracking for low SNR values.

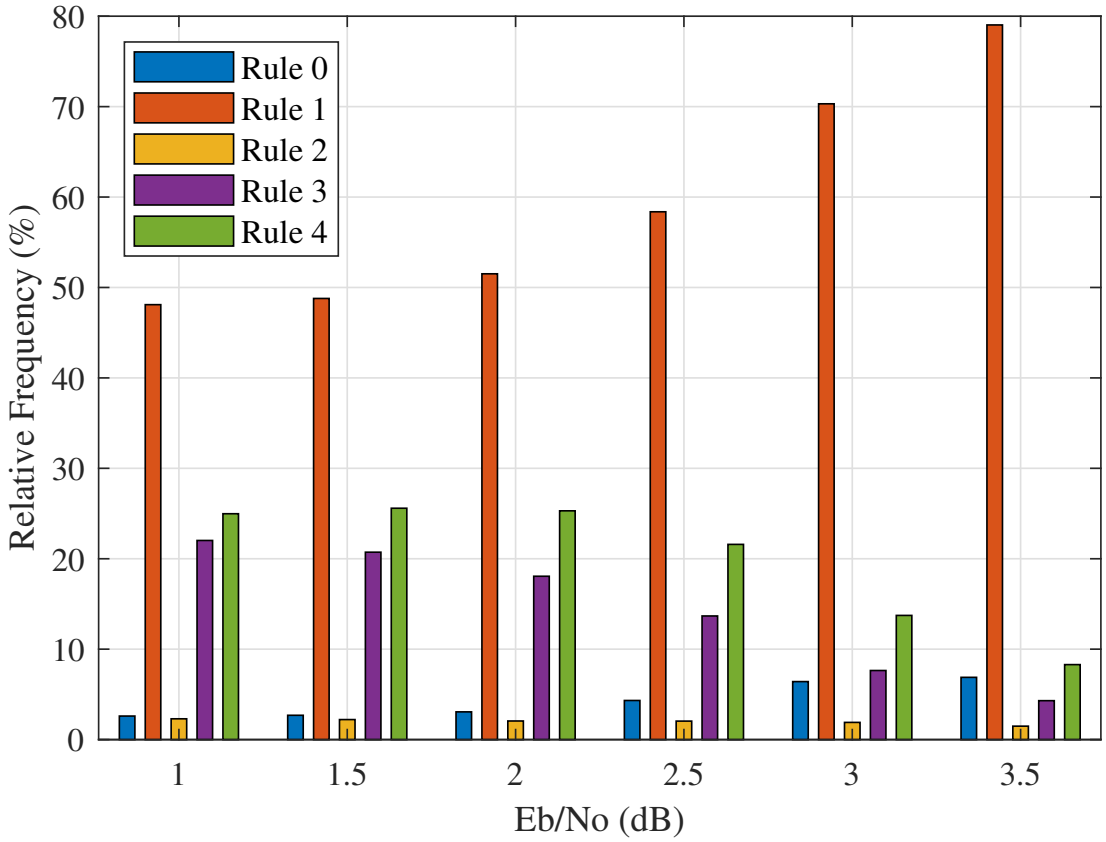

Figure 4.5: Relative frequency of the rules executed by the Fano algorithm.

4.2 Post-Synthesis Results

We implement the proposed PAC Fano decoder using Cadence® Innovus™ Implementation System with TSMC 28 nm 0.72 V library. Table 4.1 lists the post-synthesis results of the proposed PAC Fano decoder. We present the results for the PAC Fano decoder with MC = 2^{18}. The decoder occupies an area of 0.059 mm^2 from which approximately 80%, 16%, and 1% are occupied by polar demapper, Fano control unit, and branch metric unit, respectively. The power value is estimated with Cadence® Voltus™ IC Power Integrity Solution using 10^4 pseudorandom input vectors. Operating at 500 MHz clock frequency, the proposed decoder consumes 3.85 mW power.

The performance values in Table 4.1 are reported at $E_b/N_0 = 3.5$ dB, and the average values are calculated from 10^7 decoding trials. The average information throughput (TP) of the decoder is estimated by

$$\text{TP [bit/s]} = \frac{f_{\text{clk}}[\text{cycle/s}]}{\text{ACC [cycle/bit]}} \times K.$$

where f_{clk} is the operating frequency and ACC is the average number of clock cycles consumed for decoding a single frame which is obtained from Fig. 4.3. The proposed PAC Fano decoder reaches an average information throughput of 38.1 Mb/s with an average latency of 1.68 μs and worst-case latency of 526 μs. The worst-case latency and information throughput of the decoder are determined by the value of MC = 2^{18}. On the other hand, the best-case latency and information throughput are obtained at higher SNR points where the Fano algorithm performs no backtracking. In this case, the proposed PAC Fano decoder consumes 638 clock cycles to decode each codeword. This corresponds to $5N-2$ clock cycles, of which $2N-2$, $2N$, and N is consumed by the polar demapper, Fano control unit, and branch metric unit, respectively.

The low power consumption of the decoder is because a large chunk of the logic (especially a significant portion of the polar demapper) is inactive at any given time. The polar demapper architecture uses $N \log N$ process elements (f and g blocks) from which a large fraction is inactive during the decoding. The activation

Table 4.1: ASIC Implementation Results

Technology	28 nm
K	64
N	128
Supply Voltage (V)	0.72
Frequency (MHz)	500
Area (mm^2)	0.059
Power (mW)	3.85
Average Information TP† (Mb/s)	38.1
Best-Case Information TP (Mb/s)	50.16
Worst-Case Information TP (Mb/s)	0.12
Average Latency† (CC)	838
Average Latency† (μs)	1.68
Best-Case Latency (CC)	638
Best-Case Latency (μs)	1.28
Worst-Case Latency (CC)	2^{18}
Worst-Case Latency (μs)	524
Area Efficiency† (Gb/s/mm^2)	0.646
Power Density (W/mm^2)	0.065
Energy Efficiency† (pJ/bit)	101

†Average value at $E_b/N_0 = 3.5$ dB.

of largest number of process elements occurs when the Fano control unit requests z_0 or $z_{N/2}$. In this case, all the stages of polar demapper are activated and at each state j, 2^j number of process elements are activated for $j = 0, \cdots, \log N - 1$. Hence, each time the polar demapper is activated, at most $\frac{2^n-1}{Nn}$ (or $\frac{N-1}{Nn} \approx \frac{1}{n}$) portion of process elements are activated, where $n = \log N$. For $N = 128$, this corresponds to approximately 14% of the polar demapper logic. Also we would like to emphasize that the polar demapper is idle during the operation of the branch metric unit (BMU) and Fano control unit (FCU). Table 4.2 lists the power consumption of main blocks of PAC Fano decoder. From 3.85 mW total power consumption, 2.58 mW is used by the polar demapper which corresponds to approximately 67% of the total power.

Table 4.2: Power consumption of main blocks of PAC Fano decoder.

Module	Power Consumption (mW)	%
PAC Fano decoder	3.85	100
Polar Demapper	2.58	67.01
FCU	0.67	17.41
Input buffer	0.28	7.27
Vreg	0.05	1.29
Ureg	0.03	0.78
Output buffer	0.03	0.78
BMU	0.01	0.26

4.3 Comparison With Polar Decoders

In this section we compare our proposed PAC Fano decoder against five different decoders of polar codes with block lengths of $N = 128$ and $N = 1024$. Throughout this section we refer to our proposed PAC Fano decoder with a maximum allowed cycle of $\text{MC} = 2^{18}$ as PAC Fano decoder. Kestel *et al.* [26] present high throughput SC and SCL polar codes' decoders that can achieve throughput up to 516 Gbit/s in 28 nm CMOS FD-SOI technology. From the decoders of [26], we choose an SC decoder for $(N = 128, K = 64)$ polar codes, an SCL decoder with a list size of 8 and CRC length of 6 (SCL8-CRC6) for $(128, 70)$ polar codes, and an SCL decoder with a list size of 2 (SCL2) for $(1024, 512)$ polar codes.

Giard *et al.* [27] present a flexible SCL decoder with a list size of 4 (SCL4) for $(1024, 512)$ polar codes fabricated in 28 nm CMOS FD-SOI technology. Park *et al.* [28] present a rate-flexible belief propagation (BP) decoder for $(1024, 512)$ polar codes fabricated in 65 nm CMOS TSMC technology.

Fig. 4.6 compares the FER performance of the proposed PAC Fano decoder against the decoders of polar codes mentioned above. For the same block length (i.e. $N = 128$) the proposed PAC Fano decoder outperforms the SC and SCL8-CRC6 polar decoders. Compared to SC and SCL8-CRC6 polar decoders, the proposed PAC Fano decoder has a coding gain of approximately 2.1 dB and 1.2 dB at FER $= 10^{-5}$, respectively. Comparing PAC codes of length 128 with polar codes of length 1024, the proposed PAC Fano decoder performs better than the BP decoder with a coding gain of approximately 0.6 dB at FER $= 10^{-5}$. The PAC Fano decoder performs close to the SCL2 decoder of $(1024, 512)$ polar codes. However, in case of using a list size of 4, the SCL4 decoder of $(1024, 512)$ polar codes outperforms the proposed PAC Fano decoder of $(128, 64)$ PAC codes with a coding gain of 0.4 dB at FER $= 10^{-5}$.

Table 4.3 shows a comparison of our PAC Fano decoder against the other ASIC decoders of polar codes. Since the results of other decoders are for other technologies or/and different supply voltages, for a fair comparison, we provide the normalized results as well. Also in [26–28] the throughput values are reported in term of coded throughput (not information throughput). For this reason we report the coded throughput of the PAC Fano decoder and calculate the area efficiency and energy efficiency values accordingly. Comparing our PAC Fano decoder with the normalized results of polar decoders, it can be seen from Table 4.3 that the PAC Fano decoder occupies smaller area (slightly smaller than SC decoder) and has significantly lower power consumption. On the other hand, the SCL2 and SC polar decoders of [26] have the highest throughput and the best energy efficiency, respectively. These decoders use various architectural optimizations and are designed for throughputs beyond 100 Gbit/s. It should be noted, however, that the FER performance of the PAC Fano decoder is significantly better than that of the SC decoder. The energy efficiency of the proposed PAC Fano decoder is smaller than the SCL4 decoder of [27] but is significantly larger

than the rest of the polar decoders.

Benefiting from pipelining and unrolling techniques, SC and SCL decoders of polar codes are capable of achieving high throughput values. However, because of the Fano decoder's backtracking feature, the PAC Fano decoder requires to decode one codeword at a time and hence suffers from low throughput. Among all the decoders, the PAC Fano decoder has the lowest throughput. Compared to SC and SCL8-CRC6 decoders of [26], the PAC Fano decoder has lower throughput by a factor of approximately 800 and 660, respectively.

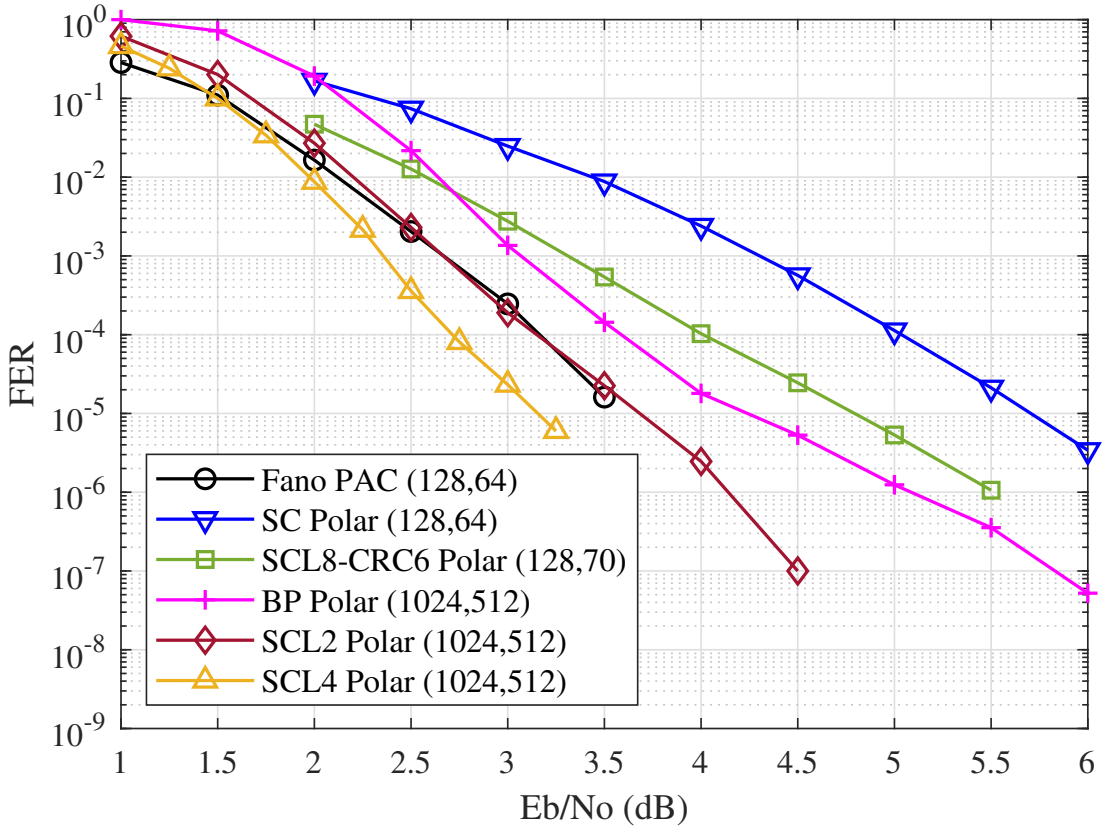

Figure 4.6: FER performance comparison of PAC Fano decoder with decoders of Polar codes.

Table 4.3: Comparison of the PAC Fano Decoder Against the ASIC Decoders of Polar Codes.

Implementation	This work	[26]	[26]	[26]	[27]	[28]
Algorithm	Fano	SC	SCL8-CRC6	SCL2	SCL4	BP (15 iter.)
(N, K)	(128,64)	(128,64)	(128,70)	(1024,512)	(1024,512)	(1024,512)
E_b/N_0 at FER $= 10^{-5}$	3.6 dB	5.7 dB	4.8 dB	3.7 dB	3.2 dB	4.2 dB
Technology	28 nm	28 nm	28 nm	28 nm	28 nm	65 nm
Supply (V)	0.72	1.0	1.0	1.0	1.3	1.0
Frequency (MHz)	500	503	418	503	721	300
Area (mm^2)	0.059	0.07	3.15	7.52	0.44	1.48
Power (mW)	3.85	80	3340	4530	128.3	477.5
Avg. Coded TP (Mb/s)	76.2†	64000	53000	516000	306.8	4675.8$^\diamond$
W.-C. Coded TP (Mb/s)	0.24	64000	53000	516000	306.8	2048
Avg. Latency (μs)	1.68	0.0179	0.1413	0.292	3.34	0.219$^\diamond$
W.-C Latency (μs)	524	0.0179	0.1413	0.292	3.34	0.5
Area Eff. (Gb/s/mm^2)	1.292	931	17	69	0.692	3.168$^\diamond$
Energy Eff. (pJ/bit)	50.5	1.28	62.46	8.79	418.3	102.1$^\diamond$
Normalized for 0.72 V and 28 nm using the scaling method in [62, 63]						
Frequency (MHz)	500	503	418	503	721	696
Area (mm^2)	0.059	0.07	3.15	7.52	0.44	0.275
Power (mW)	3.85	41.47	1731.46	2348.35	39.35	106.63
Avg. Coded TP (Mb/s)	76.2†	64000	53000	516000	306.8	10854.54$^\diamond$
W.-C. Coded TP (Mb/s)	0.24	64000	53000	516000	306.8	4754.29
Avg. Latency (μs)	1.68	0.0179	0.1413	0.292	3.34	0.0943$^\diamond$
W.-C Latency (μs)	524	0.0179	0.1413	0.292	3.34	0.2154
Area Eff. (Gb/s/mm^2)	1.292	931	17	69	0.692	39.632$^\diamond$
Energy Eff. (pJ/bit)	50.5	0.663	32.379	8.79	128.312	9.821$^\diamond$

†Average value at $E_b/N_0 = 3.5$ dB.

$^\diamond$Average value at $E_b/N_0 = 4$ dB with early termination and an average number of iterations of 6.57.

Chapter 5

Conclusion

In this thesis, we proposed a hardware architecture for Fano decoding of PAC codes. First, we introduced a hardware-friendly variant of the Fano algorithm that is suitable for sequential decoding of PAC codes. Then, we discussed the hardware implementation of the proposed PAC Fano algorithm. The proposed design consist of three main blocks: polar demapper, branch metric unit, and Fano control unit. To design the polar demapper we modified the architecture of SC decoder of [60]. We obtained a simplified branch metric function for sequential decoding of PAC codes and presented a novel branch metric unit that can be implemented with simple logic gates. The presented branch metric unit is capable of calculating the current and previous branch metric values online, without requiring any storage element or comparator. We showed that the Fano control unit, which implements the introduced Fano algorithm, can be implemented using simple logic gates. We provided estimates for the hardware complexity and combinational delays of the sub-block of PAC Fano decoder.

Table 5.1 summarises ASIC implementation results of the PAC Fano decoder and compares them with those of the SC and SCL decoders of polar codes with $N = 128$ which are reported in [26]. Post-synthesis results showed that the decoder can provide an average information throughput of approximately 38 Mb/s at 3.5 dB with a power consumption of 3.85 mW and an area of 0.059 mm^2 for a

block length of 128 and a code rate of 1/2. Compared to the SC and SCL8-CRC6 decoders of polar codes, the PAC Fano decoder has a coding gain of 2.1 dB and 1.2 dB, respectively. However, due to its backtracking nature, the PAC Fano decoder has significantly lower throughput than the polar decoders.

Because of their excellent FER performance at short block lengths and low encoding complexity, one of the potential use cases of PAC codes could be the Internet of Things (IoT), for which reliable communication is of great interest and low throughput and high decoding latency are tolerable. For example, remote electrocardiography (ECG) applications require high reliability of data transmission and low throughput of tens of Kb/s, and can tolerate end-to-end transmission latency of 2 seconds [64].

Table 5.1: Summary of ASIC Implementation Results

Implementation	This work	[26]	[26]
Algorithm	Fano	SC	SCL8-CRC6
(N, K)	(128,64)	(128,64)	(128,70)
E_b/N_0 at FER = 10^{-5}	**3.6 dB**	5.7 dB	4.8 dB
Technology	28 nm	28 nm	28 nm
Supply (V)	0.72	1.0	1.0
Frequency (MHz)	500	503	418
Area (mm^2)	**0.059**	0.07	3.15
Power (mW)	3.85	80	3340
Avg. Coded TP (Mb/s)	76.2†	**64000**	53000
Area Eff. (Gb/s/mm^2)	1.292†	**931**	17
Energy Eff. (pJ/bit)	50.5†	1.28	62.46
Normalized for 0.72 V using the scaling method in [62, 63]			
Power (mW)	**3.85**	41.47	1731.46
Energy Eff. (pJ/bit)	50.5†	**0.663**	32.379

†Average value at $E_b/N_0 = 3.5$ dB.

5.1 Suggestions for Future Work

We provide some suggestions for future research directions that are relevant to the work discussed in this thesis.

Line Architecture for Polar Demapper

To design the polar demapper block, we adopted the fully parallel FFT-like architecture of [60]. This architecture consists of $n = \log_2 N$ stages, each of which containing N process elements. The stages of polar demapper are activated sequentially, as explained in section 4.2, and at most N of the process elements of each stage are used. However, the line SC architecture introduced in [60] has only single stage which contains N process element. In this architecture, the intermediate LLRs are stored in a register bank and are connected via multiplexers to the process elements, and at each step of decoding, the proper portion of LLRs are passed to the process elements. Adopting this architecture for the polar demapper can reduce the area and power consumption of the PAC Fano decoder significantly.

Adaptive Quantization of LLRs

In this thesis, we used a fixed number of quantization bits (Q) for the channel and intermediate LLRs of the polar demapper. In [24], an adaptive quantization method is introduced that uses a variable number of bits for storing and processing the intermediate LLRs. This method partitions the polar code into smaller code segments and uses different number of bits for each segment based on maximizing the mutual information between the quantizer's input and output. Employing this method can reduce the chip area and power consumption of the PAC Fano decoder further.

Tree Search Constraining

Depending on the severity of the noise, the Fano algorithm may fail to identify the correct path and explore the wrong paths that may already result in a frame error. The authors of [17] introduce couple of early termination criteria that stop the decoder whenever the Fano decoder is likely to make a frame error and consequently, reduce the search complexity of the decoder. We believe it is worthwhile to study the suitability of these early termination techniques for hardware implementation.

Appendix A

Fixed-point Simulation

The quantization level plays an important role in the FER performance and time complexity of the Fano decoder. Increasing the number of quantization bits provides the decoder with more information on LLR values and helps the decoder perform better. But using more bits for representing the real values of LLRs requires extra resources. As an example, consider the architecture of polar demapper used in our design. This architecture has $N(\log_2 N - 1)$ intermediate nodes and N final nodes. To store the LLR values of these nodes $QN \log_2 N$ flip-flops are required. Thus, increasing Q by one increases the flip-flop usage by a factor of $N \log_2 N$. As a result, it is critical to select the number of quantization bits in such a way that performance loss and resource allocation are minimized.

In this appendix, we investigate the effect of quantization level on the computational complexity and FER performance of the proposed PAC Fano decoder. For this, we implement the proposed PAC Fano decoder on FPGA for $N = 128$ and $K = 64$ using $\mathbf{c} = (1, 0, 1, 1, 0, 1, 1)$ and $\Delta = 2$ for four different quantization levels $Q \in \{5, 6, 7, 8\}$. We choose the data index set $\mathcal{A}$ according to the Reed-Muller scoring rule and use the hard quantized bit-channel capacities as the bias vector $\mathbf{b}$.

Figure A.1 and Figure A.2 show the FER performance and Time complexity of

the proposed PAC Fano decoder under different quantization levels. With $Q = 5$ and $Q = 6$, the FER performance loss is significant but the time complexity is low (especially very low when using $Q = 5$). This demonstrates that using $Q = 5$ or $Q = 6$ does not provide enough information about the LLRs, and as a result, the decoding process ends prematurely, resulting in many incorrectly decoded codewords. On the other hand, the FER performance gain with $Q = 8$ is negligible when compared to $Q = 7$, but it results in increased time complexity at low SNR regime. For this reason, we use 7-bits for quantization (i.e. $Q = 7$) in the implementation of our PAC Fano decoder.

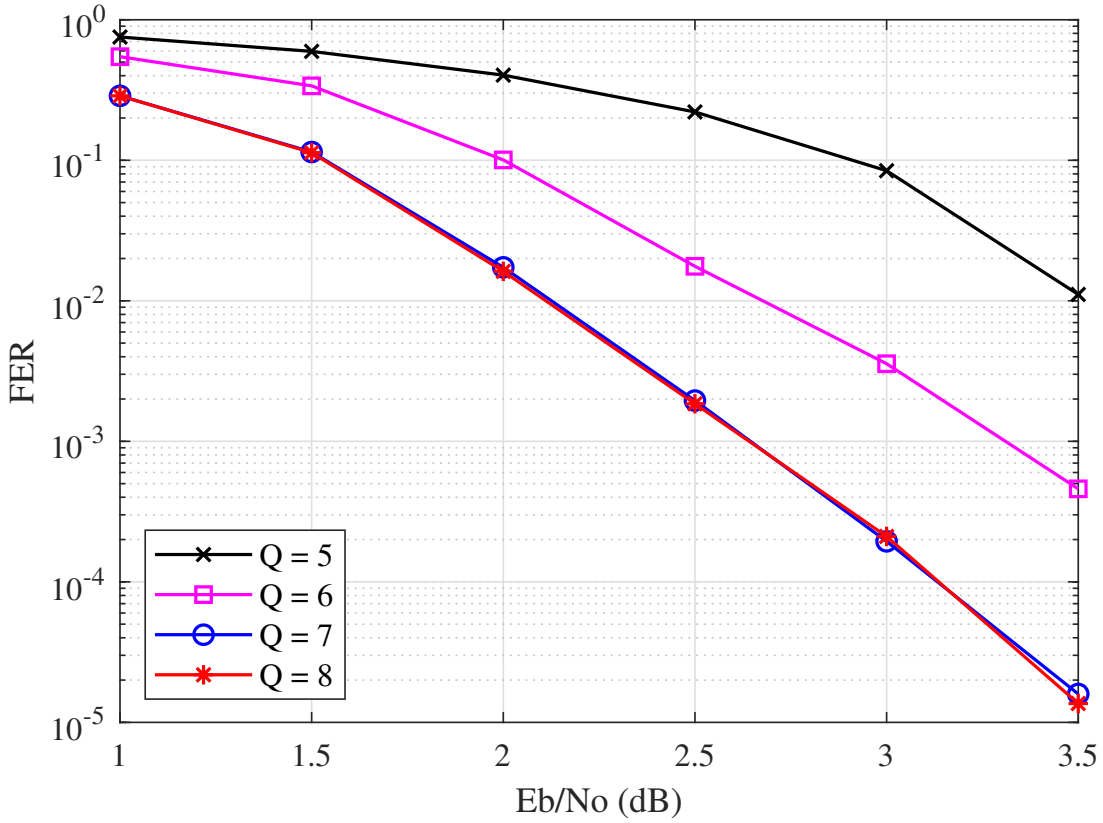

Figure A.1: FER performance of FPGA implementation of PAC Fano decoder for various number of quantization bits Q.

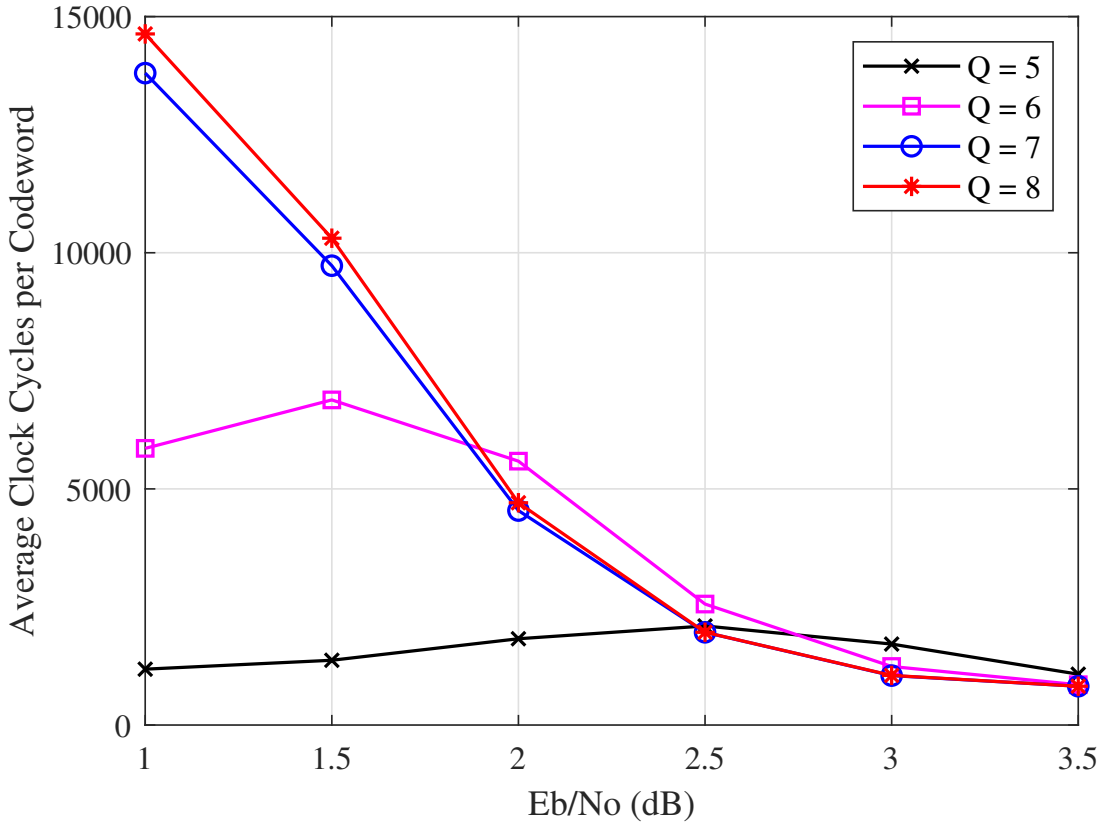

Figure A.2: Time complexity of FPGA implementation of PAC Fano decoder for various number of quantization bits Q.

Appendix B

Threshold Spacing Simulation

In this appendix, we investigate the impact of the threshold spacing Δ on the FER performance and time complexity of our proposed PAC Fano decoder. Although in our design the value of Δ can be reconfigured during the execution phase, because different values of Δ result in different FER performance and time complexity [47], it is critical to investigate this impact on the performance of our proposed decoder.

In order for the algorithm to find the correct path, the threshold must be reduced to a value less than the minimum metric along the correct path at some point. When a large Δ is used, lowering the threshold below the minimum metric of the correct path may cause the new threshold to fall below the minimum metric of several other incorrect paths. As a result, it possible for any of those incorrect paths to be decoded before the correct path [50].

The following figures in this appendix demonstrate the impact of threshold spacing Δ on the FER performance and time complexity of the proposed PAC Fano decoder for the SNR values from $E_b/N_0 = 1$ dB to $E_b/N_0 = 3.5$ dB with a step size of 0.5 dB. To obtain these curves, we use the FPGA implemented PAC Fano decoder for $N = 128$ and $K = 64$ with $\mathbf{c} = (1, 0, 1, 1, 0, 1, 1)$ and $\Delta = 2$. We choose the data index set $\mathcal{A}$ according to the Reed-Muller scoring rule and use

the hard quantized bit-channel capacities as the bias vector $\mathbf{b}$.

As expected, the value of Δ creates a trade-off between the decoder's FER performance and time complexity. Using smaller Δ values results in improved FER performance but increased time complexity; as Δ increases, the FER performance degrades, and the time complexity drops. The proper value for Δ is determined by the requirements of the application for which the PAC Fano decoder will be used. Table B.1 tabulates the estimated information throughput of the ASIC implementation of PAC Fano decoder for various values of Δ and SNR when clocked at $f_{\text{clk}} = 500$ MHz.

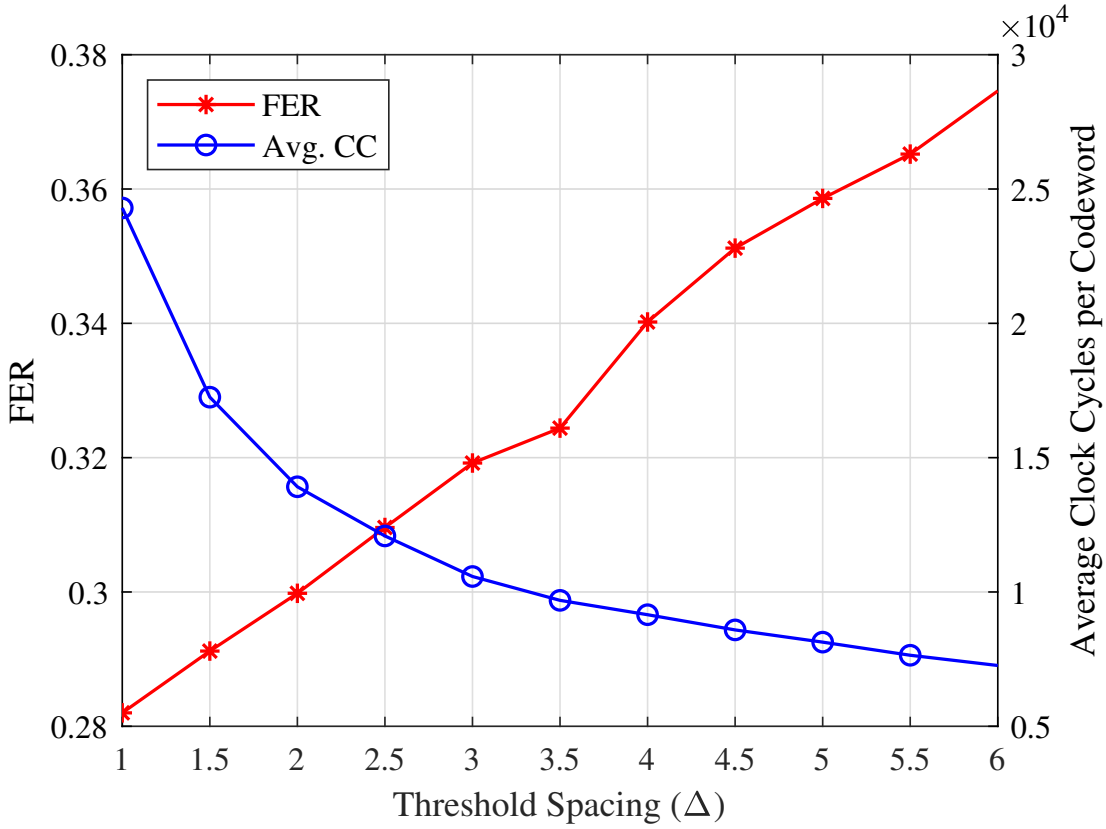

Figure B.1: Time complexity and FER performance of FPGA implementation of PAC Fano decoder versus Δ at $E_b/N_0 = 1$ dB.

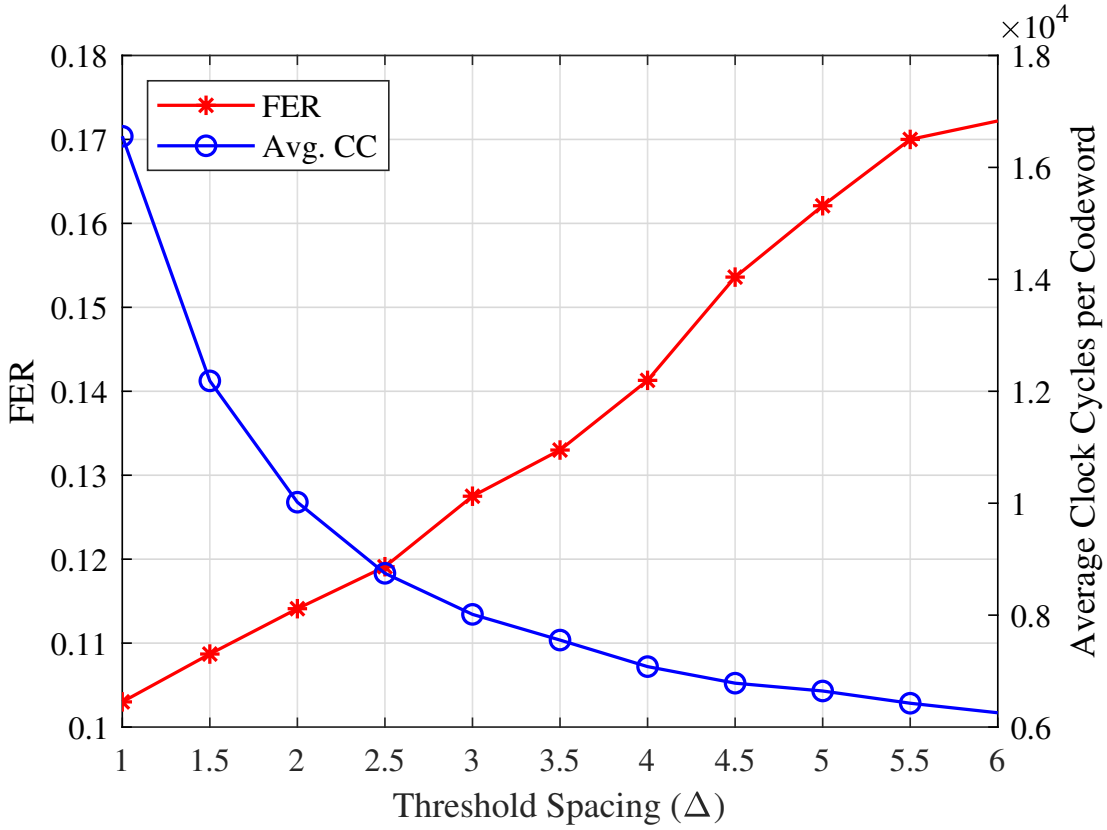

Figure B.2: Time complexity and FER performance of FPGA implementation of PAC Fano decoder versus Δ at $E_b/N_0 = 1.5$ dB.

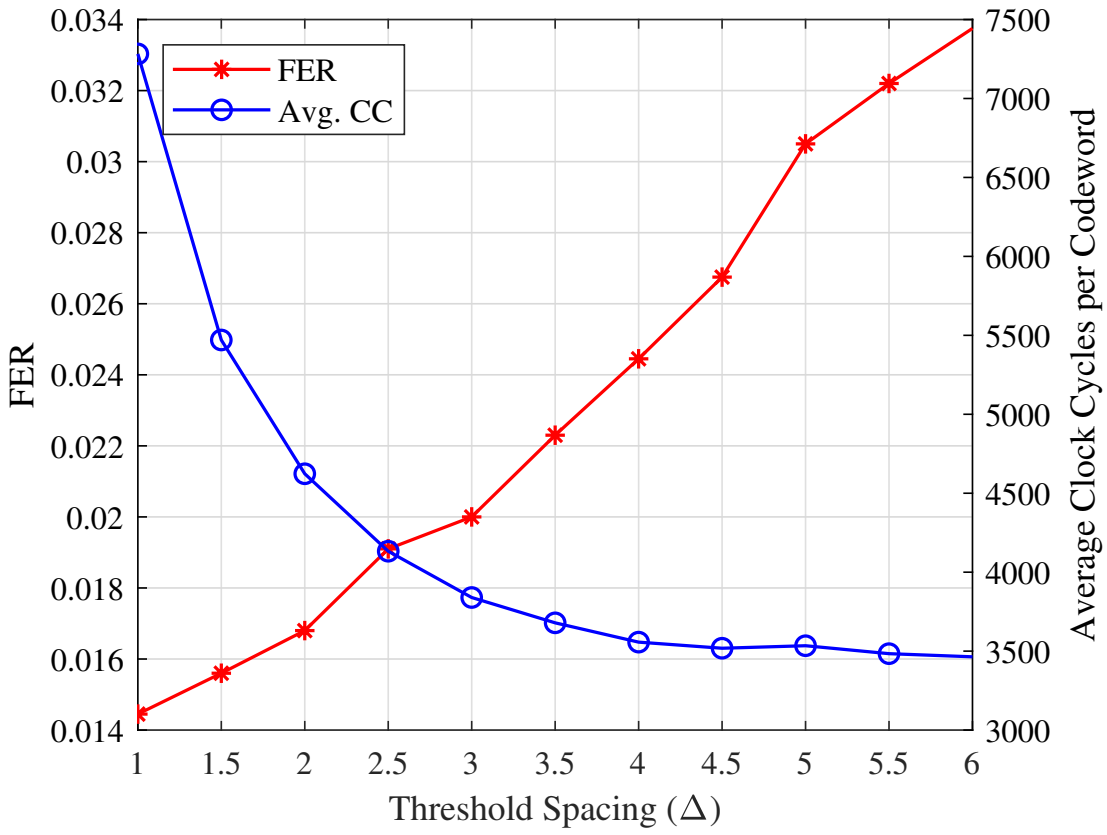

Figure B.3: Time complexity and FER performance of FPGA implementation of PAC Fano decoder versus Δ at $E_b/N_0 = 2$ dB.

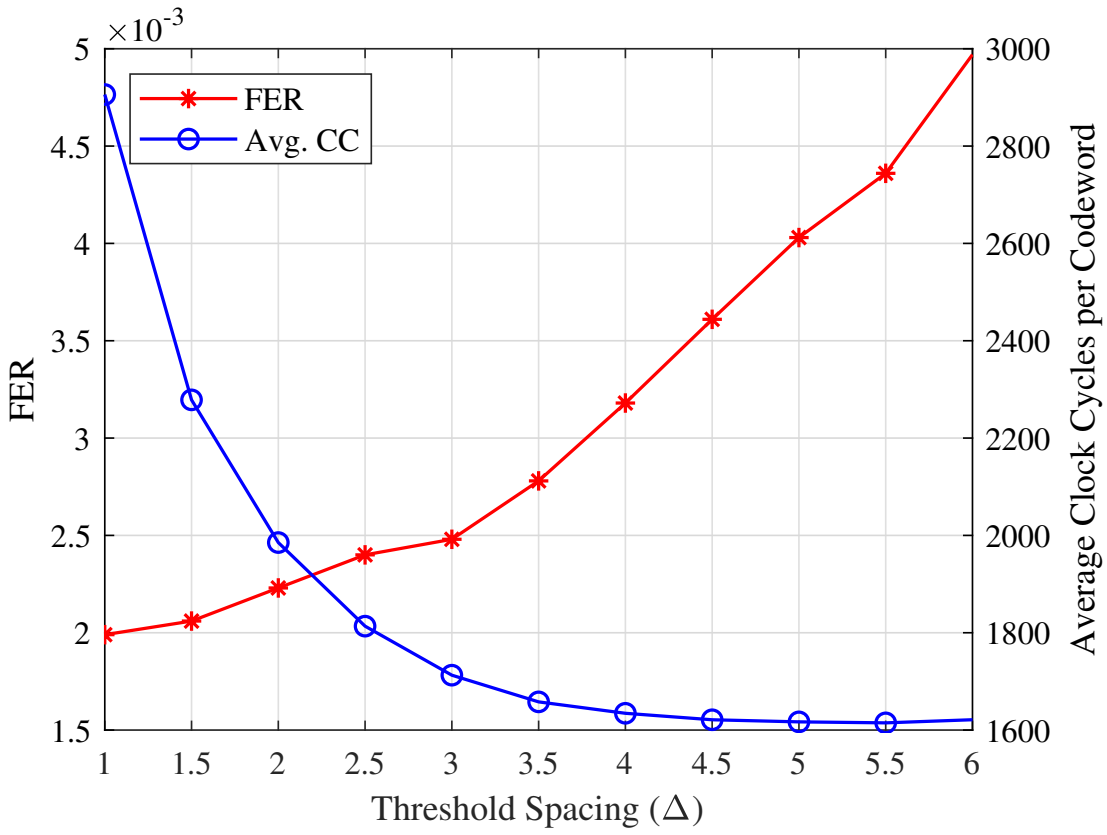

Figure B.4: Time complexity and FER performance of FPGA implementation of PAC Fano decoder versus Δ at $E_b/N_0 = 2.5$ dB.

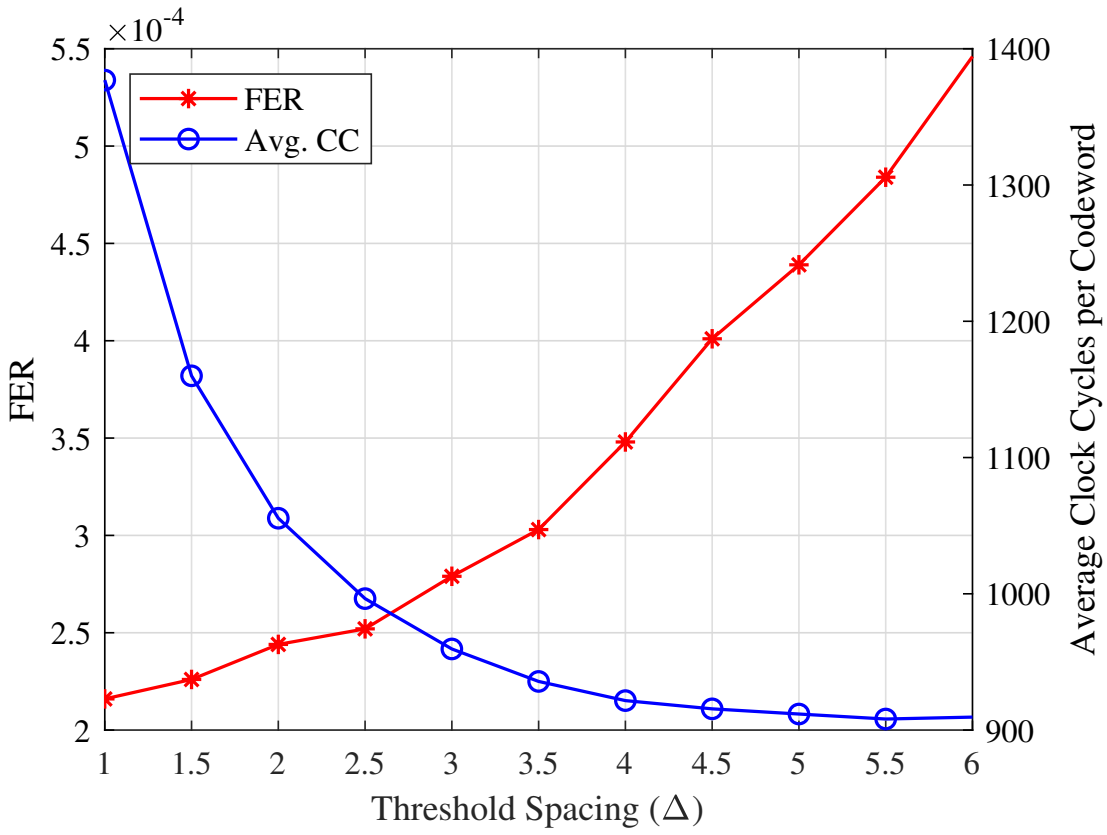

Figure B.5: Time complexity and FER performance of FPGA implementation of PAC Fano decoder versus Δ at $E_b/N_0 = 3$ dB.

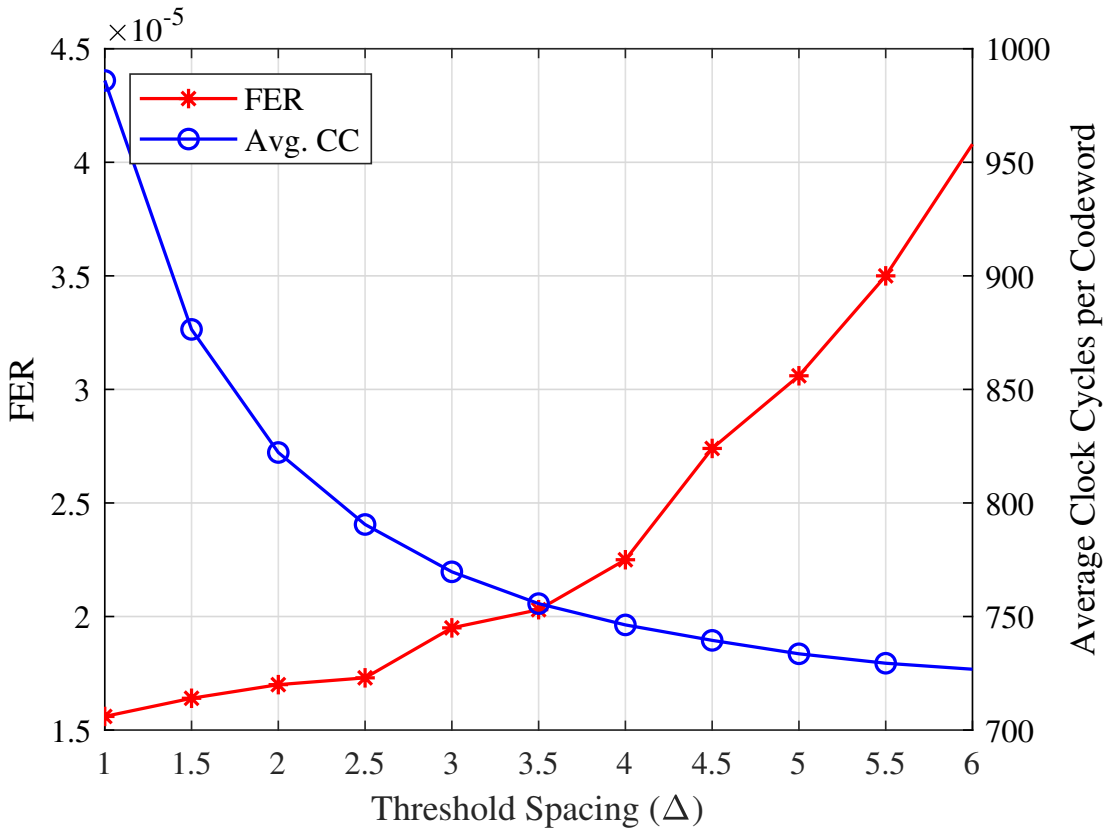

Figure B.6: Time complexity and FER performance of FPGA implementation of PAC Fano decoder versus Δ at $E_b/N_0 = 3.5$ dB.

Table B.1: Information throughput (Mb/s) of the PAC Fano decoder for various values of Δ and SNR.

		E_b/N_0 (dB)					
		1	1.5	2	2.5	3	3.5
Δ	1	1.32	1.93	4.39	11.01	23.24	32.45
	1.5	1.85	2.63	5.85	14.04	27.59	36.51
	2	2.30	3.19	6.92	16.12	30.32	38.92
	2.5	2.65	3.66	7.74	17.65	32.11	40.48
	3	3.03	3.99	8.33	18.68	33.35	41.58
	3.5	3.30	4.24	8.70	19.3	34.2	42.35
	4	3.5	4.52	9.00	19.58	34.72	42.88
	4.5	3.73	4.72	9.10	19.74	34.95	43.27
	5	3.93	4.82	9.05	19.79	35.1	43.62
	5.5	4.19	4.98	9.19	19.82	35.24	43.87
	6	4.41	5.12	9.24	19.74	35.18	44.03

Bibliography

[1] A. J. Viterbi and J. K. Omura, *Principles of digital communication and coding*. New York: McGraw-Hill, 1979.

[2] M. Benaissa and Y. Zhu, "Reconfigurable hardware architectures for sequential and hybrid decoding," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 3, pp. 555–565, 2007.

[3] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC '93 - IEEE International Conference on Communications*, vol. 2, pp. 1064–1070 vol.2, 1993.

[5] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[6] D. J. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *IMA International Conference on Cryptography and Coding*, pp. 100–111, Springer, 1995.

[7] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 32, no. 18, p. 1645, 1996.

[8] M. Sipser and D. A. Spielman, "Expander codes," *IEEE transactions on Information Theory*, vol. 42, no. 6, pp. 1710–1722, 1996.

[9] D. A. Spielman, "Linear-time encodable and decodable error-correcting codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1723–1731, 1996.

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

[11] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[12] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE communications letters*, vol. 16, no. 12, pp. 2044–2047, 2012.

[13] E. Arikan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Communications Letters*, vol. 12, no. 6, pp. 447–449, 2008.

[14] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[15] E. Arıkan, "From sequential decoding to channel polarization and back again," *arXiv preprint arXiv:1908.09594*, 2019.

[16] M. Moradi, A. Mozammel, K. Qin, and E. Arıkan, "Performance and complexity of sequential decoding of PAC codes," *arXiv preprint arXiv:2012.04990*, 2020.

[17] M. Rowshan, A. Burg, and E. Viterbo, "Polarization-adjusted convolutional (PAC) codes: Sequential decoding vs list decoding," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 2, pp. 1434–1447, 2021.

[18] H. Yao, A. Fazeli, and A. Vardy, "List decoding of Arıkan's PAC codes," *Entropy*, vol. 23, no. 7, p. 841, 2021.

[19] J. M. Wozencraft, "Sequential decoding for reliable communication," Tech. Rep. 325, Research Laboratory of Electronics, MIT, Cambridge, 1957.

[20] M. Rowshan and E. Viterbo, "List viterbi decoding of pac codes," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 3, pp. 2428–2435, 2021.

[21] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Transactions on Information Theory*, vol. 9, no. 2, pp. 64–74, 1963.

[22] K. Zigangirov, "Some sequential decoding procedures," *Problemy Peredachi Informatsii*, vol. 2, no. 4, pp. 13–25, 1966.

[23] F. Jelinek, "Fast sequential decoding algorithm using a stack," *IBM journal of research and development*, vol. 13, no. 6, pp. 675–685, 1969.

[24] A. Süral, E. G. Sezer, Y. Ertuğrul, O. Arikan, and E. Arikan, "Terabits-per-second throughput for polar codes," in *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, pp. 1–7, IEEE, 2019.

[25] O. Dizdar and E. Arıkan, "A high-throughput energy-efficient implementation of successive cancellation decoder for polar codes using combinational logic," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 3, pp. 436–447, 2016.

[26] C. Kestel, L. Johannsen, O. Griebel, J. Jimenez, T. Vogt, T. Lehnigk-Emden, and N. Wehn, "A 506 Gbit/s polar successive cancellation list decoder with CRC," in *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 1–7, 2020.

[27] P. Giard, A. Balatsoukas-Stimming, T. C. Müller, A. Bonetti, C. Thibeault, W. J. Gross, P. Flatresse, and A. Burg, "Polarbear: A 28-nm FD-SOI ASIC for decoding of polar codes," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 4, pp. 616–629, 2017.

[28] Y. S. Park, Y. Tao, S. Sun, and Z. Zhang, "A 4.68 Gb/s belief propagation polar decoder with bit-splitting register file," in *2014 Symposium on VLSI Circuits Digest of Technical Papers*, pp. 1–2, IEEE, 2014.

[29] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "237 Gbit/s unrolled hardware polar decoder," *Electronics Letters*, vol. 51, no. 10, pp. 762–763, 2015.

[30] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "Multi-mode unrolled architectures for polar decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 9, pp. 1443–1453, 2016.

[31] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, 2014.

[32] A. Süral, E. G. Sezer, E. Kolağasıoğlu, V. Derudder, and K. Bertrand, "Tb/s polar successive cancellation decoder 16nm ASIC implementation," *arXiv preprint arXiv:2009.09388*, 2020.

[33] A. Pamuk, "An FPGA implementation architecture for decoding of polar codes," in *2011 8th International symposium on wireless communication systems*, pp. 437–441, IEEE, 2011.

[34] X.-Y. Shih, J.-H. Tsai, B.-X. Li, and C.-P. Huang, "Reconfigurable hardware architecture of area-efficient multi-mode successive cancellation (SC) decoder," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2022.

[35] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Transactions on very large scale integration (VLSI) Systems*, vol. 23, no. 10, pp. 2268–2280, 2014.

[36] M. Jeong and S. Hong, "SC-Fano decoding of polar codes," *IEEE Access*, vol. 7, pp. 81682–81690, 2019.

[37] I. Jacobs, "Sequential decoding for efficient communication from deep space," *IEEE Transactions on Communication Technology*, vol. 15, no. 4, pp. 492–501, 1967.

[38] J. Layland and W. Lushbaugh, "A flexible high-speed sequential decoder for deep space channels," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 813–820, 1971.

[39] G. Forney and E. Bower, "A high-speed sequential decoder: Prototype design and test," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 821–835, 1971.

[40] A. Mozammel, "Hardware implementation of fano decoder for polarization-adjusted convolutional (pac) codes," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[41] M. Moradi and A. Mozammel, "A Monte-Carlo based construction of polarization-adjusted convolutional (PAC) codes," *arXiv preprint arXiv:2106.08118*, 2021.

[42] M. Moradi and A. Mozammel, "Concatenated Reed-Solomon and polarization-adjusted convolutional (PAC) codes," *arXiv preprint arXiv:2106.08822*, 2021.

[43] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge university press, 2008.

[44] R. Mori and T. Tanaka, "Performance and construction of polar codes on symmetric binary-input memoryless channels," in *2009 IEEE International Symposium on Information Theory*, pp. 1496–1500, 2009.

[45] H. Li and J. Yuan, "A practical construction method for polar codes in AWGN channels," in *IEEE 2013 Tencon - Spring*, pp. 223–226, 2013.

[46] F. Brannstrom, L. Rasmussen, and A. Grant, "Convergence analysis and optimal scheduling for multiple concatenated codes," *IEEE Transactions on Information Theory*, vol. 51, no. 9, pp. 3354–3364, 2005.

[47] M. Moradi, "On sequential decoding metric function of polarization-adjusted convolutional (PAC) codes," *IEEE Transactions on Communications*, vol. 69, no. 12, pp. 7913–7922, 2021.

[48] F. Brannstrom, *Convergence analysis and design of multiple concatenated codes*. Ph.D. dissertation, Chalmers University, 2004.

[49] B. Li, H. Shen, and D. Tse, "A RM-polar codes," *arXiv preprint arXiv:1407.5483*, 2014.

[50] S. Lin and D. J. Costello, *Error control coding*, vol. 2. New York: Prentice hall, 2001.

[51] A. Elkelesh, M. Ebada, S. Cammerer, and S. t. Brink, "Decoder-tailored polar code design using the genetic algorithm," *IEEE Transactions on Communications*, vol. 67, no. 7, pp. 4521–4534, 2019.

[52] V. Bioglio, C. Condo, and I. Land, "Design of polar codes in 5G new radio," *IEEE Communications Surveys Tutorials*, vol. 23, no. 1, pp. 29–40, 2021.

[53] P. Elias, "Coding for noisy channels," *IRE Conv. Rec.*, vol. 3, pp. 37–46, 1955.

[54] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[55] G. D. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[56] J. L. Massey, *Threshold decoding*. Cambridge, MA: USA: MIT Press, 1963.

[57] R. G. Gallager, *Information theory and reliable communication*, vol. 2. New York: Wiley, 1968.

[58] I. M. Jacobs and J. Wozencraft, *Principles of communication engineering*. New York: John Wiley and Sons, 1965.

[59] R. Johannesson and K. S. Zigangirov, *Fundamentals of convolutional coding*. John Wiley & Sons, 2015.

[60] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *2011 IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1665–1668, IEEE, 2011.

[61] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE transactions on signal processing*, vol. 63, no. 19, pp. 5165–5179, 2015.

[62] C.-C. Wong and H.-C. Chang, "Reconfigurable turbo decoder with parallel architecture for 3GPP LTE system," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 7, pp. 566–570, 2010.

[63] C.-H. Lin, C.-Y. Chen, A.-Y. Wu, and T.-H. Tsai, "Low-power memory-reduced traceback map decoding for double-binary convolutional turbo decoder," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 5, pp. 1005–1016, 2009.

[64] N. Golmie, D. Cypher, and O. Rébala, "Performance analysis of low rate wireless technologies for medical applications," *Computer Communications*, vol. 28, no. 10, pp. 1266–1275, 2005.