Design and implementation of an Enhanced Successive Cancellation Flip decoder and Rapid SCF Execution-Time Estimation

by

Tannaz KALATIAN

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR A MASTER'S DEGREE
WITH THESIS IN ELECTRICAL ENGINEERING
M.A.Sc.

MONTREAL, NOVEMBER 29, 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

ACKNOWLEDGEMENTS

First of all, I would like to express my profound thanks to my supervisor Professor. Pascal Giard for the consistent support and guidance during the running of this project and for keeping me motivated through my master program. I was so lucky to have such a caring and helpful supervisor. I cannot explain how grateful and fortunate I am to be his student. Additionally, I want to thank Ilshat Sagitov, my teammate in the LACIME lab, who was a great help throughout this thesis and project.

I would also like to thank the Natural Sciences and Engineering Research Council (NSERC) and the Ministère de l'Enseignement supérieur for providing me with a full scholarship to pursue my master's program in Canada.

Furthermore, I would like to thank my parents for their love, trust, and encouragement, as well as my brother, Kourosh, who always believed deeply in me and encouraged me to reach my goals.

As a final word, I want to thank my best friend and partner, Amirhossein, who stood by my side and supported me emotionally over the challenges I faced, and Leo, my sweet dog, for all the love he gave me.

Conception et mise en œuvre d'un décodeur Flip à annulation successive améliorée et étude des variables efficaces sur la taille du tampon

Tannaz KALATIAN

RÉSUMÉ

Les systèmes modernes de communication et de stockage nécessitent une détection et une correction des erreurs pour fonctionner correctement. Comme les codes polaires sont des codes correcteurs d'erreurs qui peuvent atteindre la capacité des canaux, ils ont fait l'objet d'une grande attention au cours de la dernière décennie. Selon les études actuelles, un décodeur (*Successive-cancellation (SC)*), le premier algorithme proposé pour décoder les codes polaires, peut atteindre la capacité du canal.

Bien que SC ait de bonnes performances en termes de taux d'erreur de trame pour une longueur de bloc infinie, avec des codes de longueur courte à moyenne, cet algorithme de décodage a une performance de correction d'erreur qui laisse à désirer.

Pour ces cas, divers algorithmes de décodage ont été proposés pour améliorer les performances de correction d'erreurs, comme le *Successive-cancellation Flip (SCF)*.

Le décodeur SCF a des performances supérieures, cependant, son temps d'exécution variable rend l'implémentation matérielle plus complexe, puisque le matériel doit être conçu pour supporter les temps d'exécution les plus défavorables afin d'éviter un débordement de tampon.

Dans ce mémoire, nous proposons d'abord un nouvel algorithme de décodage nommé *Enhanced Successive-cancellation Flip (ESCF)* qui est basé sur SCF et qui réduit le temps d'exécution moyen tout en maintenant les performances de correction d'erreurs du décodeur SCF. Les résultats de nos simulations montrent que le temps d'exécution est réduit surtout pour des *Signal Noise Ratios (SNRs)* plus faibles au prix d'un petit sacrifice au niveau des besoins en mémoire.

Dans la deuxième partie de la thèse, nous nous concentrons sur la proposition d'une méthode simple mais efficace qui prédit le temps d'exécution de SCF et identifie la sensibilité du temps d'exécution à divers paramètres tels que la longueur de bloc, le taux de code et la longueur du contrôle de redondance cyclique (*Cyclic Redundancy Check (CRC)*). Un mécanisme peu complexe a également été développé pour déterminer les statistiques relatives au nombre d'essais nécessaires dans diverses situations pour décoder une trame. Cette méthode sera utilisée comme élément de base dans des travaux futurs.

Mots-clés: 5G, Communication de données, codes polaires, correction et détection d'erreurs, décodage par annulation successive améliorée

Design and implementation of an Enhanced Successive Cancellation Flip decoder and Rapid SCF Execution-Time Estimation

Tannaz KALATIAN

ABSTRACT

Modern communication and storage systems require error detection and correction to function properly. Since polar codes are error-correcting codes that can achieve channel capacity, they have been receiving a lot of attention in the past decade. According to current studies, a SC decoder, the first algorithm that was proposed to decode polar codes, can achieve channel capacity.

Although SC has good performance in terms of Frame Error Rate (FER) for infinite block length, with short to moderate length codes, this decoding algorithm has an error-correction performance that leaves to be desired.

For these cases, various decoding algorithms were proposed to improve the error-correction performance such as SCF.

The SCF decoder has superior performance, however, its variable execution time makes hardware implementation more complex, since the hardware should be designed to withstand worst-case execution times to prevent buffer overflow.

In this thesis, first, a new decoding algorithm named ESCF is proposed which is based on SCF that reduces the average execution time while maintaining the error-correction performance of the SCF decoder. Our simulation results show that the execution time is reduced especially at lower SNRs at the cost of a small sacrifice in memory requirements.

In the second part of the thesis, we focus on proposing a simple but effective method that predicts the execution time of SCF and identifies the sensitivity of the execution time from various parameters such as block length, code rate, and the CRC length. A low-complexity mechanism was also developed to determine statistics related to the number of trials needed in various situations to decode a frame. This method is to be used as a building block in future works.

Keywords: 5G, Data communication, Polar Codes, Error-correction and detection, Enhanced Successive Cancellation Decoding

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

$T_{\max}$	Maximum number of trials.
AWGN	Additive White Gaussian Noise.
BPSK	Binary Phase Shift Keying.
CRC	Cyclic Redundancy Check.
EIS	Enhanced Index Selection.
ESCF	Enhanced Successive-cancellation Flip.
FER	Frame Error Rate.
FIS	Fixed Index Selection.
LDPC	Low-Density Parity-Check.
LLR	Likelihood- Ratio.
R	Code Rate.
SC	Successive-cancellation.
SCc	Successive Cancellation Stack.
SCF	Successive-cancellation Flip.
SCL	Successive Cancellation List.
SNR	Signal Noise Ratio.

INTRODUCTION

Over the last 20 years, digital circuitry has slowly taken over applications previously controlled by analog circuitry. Digital circuits allow us to detect and correct errors, which is one of the reasons for this tendency. Detecting and correcting errors are essential components of digital communication systems. The ability to detect and fix errors in digital communication made communication faster and more reliable. In the years following the second world war, Claude Shannon founded the field of information theory and estimated the limit or reliability of communications within this field. Since Shannon defined Shannon's capacity Shannon (1948), numerous researchers have tried to achieve it.

Despite Reed-Solomon and Bose-Chaudhuri-Hocquenghem (BCH) codes being well-performing and still widely used today, approaching channel capacity did not occur until the discovery of turbo codes (Berrou, Glavieux & Thitimajshima, 1993) in the 1990s. On the other hand, in 1960, Low-Density Parity-Check (LDPC) codes were discovered by Robert Gallager (Gallager, 1962) which could also be capacity approaching; however, their decoding algorithm was too complex. One of the most recent discoveries on the road to achieving channel capacity is polar codes invented by Arıkan. In contrast to Low-Density Parity-Check (LDPC) and turbo codes, Polar codes use an explicit and non-random construction, which simplifies the implementation of encoders and decoders (Arıkan, 2009). Most importantly, comparing to the mentioned decoders, Successive-cancellation (SC) has a low complexity hardware implementation (Leroux, Raymond, Sarkis, Tal, Vardy & Gross, 2012). Polar codes rely on channel polarization, where the probability of correctly estimating a codeword's bits is either 1 (perfectly reliable) or 0.5 (completely unreliable). In a recursive construction, increasing code length increases the polarization. As polar codes' length approaches infinity, they can achieve the symmetric capacity of memoryless channels by using SC decoding.

Because SC decoders have a sequential implementation, they are low-complexity algorithms but have high latency and low throughput. Furthermore, error correction performance is poor with finite block length. In order to improve the finite block length performance, more sophisticated algorithms have been introduced lately, such as the Successive Cancellation List (SCL) decoder (Tal & Vardy, 2015) and Successive Cancellation Stack (SCc) decoder (Chen, Niu & Lin, 2013). As the underlying decoder, SC is used in these algorithms, but their performance is enhanced by simultaneously exploring several ways on a decision tree, with each way resulting in one candidate codeword. SCL decoding has computational complexity $O(LNlogN)$ and memory complexity of $O(LN)$, where L is the list size. In contrast, SC stack decoding has computational complexity $O(DNlogN)$ and memory complexity of $O(DN)$, where D is the depth of the stack. Finally, in 2014, a new SC-based decoding algorithm, called SCF was proposed that offered the $O(N)$ memory complexity and an average computational complexity that is $O(NlogN)$ at high SNR. The error-correction performance of this algorithm is better than that of the SC algorithm, and it can compete with SCL decoder (Afisiadis, Balatsoukas-Stimming & Burg, 2014).

This decoder, however, incorporates a new parameter: the Maximum number of trials ($T_{\max}$), which results in varying execution times. The computation complexity is reasonable on average, but when it comes to hardware implementation, it must be prepared for the worst-case scenario in terms of delay.

Objective

Our focus of study is addressing the execution-time variability in two aspects: first, reducing the average execution-time, second, predicting the execution time. We will propose a modification to SCF decoder and measure the execution time and error-correction performance. Later, we will create a rapid system to predict the execution-time of decoder based on the variables identified to be relevant.

Thesis Contributions

ESCF is a new algorithm that reduces the execution time while keeping the error correction performance of SCF. It aims to reduce unnecessary computations to reduce the number of clock cycles required to complete the decoding. As a result of lowering the number of needed clock cycles of the decoder, the delay can be further reduced, and ultimately it will lead to a simpler hardware implementation.

Secondly, investigate different factors that affect the execution time of a SCF decoder are investigated. In addition, finding a pattern of other trials occurring in different configurations can help estimate execution times based on the various code parameter and channel condition. As a final contribution to this section, a low complexity system is implemented that shows that can generate a lookup table that associates the probability of different trials occurring based on the most significant parameters. This system can be a building block used in future works that could benefit from an early estimate of the decoder delay.

CHAPTER 1

BACKGROUND AND LITERATURE REVIEW

This chapter provides a broad overview of the fundamental ideas that appear throughout the thesis.

All digital communication systems can fit into a similar structure as illustrated by Figure 1.1. This structure was introduced by Shannon for the first time, saying that communication systems include a source encoder for compressing data, a channel encoder for ensuring redundancy, a communication channel for transmitting messages, a channel decoder to correct errors introduced by noisy channels, and a source decoder to decompress the data (Shannon, 1948). Although our main focus will be on channel encoding and decoding, I will briefly describe the components of a digital communication system.

1.1 Digital communication system

Six building blocks constitute a digital communication scheme. At the transmitter part, the operational blocks process the input message, encode, modulate, and transmit it over the communication channel. Then the transmitter blocks execute reverse processing at the receiver site to recover the original information (Proakis & Manolakis, 1992).

1.1.1 Source Encoder

Digital communication relies on integrating various data compression, encoding, and modulation techniques to reproduce the message as successfully as possible at the receiver. The source encoder's responsibility is to reduce redundant information in information sets to represent them as a more compact and independent information set. Compressed data is transmitted into the channel encoder. In other words, source coding is converting the information into a sequence of binary digits as compact as possible.

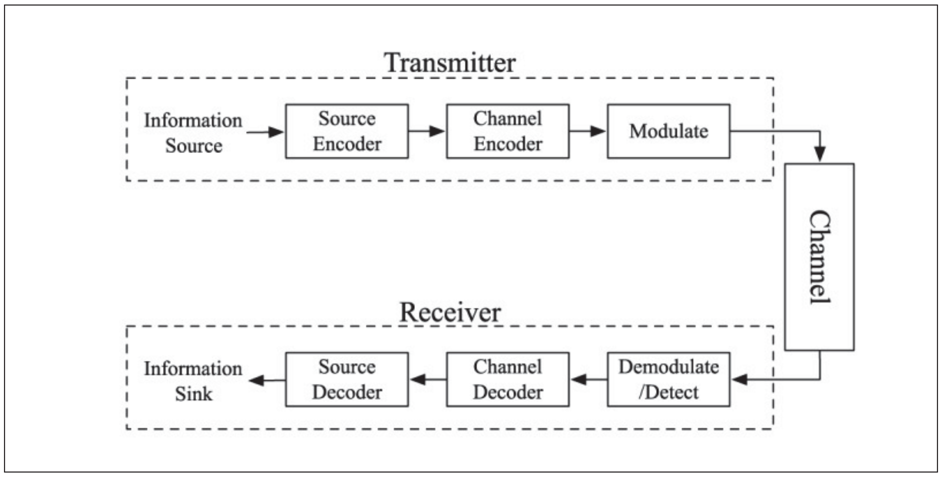


Figure 1.1 Simplified digital communication chain
Adapted from Farsad *et al.* (2013)

1.1.2 Channel Encoder

In order to handle the factors of interference and noise on the way through the communication channel, the channel encoder provides redundancy in the binary information sequence (Shannon, 1948). By adding redundancy in the information message, the reliability of the received data is increased, and the accuracy of the signal is enhanced. In this module, another parameter named code rate is taken into account to choose the number of required redundant bits. The Code Rate (R) is the ratio of the number of meaningful bits in a frame. If a frame contains 8 bits, and the code rate is ½, it means four bits in this frame contain information and the rest of the frame are parity bits. In specific case of polar codes, they are called frozen bits. This concept will be explained later.

1.1.3 Modulator

The channel encoded content is transmitted to the modulator. Modulation is a method by which digital data sequences are transformed into a waveform that is compatible with the characteristics of the channel. The encoded sequence is modulated using appropriate digital modulation techniques, i.e., Binary Phase Shift Keying (BPSK), Quadrature Phase Shift Keying (QPSK), and transferred over the channel (Farsad *et al.*, 2013).

1.1.4 Channel

The communication channel is used to transmit signals carrying the encoded information from the transmitter to the receiver. During transmission of information, noise and interference of varying kinds can affect the information signal. Depending on the channel, communications can be conducted by air, wire, or optical cable. Additive White Gaussian Noise (AWGN) is used to simulate the channel in the thesis experiments.

1.1.5 Demodulator (Detector)

During transmission, the modulated signal, likely containing some channel noise, will be demodulated at the receiver by the detector and forwarded to the decoder. The output of detector module are Likelihood- Ratios (LLRs). Those LLR values are given to the decoder.

1.1.6 Channel Decoder

Using redundant data inserted by the channel encoder in the transmitter, the channel decoder processes the received encoded sequence and decodes the message bits. After that, the source decoder re-creates the information message. Due to errors in the channel, the reconstructed information message at the receiver is likely a close approximation of the original message, not the exact transmitted data. In other words, when the signal is transferred through the noisy channel, data in the signal will be effected. Transmitters and receivers are implemented to overcome the noise caused by the channel. There are several coding algorithms such as Turbo codes, LDPC, and Polar codes, which aim to reduce the error probability by detecting and correcting the errors. A comprehensive discussion of polar codes is presented in this thesis.

1.2 Polar Codes

Polar codes are linear block error-correcting codes invented by Erdal Arıkan in 2009 (Arıkan, 2009). As opposed to LDPC and turbo codes, the newest error-correcting codes have an explicit construction that makes their implementation less complex than those mentioned error-correcting codes. Polar codes operate by the channel polarization effect, which increase the success with codeword prediction. Polar codes consider each bit in the frame either reliable or unreliable. With increasing code length, the probability of recreating the exact codeword will approach their limit. Polar codes were found to be able to achieve the symmetric capacity of memoryless channels once the channel length reaches infinity (Arıkan, 2009). The least reliable bits of a polar code is set to a known number by channel encoder and decoder (usually zero) and are called frozen bits. The rest of the bits are non-frozen bits containing information. Polar code construction consists in determining which bit locations should be frozen. The determination process is out of scope of this thesis.

1.2.1 Polar Code Encoder

In the encoding process, $x = u\boldsymbol{G}$, where x is the encoded information, called codeword, u is a source vector containing the information bits and frozen bits, and $\boldsymbol{G}$ is the generator matrix. There are k information bits in u, with $k \leq N$ and the code rate is k/N, where N is the length of frame. The generator matrix $\boldsymbol{G}$ is constructed through the recursive construction of a square kernel matrix $\boldsymbol{F}$ with a size that depends on the length of u. Given a matrix u of length N, $\boldsymbol{G}$ will be a $N \times N$ matrix.

$$\boldsymbol{G}_N = \boldsymbol{F}_2^{\otimes \log_2 N}, \boldsymbol{F}_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \tag{1.1}$$

and $^{\otimes}$ is the Kronecker power. Moreover, the number of information bits in u is defined as k, with $k \leq N$. Therefore, the code rate is k/N.

Figure 1.2 is showing the encoder for 2 bits. In this figure, the input frame of this encoder is (u_0, u_1). The output, which is the encoded codeword, is (x_0, x_1). Inside this module, an XOR operation calculates the value of x_0. The encoded frame will be $(u_0 \oplus u_1, u_1)$. In the output of the encoder, it can be seen that u_1 is repeated in 2 bits of output. If there is an error in the x_0, there is still a chance to recover u_1 based on the x_1 value. However, u_0 does not have the same chance of being recovered. Therefore, it is a more reliable location to carry information than u_0.

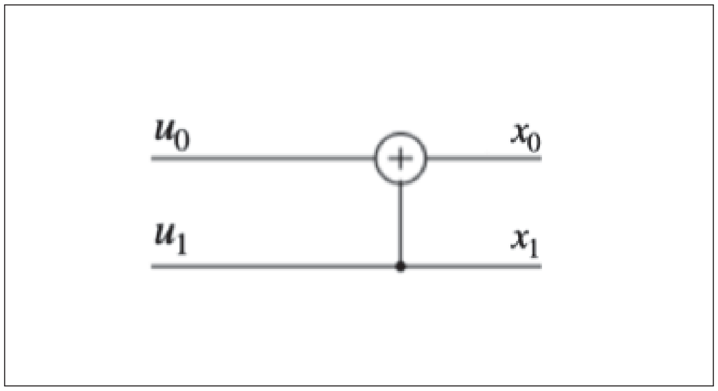


Figure 1.2 2-bit encoder

As another instance, if u is a 4-bit vector, $N = 4$ and $\boldsymbol{G}$ will be a 4×4 matrix named $\boldsymbol{G}_4$, where

$$\boldsymbol{G}_4 = \boldsymbol{F}_2^{\otimes 2} = \begin{bmatrix} \boldsymbol{F}_2 & 0 \\ \boldsymbol{F}_2 & \boldsymbol{F}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \tag{1.2}$$

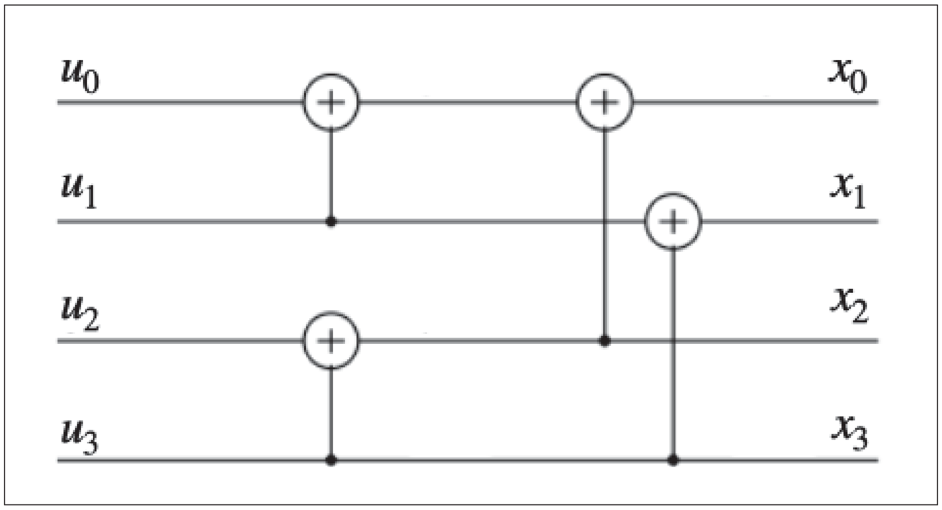


Figure 1.3 4-bit encoder

Figure 1.3 is showing the structure of encoding 4 bits. The input frame of this 4-bits encoder is (u_0, u_1, u_2, u_3) and the output which is the encoded frame is (x_0, x_1, x_2, x_3). This frame can also be written as $(u_0 \oplus u_1 \oplus u_2 \oplus u_3,\ u_1 \oplus u_2 \oplus u_3,\ u_2 \oplus u_3,\ u_3)$. It is shown that u_3 is repeated 4 times. Consequently, it has 4 times more chance than other to be recovered which make it more reliable bit location.

1.2.2 Successive Cancellation Decoder

In SC decoding, the decoder tree is traversed depth-first, choosing left edges from the decoder tree before going rightwards again until it reaches the size-1 frozen and information leaf nodes. The child nodes receive LLRs, and the parents receive estimated bits denoted α and β. Left child node l messages are calculated using F function:

$$\begin{aligned}\alpha_l[i] &= F(\alpha_v[i], \alpha_v[i + {}^{N_v}/2]) \\ &= \text{sign}(\alpha_v[i])\text{sign}(\alpha_v[i + {}^{N_v}/2])\min(|\alpha_v[i]|, |\alpha_v[i + {}^{N_v}/2]|) \qquad (1.3)\end{aligned}$$

In this equation, given LLRs to child are defined as α_v and the size of corresponding constituent code is N_v. In the decoder tree, right nodes r are calculated using G function, and β_l is the estimated bit from the left child node:

$$\alpha_r[i] = G(\alpha_v[i], \alpha_v[i][i + {}^{N_v}/2], \beta_l[i])$$

$$= \begin{cases} \alpha_v[i][i + {}^{N_v}/2] + \alpha_v[i] & \text{when } \beta_l[i] = 0 \\ \alpha_v[i][i + {}^{N_v}/2] - \alpha_v[i] & \text{otherwise} \end{cases} \tag{1.4}$$

When the decoder reaches to a leaf node, it is a hard decision on the LLRs unless that bit is known to be frozen, in which case, it is set to be the predefined value. If the calculated LLR value is positive, that bit is set to 0; otherwise, it will set that bit to 1. Note that in polar codes, the encoder and decoder set unreliable bit locations to a known number (usually zero). Therefore, the decoder knows the frozen-bit locations and assigns them to zero while decoding. Equation 1.5 shows the hard decision process.

$$\beta[i] = \begin{cases} 0 & \text{when } LLR[i] > 0 \\ 1 & \text{when } LLR[i] < 0 \\ 0 & \text{If it is frozen bit} \end{cases} \tag{1.5}$$

In the last step, when both children of a node are calculated, the combination operation is done, and estimated values are returned to the parent. This operation is defined as:

$$\beta_v[i] = \begin{cases} \beta_l[i] \oplus \beta_r[i] & \text{when } i < {}^{N_v}/2 \\ \beta_r[i - {}^{N_v}/2] & \text{otherwise} \end{cases} \tag{1.6}$$

$\oplus$ operation is XOR.

All notations in this section are adopted from (Giard, Thibeault & Gross, 2017). An illustration of a polar decoder with the successive cancellation algorithm for a 4-bit vector is presented below to better understand the decoder structure. The memory layout is explained in this example as well.

1.2.2.1 Successive cancellation decoder and memory layout

As mentioned previously, the SC is a sequential decoder. The memory layout is illustrated and described for a 4-bit code for the sake of simplicity but it can easily be generalized for code with a length that is power of 2. The first step of filling memory for a 4-bit frame is shown in Figure 1.4. In the first step, the channel LLRs are given to the decoder, and they are stored in the memory. Afterward, the intermediate LLRs are calculated and saved in the memory to be consumed in the next step. In the memory layout, L_0 and L_1 are equal to $F(y_0, y_2)$ and $F(y_1, y_3)$ respectively (1.3). Afterward, using the intermediate LLRs, the decoder calculates the LLR value of the first bit by $F(L_0, L_1)$. The decoder uses the hard decision operation on the L_2 value. As previously explained, when decoder gets to the leaf node, if the LLR value of that lead node is positive, the hard decision value will be 0; otherwise, it will be 1. Eventually, the decoder fills the hard decision vector with the $\hat{u}_0$ (1.5). Note that almost always, $\hat{u}_0$ is frozen bit.

Figure 1.5 illustrates the decoder deciding the value of the second bit of the frame. It is shown that the channel LLRs and intermediate LLRs remain in the memory without any changes. Therefore, the only necessary step is step 4, shown in the figure that calculates the G function (1.4). The decision LLR is overwritten with L_3 which is an estimation for u_1. Additionally, $\hat{u}_1$ is written in the hard decisions vector.

Figure 1.6 demonstrates the decoding process of u_2. Notice that the intermediate LLRs are changed when the decoder calculates the decision LLRs for the right sub-tree. However, the channel LLRs remain constant in the memory. In step 5, L_4 and L_5 are calculated as $G(y_0, y_2, \hat{u}_0 \oplus \hat{u}_1)$ and $G(y_1, y_3, \hat{u}_1)$. In this step, there is a combination operation that provide the result of $\hat{u}_0 \oplus \hat{u}_1$ for the first G function. Each time G function is called, if more than one hard decision is involved, combination operation needs to prepare the relevant input.

In step 6, the L_6 is overwritten on L_3 and is an estimation for u_2. Besides, $\hat{u}_2$ is added to the hard decision vector.

Finally, Figure 1.7 shows the estimation process of u_3. As it is demonstrated in this figure, the intermediate LLRs are not updated. Accordingly, the only remaining step is calculating the $G(L_4, L_5, \hat{u}_2)$ to estimate the last bit of this frame, followed by writing $\hat{u}_3$ in the hard decision vector. The hard decision vector is the estimated codeword. When the hard decision vector is filled completely, the SC decoding is finished.

It is worth reminding that, in general case, the first $N/2$ intermediate LLRs are only getting updated when the decoder transits from left sub-tree to right-tree. In a 4-bit example, when the decoder moves in a sub-tree, it uses the intermediate values to estimate the next bit. In the other words, in this example, to estimate u_2, the intermediate LLRs calculated for u_1 are not needed, while for estimating the u_1, the intermediate LLRs are necessary. As shown, the required memory to decode a frame is $Q \times (2N - 1)$, where N is the length of the frame and Q is the number of bits required to store each LLR which is usually 6 bits (Giard *et al.*, 2017). Moreover, a memory size of N is needed to store the hard decision values.

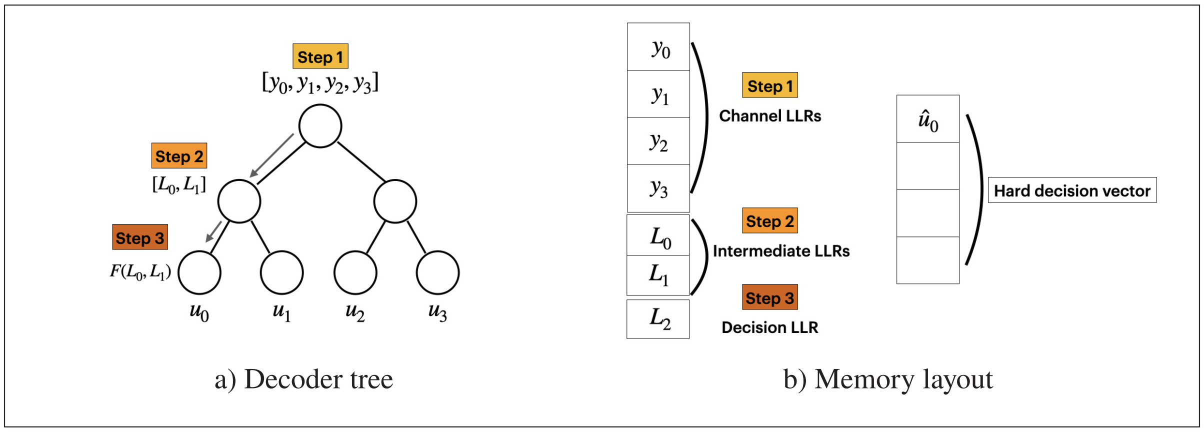


Figure 1.4 4-bit decoder - Estimating u_0

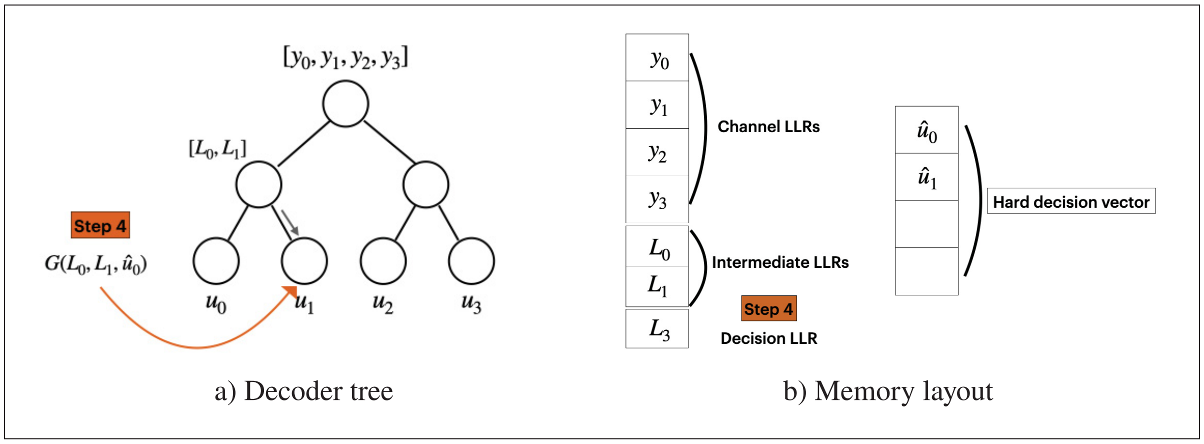


Figure 1.5 4-bit decoder - Estimating u_1

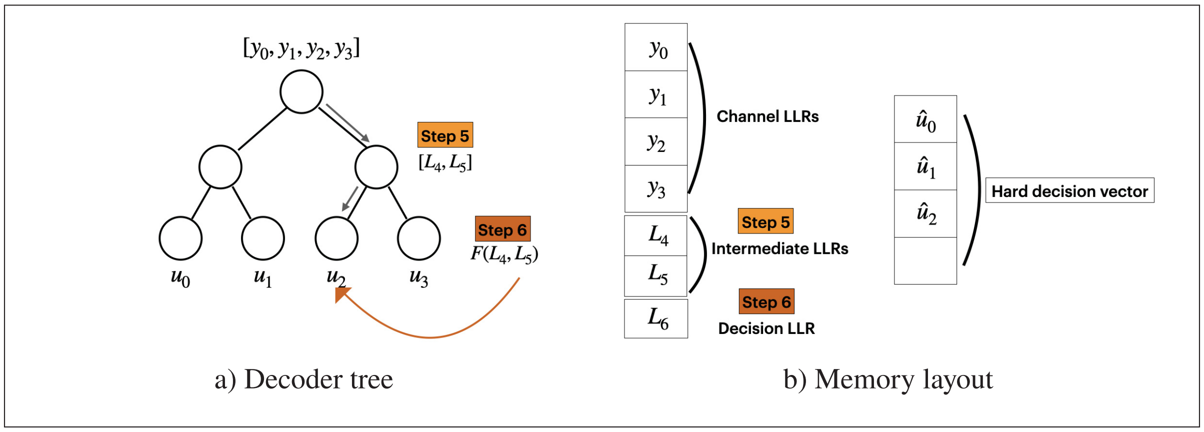


Figure 1.6 4-bit decoder - Estimating u_2

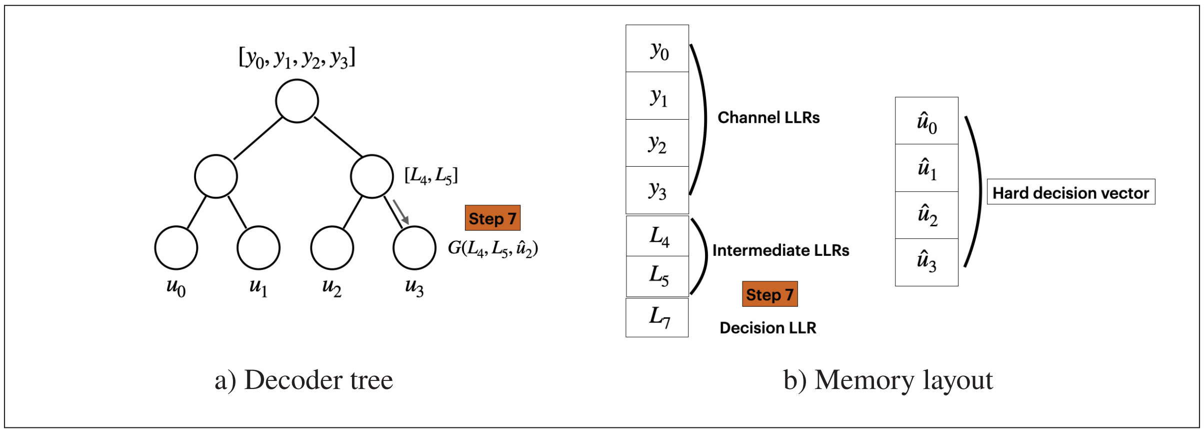


Figure 1.7 4-bit decoder - Estimating u_3

1.2.3 Successive Cancellation Flip Decoder

The SCF decoder estimates the first codeword using the SC decoding algorithm. Then, a decoder check if the estimated codeword is correct using Cyclic Redundancy Check (CRC). It is a method for detecting errors. To benefit from CRC method, for encoding the information bits, a CRC code with a C-bit remainder is added to the information bits and stored in the frame. The length of CRC code is considered in code rate. For example, it block length is 1024 bits, and $R = 1/2$, 512 bits are information bits and CRC bits. In that case, if CRC length is 8, the frame contains 504 information bits.

Note that the Maximum number of trials ($T_{\max}$) is a variable that defines how many times the SCF algorithms can attempt to estimate the codeword. The codeword is presumed to be accurate if the CRC matches. Otherwise, when the CRC does not match, the lowest $T_{\max}$ LLRs, which indicate the level of reliability of the bit estimations, are sorted and stored as the flipping candidate list. Then the SCF decoder attempts to repeat the decoding using SC decoder. When the decoder reaches to the first flipping candidate, the hard decided value is flipped and decoder continues decoding till end of the frame. This process is repeated until the number of attempts reaches to the $T_{\max}$ or the decoder estimates the correct codeword which is equal to find a CRC match (Condo, Ercan & Gross, 2018).

Note that the $T_{\max}$ is a variable that defines how many times the SCF algorithms can attempt to estimate the codeword. A codeword is defined undecodable if CRC does not match after $T_{\max}$ attempts to decode.

To sum it up, the SCF algorithm can be broken down into the four sections:

1. Decode the frame with SC algorithm and verify if the CRC matches.
2. If CRC passes, decoding is finished. Otherwise, the list of LLRs will be sorted from low reliability to high reliability. Those LLRs associate to a bit location. First $T_{\max}$ numbers in sorted LLR paired with their bit location are chosen as an array named flipping candidates.
3. SC decoding will be restarted, which means iteration from the beginning of the frame to the end. When the decoder reaches the next bit location in flipping candidate, after the hard

decision, the estimated bit value will be changed, which will also affect the rest of the frame. If it was predicted as 1, it would be flipped to 0, vice versa.

4. After flipping the candidate, the SC decoder will be continued until reaching the end of the frame. The decoder goes back to step 2 unless the number of times that this process repeated reached the $T_{\max}$, that codeword is counted as an undecodable frame.

1.2.3.1 Complexity of successive cancellation flip decoder

As long as the SNR is large enough, the SCF decoder will have a high probability of estimating the correct codeword without flipping any bit, since in better channel condition, the probability of having error in a frame gets low. In this case, the SCF execution time is the same as SC execution time. On average, SCF's computational complexity in $O(N \log N)$ as described in (Afisiadis *et al.*, 2014). The SCF decoding has a computational complexity of $O(N \log N[1+T_{max}\times\Pr(R, \mathrm{SNR})])$ (Afisiadis *et al.*, 2014), where $\Pr(R, \mathrm{SNR})$ represents the FER of a polar code with rate R for some SNR under SC decoding. Accordingly, the average computational complexity for SCF decoding is related to the average number of iterations, which itself depends on $T_{\max}$ and $Pr(R, \mathrm{SNR})$. In the worst-case scenario, there is $T_{\max}$ attempts to decode, which means the execution time complexity is $O(TN\mathrm{log}N)$. As we explained how SCF works, the number of attempts to decode a frame can be in range 1, which corresponds to SC decoder, to $T_{\max}$. This variable that varies from from 1 to $T_{\max}$ is called required trial (T). Consequently, the SCF decoder has variable execution time. When it comes to variable execution time, the worst-case scenario should be considered in hardware implementation. If the hardware is not designed and prepared for the highest possible delay, there might be a chance of packet loss. The decoder has a buffer that stored the coming frames when it is busy decoding the previous frames. If the buffer is not big enough, the system might either have packet loss or buffer overflow.

1.2.4 Related works

In this section, three relevant papers are described briefly.

In the (Afisiadis *et al.*, 2014) paper, the SCF was introduced that keeps the memory complexity as low as SC. Moreover, the average computational complexity is as low as $O(N\log N)$, which is the same as the average complexity of SC. Nonetheless, in lower SNR values the energy consumption is higher than SC decoding. Our proposed algorithm uses SCF decoder as the underlying algorithm.

In lower SNR regions, the number of iterations in SCF is high and close to $T_{\max}$. In the (Condo *et al.*, 2018) paper, two methods are proposed to alleviate the mentioned problems and reduce the average execution time: Fixed Index Selection (FIS) and Enhanced Index Selection (EIS). The former detects the indices of high likely erroneous bit locations based on the channel and SNR region. It prioritizes those bit locations when SCF chooses the flipping candidates. The latter proposed decoder suggests the idea that in some cases, the low channel LLR for a bit does not indicate the low reliability of that bit locations. This algorithm also uses the bit location and channel LLR to choose the flipping candidate. The results of their proposed solutions show adequate error-correction performance when the code rate is low with a decrease in execution time. The error-correction performance of both presented decoders depends on the channel condition.

Another article, (Giard & Burg, 2018), proposed an algorithm that combined state-of-the-art high-speed Successive Cancellation with SCF and named that algorithm Fast-SSC-Flip. They proposed mixed node types, and for each node type, a method for LLR calculation is presented to reduce the latency of computations. The Fast-SSC algorithm predicts multiple none-frozen bits simultaneously with various decoders.

The error-correction performance of the Fast-SSC-Flip algorithm can be the same as SCF decoding algorithm. Most importantly, the proposed algorithm is significantly faster than SCF. However, the error-correction performance of the proposed algorithm with SPC nodes when

$T = 8$ is less than SCF. Additionally, the throughput depends on the frozen-bit locations and node types. The algorithm cannot solve the variable execution time problem, although it has an improvement in the latency of SCF, especially in low SNR regions.

In conclusion, one of the last two papers ran the simulation offline to find the most likable erroneous bit location. The other one categorized the nodes in the decoder tree and started the decoding in parallel. Both papers proposed improvements to the SCF algorithm resulting in a significant latency reduction. Our proposed algorithm is compatible with these ideas and could be integrated in the future.

In the next section, the proposed modifications to SCF decoder and turning it to ESCF decoder is described.

CHAPTER 2

ENHANCED SUCCESSIVE CANCELLATION FLIP DECODING

2.1 Overview

As we discussed previously in section 1.2.3, for short to medium-length codewords, the SC decoding algorithm provides poor error-correction performance. A new decoding algorithm called SCF decoding has been proposed to improve the error-correction performance of SC decoder. However, SCF has variable execution time, which leads to hardware design with a worst-case latency.

In this chapter, a new decoding algorithm named Enhanced Successive-cancellation Flip (ESCF) is presented with the aim of reducing the average execution time. The idea of this decoder is skipping some steps in SCF decoder. However, there is a trade-off between the execution time and memory requirement. ESCF decoder is implemented with a slight increase in required memory. The memory layout of this algorithm is explained in this chapter. Throughout the rest of this chapter, error-correction performance and execution time differences among SC, SCF, and ESCF algorithms will be discussed after describing our proposed modifications to SCF.

2.2 Proposed modifications to SCF decoding algorithm

To remind how the SCF decoder works, at the first attempt to decode a frame, the SC algorithm is used to estimate the codeword and checks for CRC. If the CRC passes, the decoding process is finished; otherwise, the SCF decoder restarts the decoding process from the beginning, flips the flipping candidate, and continues decoding. Those flipping candidates are the least reliable ones based on the channel LLRs. This process repeats until CRC matches or the decoder reaches $T_{\max}$. More details about SCF are available in section 1.2.3.

In the best-case scenario, a frame can be decoded at its first attempt. Otherwise, the SCF repeatedly decodes and changes the flipping candidates until finding the CRC match. The number of trials can vary from 1 to $T_{\max}$. Therefore, SCF has variable execution time.

As mentioned in section 1.2.2.1, when the decoder moves in a sub-tree, it reuses the intermediate calculations for estimating the next bit in that sub-tree, while the decoder transits from left sub-tree to right sub-tree, intermediate LLRs are entirely updated. It means that for decoding bits in the right sub-tree, calculations are independent of the previous intermediate LLRs in the left sub-tree. On the other hand, after SCF chooses the flipping candidate, regardless of the candidate's position, it restarts the decoding from the beginning. However, as we will explain, it is not always necessary.

Figure 2.1 shows a decoder tree for decoding a 4-bit frame. Assume the flipping candidate is u_2. Knowing that the decoder already has the estimated value for u_0 and u_1 in its memory as well as the channel LLRs, the decoder does not need to restart the decoding from the root of the tree. It can directly resume from u_2, calculate the necessary intermediate values for that bit and continue to the end of the frame.

In this step, to decode the right sub-tree, the input values for G function are needed which means decoder needs the result of combination operation from previous bits. When decoding is started from the beginning, the input values for G functions are calculated and stored in a separated memory. In the other word, these input values are calculated and stored during the first trial. In the Figure 2.1, at the Step 2, the values L_0 and L_1 are filled with $G(y_0, y_2, PS[0])$ and $G(y_1, y_3, PS[1])$ respectively where $PS[0] = \hat{u}_0 \oplus \hat{u}_1$ and $PS[1] = \hat{u}_1$. As we showed that the bit values in the right sub-tree are independent of the intermediate values, also channel LLR and previous hard decided values are enough to decode the bits located in the right sub-tree. We use this fact in our suggested algorithm.

We showed that the calculations in the right sub-tree are independent of the intermediate LLRs calculated in the left sub-tree. Therefore, after the first attempt to decode the frame and failing to match the CRC, if a flipping candidate is in the right sub-tree, the decoder can restart the decoding from the middle of the frame. In the proposed algorithm, the location of the bit flip candidate determines whether the decoding process needs to restart from the beginning of the middle. The middle of a frame is called target and defined as $target = N/2$. When the ESCF starts decoding from the middle of the frame, the input values for G functions are needed. We assume that while the first iteration of decoding is in process, the input values for the first G function for the right sub-node are stored in an additional memory named PS with the size of $^N/_2$. Adding this memory is the slight sacrifice in terms of memory requirement that might eventually lead to a decrease in execution time.

As described in section 1.2.2.1, a frame with size of N needs a memory size of $(2 \times N) - 1$ including channel LLRs and intermediate LLRs. Besides, a memory with size of N is needed to store the hard decisions. Therefore, the total size of a memory with ESCF is $(2 \times N) - 1 + N + {^N/_2}$ while the total size of required memory for SCF is $(2 \times N) - 1 + N$. In other words, both algorithms' memory sizes are $O(N)$. Note that each LLR cannot be stored in a bit memory. Assume that each LLR can be stored in Q bits which equals to 6 bits (Giard *et al.*, 2017). Therefore, the memory size that ESCF requires is $Q \times (2 \times N) - 1 + N + {^N/_2}$. However, to simplify our computations, this value is ignored since it does not impact our calculations.

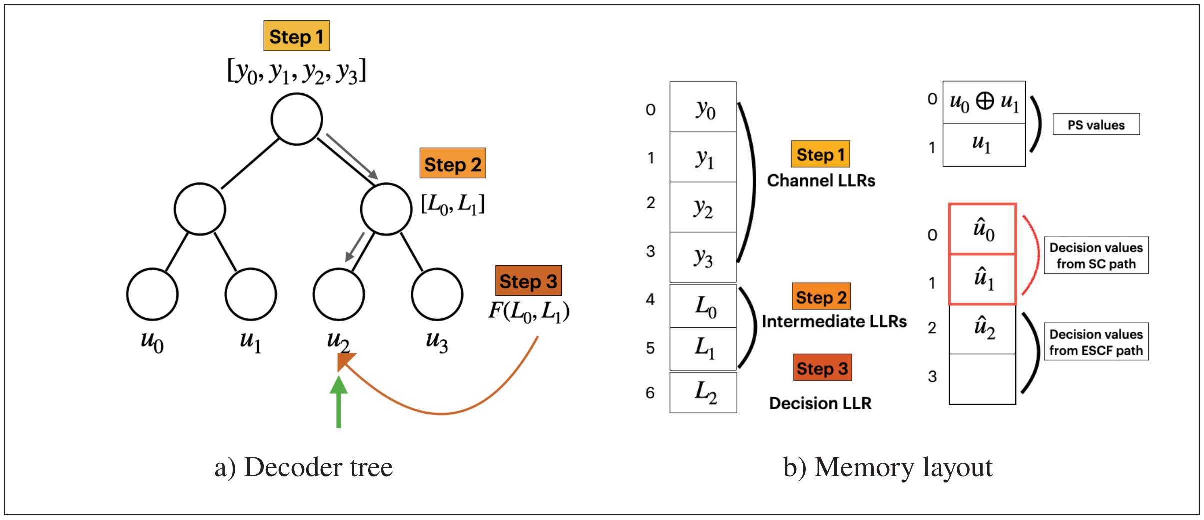


Figure 2.1 4-bit ESCF decoder - Flipping candidate = u_2

The ESCF algorithm has 4 steps:

1. At the first step, the decoder iterates the frame, same as SC decoder. In this stage, the decoder's memory complexity is $O(N\log N)$ to keep channel LLRs at all times and rewrite intermediate LLRs for each bit. Moreover, the PS memory is filled with the required input of the G function for the right sub-tree. If the CRC doesn't match, the decoder goes to step 2.
2. In the second step, the ESCF decoder sorts the list of decoder's absolute LLR values based on their reliability. The frozen bit locations are set at 10,000, which is a very big number comparing the regular LLRs in this list with the purpose of not affecting the sorting process. The first T items are chosen as flipping candidates.
3. Depending on the location of the bit flip candidate, the decoder decides whether to restart the process from the beginning or the middle of the frame. If the flipping candidate location is bigger than the $target$ location, the decoder resumes decoding from the target in the decoder tree. Otherwise, the decoder starts decoding from the beginning.
4. In this step, if CRC matched, the decoding process is finished. Otherwise, the decoder goes to step 3. This process continues until reaching to the $T_{\max}$ or CRC pass.

This algorithm reduces the average execution time by using the available intermediate calculation results stored in the memory. On the other side, the execution time reduction cannot be gained for free. As described in the Equation 1.4, G function needs another set of inputs that are calculated in the first iteration and stored in the PS memory. This combine operation to make the inputs ready for G function is common among SC, SCF, and ESCF. Therefore, instead of recalculating the intermediate variables, in the first iteration of decoding the frame, the required inputs for the first G function of the right sub-tree are kept in the new dedicated memory, and the combination operation is still required to prepare the 3rd input value for G.

We hypothesize that this modification should reduce latency without affecting the error correction performance with a small sacrifice on memory requirements.

2.3 Methodology

The goal of the proposed ESCF decoding algorithm is to reduce the execution time without sacrificing the error-correction performance. To show the effect of the described modification in the section 2.2 a simple communication system is implemented based on Figure 1.1. However, to simplify the implementation, the source encoder and decoder are not implemented. Instead, the information bits are generated from a random source with a uniform distribution.

1. Simulate SC, SCF, and ESCF on same data-set to compare error-correction performance of these algorithms.
2. Estimate the execution time in terms of number of clock cycles for SCF and ESCF to compare their execution time. We showed that a list of F and G functions are needed to calculate each bit of a frame. To estimate the number of clock cycles required, the number of operations, a counter, count how many operations including these two functions are needed, assuming that both functions need the same number of clock cycles. It means completing these two operations costs equally for the processor in our assumption. There's another operation that combines the partial-sum as input for the G function. However, this operation is done in both SCF and ESCF the same number of times. Therefore, to simplify the calculations, the cost of combination operation for third input of G function is not considered.

   Moreover, 10,000 simulations have been done in different SNR points to see if different E_b/N_0 values can have an impact on the number of clock cycles. As channel condition is improved, fewer and fewer trials are required to decode a frame. Consequently, the SNR value can have a high impact on the number of required clock cycles in both SCF and ESCF decoding algorithms. Three different code rates including $1/3$ as low code rate, $1/2$ as moderate and $2/3$ as high code rate were investigated to cover most of the cases.

This simulation implementations consists of various components including:

- **Initialization:**

  Initialization is the first step. Information bits are set randomly, and frozen bits are set to zero.

- **Encoding:**
  In this part, the frame produced in the initialization section is given to the encoder. Based on the explanation in section 1.2.1, the frame is encoded. This module can generally encode bits ranging from 8 to 1024 bits, as demonstrated in the test cases.
- **Modulation - Passing through the Channel - Demodulation:**
  Binary Phase Shift Keying (BPSK) modulates the encoded frame and passes it through the AWGN channel. The Signal Noise Ratio (SNR)(E_b/N_0) determines noise level in AWGN channels. E_b/N_0 represents the energy per bit compared with noise power density, which is a normalized SNR. E_b is the signal energy corresponding to each user data bit. In other words, it is equivalent to the signal power divided by the user bit rate (not the channel symbol rate), also N_0 is the noise power per unit of bandwidth. When a channel's E_b/N_0 value is higher, fewer bits will get changed accidentally. After carrying through the channel, the demodulation step is done and channel LLRs are ready to be decoded.
- **Decoding**:
  Our goal in this section is to decode noisy codewords and fix incorrect bits. Therefore, the output of the detector is passed to the SC, SCF, and ESCF. In this step, the decoder returns the estimated frame, as well as the number of total required clock cycles to decode that frame.
- **Comparison:**
  In this step, the comparison module compares the input and output of the decoder and calculates the FER in 10,000 simulation for SC, SCF, and ESCF. Moreover, it calculates the average clock cycle of SCF and ESCF clock cycles per frame in 10,000 number of simulation. The number of clock cycles for SC is also calculated to compare with these two flipping based decoders. Since both flip-based decoders use SC as their core, the latency of SC is expected to be lowest among the three.

Table 2.1 Simulation configuration table

Polar decoding algorithms : SC - SCF - ESCF			
Channel	AWGN		
Modulation method	BPSK		
Codeword length	256 bits		
Code Rate	1/3	1/2	2/3
$T_{\max}$	8		

Python (Rossum, 1991) is used in this implementation, along with the PyCharm IDE (Dmitriev & Kipyatkov, 2011). Matplotlib (Hunter, 2012) was used to create the plots, and Numpy (Hugunin, 2005) is consumed for vector operations to speed up calculations. According to the configuration Table 2.1, simulations were performed 10,000 times with different random data sets.

2.4 Simulation Result

It is expected that the proposed algorithm should have the same error-correction performance as SCF. To prove that, first, we show the FER in all three decoding algorithms and later in different SNR points. In addition, ESCF should have a smaller number of clock cycles than SCF.

The results are divided into two categories. In the first section, it is demonstrated whether the proposed modification affects the error-correction performance or not. In the second part, the improvement of execution time in the ESCF is discussed.

Error-correction performance comparison

Three different code rates were studied to compare the error-correction performance of SC, SCF, and ESCF. The block length is kept constant at 256 bits, and $T_{\max} = 8$.

The error-correction performance is measured by reaching the lower FER in the lower SNR value. It means at the same channel condition, each of them provides smaller FER, has the better error-correction performance.

Figure 2.2 is a line graph that shows the comparison between error-correction performance of SC, SCF, and ESCF with a code rate $CodeRate(R) = 2/3$, which is considered a high code rate. The FER of three algorithms are shown in the Y-axes, while the X-axes shows different SNR values used in the experiments. The higher values in the Y-axes show the higher number of incorrect frames. On the X-axes, when the data point approaches the right, it indicates the higher signal power, which means the less noisy channel and fewer errors in the signals that enter the decoder. Our tendency is reaching to lower FER when the channel condition is in worse situation (lower SNR value). The reason behind this tendency is lower SNR costs are less expensive. As presented in the graph, as we raise the SNR (E_b/N_0 dB), the proportion of erroneous frames decreased as expected. Our goal is to create an algorithm that can have a smaller FER in worse channel conditions.

The goal of SCF and ESCF is having better error-correction ability than SC which is came true. For instance, while using the SCF and ESCF decoders, the FER 10^{-2} can be achieved when $E_b/N_0 = 3.5$ dB approximately. The same FER while using the SC decoder can be gained when $E_b/N_0 = 4$ dB. Moreover, this line graph shows that SCF and ESCF have overlapped at all SNR values, which means the proposed modification did not affect the error-correction performance as it was expected.

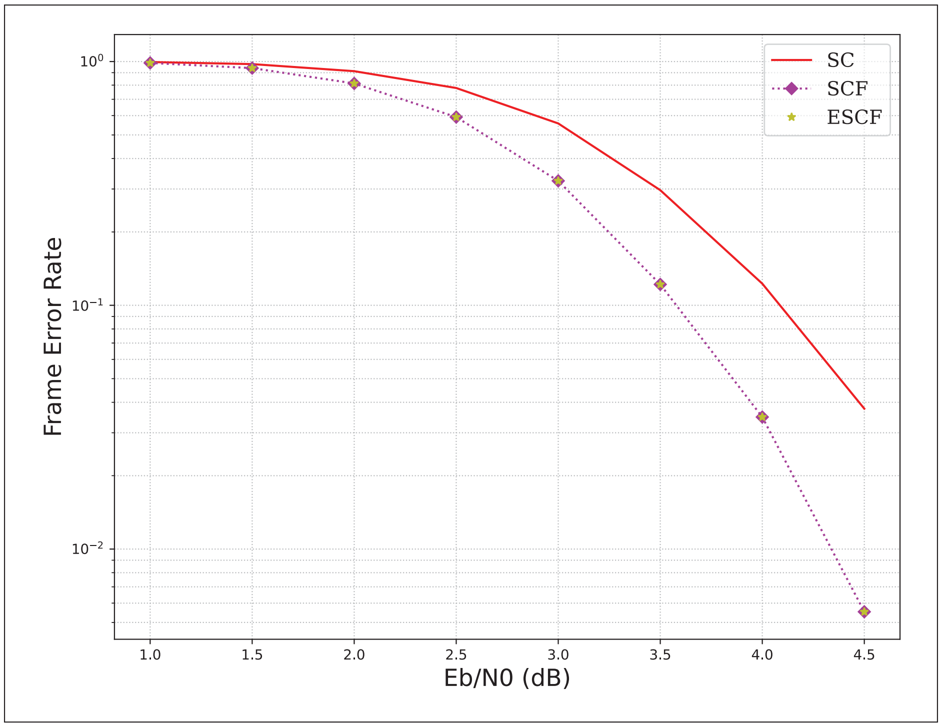


Figure 2.2 Comparing error-correction performance among SC, SCF, ESCF for $T = 8$, $N = 256$, $R = 2/3$

In Figure 2.3, the FER is compared among three decoding algorithms including SC, SCF, and ESCF when the code rate $R = 1/2$. It can be observed that a FER of 10^{-2} is reached at $E_b/N_0 = 2.5$ dB with the SCF or ESCF decoding algorithms.

In contrast, E_b/N_0 should be about 3.2 dB for SC to get the same FER. It means our proposed algorithm provides improved error-correction performance compared to SC. Again, as expected, the line graphs show that error-correction performance of the SCF and ESCF decoders are the same since both line graphs corresponding to these decoders have overlap.

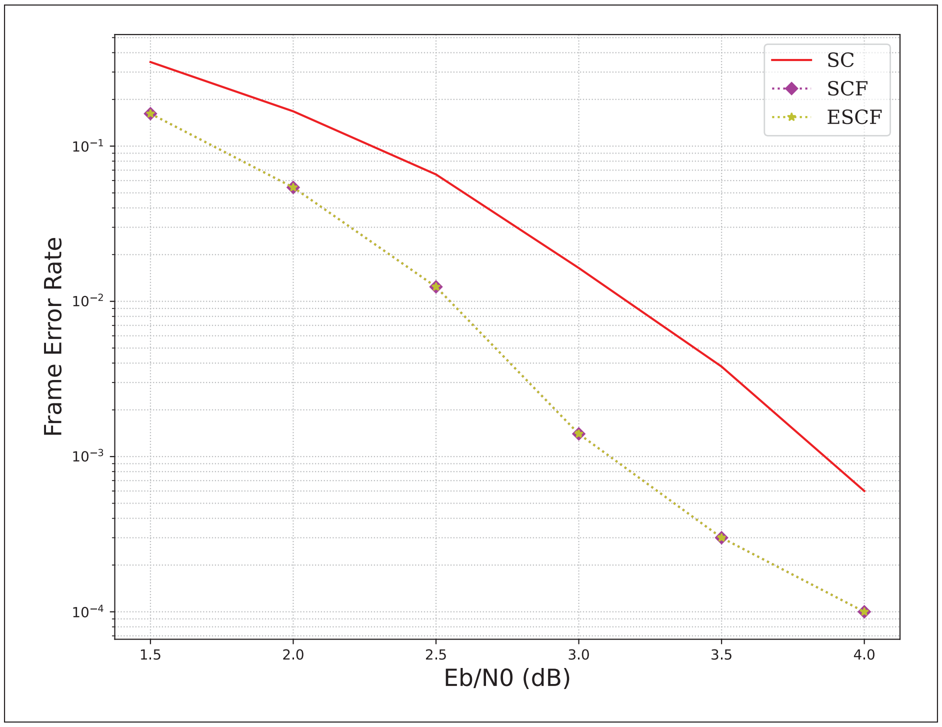


Figure 2.3 Comparing error-correction performance among SC, SCF, ESCF for $T = 8$, $N = 256$, $R = 1/2$

Lastly, Figure 2.4 displays the error-correction performance of the SC, the SCF, and the ESCF decoder for $R = 1/3$, which is considered a low code rate. Both of SCF and ESCF error-correction performance are higher than SC which is supposed to be, and again ESCF's error-correction performance is the same as SCF.

Eventually, these figures showed that considering the high code rate and low code rate, the modification that was made to SCF did not sacrifice the error-correction performance. Next, we'll compare how many clock cycles each algorithm requires to decode a frame in different SNR values.

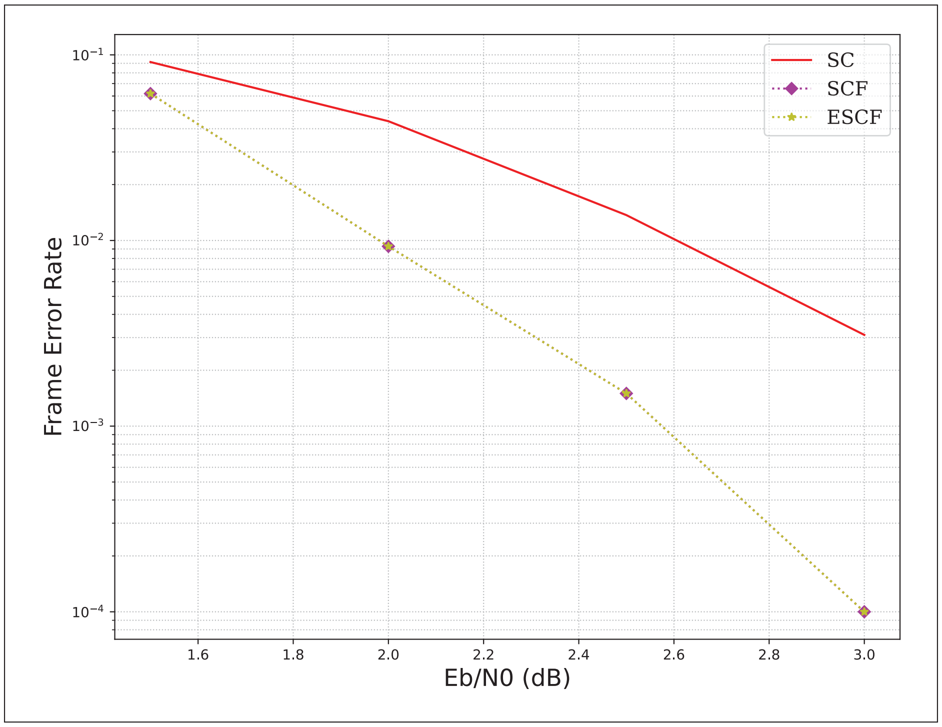


Figure 2.4 Comparing error-correction performance among SC, SCF, ESCF for $T = 8$, $N = 256$, $R = 1/3$

Execution time comparison

As previously discussed, the number of computed clock cycles is an indicator of each algorithm execution time. In the experiments to calculate the execution time, the average clock cycle of SC, SCF, and ESCF decoding algorithms are calculated in 10,000 simulations and different SNR points. To have a better comparison, those numbers are normalized. It means that the highest number of average clock cycles corresponds to 1, and the rest of the values are normalized based on this maximum execution time.

The bar graph in the Figure 2.5 compares the normalized execution time of three algorithms, SC, SCF, and ESCF, to decode a 256-bit frame with $R = 2/3$. To have a fair comparison, we have used the same noisy codewords in our simulations.

It is shown that the normalized execution time of the ESCF decoder is lower than the normalized execution time of the SCF decoder. As a result, the latency of the ESCF decoder is lower than SCF. Additionally, as the SNR value increases, we can see a reduction in the execution time in SCF and ESCF decoders. This is expected as the probability that an error will occur decreases as we increase the signal's power. When the SNR has a higher value, the channel condition is better, which means the frames are less noisy, and the probability of having to flip a bit to decode a frame correctly gets smaller. Consequently, the execution time for both algorithms is getting lower. As a result, there is a smaller gap between the clock cycles of these algorithms in higher SNR values. The gap between the normalized execution time of ESCF and SCF ranges from 22% to 10% for SNRs from 1 dB to 4 dB. Figure 2.5 also includes the normalized execution time of SC as a reference to show that our proposed algorithm has a closer execution time to SC decoder compared to SCF decoder. As shown in the figure, the execution time of SC decoder is constant in all SNR points since the SC decoder does not have a variable execution time.

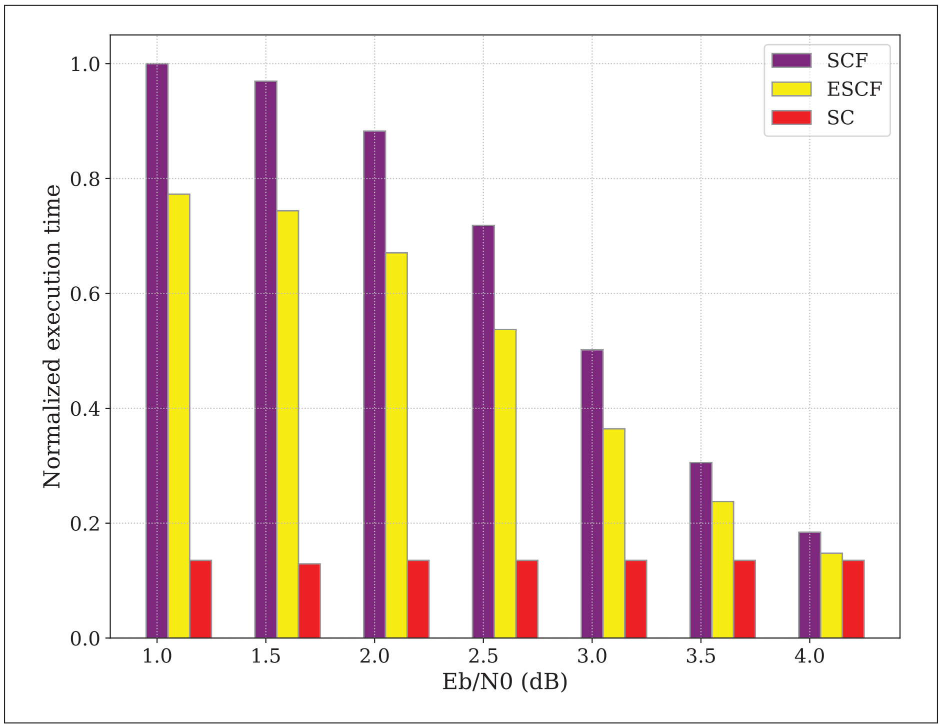


Figure 2.5 Comparing the normalized execution time between SC, SCF and ESCF for $T = 8$, $N = 256$, $R = 2/3$

Figure 2.6 shows the normalized execution time for the SC, SCF, and ESCF when the $R = 1/2$ and block length is 256 bits. At all E_b/N_0 values, the normalized execution time of ESCF is lower than SCF, which means a lower latency. Moreover, the gap between these execution times for SCF and ESCF is lower in higher SNR points, for the same reason of decoding the frame with $R = 2/3$ as described. By increasing the SNR, the number of attempts to flip bits is reduced, so the mentioned gap between these algorithms' execution time is getting smaller. This gap ranges from 21 % to 5 % approximately. When code rate $R = 2/3$, the lowest normalized execution time is 10% which is higher than the lowest normalized execution time when $R = 1/2$. The reason is as much as the code rate is higher, the probability of having an erroneous frame gets higher. Consequently, the probability of requiring flipping a bit gets higher, and the execution time increases. Again, the execution time of SC is constant at all SNR points for the same reason in the previous figure. Moreover, the execution time of SC decoder is smaller than both SCF and ESCF execution times, although ESCF's execution time is closer to SC execution time.

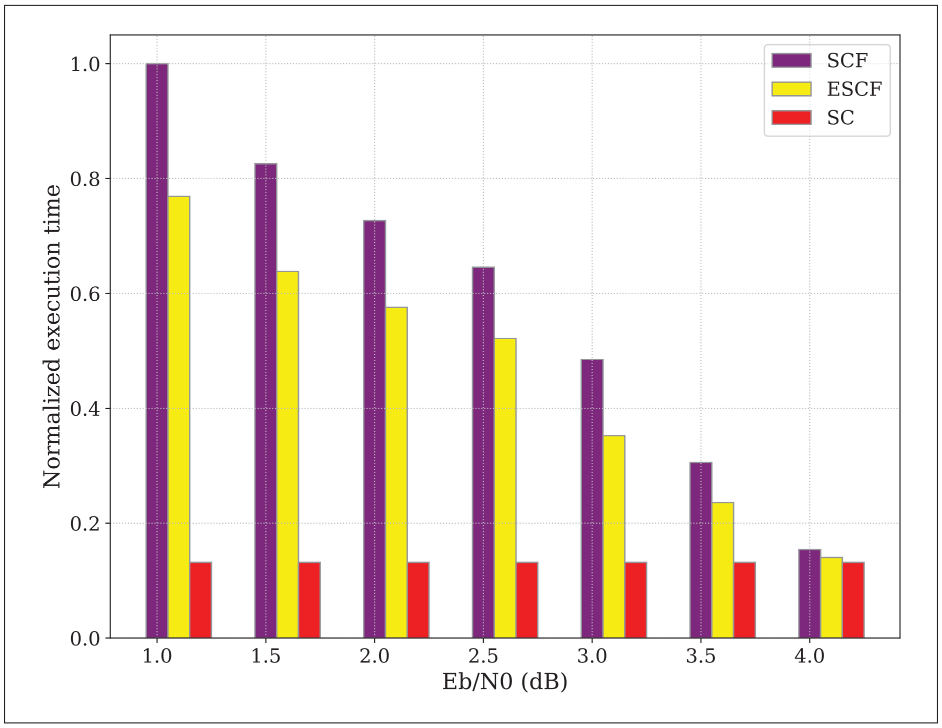


Figure 2.6 Comparing the normalized execution time between SC, SCF and ESCF for $T = 8$, $N = 256$, $R = 1/2$

A final comparison of the normalized execution time between the SC, ESCF, and SCF can be seen in the Figure 2.7 for code rate $R = 1/3$. The ESCF has smaller execution time comparing to SCF continuously. In addition, the gap between the execution time of ESCF and SCF frame decreases with an increase in E_b/N_0. This difference ranges from 17 % to 1 % approximately. As previously mentioned, as much as a code rate is smaller, the probability of having error in frame get smaller. This fact explains why the lowest normalized execution time when $R = 1/3$ is smaller than the situation when $R = 1/2$ and $R = 2/3$. In this figure, the SC decoder's execution time is constant for the same reason explained in Figure 2.5. Again, our proposed algorithm's execution time is closer to SC execution time.

To sum it up, these results showed that in three different code rates, the normalized average execution time of decoding a frame when ESCF is the decoder is smaller than SCF in all SNR points. However, as much as the channel condition gets better, this gap gets lower. Another result that is worth mentioning is a trend of having shorter execution time in lower code rates. It means in a situation that execution time plays a very important role, a lower code rate can be beneficial. We also showed that SC decoder has a constant execution time.

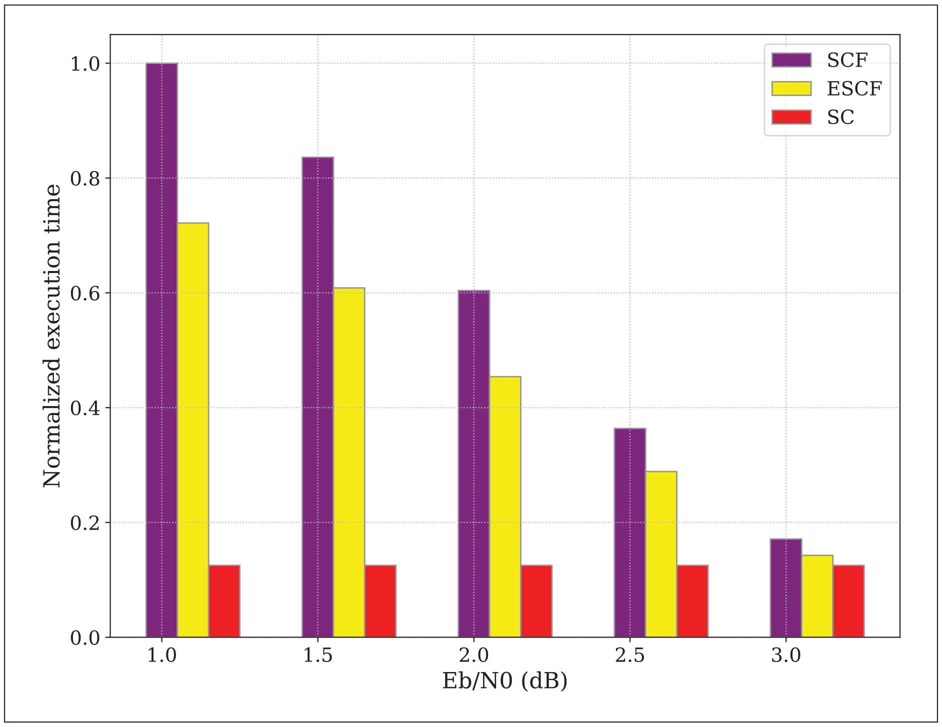


Figure 2.7 Comparing the normalized execution time between SC, SCF and ESCF for $T = 8$, $N = 256$, $R = 1/3$

2.5 Conclusion

The previous section examined the error-correction performance of the proposed algorithm for three different code rates which $R \in \{2/3, 1/2, 1/3\}$ and compared it with SC and SCF algorithms. Our results show that not only the time complexity of SCF has been reduced, but the error-correction performance remains exactly the same as SCF.

Studying the execution times of the SC, SCF, and ESCF decoders with the same code rates and configurations, we found that ESCF decoders are faster than SCF decoders at all SNR points and all rates. Moreover, its execution time is closer to SC execution time. Furthermore, we showed that when the code rate is higher, the gap between average execution time of these two flipping decoding algorithm is higher. Our results also showed that in smaller code rates, the normalized average execution time is lower in both SCF and ESCF decoding algorithms.

2.5.1 Future works

To have a suggestion for the future works, as described in section 1.2.4, our solution can be integrated with the idea of (Condo *et al.*, 2018). It is suggested that first, an offline simulation is run, and some bit locations should be chosen as more likely to carry erroneous information. After simulating with (Condo *et al.*, 2018) presented decoder, the rest of the decoding can be done based on ESCF decoder. Moreover, our decoding algorithm is compatible with (Giard & Burg, 2018). When the nodes are categorized and decoding gets started, ESCF decoder can decide which node is better candidate to restart in case the CRC could not get matched.

CHAPTER 3

LOW-COMPLEXITY SYSTEM TO ESTIMATE THE PROBABILITY ASSOCIATED WITH A GIVEN NUMBER OF TRIALS IN SCF

3.1 Overview

As discussed in section 1.2.3, a new flavor of polar decoders named SCF was proposed to improve the error-correction performance of SC. However, SCF is an iterative decoding algorithm that will conduct up to a predefined Maximum number of trials (T_{max}) in an attempt to decode a noisy codeword. Thus, it is an algorithm that exhibits a variable execution time, making its integration into receivers more challenging than deterministic algorithms.

This chapter describes results on the investigation on various factors that might affect the variability of the execution time, which depends on the number of trials (T) in the SCF decoder. Generally speaking, when it comes to hardware implementation, variable execution time is not ideal because hardware should be designed and prepared for the worst-case scenario.

The objective in this chapter is to identify the influential code parameters and channel conditions on the number of required trials in various configurations. Those findings will eventually lead to a low-complexity estimation system for required trials, given the condition.

First, the number of necessary trials to decode a frame under certain conditions is investigated. Hence, a series of experiments are run to identify the impact of the block length, code rates, CRC length, and FER on the required number of trials.

Finally, based on the result of the experiments and built statistics, a lookup table is generated, which associates each number of trials with a probability. The lookup table generator can create the table using the channel conditions and code parameters.

This lookup table can be used in future works as a building block in the communication system that can predict the decoder delay.

In the following chapter, first, our methodology will be described, followed by a description of the simulation setups and their results. Afterward, the code structure that leads to the lookup table accompanied by a few recommendations for the future is explained.

3.2 Methodology

The first step is to establish the impact and relationship of various code parameters on the required trials. These code parameters are block length, code rates, CRC length, and for sure $T_{\max}$. Additionally, the effect of the channel condition is examined using the FER parameter. The goal of this investigations was described in section 3.1, which is preparing the base layer to create the mentioned lookup table.

To find the impact of those variables, multiple experiments have been performed. In each experiment, we keep the parameters constant except one of them to observe the effect of that parameter on the number of required trials. Note that the error-correction performance in all experiments was kept approximately the same for fair comparison around 3×10^{-2}. Note that FER is not controllable in our simulations. For each configuration, the channel condition is the control module that helped us reach that specific FER.

Afterward, the second step is visualizing the number of required trials based on each factor to see the relationship. Visualization helps us to see the effect of each parameter on the required trials and identify the pattern. Using the MATLAB simulations and plotting the results, it can be seen that how significantly each variable affects the value of T. Note that, for each simulation, a list with size of $T_{\max}$ +1 is created. The decoder and encoder of the system are written in C language to minimize the simulation time.

Therefore, the base layer of creating the lookup table is prepared. As previously discussed, the contribution aims to provide an early-estimation system that can associate a probability to each number of required trials based on the given condition. These conditions are block length, code rate, and channel condition. The description of simulations is brought in the next section.

3.3 Simulation

To build the delay estimation system, the first step is running the experiments to see the effect of each code parameter and channel condition. The second step is to prepare the data to save in the lookup table and later use that data. In this section, the simulations setup are discussed, followed by the result of those simulations. The code parameters that are under investigation include block length, code rate, and CRC length. On the other hand, the effect of channel condition is another subject to review, which is measured by FER of the simulation.

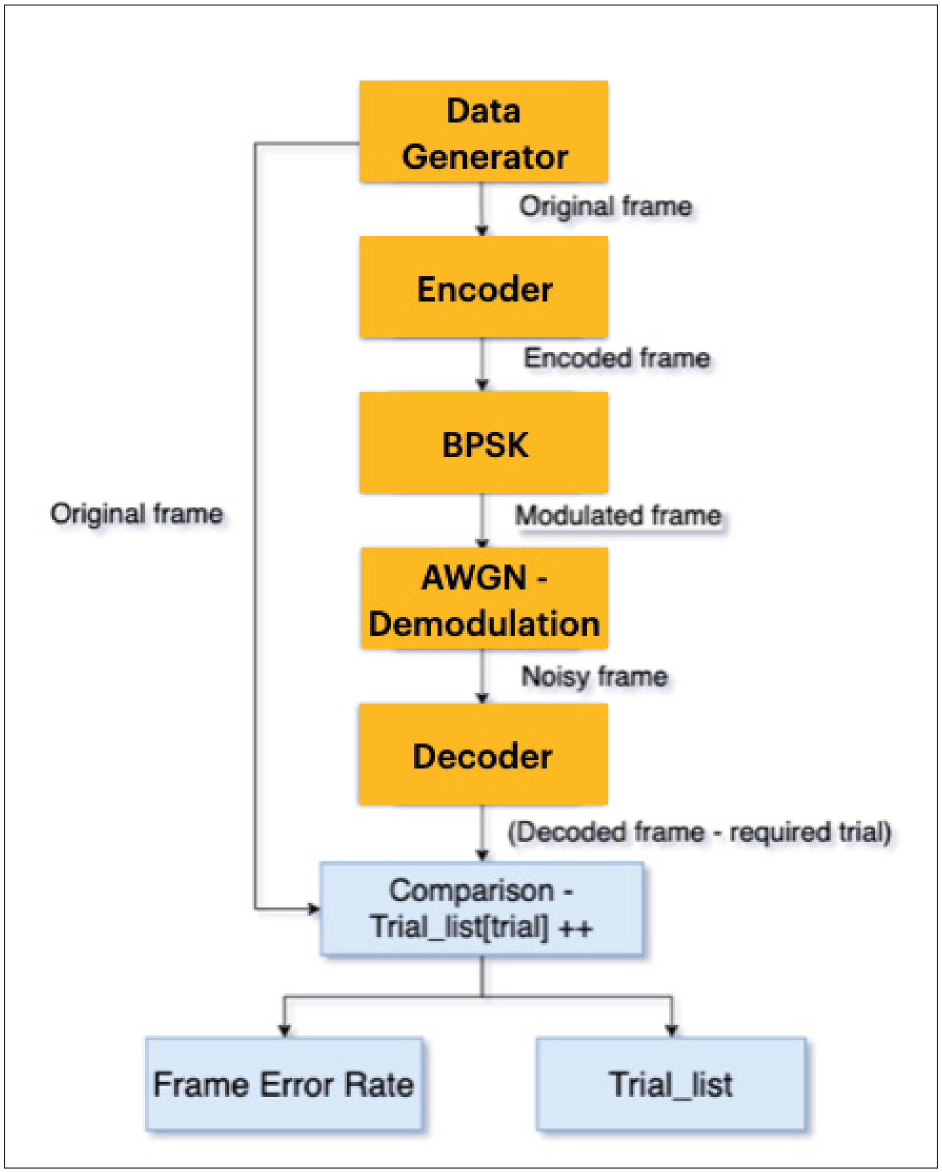


Figure 3.1 Simulation setup for running experiments researching on block length, CRC length, Code Rate, $T_{\max}$, and FER

Figure 3.1 shows one iteration of simulation. To have a fair comparison, this simulation is repeated 100,000 times with random data sets.

Based on the Table 3.1, the configuration file is prepared with the block length $N = 512$ and $R = 1/2$. As can be seen in the Figure 3.1, the decoder returns the decoded frame as well as the trial needed to decode that frame.

A list of trials with the size of 11 is created to store the number of times each trial is required to decode the frame. The last number in the list with index = $T_{\max} + 1$ shows the number of failed decoded frames after $T_{\max}$ attempts. Afterward, this procedure is repeated for block length = 1024, and another Trial_list is created to be used later.

Table 3.1 Variable block length

Block length	1024 bits	512 bits
Message length	512 bits	256 bits
$T_{\max}$	10	
Targeted FER	10^{-2}	
CRC length	8 bits	

Table 3.2 shows the configuration for the second round of simulations. Three different code rates are under investigations to see if this code parameter can effect the required trials distribution. According to Figure 3.1, the simulation is running where R is $1/2$, $1/3$, and $1/4$ respectively. For each of those configurations, Trial_list is stored separately.

Table 3.2 Variable code rate (message length)

Block length	1024 bits		
Message length	512 bits	341 bits	256 bits
$T_{\max}$	10		
Targeted FER	10^{-2}		
CRClength	8 bits		

Afterward, the simulation is run based on the Table 3.3 to see the effect of different CRC length and polynomials. Three different CRC polynomials were investigated based on the Table 3.4. For each of those configurations, the Trial_list is saved separately.

Table 3.3 Variable CRC length

Block length	1024 bits		
Message length	512 bits		
$T_{\max}$	10		
Targeted FER	10^{-2}		
CRC	8 bits	10 bits	16 bits

Table 3.4 CRC Polynomials

CRCLength	**CRCPolynomial**
8	$x^8 + x^7 + x^4 + x^2 + x + 1$
10	$x^{10} + x^9 + x^5 + x^4 + x + 1$
16	$x^{16} + x^{10} + x^8 + x^7 + x^3 + 1$

Another set of simulations was done to see the effect of various $T_{\max}$ values on trial distributions. Therefore, according to Figure 3.1, the simulation was first run where $T_{\max} = 5$. Note that Trial_list's size is 6. Then, based on the Table 3.5, the $T_{\max}$ set to 10 to fill the Trial_list with size of 11. Finally, another round of simulations is run where $T_{\max} = 15$ to see the distribution of required trials. In the last configuration, the Trial_list's size is 16.

Table 3.5 Variable $T_{\max}$ values

Block length	1024 bits		
Message length	512 bits		
$T_{\max}$	5	10	15
Targeted FER	10^{-2}		
CRClength	8 bits		

After investigating the impact of different code parameters, we should consider channel conditions to observe whether they can affect the required trials to decode a frame. To be able to measure that, the simulation is run on various SNR points, and it stops when it reaches a specific FER. Table 3.6 shows the different setups for the configuration file.

Table 3.6 Variable Frame Error Rate (FER)

Block length	1024 bits		
Message length	512 bits		
$T_{\max}$	10		
Targeted FER	0.89	0.55	0.026
CRC length	8 bits		

In the simulation setup, the decoder and encoder are written in C language for faster execution whereas the simulation was on the MATLAB platform. To be able to call functions that are written in C by MATLAB simulation, the MEX function is used (MathWorks, 2019). MEX is a program that is automatically loaded and can be called as a MATLAB function.

After finishing the simulations based on the tables above, several Trial_list exist. Those lists will be used to create the lookup tables. We present the simulations results corresponding to each of the setups described above. Later, the detail of the lookup tables is explained.

3.3.1 Simulation Results

Figure 3.2 shows how different block lengths affect the probability distribution of different trials. The X-axes represent the number of trials from 1 to $T_{\max}$. The value of $T_{\max}$ has to be a value smaller than the number of information bits and in these experiments, $T_{\max} = 10$. The Y-axes show the probability of required trials. For example, when X-axes show the $T = 1$, the Y-axes show 99%. It means 99% of the times frames can be decoded at the first attempts given the condition based on the Table 3.1. Additionally, the last number on the X-axes represents the failure rate of decoding frames. As can be seen in this figure, both graphs have the same probability at this X-axes value. The reason is they both have the same FER = 0.035, and that number shows that 3.5% of the time, the codes are undecodable with $T_{\max} = 10$.

The general tendency of the graphs shows that in most of the cases, with these configurations, SCF decoder is able to decode frames in the first trial. For example, the percentage of requiring ten trials to successfully decode a frame when $N = 1024$ is less than 0.1%.

Based on this figure, the block length impact on the distribution of the trial is considerable. As an example, the probability of having $T = 2$ for $N = 1024$ is approximately 0.035, while this number of $N = 512$ is 0.022. As a result, it can be said that the probability of a different number of trials can get changed with different packet lengths. It can be seen that when $T \geq 4$, the probability of having higher trials is bigger when $N = 512$ rather than $N = 1024$. The reason is when the block length is bigger, the probability of transmitting correct codes get higher which means the decoder needs smaller number of attempts to decode a frame correctly.

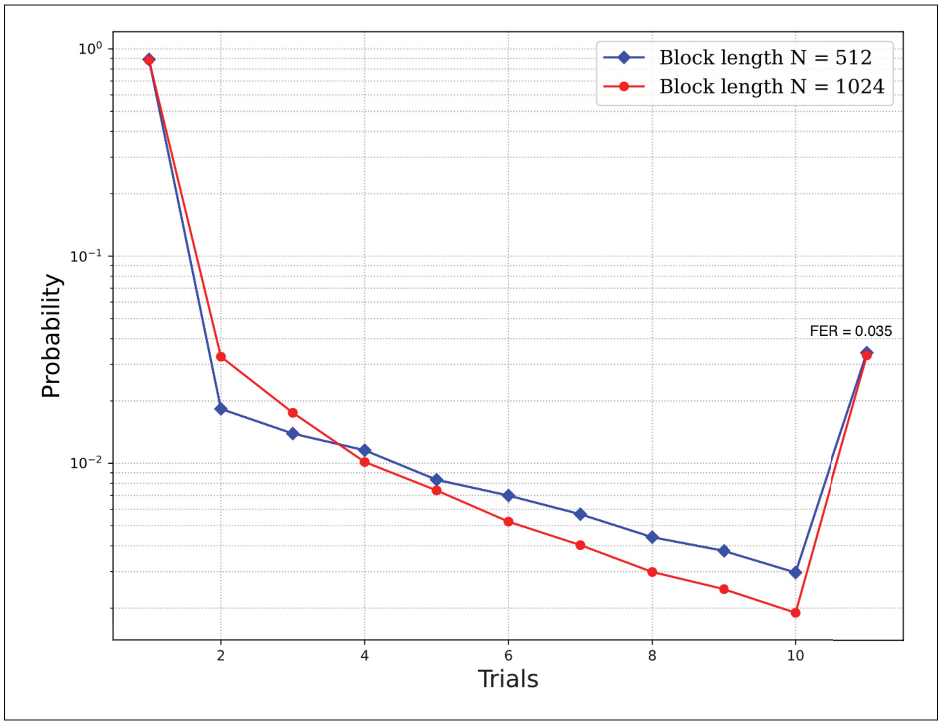


Figure 3.2 Block length effect on probability distribution of T, $N \in \{512, 1024\}$, $CRC = 8$, $T_{\max}$= 10, R = 1/2, FER = 10^{-2}

In Figure 3.3, it can be seen the effect of different code rates on the probability distribution of different trials. The last index of X-axes indicates the likelihood of frames that are not correctly decoded. Accordingly, the configuration is determined by Table 3.2. The figure shows that the code rate does not significantly have an impact on the distribution of the trial. There are some gaps when $T = 2$. At the higher value of trials, the three graphs are highly overlapped. Although the effect of code rate on trial distribution is not considerable, there is still some gap.

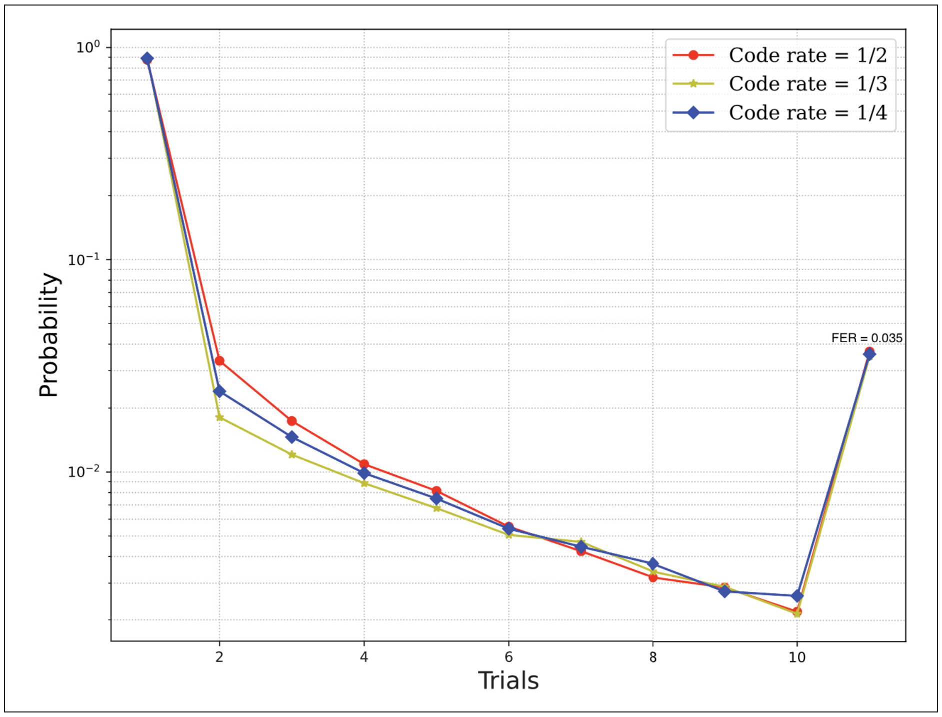


Figure 3.3 Code rate effect on probability distribution of T, $N = 1024$, $CRC = 8$, $T_{\max} = 10$, $R \in \{1/2, 1/3, 1/4\}$, FER = 10^{-2}

Another code parameters is CRC length. Figure 3.4 illustrates the effect of different CRC lengths on the probability distribution of different trials. The last index of the X-axes in this graph shows the frames that failed to be decoded correctly. Table 3.3 determines the configuration that leads to these results. As the figure shows, up to this point, this factor has the least impact on the trial distribution, with a great deal of overlap between the three line graphs. It means that the probability of a different number of trials does not change significantly based on the length of CRC. Consequently, the CRC length does not significantly affect the execution time for SCF. In the polar code with configuration CRC = 10, the FER is a bit more about 0.01%. This can explain the gap between those graphs when $T = 9$.

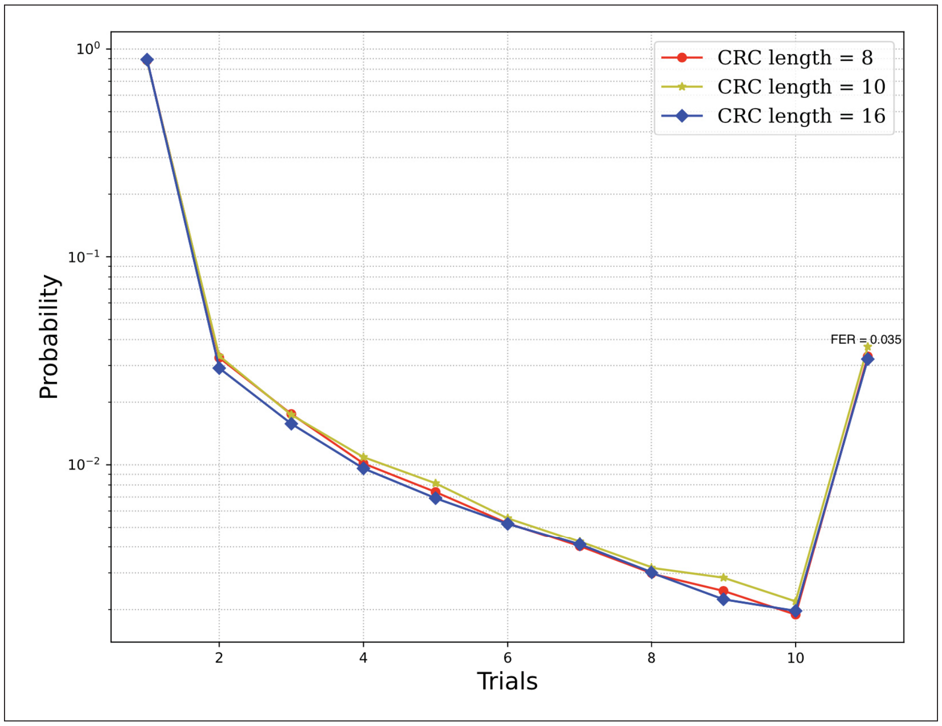


Figure 3.4 CRC-length effect on probability distribution of T, $N = 1024$, $CRC \in \{8, 10, 16\}$, $T_{\text{max}} = 10$, $R = 1/2$, FER $= 10^{-2}$

Next, we investigated how T_{max} can change the trial distribution pattern. In Figure 3.5, the impact of different maximum number of trials is shown, and the configuration table is in Table 3.5. This comparison indicates how the distribution looks based on four different T_{max} in the given figure. As can be seen, there is a pattern associated with the T_{max}. For example, the probability of being able to decode a frame on the first trial ($T = 1$) is approximately of 90% regardless of the value of T_{max}. It is been decided to put the probability of undecodable frames on the $T = T_{\text{max}} + 1$ slots. It can be seen that those values are approximately the same, because the targeted FER was also the same.

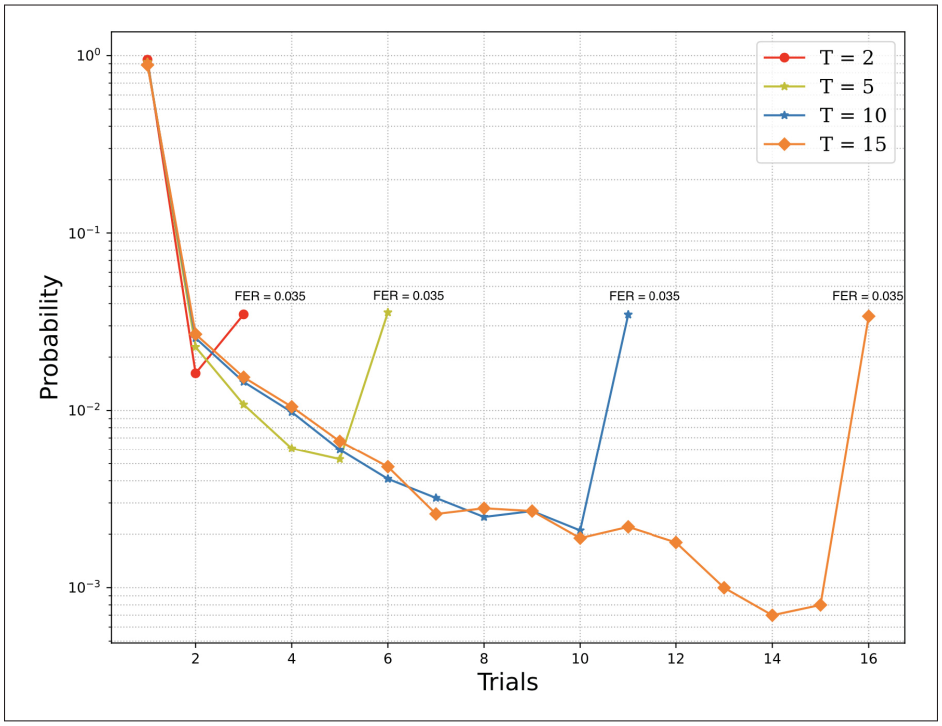


Figure 3.5 Effect of variable $T_{\max}$ values on trials distribution, $N = 1024$, $CRC = 8$, $T_{\max} \in \{5, 10, 15\}$, $R = 1/2$, FER = 10^{-2}

On Figure 3.6, the channel condition effect is under investigation, which is measured by FER. It can be seen that the distribution of trials will change significantly once other variables are kept constant except FER. It is worth mentioning that it is not possible to directly control and change the FER value. This value is controlled by changing the channel condition (E_b/N_0). The simulation is done based on Table 3.6. Note that for FER = 0.026, 100,000 simulations were not enough to get a smooth curve. We increased the number of simulations to 150,000 to get this curve. When FER is higher, it means the number of attempts to flip a bit is higher than 1. Therefore, the probability of $T = 1$ is smaller comparing to lower FERs. If the FER is as small as 10^{-2}, then the probability of requiring more trials is significantly smaller. It means that the channel condition is better that leads to a low FER and the chance of needing to flip more than a bit gets smaller.

These experiments lead to creating a lookup table that can provide the probability that a given number of trials will be required based on the code parameters and channel conditions. As we described, block length and code rates are more effective than CRC length. Therefore, in the lookup table, the length of the frame and the rate of code are considered. Moreover, we showed that channel condition (E_b/N_0) is another influential variable in the trials probability distribution. Consequently, the FER is regarded in the lookup table as well.

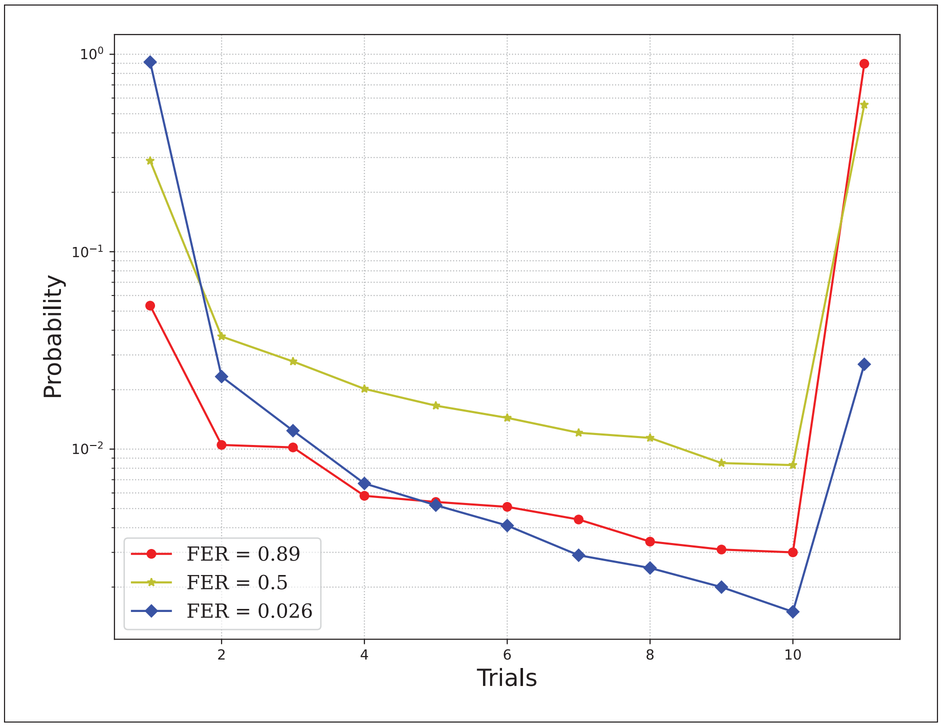


Figure 3.6 Effect of variable FER values on trials distribution, N = 1024, CRC = 8, T = 10, R = 1/2, FER range from 10^{-1} to 10^{-2}

3.4 Implementation of Lookup Table

In order to create a lookup table that displays the distribution of trials probability, two functions are required: to develop and fill the table and to read and use the numbers for execution time estimation purposes. For both functions, the same addressing system should be used. Generally, to find the relevant row in the table, a function gets the channel condition and code parameters. It returns the row of the lookup table that includes the list of trials' probability given those inputs. Figure 3.7 indicates the overall picture of the block creating that address for writing in the lookup table or reading the values.

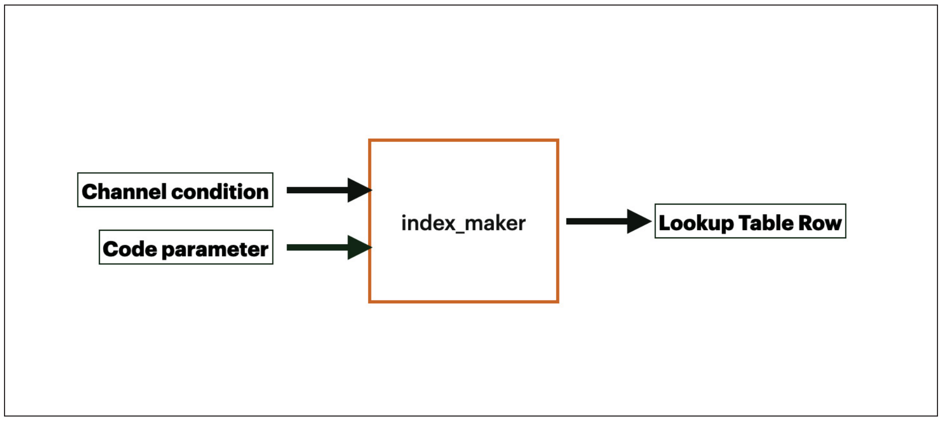


Figure 3.7 Index_maker module

There are three separated tables at the beginning of the process for each $T_{\max} \in \{5, 10, 15\}$. A function called `index_maker` uses FER, block length, and code rate to determine the correct address. As an indicator, a number of bits are assigned to each of these parameters.

Figure 3.8 shows the memory addressing mechanism in detail. As previously explained, we have three separated tables based on the Maximum number of trials ($T_{\max}$). To fill each of them, the addressing system can be seen in this figure. In terms of the code rate, we want to establish this lookup table for three rates where $R \in \{1/2, 1/3, 1/4\}$. To distinguish between these three types, 2 bits are required. To cover three types of frame length where $N \in \{512, 1024, 2048\}$ 2 bits are needed. Eventually, to distinguish two types of FER including 10^{-2} and 10^{-3}, 1 bit is required. After classifying the required bits to cover different states, these bits are concatenated in the mentioned order and converted to a decimal number. The output will be the table row. After creating the addressing mechanism, the values in the lookup table should be set. We previously discussed in the Figure 3.1 that after each simulation, the SCF decoder returns the required trial to decode the frame as an output accompanying the decoded frame. Those values of trials are stored in the various Trial_lists. In this step, using those lists, the lookup tables are completed. We created a separated lookup table for each $T_{\max}$ value. A function named `write_LUK` has this responsibility: reading the Trial_lists from the stored values and inserting them into the correct row of the table. The code of this function is available in Source code-A I.2. Moreover, Source code-A I.1 is the `index_maker` function.

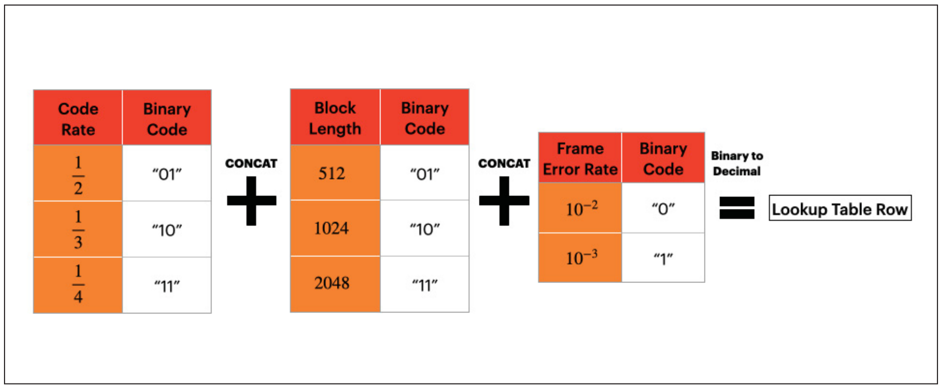
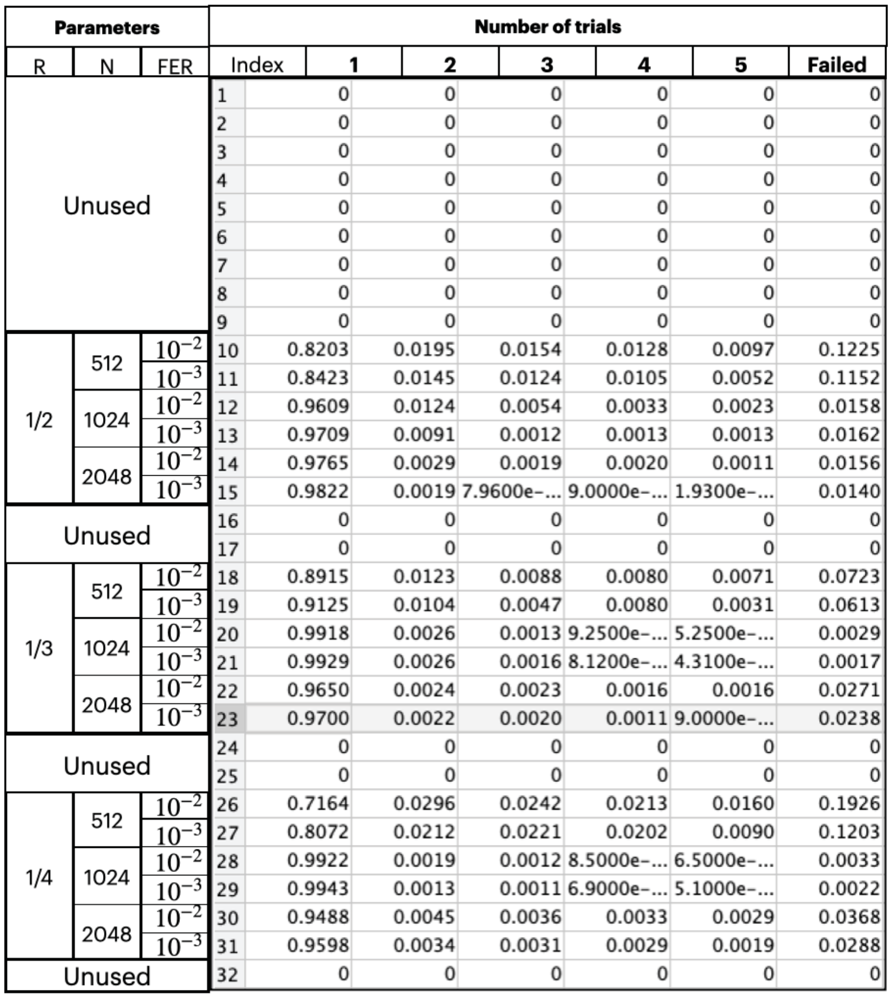


Figure 3.8 Allocating a number of bits to each parameter and creating the address of lookup table

Figure 3.9 illustrates the lookup table where $T_{\max} = 5$. This is the actual lookup table created in the MATLAB environment.

As an example to show how it works, let's say the code rate is ½, the block length is 2048 bits and the FER is 10^{-2}. The concatenated form is [10111], which is equal to 23 in decimal format. Assume that experiments are done previously with this configuration, and the Trial_list is stored with the size of 6. In this example, the probability of decoding the frame with this configuration when $T = 1$ is approximately 97%. This row of the table is shown with the gray color in this figure. The 6th column shows the percentage of the decoder failing to decode the frame in this condition.

It can be seen that there are some rows such as row 1 that are filled with zero because five bits are used to cover the different cases, but with five bits, 32 stages can be covered. However, there are three code rates, three block lengths, and two FERs for a total of 18 combinations.



Parameters			Number of trials						
R	N	FER	Index	1	2	3	4	5	Failed
Unused			1	0	0	0	0	0	0
			2	0	0	0	0	0	0
			3	0	0	0	0	0	0
			4	0	0	0	0	0	0
			5	0	0	0	0	0	0
			6	0	0	0	0	0	0
			7	0	0	0	0	0	0
			8	0	0	0	0	0	0
			9	0	0	0	0	0	0
1/2	512	10^{-2}	10	0.8203	0.0195	0.0154	0.0128	0.0097	0.1225
		10^{-3}	11	0.8423	0.0145	0.0124	0.0105	0.0052	0.1152
	1024	10^{-2}	12	0.9609	0.0124	0.0054	0.0033	0.0023	0.0158
		10^{-3}	13	0.9709	0.0091	0.0012	0.0013	0.0013	0.0162
	2048	10^{-2}	14	0.9765	0.0029	0.0019	0.0020	0.0011	0.0156
		10^{-3}	15	0.9822	0.0019	7.9600e-...	9.0000e-...	1.9300e-...	0.0140
Unused			16	0	0	0	0	0	0
			17	0	0	0	0	0	0
1/3	512	10^{-2}	18	0.8915	0.0123	0.0088	0.0080	0.0071	0.0723
		10^{-3}	19	0.9125	0.0104	0.0047	0.0080	0.0031	0.0613
	1024	10^{-2}	20	0.9918	0.0026	0.0013	9.2500e-...	5.2500e-...	0.0029
		10^{-3}	21	0.9929	0.0026	0.0016	8.1200e-...	4.3100e-...	0.0017
	2048	10^{-2}	22	0.9650	0.0024	0.0023	0.0016	0.0016	0.0271
		10^{-3}	23	0.9700	0.0022	0.0020	0.0011	9.0000e-...	0.0238
Unused			24	0	0	0	0	0	0
			25	0	0	0	0	0	0
1/4	512	10^{-2}	26	0.7164	0.0296	0.0242	0.0213	0.0160	0.1926
		10^{-3}	27	0.8072	0.0212	0.0221	0.0202	0.0090	0.1203
	1024	10^{-2}	28	0.9922	0.0019	0.0012	8.5000e-...	6.5000e-...	0.0033
		10^{-3}	29	0.9943	0.0013	0.0011	6.9000e-...	5.1000e-...	0.0022
	2048	10^{-2}	30	0.9488	0.0045	0.0036	0.0033	0.0029	0.0368
		10^{-3}	31	0.9598	0.0034	0.0031	0.0029	0.0019	0.0288
Unused			32	0	0	0	0	0	0

Figure 3.9 Lookup Table for $T_{\max} = 5$, $R \in \{1/2, 1/3, 1/4\}$, $N \in \{512, 1024, 2048\}$, FER $\in \{10^{-2}, 10^{-3}\}$ in MATLAB environment

After those steps, there are three lookup tables that store the trials probability distribution given the code parameters and channel condition. To use those data, a function named `read_LUK` is provided to read the list of probability in a specific setup and available in Source code-A I.3. Note that, to read from the lookup table, first, the `index_maker` calculates the relevant address. The output of this function is used as an input for the `read_LUK`. This function gets the $T_{\max}$ as an input to chose the correct lookup table.

3.5 Conclusion

Our purpose in this chapter was to create a low complexity system that is able to provide the probability that given a number of trials will be required. We examined possible code parameters that might influence the trials, including frame length, code rate, and CRC length. Additionally, the effect of channel condition was examined and measured using FER parameter. Simulating and visualizing the mentioned factors, we could see that the CRC length does not impact the required trials. Consequently, having the block length, code rate, and FER is enough to estimate the trials' distribution. Later, we created lookup tables that save the probability distribution of required trials given the condition with those data from simulations. A method gets the code parameters and channel condition as the inputs and returns the probability distribution as output. Those lookup tables can predict the possible delay based on the code parameters and channel condition, which can potentially lead to predicting the required resources in terms of memory requirement.

3.5.1 Future works

When the block length and code rate were under investigation, a linear pattern could be seen in the simulation results. It is suggested that in the future, instead of storing the probability of required trials in additional memories (Lookup Table), a mathematical equation calculates the probability of requiring each trial based on these two factors. The effect of FER should also take into consideration.

Although the cost of this equation is running a small number of simulations to create the linear pattern, the additional memory to store the lookup table is no longer needed.

Additionally, in the future, a prediction module can be added to the communication system that is able to get code parameters and possible channel conditions as the inputs and returns the estimated delay. In this block, other parameters such as source encoder and decoders should also be considered.

CONCLUSION AND RECOMMENDATIONS

Polar codes were discovered in 2009 and made it into the 5G standard within a decade. In contrast to LDPC and turbo codes, polar codes have an explicit construction. Therefore their encoder and decoder are easier to be implemented (Arıkan, 2009). Additionally, polar codes can achieve the channel capacity when the frame length is infinite. Later Successive-cancellation Flip (SCF) decoders were invented that could improve the error-correction performance of polar codes when the frame length is finite with a cost of variable execution time. The worst-case scenario should be considered when a decoder is designed for an algorithm with variable execution time, which is not always necessary. In this thesis, a new decoder is presented based on SCF decoder with a reduced execution time and a small increase in memory requirements. This decoder is named Enhanced Successive-cancellation Flip (ESCF) decoder.

In chapter 2, the ESCF algorithm was presented with the same error-correction performance of SCF decoder. The execution time of ESCF decoder is 25% to 10% smaller than SCF's execution time in various SNR points.

Therefore, ESCF could have the error-correction performance of SCF with lower execution time. The execution time of SC, SCF, and ESCF algorithms were calculated and showed that the ESCF has closer execution time to SC decoder. The execution time was measured by counting the number of clock cycles per operations in all three decoders. It is also showed that the gap between SCF and ESCF execution time increases at the lower E_b/N_0 values. Consequently, when the signal power is low, the ESCF decoder requires approximately 25% smaller execution time while it has the same error-correction performance of SCF decoder. This decoder can later integrated with FIS and EIS decoders for future. In those two decoders, after they choose the flipping candidates, ESCF can decide on where to restart decoding based on the location of flipping candidates (Condo *et al.*, 2018). Additionally, ESCF decoder is compatible with Fast-SSC-Flip decoder. When Fast-SSC-Flip decoder categorizes the different types of nodes and chooses the flipping candidates, if that node is in the right sub-tree, ESCF can play its role and increase the execution time.

In chapter 3, the variability of SCF's execution time was investigated. Our goal was to create a low-complexity system that can predict the probability of requiring each trial based on the code parameters and channel condition. This low complex is the system was implemented as a lookup table. At first several simulations were run to build the required statistics for the lookup table and find which code parameter's effect is more significant on the number of trials needed to decode a frame. Our system was simulated with variable code length, code rates, CRC length. Also the system was simulated in different channel conditions to test if different required FER can affect the required trials. Our results showed that trial distribution is not affected significantly by changing the CRC length. However, block size, code rate, and channel condition could affect the distribution.

We used those results and statistics to implement an auto-generated lookup table based on the block length, code rate, and FER. The generated table can provide a probability distribution of trials given the mentioned system configuration. This lookup table gives us a realistic probability distribution that enables us to predict the execution time with a limited number of simulations.

This lookup table can later be added as a building block that is able to provide an estimation of the system delay based on the mentioned parameters. Moreover, we described a linear relationship between those parameters with the required trials, which can lead to creating a mathematical equation. This equation could be able to formulize the required trial based on the code parameters and channel condition.

APPENDIX I

SOURCE CODES FOR CHAPTER 3

1. Lookup Table Source Codes

1.1 Address maker

```
function row=indexMaker(R, N, FER)
    r='00';
    n='00';
    f='0';
    switch R
        case 1/2
            r='01';
        case 1/3
            r='10';
        case 1/4
            r='11';
    end

    switch N
        case 512
            n='01';
        case 1024
            n='10';
        case 2048
            n='11';
    end

    switch FER
        case 100
            f='0';
        case 1000
            f='1';
    end
    row=bin2dec(strcat(r,n,f));
return;
```

Source code-A I.1 Address maker for low complexity estimation system

1.2 Writing in the Lookup Table

```
function lookUpTable=writeLUK(R, N, FER,TType, probability)
    prob=zeros(1,TType+1);
    luk5=zeros(32,6);
    luk10=zeros(32,11);
    luk15=zeros(32,16);
    for counter=1:TType+1
        prob(counter)=probability(counter);
    end
    row=indexMaker(R,N,FER);
    switch TType
        case 5
            for i=1:6
                luk5(row,i)=prob(i);
            end
            lookUpTable=luk5;
            %save('luk5.mat','luk5','-append');
        case 10
            for i=1:11
               luk10(row,i)=prob(i);
            end
            %save file
            lookUpTable=luk10;
        case 15
            for i=1:16
               luk15(row,i)=prob(i);
            end
            lookUpTable=luk15;
            %save file
    end
return;
```

Source code-A I.2 Write into the lookup table

1.3 Reading from the Lookup Table

```
function [probability] = readLUK(index,TType)
    lookUpTable5=zeros(32,6);
    lookUpTable10=zeros(32,6);
    lookUpTable15=zeros(32,6);
    %row in lookup table shows the configuration system
    switch TType
        case 5
            temp=matfile('luk5.mat');
            lookUpTable5=temp.luk5;
            probability=lookUpTable5(index,:);
        case 10
            temp=matfile('luk10.mat');
            lookUpTable10=temp.luk10;
            probability=lookUpTable10(index,:);
        case 15
            temp=matfile('luk15.mat');
            lookUpTable15=temp.luk15;
            probability=lookUpTable15(index,:);
     end

return;
```

Source code-A I.3 Read from the lookup table

BIBLIOGRAPHY

Afisiadis, O., Balatsoukas-Stimming, A. & Burg, A. (2014). A low-complexity improved successive cancellation decoder for polar codes. *2014 48th Asilomar Conference on Signals, Systems and Computers*, pp. 2116–2120.

Arıkan, E. (2009). Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7), 3051–3073.

Berrou, C., Glavieux, A. & Thitimajshima, P. (1993). Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. *Proceedings of ICC'93-IEEE International Conference on Communications*, 2, 1064–1070.

Chen, K., Niu, K. & Lin, J. (2013). Improved successive cancellation decoding of polar codes. *IEEE Transactions on Communications*, 61(8), 3100–3107.

Condo, C., Ercan, F. & Gross, W. J. (2018). Improved successive cancellation flip decoding of polar codes based on error distribution. *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 19–24.

Dmitriev, S. & Kipyatkov, V. (2011). *Pycharm 2020.3*. Pycharm Release 2020. Consulted at https://www.jetbrains.com/pycharm/whatsnew/2020-3/.

Ercan, F., Condo, C. & Gross, W. J. (2018). Improved bit-flipping algorithm for successive cancellation decoding of polar codes. *IEEE Transactions on Communications*, 67(1), 61–72.

Farsad, N., Guo, W. & Eckford, A. W. (2013). Tabletop molecular communication: Text messages through chemical signals. *PloS one*, 8(12), e82935.

Gallager, R. (1962). Low-density parity-check codes. *IRE Transactions on information theory*, 8(1), 21–28.

Giambene, G. (2005). *Queuing theory and telecommunications*. Springer.

Giard, P. & Burg, A. (2018). Fast-SSC-flip decoding of polar codes. *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 73–77.

Giard, P., Thibeault, C. & Gross, W. J. (2017). *High-speed decoders for polar codes*. Springer.

Hagenauer, J., Offer, E. & Papke, L. (1996). Iterative decoding of binary block and convolutional codes. *IEEE Transactions on information theory*, 42(2), 429–445.

Hugunin, J. (2005). *NumPy 1.21.0*. Numpy. Consulted at https://numpy.org/doc/.

Hunter, J. D. (2012). *Matplotlib 3.4.2*. Matplotlib. Consulted at https://matplotlib.org/.

Leroux, C., Raymond, A. J., Sarkis, G., Tal, I., Vardy, A. & Gross, W. J. (2012). Hardware implementation of successive-cancellation decoders for polar codes. *Journal of Signal Processing Systems*, 69(3), 305–315.

MacKay, D. J. & Mac Kay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

MacKay, D. J. & Neal, R. M. (1996). Near Shannon limit performance of low density parity check codes. *Electronics letters*, 32(18), 1645.

MathWorks. (2019). MEX Version 15. Consulted at https://www.mathworks.com/help/matlab/ref/mex.html.

Moler, C. (1980). *MATLAB R2021a*. MATLAB. Consulted at https://www.mathworks.com/help/matlab/release-notes.html.

Proakis, J. G. & Manolakis, D. G. (1992). Digital signal processing. *MPC, New York*.

Rossum, G. V. (1991). *Python 3.9*. Python Release 3.9.6. Consulted at https://docs.python.org/3/whatsnew/3.9.html.

Sarkis, G. (2016). *Efficient encoders and decoders for polar codes: Algorithms and implementations*. McGill University (Canada).

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, 27(3), 379–423.

Tal, I. & Vardy, A. (2015). List decoding of polar codes. *IEEE Transactions on Information Theory*, 61(5), 2213–2226.