# DIGITAL DESIGN AND OPTIMIZATION OF ULTIMATE-SHANNON-LIMIT-APPROACHING CHANNEL CODES

SHENG JIANG

PhD

The Hong Kong Polytechnic University

2021

The Hong Kong Polytechnic University

Department of Electronic and Information Engineering

# Digital Design and Optimization of Ultimate-Shannon-Limit-Approaching Channel Codes

SHENG JIANG

A thesis submitted in partial fulfilment of

the requirements for the degree of

Doctor of Philosophy

September 2020

# Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signed)

Sheng Jiang (Name of student)

# Abstract

This thesis focuses on digital communication systems in which the received signal-to-noise ratio is extremely low. Application examples include space communications and multiple-user environments using code division multiple access and interleaver division multiple access. Hence, only channel codes with performance very close to the ultimate Shannon limit, i.e., bit-energy-to-noise-power-spectral-density ratio $E_b/N_0 = -1.59$ dB, are considered. In this thesis, we propose digital system designs of two ultimate-Shannon-limit-approaching codes, namely turbo Hadamard codes and concatenated zigzag Hadamard codes. Moreover, we propose ways to design punctured turbo Hadamard codes and to lower the error floor of turbo Hadamard codes. We also figure that efforts are needed to evaluate cycles in the design of a turbo Hadamard code. To estimate the computation effort required, we generalize the problem and propose a new method to evaluate the number of closed paths in an all-one base matrix.

Firstly, we propose a pipelined digital design of a turbo Hadamard encoder/decoder system. To accomplish a high throughput, we make use of a multiple sub-decoder architecture to process multiple codes at the same time. Each sub-decoder is processing the data of one codeword at anytime; and data from the same codeword will be processed by different sub-decoders at different times. One iteration is completed when the data from the same codeword is processed by all the sub-decoders once. Moreover, tens of iterations are required to complete the decoding. Hence, the design challenges include control of data flow within a sub-decoder; control of data flow among sub-decoders; proper data storage to avoid data access conflicts; conversion of data formats

to facilitate computations; effective interleavers that cause minimum latency. In order to achieve performance close to the ultimate Shannon limit, code lengths of 216450, 287235 and 358020 are used. Since the code lengths are relatively long, effective use of data storage is crucial. Also, the transmitter is required to send code bits in a way that facilitates storing and processing of data at the receiving end. Note that new codeword data are being received continuously and need to be stored as the decoder is processing the existing codewords. The theoretical throughput of our digital design is derived based on the given parameters such as hardware operating frequency, code length, number of iterations, and length of the turbo trellis. Results indicate that at an operating frequency of 100 MHz, the proposed digital turbo Hadamard encoder/decoder system achieves throughputs of 1.92 Gbps, 2.56 Gbps and 3.2 Gbps at code lengths of 216450, 287235 and 358020, respectively. The encoder/decoder system also realizes a bit error rate of $10^{-5}$ at $E_b/N_0 = -0.45$ dB, i.e., 1.14 dB from the ultimate Shannon limit.

Secondly we realize that in the design of the turbo Hadamard encoder/decoder system, relatively complex Bahl-Cocke-Jelinek-Raviv (BCJR) decoding is required and it limits the operating frequency to 100 MHz. Thus we propose a pipelined digital design of a concatenated zigzag Hadamard encoder/decoder system, in which BCJR decoding is not needed. Again we propose a multiple sub-decoder architecture to process multiple codes at the same time. However, different from the sub-decoders in the turbo Hadamard decoder system, each sub-decoder in the concatenated zigzag Hadamard decoder system processes a set of multiple codes at the same time; and the same set of multiple codes is processed by different sub-decoders at different times. Such an arrangement aims to improve the throughput and also the hardware utilization rate. We overcome challenges similar to those occurring in the design of turbo Hadamard encoder/decoder systems. The final concatenated zigzag Hadamard encoder/decoder system is found to work with 50% increase in operating frequency compared with the turbo Hadamard encoder/decoder system and with similar error performance. The

drawbacks of the concatenated zigzag Hadamard encoder/decoder system, however, are higher decoding latency and higher memory requirement.

Thirdly, we propose ways to optimize the turbo Hadamard codes. Punctured Hadamard codes and punctured turbo Hadamard encoder/decoder systems are first investigated. By puncturing some of the code bits and not sending those bits through the channel, the rate of a code is improved and sometimes the bit error rate performance can be improved too. Here, two methods to select the punctured Hadamard code bits, or equivalently the puncturing patterns, are proposed. The first scheme aims to maximize the minimum Hamming distance of the punctured Hadamard codes. The upper-bound of the minimum Hamming distance of punctured Hadamard codes is derived and then an algorithm is proposed to find punctured Hadamard codes achieving close to this bound. The second scheme aims to minimize the cross-correlations among the punctured Hadamard codes. For punctured code sets having the same minimum cross-correlation, a new metric has been proposed to identify sets that can further enhance the reliability of decoding. For Hadamard codes, the two proposed puncturing schemes have shown error improvements over the use of random puncturing. Moreover, the scheme that minimizing the cross-correlations outperform that maximizing the minimum Hamming distance. By applying the more superior scheme to puncture Hadamard codes in a turbo Hadamard encoder/decoder system, the code rate is improved with little change in error performance. Another way to optimize the turbo Hadamard codes is to lower its error floor. At the high $E_b/N_0$ region, we observe that the error rate of the turbo Hadamard code may become flat. To tackle this issue, we investigate the overall turbo Hadamard code structure by looking into an associated parity-check matrix. By re-designing the interleavers and hence removing short cycles in the parity-check matrix, we observe that the error floor can be lowered.

Finally, while studying the cycles of the turbo Hadamard code structure, we come up with a new method to evaluate the number of cycles for a given parity-check matrix. We further generalize the method and use it to evaluate the number of closed

paths in an all-one base matrix. Theoretical results up to closed paths of length 10 have been derived and are verified by the exhaustive search method. Based on the theoretical work, results for closed paths of length larger than 10 can be further derived. The results are particularly useful for estimating computational resources required in designing parity-check matrices of low-density parity-check codes.

# Publications

## Journal papers

- **S. Jiang** and F. C. M. Lau, "An Approach to Evaluating the Number of Closed Paths in an All-One Base Matrix," *IEEE Access*, vol. 6, pp. 22332-22340, 2018, DOI: 10.1109/ACCESS.2018.2819981.

- **S. Jiang**, P. W. Zhang, F. C. M. Lau and C. -W. Sham, "An Ultimate-Shannon-Limit-Approaching Gbps Throughput Encoder/Decoder System," *IEEE Transactions on Circuits and Systems II*, available for Early Access on 18 December 2019, DOI: 10.1109/TCSII.2019.2960613.

- **S. Jiang**, F. C. M. Lau and C. -W. Sham, "Hardware Design of Concatenated Zigzag Hadamard Encoder/Decoder System with High Throughput," *IEEE Access*, vol. 8, pp. 165298-165306, 2020, DOI: 10.1109/ACCESS.2020.3022537.

## Conference papers

- **S. Jiang**, F. C. M. Lau, W. M. Tam and C. -W. Sham, "Design and Error Performance of Punctured Hadamard Codes," in *23rd Asia-Pacific Conference on Communications (APCC)*, Perth, Australia, 2017, pp. 1-5.

- **S. Jiang** and F. C. M. Lau, "An Approach to Evaluating the Number of Potential Cycles in an All-One Base Matrix," in *20th International Conference on Ad-*

*vanced Communication Technology (ICACT)*, Phoenix Park, Korea, 2018, pp. 104-108.

- **S. Jiang**, P. W. Zhang, F. C. M. Lau, C. -W. Sham and K. Huang, "A Turbo-Hadamard Encoder/Decoder System with Hundreds of Mbps Throughput," in *IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, Hong Kong, 2018, pp. 1-5.

# Acknowledgments

I would like to express my sincere gratefulness to my supervisor Prof. Francis. C. M. Lau for his continuous support, patient guidance and valuable advice. Prof. Francis C. M. Lau is very active and open-minded, and always encourages me to think deeply and freely during the course of this project. His enthusiasm in research and insights into the area of channel coding have inspired me a lot. Without his help, I could not have accomplished this Ph.D. study.

I would also like to thank Dr. Bruce C. -W. Sham, Dr. Wai-man Tam and Mr. Pengwei Zhang for many inspiring discussions on the topic of channel coding. The numerous discussions with them have benefited me immensely.

Finally, I would like to thank my parents and my wife. Their love, care, and support make me stronger.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

Digital communication has been embedded in our daily lives. The most common examples include 3G/4G/5G communications, Wi-Fi, and Internet connections via cables or optical fibres. What these applications share in common is that information is transmitted or communicated from one point to another. Fig. 1.1 depicts the typical blocks of a digital communication system.

Firstly, the original information is compressed into sequences of $M$-ary symbols by the source encoder. (In this thesis, we only consider the situation of binary transmission, i.e., $M = 2$.) Then the channel encoder encodes the data by introducing redundancy to the information sequences which can help recovering data from noise and interference. The digital modulator modulates the encoded sequences into waveforms that are suitable for transmission and sends them to the physical channel [1]. For example, wireless signals will be sent via antennas to the air; while modulated light signals will be sent along optical fibers. The physical channel, such as air and optical fibre, introduces attenuation and distortion (e.g., interference, additive noise) to the waveforms. At the receiving side, the operations are reversed. The demodulator demodulates the received waveforms and produces channel observations. Based on

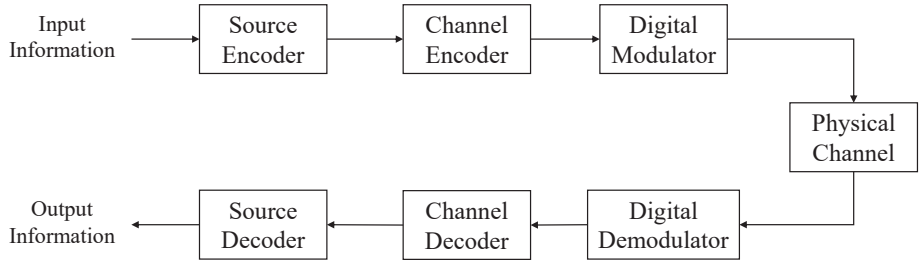


Figure 1.1: Block diagram of a typical digital communication system.

the channel observations, the channel decoder decodes the code bits; and based on the decoded bits, the source decoder decompresses and recovers the source information. In particular, the role of the channel encoder/decoder pair is to improve the reliability of the transmission process.

Channel coding techniques mainly consist of two categories: automatic repeat request (ARQ) schemes and forward error correction (FEC) schemes [2]. In ARQ schemes, channel codes only detect errors. Once one or more errors are detected, the receiver will send a request to the transmitter to resend the codes. In FEC schemes, channel codes add redundancy to the information and allow error correction through a decoding algorithm at the receiving end. In some situations, more flexible schemes may be desirable to accommodate different error protection requirements [2] and hybrid FEC/ARQ protocols are used [3]. In this thesis, we focus on the FEC codes. Note that all FEC codes can be used in a hybrid FEC/ARQ scheme where the code is used for both error correction and detection [4].

Among the FEC codes, turbo code, low-density parity-check (LDPC) code, and polar code are the most widely studied and implemented error-correction codes over the last two-and-a-half decades because of their capacity-approaching capabilities [5–13]. Turbo code was discovered in 1993 by Berrou and Glavieux [5] and was the first capacity-approaching channel code ever widely used. LPDC code, though first studied in 1960's by Gallager [7], did not receive much attention until Mackey "rediscovered" it in 1996 [8]. From the hardware design point of view, LDPC code has an advantage

over the other two codes because it allows parallel processing with reasonable complexity and hence can reduce the decoding latency. Among the aforementioned three types of codes, only polar codes can theoretically reach the channel capacity, even though the code length has to become infinitely long. Nonetheless, all of them can achieve very low error rates with reasonable code lengths as they approach the capacity limit. Moreover, some of the code bits can be punctured and not sent to the receiver. In [14, 15], both the information bits and the parity-check bits are punctured to get a good "turbo-like" code. Also, compatible punctured codes can be employed in hybrid forward-error correction/automatic repeat-request (ARQ) systems [2, 3, 16, 17].

Both turbo code and LDPC code, when used in conjunction with Hadamard code, have been shown to achieve performance very close to the ultimate Shannon limit, i.e., bit-energy-to-noise-power-spectral-density ratio $E_b/N_0 = -1.59$ dB [18, 19]. Another code with comparable performance is the concatenated zigzag Hadamard code [20]. In [18], it has been simulated that using 50 decoding iterations, a turbo-Hadamard code can achieve a bit-error rate (BER) of $10^{-5}$ at $E_b/N_0 = -1.20$ dB, i.e., within 0.39 dB of the ultimate Shannon limit. Also, the LDPC-Hadamard/concatenated zigzag Hadamard code can achieve a bit-error rate (BER) of $10^{-5}$ at $E_b/N_0 = -1.44/-1.15$ dB, i.e., within $0.15/0.44$ dB of the ultimate Shannon limit [19].

## 1.2 Motivation

This thesis focuses on digital communication systems in which the received signal-to-noise ratio is extremely low. Application examples include space communications and multiple-user environments using code division multiple access and interleaver division multiple access. Hence, only channel codes with performance very close to the ultimate Shannon limit are considered.

While turbo code or LDPC code used in conjunction with Hadamard code can achieve performance very close to the ultimate Shannon limit [18, 19], the code lengths

used are quite long ($> 100,000$). Such long lengths pose huge design challenges (such as control of data flow, proper data storage to avoid data access conflicts, conversion of data formats to facilitate computations, and effective interleavers) for the decoders, especially when high throughput and low latency are required. Thus, no such encoder/decoder system designs have ever been proposed or realized. In this thesis, we design and realize turbo Hadamard encoder/decoder system and concatenated zigzag Hadamard encoder/decoder system. Pipelined digital designs are used to increase the throughputs.

We have also proposed techniques to improve the code rate and error floor of turbo Hadamard codes. By puncturing the code bits appropriately and not sending them to the receiver, the code rate is improved. To lower the error floor of turbo Hadamard codes, we study the property of a related parity-check matrix. By eliminating short cycles of the matrix, the error floor is lowered.

Cycles in parity-check matrices affect performance of not only turbo Hadamard codes, but also other high-performance channel codes such as LDPC codes. Techniques such as density evolution [21] and extrinsic information transfer chart (EXIT chart) [22, 23] have been used to design LDPC codes by optimizing the degree distributions. However, LDPC codes with the same degree distributions may have very different error performance due to the existence of short cycles. Maximizing the girth (shortest cycle) of LDPC codes is thus important in the design of quasi-cyclic LDPC (QC-LDPC) code [24], QC-LDPC-convolutional code [25] and protograph LDPC code [26]. Here, we propose a new method to evaluate the number of cycles in an all-one base matrix. The technique is useful in estimating the resources required to design good performing channel codes such as turbo Hadamard codes and LDPC codes.

## 1.3 Thesis Organization

This thesis is organized as follows.

Chapter 2 provides a literature review. Key results of Shannon's theorems are reviewed. The structure and decoding of turbo codes, LDPC codes, Hadamard codes and concatenated Hadamard codes are presented.

Chapter 3 presents the detailed design of a turbo Hadamard encoder/decoder system. The decoder consists of multiple sub-decoders which can process multiple codes at the same time. The theoretical throughput of the digital design is derived based on the given parameters such as hardware operating frequency, code length, number of iterations, and length of the turbo trellis. Results indicate that at an operating frequency of 100 MHz, the proposed digital turbo Hadamard encoder/decoder system achieves throughputs of 1.92 Gbps, 2.56 Gbps and 3.2 Gbps at code lengths of 216450, 287235 and 358020, respectively. The encoder/decoder system also realizes a bit error rate of $10^{-5}$ at $E_b/N_0 = -0.45$ dB, i.e., 1.14 dB from the ultimate Shannon limit.

Chapter 4 proposes a pipelined digital design of a concatenated zigzag Hadamard encoder/decoder system. Again a multiple sub-decoder architecture is employed to process multiple codes at the same time. Different from the sub-decoders in the turbo Hadamard decoder system, each sub-decoder in the concatenated zigzag Hadamard decoder systems processes a set of multiple codes at the same time; and the same set of multiple codes is processed by different sub-decoders at different times. The final concatenated zigzag Hadamard encoder/decoder system is found to work with 50% increase in operating frequency compared with the turbo Hadamard encoder/decoder system and with similar error performance.

Chapter 5 optimizes the turbo Hadamard code from two perspectives — improving the code rate and lowering the error floor. To improve the code rate, two methods to select the punctured Hadamard code bits are proposed. The first scheme aims to maximize the minimum Hamming distance of the punctured Hadamard codes while the second scheme aims to minimize the cross-correlations among the punctured codes. Then the more superior scheme is applied to puncture Hadamard codes in a turbo Hadamard encoder/decoder system, and the error performance of the punctured code

is simulated. To lower the error floor of a turbo Hadamard code, the parity-check matrix related to the code is investigated. By re-designing the interleavers and hence removing short cycles within the parity-check matrix, the error floor can be lowered.

Chapter 6 proposes a new technique for evaluating the number of closed paths in an all-one base matrix. Theoretical results up to closed paths of length 10 have been derived and are verified by the exhaustive search method.

Chapter 7 concludes the thesis. Major contributions are summarized and some potential future works are presented.

# Chapter 2

# Literature Review

In this chapter, we review the concept of the channel capacity; and the structure and decoding algorithm of turbo codes, LDPC codes, Hadamard codes and concatenated Hadamard codes.

## 2.1 Channel capacity

In 1948, Claude Shannon's paper "*A Mathematical Theory of Communication*" defined information as a measurable quantity [27]. The entropy $H$ of a random variable measures its average uncertainty. Supposing a random variable $x$ has a discrete probability distribution $p(x) = \{p_1, p_2, ..., p_n\}$, then the entropy of $x$ is defined as

$$H(x) = -\sum_{i=1}^{n} p_i \log p_i. \tag{2.1}$$

Similarly, for a random variable $x$ with a continuous distribution and probability density function (PDF) $p(x)$, its entropy is defined as

$$H(x) = -\int_{-\infty}^{\infty} p(x) \log p(x) dx. \tag{2.2}$$

In a digital communication system, both the input and output of the channel hold information. Suppose a discrete memoryless channel has an input $X$ and an output $Y$. The joint entropy of the random variables $(X, Y)$ is denoted as $H(X, Y)$; and the conditional entropy of the random variables $(X, Y)$ is denoted as $H(Y|X)$ or $H(X|Y)$. Moreover, the following relationship holds [28].

$$H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y). \tag{2.3}$$

The mutual information of two random variables measures the amount of information that one random variable contains about another. The mutual information of $(X, Y)$ is defined as [29]

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X). \tag{2.4}$$

In this example, $I(X; Y)$ measures the uncertainty reduction of the information $X$ when we received the channel output $Y$.

The capacity of the channel, denoted by $C$, is thus defined as the maximum mutual information of input $X$ and output $Y$ for all distributions of source $X$, i.e.,

$$C = \max_{P_X} I(X; Y). \tag{2.5}$$

A typical example is the Additive-White-Gaussian-Noise (AWGN) channel with 0 means and variance $\sigma_n^2$, which will be considered as the most common case in this thesis. Considering the entropy of a one-dimensional Gaussian distribution with variance $\sigma^2$, i.e.,

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{x^2/2\sigma^2}, \tag{2.6}$$

we have

$$H(x) = -\int_{-\infty}^{\infty} p(x) \log p(x) dx = \frac{1}{2} \log_2 2\pi e\sigma^2. \tag{2.7}$$

The entropy of an AWGN noise is then given by

$$H(n) = \frac{1}{2}\log_2 2\pi e\sigma_n^2. \tag{2.8}$$

The maximum mutual information of transmitted signal $X$ and received signal $Y$ occurs when $Y$ also forms a white Gaussian ensemble since this is the greatest possible entropy for a limited average power situation [27]. Assuming the input power is $\sigma_p^2$, then the capacity of the AWGN channel is

$$\begin{aligned} C_{awgn} &= \max(H(Y) - H(Y|X)) \\ &= \max H(Y) - H(n) \\ &= \frac{1}{2}\log_2 2\pi e(\sigma_p^2 + \sigma_n^2) - \frac{1}{2}\log_2 2\pi e\sigma_n^2 \\ &= \frac{1}{2}\log_2\left(1 + \frac{\sigma_p^2}{\sigma_n^2}\right). \end{aligned} \tag{2.9}$$

For an AWGN channel with power $S$, bandwidth $W$ and noise power $N$, the capacity in (2.9) becomes

$$\begin{aligned} C_{awgn} &= 2W \times \frac{1}{2}\log_2\left(1 + \frac{S}{N}\right) \\ &= W\log_2\left(1 + \frac{S}{N}\right) \text{ bits per second.} \end{aligned} \tag{2.10}$$

Now let us consider the minimum energy per bit ($E_b$) required for an AWGN channel. Denoting the bit rate by $R$ and the noise power spectral density by $N_0$, then $S = RE_b$ and $N = WN_0$. Using these expressions, (2.10) can be written as

$$\begin{aligned} R < C &= W\log_2\left(1 + \frac{RE_b}{WN_0}\right) \\ \frac{R}{W} &< \log_2\left(1 + \frac{R}{W}\frac{E_b}{N_0}\right) \\ \frac{E_b}{N_0} &> \frac{2^{R/W} - 1}{R/W}. \end{aligned} \tag{2.11}$$

When the data rate $R$ is small compared to the bandwidth $W$, we reach the ultimate

Shannon limit, i.e.,

$$\frac{E_b}{N_0} > \frac{2^{R/W}-1}{R/W} > \lim_{R/W\to 0} \frac{2^{R/W}-1}{R/W} = \ln(2) = -1.59 \text{ dB}. \tag{2.12}$$

For most other channels, there is no closed-form solution for the channel capacity [30]. For example, the capacity of the general relay channel remains a challenge. Only an upper bound has been obtained [31]. Besides studying the channel capacity, Shannon also proved the channel coding theorem [27].

**Theorem 1** (Shannon's Noisy-Channel Coding Theorem): *For any discrete memoryless channel with capacity C the following statements hold: For any* $\epsilon > 0$ *and rate* $R < C$*, for sufficiently large N, there is a code of length N and rate* $\geq R$ *and a decoding algorithm, such that the probability of block error is* $\leq \epsilon$*.*

Shannon's noisy-channel coding theorem shows the performance limitation of channel coding given a certain channel. That is to say, it is impossible to find a code that can achieve infinitely small error probabilities when the signal-to-noise ratio (SNR) of the communication system is smaller than $-1.59$ dB over an AWGN channel.

## 2.2 Turbo Codes

Shannon's work did not provide the method to construct capacity-approaching channel codes. This problem motivated intensive research efforts on good codes. However, until early 90's the performance of channel codes still were 3 dB or more away from the theoretical Shannon limit. The discovery of turbo code [5,6] and block turbo code [32, 33] greatly shortened the gap to capacity. The turbo codes were soon put into practical use such as 3G wireless phones, Digital Video Broadcast (DVB) systems, and Wireless Metropolitan Area Networks (WMAN). Turbo codes are mainly categorized into two groups: parallel concatenated convolutional code (PCCC) [34] and serial concatenated convolutional code (SCCC) [35, 36].

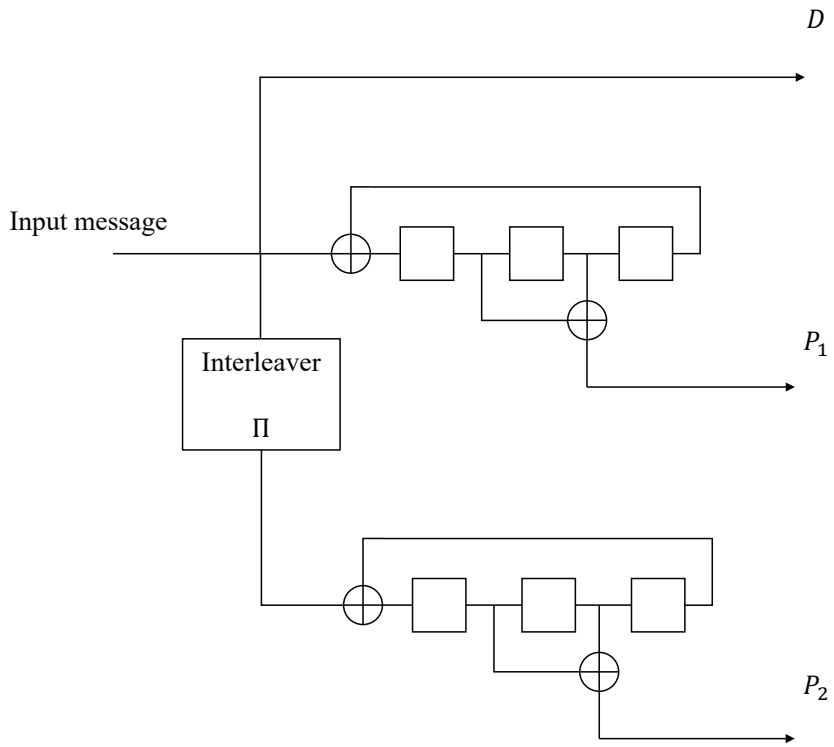


Figure 2.1: The encoder of a classical turbo code.

The classical turbo code, i.e., PCCC, consists of several parallel concatenated convolutional codes, which share the same but interleaved (permuted) messages. A random interleaver usually performs better than the familiar block interleaver [37]. The encoder of such turbo code is illustrated in Fig. 2.1. In the figure, the message $D$ is passed into two recursive systematic convolutional (RSC) encoders which produce two sets of parity-check bits $P_1$ and $P_2$. RSC codes are usually preferred in the turbo codes because the output weight of the RSC codes are approximately uncorrelated to the input weight of the codes, which greatly reduces the generation of low-weight codes that are harmful to the decoding performance. Generally, the principle of designing turbo codes is to choose the best component codes by maximizing the effective distance of the codes [38]. At high SNR regime, it is equivalent to maximizing the minimum-weight codeword; however at low SNR regime, it is more important to optimize the weight distribution of the code [39].

Iterative decoding, which adopts "soft-in/soft-out" algorithms instead of dealing

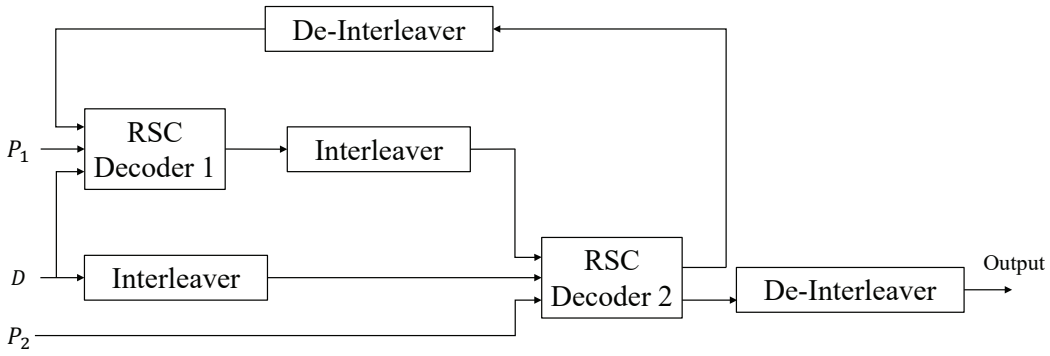


Figure 2.2: The decoder of a classical turbo code.

with hard decision algorithms, is used in the decoding of turbo codes. The turbo decoder shown in Fig. 2.2 consists of two RSC decoders to decode the two RSC component codes, respectively. Both RSC decoders are soft-in soft-out maximum a posteriori (MAP) decoders.

A unified framework of iterative decoding was first raised in [40]. For general purpose, assume a code $\boldsymbol{c}$ is transmitted in a channel and a vector $\boldsymbol{y}$ is received. Also denote $u$ as an information bit of the code $\boldsymbol{c}$. A "soft-in/soft-out" decoder is shown in Figure. 2.3. The decoder basically has two inputs and two outputs.

1. Input 1: The log-likelihood-ratio (LLR) of all the received channel bits denoted as $L_{\boldsymbol{c}}y$.

2. Input 2: The log-likelihood-ratio (LLR) of *a priori* values for all information bits denoted as $L(u)$.

3. Output 1: The log-likelihood-ratio (LLR) of *a posteriori* values for all information bits denoted as $L(\hat{u})$.

4. Output 2: The log-likelihood-ratio (LLR) of extrinsic values for all information bits denoted as $E(\hat{u})$ .

The extrinsic information $E(\hat{u})$ represents the soft information that $u$ collects from all the other coded bits. Moreover, the bit decisions are made based on output 1, which


Figure 2.3: Illustration of a soft-in soft-out decoder.

is defined as the *a posteriori* log-likelihood-ratio of the transmitted bit being a "0" to that being a "1", i.e.,

$$L(\hat{u}) = L(u|\boldsymbol{y}) = \ln\frac{P(u = \text{“0”}|\boldsymbol{y})}{P(u = \text{“1”}|\boldsymbol{y})}. \tag{2.13}$$

The iterative decoding process is as follows.

1. For uniformly-transmitted information bits, the *a priori* information $L(u) = 0$ during the first iteration of the decoding. For subsequent iterations, the extrinsic information from the previous decoder is used as the *a priori* information of the current decoder.

2. The soft-in soft-out decoder calculates the *a posteriori* information $L(\hat{u})$ and the extrinsic information $E(\hat{u})$ based on $L(u)$ and $L_c y$.

3. The extrinsic information is passed to the next decoder and act as the *a priori* information of that decoder.

4. Repeat the above steps until the maximum number of iterations are reached. Make hard decisions of the codewords based on $L(\hat{u})$.

Note that generally the reliability of the decoded bits increases with an increasing number of iterations.

## 2.3 Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes were first proposed by Gallager in 1962 [7]. Due to the lack of the computational capability in that era, LDPC codes had remained neglected for over 35 years. In 1996, McKay introduced a new class of block codes and soon it was recognized that these block codes were in fact a "rediscovery" of the LDPC codes [8]. Much research had been done to irregular LDPC codes [21, 41–43] which could easily outperform turbo codes and approach the Shannon limit a lot closer [44]. Because of an increasing computational capability, LDPC codes can be deployed as a key error correction component in many digital communication systems like WiFi [45], WiMAX [46], DVB-S2 [47], CCSDS [48], ITU G.hn [49], and 4G/5G.

LDPC codes are linear block codes with sparse parity-check matrices, i.e., the number of non-zero entries in the parity-check matrices is very small. It is the sparseness of the parity-check matrix that guarantees a minimum Hamming distance increasing linearly with the code length, and thus a good error performance of LDPC codes with long code length. An LDPC code is totally specified by a $m \times n$ parity-check matrix $\boldsymbol{H}$. We denote the number of non-zero elements in each column of $\boldsymbol{H}$ by $w_c$ and the number of non-zero elements in each row of $\boldsymbol{H}$ by $w_r$. Then the LDPC code is called $(w_c, w_r)$-regular if $w_c$ and $w_r$ remain invariant for all columns and rows. Otherwise the code is called *irregular*. The parity-check matrix of a $(2, 3)$-regular LDPC code is shown in (2.14). Note that the matrix cannot be considered as sparse but it is only used for illustrating the concept of a "regular" code.

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{2.14}$$

In [50], Tanner proposed using a graph to visualize the factorization of a code.

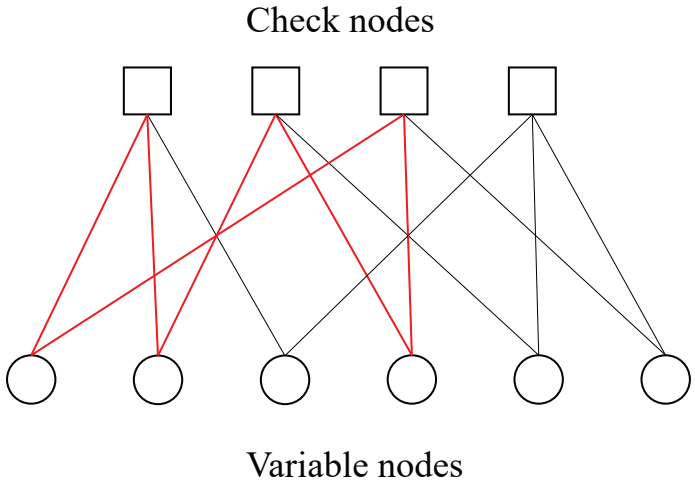


Figure 2.4: The Tanner graph of $\boldsymbol{H}$ given in (2.14). A cycle of length-6 is drawn as red.

A "Tanner" graph is a bipartite graph associated with a parity-check matrix $\boldsymbol{H}$. The graph has $n$ variable nodes corresponding to the components of the codeword (columns of $\boldsymbol{H}$), and $m$ check nodes corresponding to the parity-check constraints (rows of $\boldsymbol{H}$). An edge will be present between variable node $j$ and check node $i$ if the corresponding element in the parity-check matrix $\boldsymbol{H}$, denoted as $h_{ij}$, is non-zero. Fig. 2.4 draws the Tanner graph of $\boldsymbol{H}$ given in (2.14). A cycle in a Tanner graph is a sequence of connected vertices which starts and ends at the same vertex in the graph, and which contains other vertices no more than once. The length of a cycle is the number of edges it contains. A cycle of length-6 is illustrated in Fig. 2.4 and drawn as red.

The classical decoding algorithm of LDPC codes is called a message-passing algorithm because the decoding can be explained as the passing of messages along the edges in the Tanner graph. The message-passing algorithm is also an iterative decoding algorithm. The messages are passed from variable nodes to check nodes and vice versa iteratively until the decoding is completed. Hard-decision message-passing algorithms are called bit-flipping algorithms [51–53] where the messages being passed are binary. Soft-decision message-passing algorithms are called belief propagation (BP) algorithms [43,54] where the messages passed are probabilities that represent the amount of belief about the codeword bits. In this section, we will briefly review the

BP algorithm.

As in [55, 56], we denote

- $\mathcal{M}(n)$ as the set of check nodes that are connected to variable node $n$;
- $\mathcal{N}(m)$ as the set of variable nodes that are connected to check node $m$;
- $\mathcal{M}(n) \setminus m$ as the exclusion of $m$ from the set $\mathcal{M}(n)$;
- $\mathcal{N}(m) \setminus n$ as the exclusion of $n$ from the set $\mathcal{N}(m)$;
- $Z_{n\to m}^{(i)}$ as the LLRs passing from variable node $n$ to check node $m$ at iteration $i$;
- $L_{m\to n}^{(i)}$ as the LLRs passing from check node $m$ to variable node $n$ at iteration $i$;
- $Z_n^{(i)}$ as the a posteriori LLR of variable node $n$ at iteration $i$;
- $\boldsymbol{x} = (x_0, x_1, ..., x_{N-1})$ as the codeword transmitted over an additive white Gaussian noise (AWGN) channel;
- $\boldsymbol{y} = (y_0, y_1, ..., y_{N-1})$ as the noisy vector received.

We also assume the noise variance is $\sigma^2$ and binary-phase-shift-keying (BPSK) modulation is used. The channel LLRs can be calculated as $r_n = \frac{2y_n}{\sigma^2}$ [57]. Then the BP algorithm can be summarized as follows [55, 56].

1. *Initialization*: Set the a priori information at iteration 0 as the channel LLRs, i.e., $Z_{n\to m}^{(0)} = r_n$.

2. *Check-node updating*: For each $m$ and for $n \in \mathcal{N}(m)$, the check-to-variable messages are updated by

$$L_{m\to n}^{(i)} = 2\tanh^{-1}\Big(\prod_{n'\in\mathcal{N}(m)\setminus n} \tanh\Big(\frac{Z_{n'\to m}^{(i-1)}}{2}\Big)\Big). \tag{2.15}$$

3. *Variable-node updating*: For each $n$ and for $m \in \mathcal{M}(n)$, the variable-to-check messages are updated by

$$Z_{n\to m}^{(i)} = r_n + \sum_{m' \in \mathcal{M}(n)\setminus m} L_{m'\to n}^{(i)} \tag{2.16}$$

and the a posteriori information of all variable nodes are calculated as

$$Z_n^{(i)} = r_n + \sum_{m' \in \mathcal{M}(n)} L_{m'\to n}^{(i)}. \tag{2.17}$$

4. *Decision*: Make the decision on $\hat{\boldsymbol{x}} = (\hat{x}_0, \hat{x}_1, ..., \hat{x}_{N-1})$ such that $\hat{x}_n = 0$ if $Z_n^{(i)} \geqslant 0$; and $\hat{x}_n = 1$ if $Z_n^{(i)} < 0$. If the decoded codeword is a valid codeword, i.e., $\hat{\boldsymbol{x}}\boldsymbol{H}^T = \mathbf{0}$, or the maximum number of iterations is reached, the decoding process halts; otherwise repeat Steps 2), 3) and 4) in the next iteration.

The BP algorithm iteratively computes the maximum a posteriori (MAP) of the codewords. However, the MAPs obtained are only approximations unless the Tanner graph of the code is cycle-free. It is because the extrinsic information provided to variable node $n$ becomes dependent on the original a priori LLR $r_n$ if the original a priori LLR is returned to variable node $n$ via a cycle in the Tanner graph [58]. That is to say, the existence of cycles (especially short cycles) in the Tanner graph of a code reduces the effectiveness of the iterative decoding process. An LDPC code with large girth (minimum cycle length) is always preferred and short cycles (like cycle-4) should always be avoided when designing LDPC codes as well as other high-performing channel codes.

## 2.4 Hybrid Concatenated Hadamard Codes

Both turbo code and LDPC code, when used in conjunction with Hadamard code, have been shown to achieve performance very close to the ultimate Shannon limit $-1.59$ dB [18, 19]. Another code with comparable performance is the concatenated

zigzag Hadamard code [20]. In [18], it has been shown that using 50 decoding iterations, a turbo Hadamard code with an information length of 65534 and a code length of approximately $3,500,000$ (code rate $\approx 0.019$) can achieve a bit-error rate (BER) of $10^{-5}$ at $E_b/N_0 = -1.2$ dB, i.e., within 0.4 dB of the ultimate Shannon limit. The concatenated zigzag Hadamard codes with long block lengths achieve $10^{-5}$ BER at $E_b/N_0 = -1.15$ dB, about 0.45 dB from the ultimate Shannon limit. Also, the LDPC-Hadamard/concatenated zigzag Hadamard code with an information length of 65536/65536 and a code length of approximately $22,000,000/3,500,000$ (code rate $\approx 0.003/0.019$) can achieve a bit-error rate (BER) of $10^{-5}$ at $E_b/N_0 = -1.44/-1.15$ dB, i.e., within 0.16/0.45 dB of the ultimate Shannon limit [19,20]. Such codes can be used in a multi-user environment such as in a code-division multiple-access or an interleave-division multiple-access (IDMA) [59] system, and for narrow-band interference suppression [60]. They can also be used to carry embedded messages in point-to-point wireless/wired communications. There are extended Hadamard codes such as concatenated twist Hadamard code [61] and parallel concatenated tree Hadamard codes [62] which have shown good error floor performance when the code length is short.

### 2.4.1 Hadamard Code

A Hadamard code can be constructed from a Hadamard matrix of the same order. An order-$r$ Hadamard matrix $\boldsymbol{H}_n$ where $n = 2^r$ can be constructed recursively using

$$\boldsymbol{H}_n = \begin{bmatrix} +\boldsymbol{H}_{n/2} & +\boldsymbol{H}_{n/2} \\ +\boldsymbol{H}_{n/2} & -\boldsymbol{H}_{n/2} \end{bmatrix} \tag{2.18}$$

with $\boldsymbol{H}_1 = [+1]$. For example, a Hadamard matrix of order $r = 3$ is given by

$$\boldsymbol{H}_8 = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{bmatrix}. \tag{2.19}$$

Moreover, $-\boldsymbol{H}_8$ is given by

$$-\boldsymbol{H}_8 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 \\ -1 & -1 & +1 & +1 & -1 & -1 & +1 & +1 \\ -1 & +1 & +1 & -1 & -1 & +1 & +1 & -1 \\ -1 & -1 & -1 & -1 & +1 & +1 & +1 & +1 \\ -1 & +1 & -1 & +1 & +1 & -1 & +1 & -1 \\ -1 & -1 & +1 & +1 & +1 & +1 & -1 & -1 \\ -1 & +1 & +1 & -1 & +1 & -1 & -1 & +1 \end{bmatrix}. \tag{2.20}$$

The codeword set of an order-$r$ Hadamard code is formed by the columns of $\pm\boldsymbol{H}_n$ (or rows of $\pm\boldsymbol{H}_n$ because $\boldsymbol{H}_n = \boldsymbol{H}_n^T$ where $(\cdot)^T$ denotes the transpose operator). Moreover, the codewords are denoted as $\{\pm\boldsymbol{h}^j : j = 0, 1, 2, \ldots, 2^r - 1\}$ where $+\boldsymbol{h}^j$ and $-\boldsymbol{h}^j$ represent the $j$-th columns of $+\boldsymbol{H}_n$ and $-\boldsymbol{H}_n$, respectively.

The code length of an order-$r$ Hadamard code is $2^r$. Denoting the code-bit positions by $\{0, 1, 2, 4, 8, \ldots, 2^{r-1}\}$, the remaining bit indices are positions for parity-check bits.

Since the information length is $r + 1$, the code rate $r_h$ is therefore given by

$$r_h = \frac{r+1}{2^r} \tag{2.21}$$

With an increase of order $r$, the code rate of Hadamard code decreases exponentially.

We assume that all Hadamard codewords are transmitted with the same probability over an additive white Gaussian noise (AWGN) channel having a noise variance $\sigma^2$. Let $\boldsymbol{c} = \{c[i] : i = 0, 1, 2, \ldots, 2^r - 1\}$ be a Hadamard codeword and $\boldsymbol{x} = \{x[i] : i = 0, 1, 2, \ldots, 2^r - 1\}$ be the received noisy observation. The logarithm-likelihood-ratio (LLR) value of the $i$-th bit of the codeword is given by [18]

$$\begin{aligned} L[i] =& \ln \frac{\Pr(c[i] = +1|\boldsymbol{x})}{\Pr(c[i] = -1|\boldsymbol{x})} \\ =& \ln \frac{\Pr(\boldsymbol{x}|c[i] = +1)\Pr(c[i] = +1)/p(\boldsymbol{x})}{\Pr(\boldsymbol{x}|c[i] = -1)\Pr(c[i] = -1)/p(\boldsymbol{x})} \\ =& \ln \frac{\Pr(\boldsymbol{x}|c[i] = +1)}{\Pr(\boldsymbol{x}|c[i] = -1)} \\ =& \ln \frac{\sum\limits_{c[i]=+1} \Pr(\boldsymbol{x}|\boldsymbol{c})}{\sum\limits_{c[i]=-1} \Pr(\boldsymbol{x}|\boldsymbol{c})}. \end{aligned} \tag{2.22}$$

where the summations $\sum\limits_{c[i]=\pm 1} \Pr(\boldsymbol{x}|\boldsymbol{c})$ are over all Hadamard codewords with $c[i] = +1$ or $-1$. As the code is transmitted in AWGN channel, (2.22) can be further shown equal to

$$\begin{aligned} L[i] =& \ln \frac{\sum\limits_{c[i]=+1} \exp(-\frac{\|\boldsymbol{c}-\boldsymbol{x}\|^2}{2\sigma^2})}{\sum\limits_{c[i]=-1} \exp(-\frac{\|\boldsymbol{c}-\boldsymbol{x}\|^2}{2\sigma^2})} \\ =& \ln \frac{\sum\limits_{c[i]=+1} \exp(\frac{\langle \boldsymbol{c}, \boldsymbol{x} \rangle}{\sigma^2})}{\sum\limits_{c[i]=-1} \exp(\frac{\langle \boldsymbol{c}, \boldsymbol{x} \rangle}{\sigma^2})} \end{aligned} \tag{2.23}$$

where $\langle \boldsymbol{c}, \boldsymbol{x} \rangle$ represents the inner product of $\boldsymbol{c}$ and $\boldsymbol{x}$.

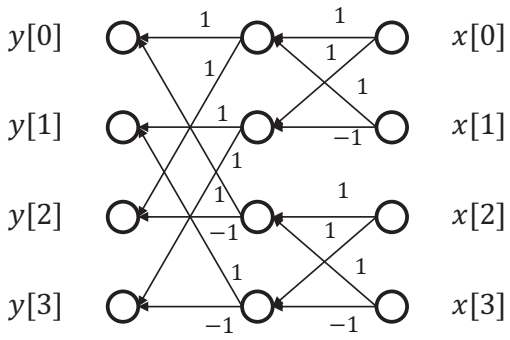


Figure 2.5: The "butterfly" structure of FHT (Hadamard matrix of order = 2).

As (2.23) shows, it is crucial to calculate the inner product of the received channel LLRs and Hadamard codewords with $c[i] = \pm 1$. The calculation can be easily accomplished by a technique called Hadamard transform (HT). The Hadamard transform is useful in many applications such as image coding [63] and algebraic space-time block codes [64]. Fast Walsh-Hadamard transform or fast Hadamard transform (FHT) is an efficient algorithm to compute the Hadamard transform. FHT is similar to fast Fourier transform as both of their calculations follow a "butterfly" pattern. An example of FHT with $r = 2$ is shown in Fig. 2.5. The FHT transforms the *a priori* LLR of the channel observations $\boldsymbol{x} = x[i]$ $(i = 0, 1, 2, \ldots, 2^r - 1)$ into $\boldsymbol{y} = y[i]$ $(i = 0, 1, 2, \ldots, 2^r - 1)$, i.e.,

$$\boldsymbol{y} = \boldsymbol{H}_{2^r}\, \boldsymbol{x}. \tag{2.24}$$

Moreover, (2.23) can be efficiently calculated by using a-posteriori-probability-FHT (APP-FHT) or dual-FHT (DFHT). An example of DFHT with $r = 2$ is shown in Fig. 2.6. The DFHT calculates the numerator and denominator of (2.23) from the results of FHT and then decodes the Hadamard code bit-wisely. To summarize, the APP decoding of Hadamard code contains the following two stages.

1. Perform a fast-Hadamard-transform (FHT) on the input *a priori* information to obtain $\boldsymbol{y}$.

2. Perform a-posteriori-probability-FHT (APP-FHT) to update the *a posteriori* LLR

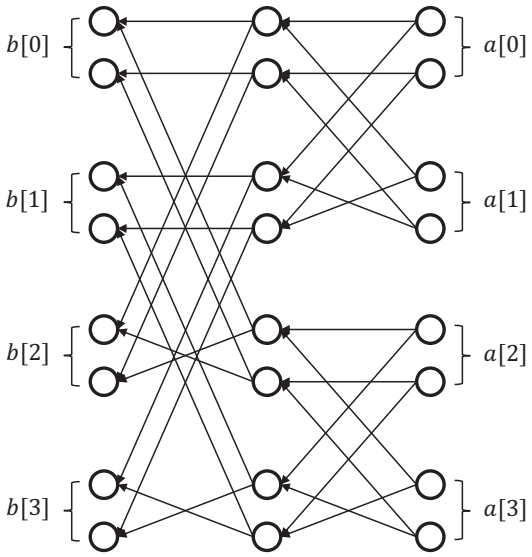


Figure 2.6: The "butterfly" structure of DFHT (Hadamard matrix of order = 2).

of the information bits.

### 2.4.2 Turbo Hadamard Code

A turbo Hadamard code concatenates a turbo code with Hadamard codes. Fig. 2.7 shows the encoder block diagram and code structure of a convolutional Hadamard code and Fig. 2.8 shows the code structure of a turbo Hadamard code [18]. A convolutional Hadamard code is a concatenation of a single-parity-check (SPC) code, an $S$-state recursive convolutional code (RCE) and a Hadamard code. A turbo Hadamard code is the combination of a number of convolutional Hadamard codes, say $M$ convolutional Hadamard codes, carrying the same but interleaved information bits.

We refer to Fig. 2.7(b). In a convolutional Hadamard code, each message $\boldsymbol{D}$ contains $L$ bits and is segmented into $K$ blocks where each block $\boldsymbol{d}_k$ $(k = 1, 2, \ldots, K)$ contains $r$ bits, i.e., $L = rK$. The parity bit $q_k{}'$ of $\boldsymbol{d}_k$ is computed and sent through an $S$-state rate-$1/2$ systematic recursive convolutional encoder producing convolutional codes denoted as $(q_k{}', q_k)$. Finally $(\boldsymbol{d}_k, q_k)$ is encoded into an order-$r$ Hadamard code $\boldsymbol{c}_k = (\boldsymbol{d}_k, q_k, \boldsymbol{p}_k)$, where $\boldsymbol{p}_k$ represents the parity bits in the Hadamard code. The output of the convolutional encoder therefore forms part of the input information block to the

Hadamard encoder.

The code length $l$ of an order-$r$ turbo Hadamard code is given by $l = rK+MK(2^r-r)$ and the information length is $rK$. The code rate $r_c$ is therefore given by

$$r_c = \frac{rK}{rK + MK(2^r - r)}. \tag{2.25}$$

The recursive convolutional codes $\{q_k\}, k = 1, 2, ..., K$ and random interleavers reduce the correlation of input and output weights of each component codes. It is shown in [18] that the input and output weights of a turbo Hadamard code are (approximately) uncorrelated for an input weight larger than 1. According to the above assumption, the parity weight of a turbo Hadamard code approaches Gaussian, which indicates a close resemblance between a turbo Hadamard code and a random code. Thus a low-rate turbo Hadamard code approaches the channel capacity when the code length approaches infinity.

The decoding of a turbo Hadamard code follows a similar principle as other turbo-like codes. Assume a convolutional Hadamard codeword $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, ..., \boldsymbol{c}_K]$ is transmitted in an AWGN channel with noise variance $\sigma^2$ and a vector $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, ..., \boldsymbol{x}_K]$ is received. The *a-posteriori-probability* likelihood ratio of the $i$th bit in the $k$th block is given by [18]

$$L_k[i] = \ln \frac{\sum\limits_{H[i,j]=\pm 1} \gamma_k(\pm \boldsymbol{h}^j)\alpha(\boldsymbol{s}_k)\beta(\boldsymbol{s}_{k+1})}{\sum\limits_{H[i,j]=\mp 1} \gamma_k(\pm \boldsymbol{h}^j)\alpha(\boldsymbol{s}_k)\beta(\boldsymbol{s}_{k+1})} \tag{2.26}$$

where

$$\gamma_k(\pm \boldsymbol{h}^j) = \Pr(\boldsymbol{x}_k | \boldsymbol{c}_k = \pm \boldsymbol{h}^j) \tag{2.27}$$

is the *a priori* information and is calculated based on the channel LLRs

$$\boldsymbol{L}_k = \frac{2\boldsymbol{x}_k}{\sigma^2}; \tag{2.28}$$

and $\alpha(\boldsymbol{s}_k)$ and $\beta(\boldsymbol{s}_{k+1})$ are calculated using the forward-backward recursion algorithm

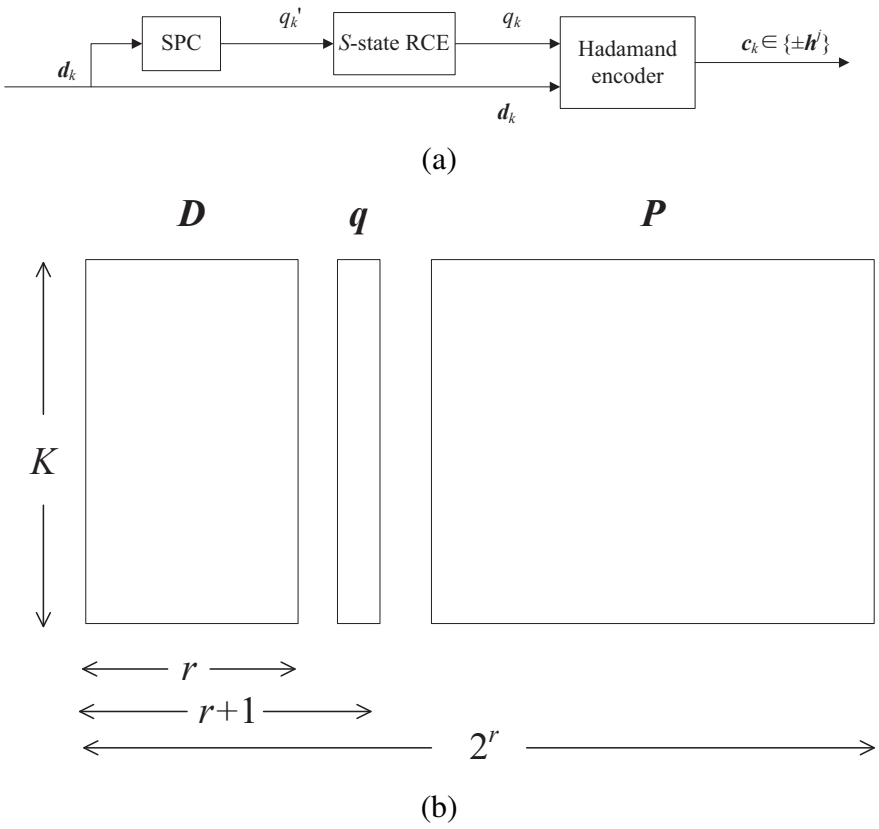


Figure 2.7: convolutional Hadamard code. (a) Encoder block diagram and (b) code structure. SPC: single-parity check; RCE: recursive convolutional encoder.

in the Bahl-Cocke-Jelinek-Raviv (BCJR) decoder [65].

Referring to Fig. 2.9, a turbo Hadamard decoder consists of $M$ convolutional Hadamard (component) decoders ($\text{DEC}_1$, $\text{DEC}_2$, ..., $\text{DEC}_M$) and the *a posteriori* LLR information of a component code becomes the input of the next component code. Moreover, each component decoder is composed of three main stages.

1. Perform a fast-Hadamard-transform (FHT) on the input *a priori* information to prepare information ($\gamma_k(\pm \boldsymbol{h}^j)$) for the next two stages.

2. Perform Bahl-Cocke-Jelinek-Raviv (BCJR) decoding.

3. Perform a-posteriori-probability-FHT (APP-FHT) to the data obtained from the first two stages and updating the *a posteriori* LLR of the information bits.

The theoretical code performance has been evaluated using the extrinsic informa-

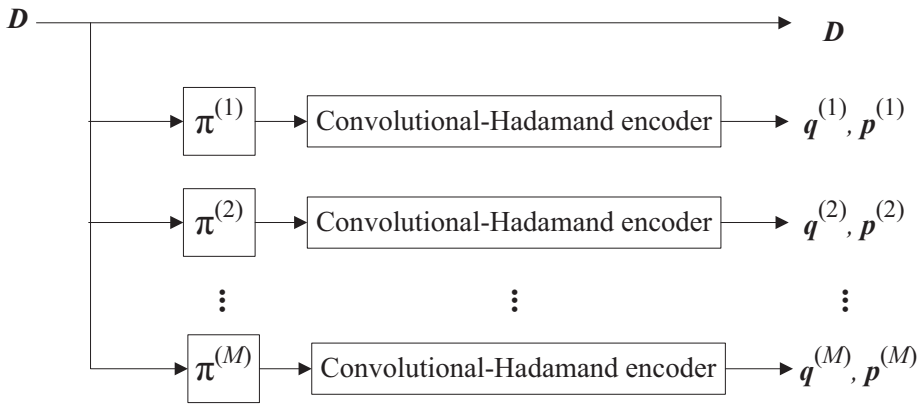


Figure 2.8: Turbo Hadamard code structure.

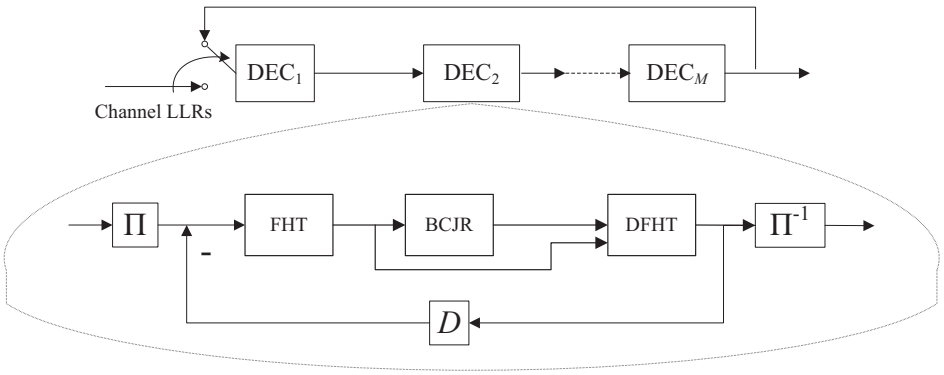


Figure 2.9: Decoder structure.

tion transfer (EXIT) algorithm [66], and the best achievable performance of the turbo Hadamard code is $E_b/N_0 = -1.3$ dB, about 0.29 dB from the ultimate Shannon limit.

### 2.4.3 Concatenated Zigzag Hadamard Code

A zigzag Hadamard code (ZHC) is graphically described in Fig. 2.10(a) where each segment represents an order-$r$ Hadamard code [20]. The overall code structure is also shown in Fig. 2.10(b). Assuming an information block $\boldsymbol{D}$ with length $L = rK$ is segmented into $K$ sub-blocks. For the $k$th segment ($k = 1, 2, \ldots, K$), the information bits $\boldsymbol{d}_k = [d_k(1), d_k(2), ..., d_k(r)]$ are represented by blank nodes (area) and the remaining parity-check bits are represented by grey nodes (area). Moreover, the last parity bit of each segment is copied to the first input of the next segment and is denoted as the common bit (black nodes/area in the figures). Note that the first input

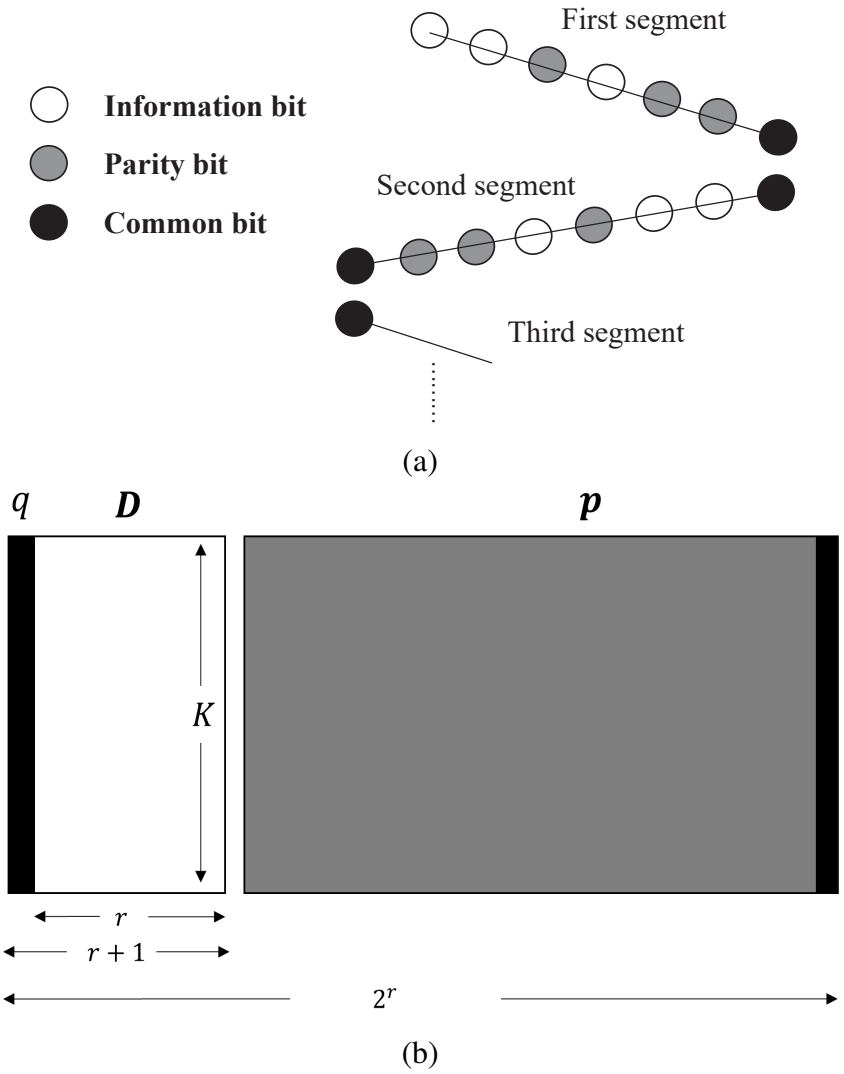


Figure 2.10: A zigzag Hadamard code. (a) Graphical representation and (b) overall structure. White: information bits; grey: parity bits; black: common bits.

bit of the first segment is fixed as 0 and is omitted. We denote the Hadamard codeword in the $k$th segment as $\boldsymbol{c}_k = [c_k(0), ..., c_k(2^r - 1)]$, where $c_k(0) = c_{k-1}(2^r - 1)$ and $c_k(2^{j-1}) = d_k(j)$, $j = 1, 2, ..., r$; and denote the common bit $q_k = c_k(0) = c_{k-1}(2^r - 1)$ and the parity bits $\boldsymbol{p}_k = \{c_k(i), i \neq 0, i \neq 2^{j-1}, j = 1, 2, ..., r\}$. The $k$th segment of a ZHC codeword can then be rewritten as $\boldsymbol{c}_k = (\boldsymbol{d}_k, q_k, \boldsymbol{p}_k)$. The encoding process of ZHC is a Markov process and the correlation between any two consecutive segments depends only on the common bit.

Note that the order $r$ plays an important role in the error performance. For an even-order systematic ZHC, the code is said to be "weight-recursive" [20], i.e., any information sequence with Hamming distance one will produce a codeword with Hamming distance weight $\rightarrow \infty$ when its length $\rightarrow \infty$. For the case of odd-order systematic ZHC, the aforementioned property does not hold. The weight-recursive property is the necessary condition for a code to achieve error free performance in an AWGN channel and it implies the existence of a threshold [34]. (In this thesis, we will only consider systematic CZHC with an even order.)

To decode the ZHC, a two-way decoding algorithm with two stages can be used [20, 67].

1. Forward recursion: Starting from the first segment to the $(k-1)$th segment, perform FHT and DFHT on the current segment to obtain the APP LLRs of the bits based on the aforementioned discussion; then use the APP LLR of the last bit of the current segment to update the *a priori* LLR of the first bit of the next segment.

2. Backward recursion: Starting from the $K$th segment to the first segment, perform FHT and DFHT on the current segment to obtain the APP LLRs of the bits (including information bits); then use the extrinsic LLR of the first bit of the current segment to update the *a priori* LLR of the last bit of the previous segment.

Fig. 2.11 shows the code structure of a CZHC [20] with $M$ component codes. $M$

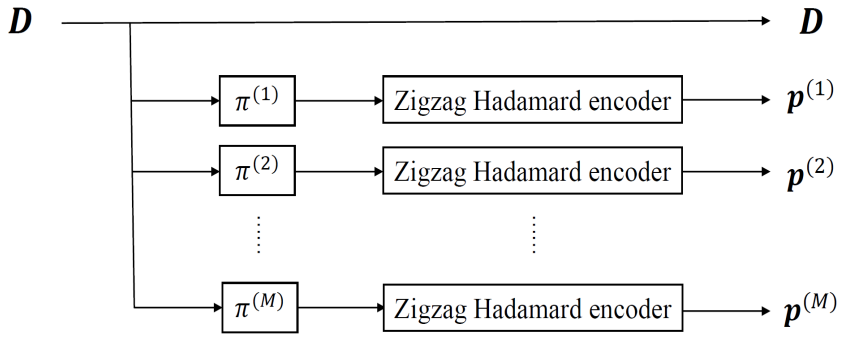


Figure 2.11: Structure of a concatenated zigzag Hadamard code.

copies of same but interleaved information bits are sent to $M$ zigzag Hadamard encoders producing $M$ copies of parity bits. The information $\boldsymbol{D}$ together with the parity bits $\boldsymbol{p}^{(1)}, \boldsymbol{p}^{(2)}, ..., \boldsymbol{p}^{(M)}$ are sent to the channel. The decoding of CZHC involves the interleaving and passing of LLRs among different zigzag Hadamard codes (or component codes), which is similar to THC code. The code performance can be evaluated using the EXIT functions [68].

## 2.5 Summary

In this chapter, we provide a literature review on some basic concepts of channel coding. We first review the Shannon's noisy channel coding theory. The Shannon's theory provides an upper-bound of the performance that a code can achieve. Then we review turbo codes and LDPC codes which adopt iterative decoding mechanisms and can approach the Shannon limit closely. Finally, we review two of the ultimate-Shannon-capacity-approaching codes, turbo Hadamard codes and concatenated zigzag Hadamard codes.

# Chapter 3

# Design of Turbo Hadamard Encoder/Decoder System

In Chapter 2, we have reviewed the turbo Hadamard code. In this chapter, we propose a pipelined digital design of a turbo Hadamard encoder/decoder system. To accomplish a high throughput, we make use of a multiple sub-decoder architecture to process multiple codes at the same time. Each sub-decoder is processing the data of one codeword at anytime; and data from the same codeword will be processed by different sub-decoders at different times. One iteration is completed when the data from the same codeword is processed by all the sub-decoders once. Moreover, tens of iterations are required to complete the decoding. Hence, the design challenges include control of data flow within a sub-decoder; control of data flow among sub-decoders; proper data storage to avoid data access conflicts; conversion of data formats to facilitate computations; effective interleavers that cause minimum latency. In order to achieve performance close to the ultimate Shannon limit, code lengths of 216450, 287235 and 358020 are used. Since the code lengths are relatively long, effective use of data storage is crucial. Also, the transmitter is required to send code bits in a way that facilitates storing and processing of data at the receiving end. Note that new codeword data are being received continuously and need to be stored as the decoder is processing the existing

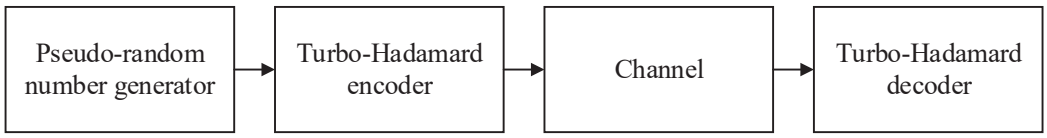


Figure 3.1: Overall design of the turbo Hadamard encoder/decoder system.

codewords. The theoretical throughput of our digital design is derived based on the given parameters such as hardware operating frequency, code length, number of iterations, and length of the turbo trellis. Results indicate that at an operating frequency of 100 MHz, the proposed digital turbo Hadamard encoder/decoder system achieves throughputs of 1.92 Gbps, 2.56 Gbps and 3.2 Gbps at code lengths of 216450, 287235 and 358020, respectively. The encoder/decoder system also realizes a bit error rate of $10^{-5}$ at $E_b/N_0 = -0.45$ dB, i.e., 1.14 dB from the ultimate Shannon limit.

## 3.1 Hardware Design

The overall design of our turbo Hadamard encoder/decoder system is shown in Fig. 3.1. The design consists mainly of three parts (besides the pseudo-random number generator (PRNG) that generates the information bits): (i) turbo Hadamard encoder, (ii) channel that simulates the quantized signal after channel noise is added, and (iii) turbo Hadamard decoder. The encoder continuously send turbo Hadamard code bits to the channel (noise generator), which outputs quantized noisy signals to the decoder. The decoder is required to estimate the transmitted codeword based on the noisy observations. Our objective is to embed the encoder/transmitter, the channel and the receiver/decoder in one FPGA board.

### 3.1.1 Encoder

The main stages of the encoder include:

1. Generate information bits using a pseudo-random number generator (PRNG)

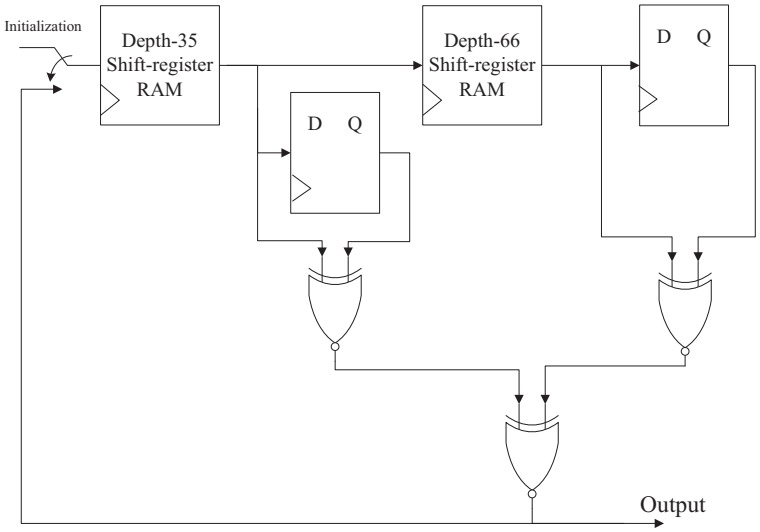


Figure 3.2: A PRNG based on linear feedback shift registers.

which is implemented with linear feedback shift registers (LFSRs) [69, 70]. Fig. 3.2 shows the detailed structure of one PRNG used. It contains 102 shift registers that implements the feedback equation $1 + x^{35} + x^{36} + x^{101} + x^{102}$. Similar PRNGs are also used in the channel to generate noise.

2. Input the information bits to the first convolutional-Hadamard encoder and produce the first set of parity bits.

3. Input the original information bits to the interleaver $\prod_{(1)}$. Send the interleaved information bits to the second convolutional-Hadamard encoder and produce the second set of parity bits.

4. Repeat Step 3) with different interleavers for the generation of the third, fourth, ..., $M$-th sets of parity bits.

5. The information bits and parity bits are stored in two different buffers (see Fig. 3.3) before sent to the "Channel" block.

The hardware structure of the encoder is shown in Fig. 3.4. The encoder module follows a pipeline manner and is very simple. Thus the delay in encoder is negligible compared with that in the receiver/decoder.

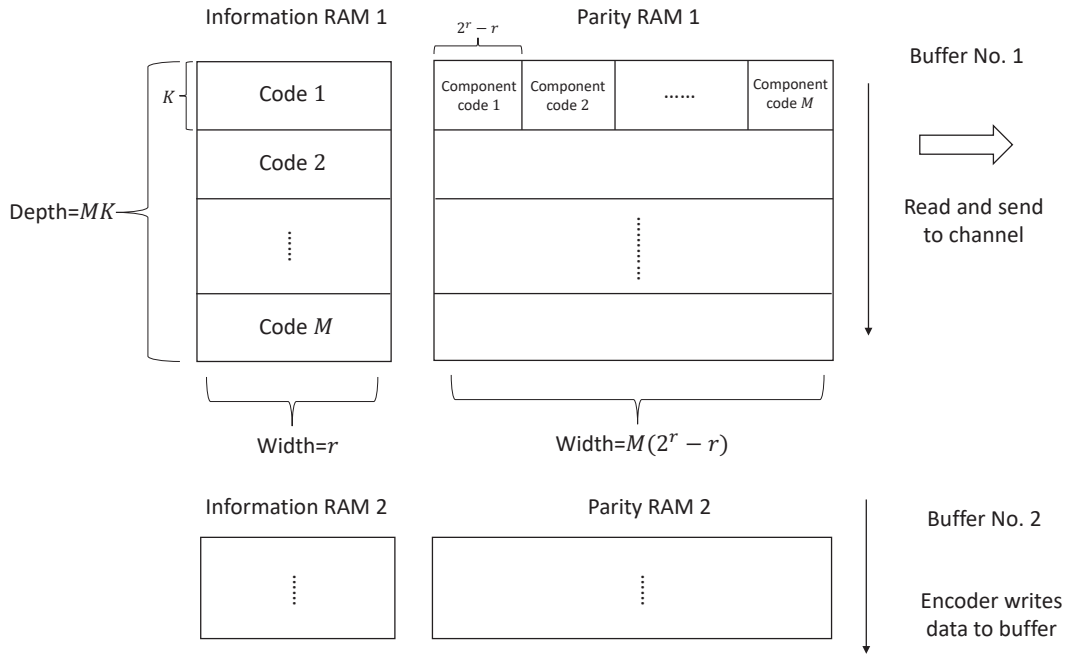


Figure 3.3: The structure of two buffers, each of which stores $M$ turbo Hadamard codewords.

Referring to Fig. 3.3, our encoder design uses two sets of buffers to ensure uninterrupted data transmission, and each set consists of a RAM block storing the information bits and another block storing the parity check bits. To improve the throughput, our encoder/decoder system is designed to process $M$ turbo Hadamard codes at the same time. Accordingly, the information RAM in the encoder stores $M$ sets of information bits and the parity RAM stores $M^2$ sets of parity bits (recall that $M$ sets of (component) parity bits are generated for each set of information bits). The width of the information RAM is $r$ and the width of the parity RAM is $M(2^r - r)$. Thus $K$ rows in each of the two RAMs are required to store a complete THC codeword. As a consequence, the depths of both RAMs are $MK$. Note that the $M$ THC codewords are stored and transmitted in such an arrangement so as to facilitate the receiver storing the channel messages and decoding the codes.

The work flow of the buffers is as follows:

1. The encoder generates information and parities for the first $M$ turbo Hadamard codes and writes them into the first buffer.

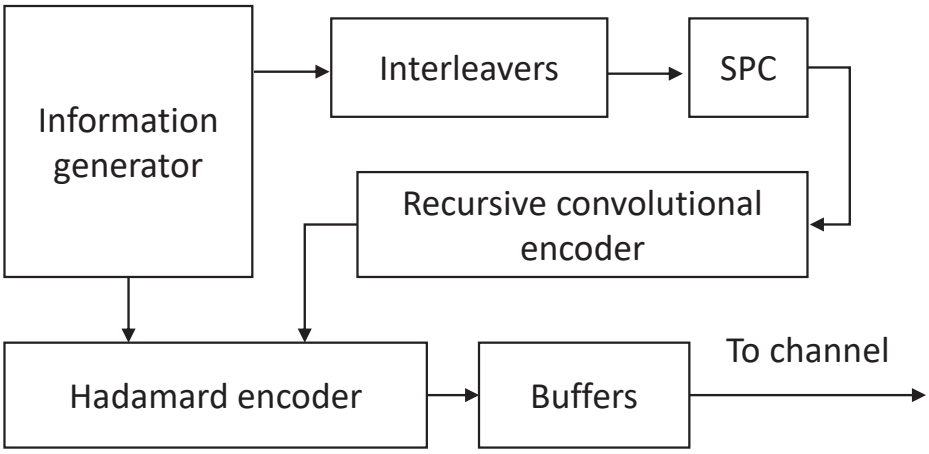


Figure 3.4: Hardware structure of a turbo Hadamard encoder.

2. The transmitter reads data from the first set of buffers and sends them to the "channel". Simultaneously, the encoder generates information and parities for the next $M$ turbo Hadamard codes and writes them into the second buffer.

3. When the transmission of the $M$ turbo Hadamard codes in the first set of buffers is completed, the writing to the second set of buffers is also completed. The $M$ turbo Hadamard codes in the second set of buffers are ready for transmission. The encoder then generates the next $M$ turbo Hadamard codes and writes into the first set of buffers.

4. The transmitter alternately transmits codes from the two set of buffers by repeating step 2) and 3).

The transmitter reads data from the buffers row-by-row. For each row/block, the total number of bits (information plus parity) is $r+M(2^r-r)$. Assume $W_t$ bits are sent in one clock cycle, and for convenience, $W_t$ divides $r+M(2^r-1)$. It then takes $\frac{MK(r+M(2^r-r))}{W_t}$ clock cycles to completely transmit $M$ consecutive turbo Hadamard codes. Note that the decoders must be capable of decoding the previous $M$ consecutive turbo Hadamard codes during the transmission. Assume the operating frequency of transmitter is $f_t$.

Then from the transmitter's perspective, the throughput of the whole system $T_t$ is

$$T_t = f_t W_t \quad \text{bits/s.} \tag{3.1}$$

Note that the throughput of the decoder, denoted by $T$, should be larger than or equal to $T_t$.

### 3.1.2 Interleaver

The function of the interleavers used in a turbo Hadamard code is slightly different from those used in turbo codes. The interleavers in a turbo Hadamard code not only shuffle the order of information bits, but also change subsequent SPCs and Hadamard codes. The principle used to design interleavers for turbo codes is therefore no longer adequate, but is still a good reference. Random interleavers help to avoid low-weight turbo Hadamard codewords [18, 37, 71]. However, the design of large-size random interleavers increases the latency/complexity of the encoder/decoder significantly. Our goal is to keep the randomness property of the interleavers, and to perform interleaving in parallel and thus increase the throughput of the whole system.

The main idea of performing interleaving in parallel is by dividing a size-$N$ interleaver into $m$ windows of size $W$ (i.e., $N = mW$). During the encoding/decoding process, contentions may happen if two or more processors try to access the same memory window. Contention-free interleavers [72] allow the interleaving process to operate simultaneously in each window. The throughput of the turbo decoder is thus theoretically increased by $m$ times.

Inter-window shuffle (IWS) interleavers is a class of contention-free interleavers [72–75]. In [72], two kinds of IWS interleavers have been illustrated: fixed inter-window shuffle (FIWS) with variable intra-window permutations and variable inter-window shuffle (VIWS) with a fixed intra-window permutation. Both interleavers consist of two stages: i) an intra-window permutation and ii) inter-window shuffle

patterns. The FIWS interleavers fix the inter-window shuffle pattern and $m$ different sub-interleaver patterns need to be designed for $m$ windows. For the VIWS interleavers, the intra-window permutation, i.e., the sub-interleavers used in every window are fixed, while the inter-window shuffle patterns that processors access are scrambled. For a turbo Hadamard code, a FIWS interleaver will shuffle the order of bits in each column of the information block $\boldsymbol{D}$, while a VIWS interleaver will shuffle the order of bits in each row of $\boldsymbol{D}$. Fig. 3.5 shows the operation of a FIWS interleaver and a VIWS interleaver.

Usually FIWS interleavers are not suitable for turbo or turbo-like codes. The reason is that it is equivalent to dividing a big interleaver into $m$ smaller interleavers. The whole codeword is also divided into $m$ smaller concatenated codewords, leading to an increase of low-weight codewords. However in a turbo Hadamard code, the situation is different. Assuming $\boldsymbol{D}$ is a weight-1 information word and interleavers $\pi_1, \pi_2, ...\pi_r$ in Fig. 3.5 are random, the only non-zero information bit can be in any place of $\boldsymbol{D}$ after interleaving, thus resulting the output weight of parity $\boldsymbol{P}$ nearly independent of the input weight 1. Considering that this is a linear code, any codeword can be regarded as the composition of codewords with weight-1 information. It implies a very low correlation between the input weight and the output weight in a turbo Hadamard code using FIWS interleaving. The chance of generating low-weight codewords does not increase and the performance of the FIWS interleavers should be close to unstructured random interleavers.

On the contrary, VIWS interleavers should be avoided in turbo Hadamard codes. VIWS interleaver shuffles the order of bits in windows but the intra-window order remains the same, resulting the same SPCs for all component codes. In Fig. 3.5, changing the order of bits in every row will not change $\boldsymbol{q}^{'}$. It can be easily seen that low-weight codewords will remain low-weight after VIWS interleaving.

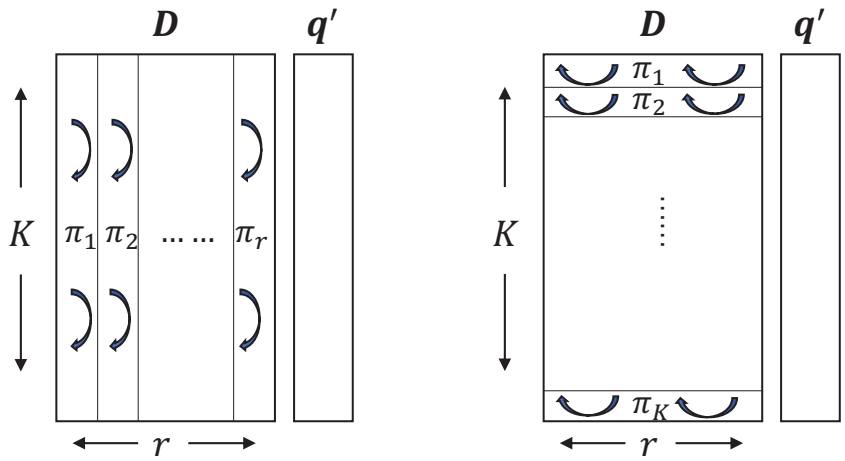


Figure 3.5: Operation of a FIWS interleaver (left) and a VIWS interleaver (right) in a turbo Hadamard code.

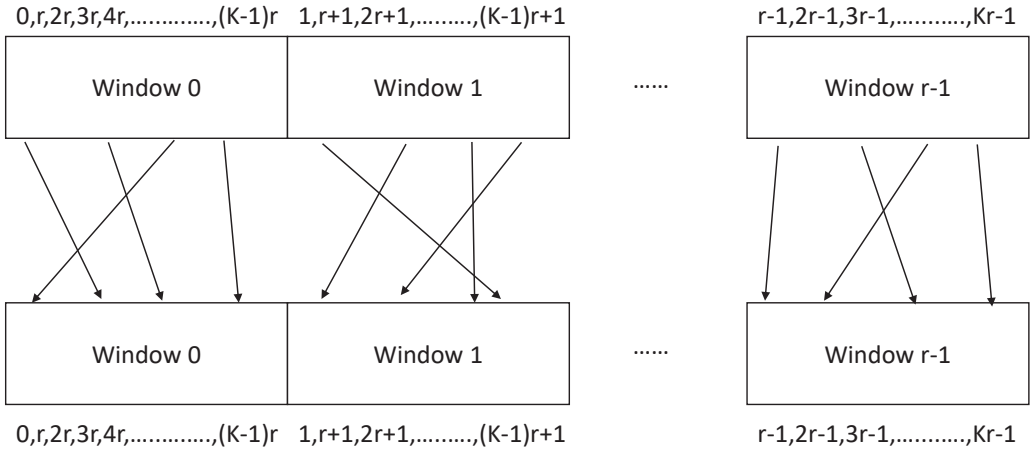


Figure 3.6: Illustration of the FIWS interleaver used in our encoder/decoder system.

For turbo Hadamard code, we have

$$m = r \quad \text{and} \quad W = K. \tag{3.2}$$

Fig. 3.6 shows the structure of the FIWS interleaver used in our system. To compute SPCs more efficiently, the bit order in Fig. 3.6 is different from that in [75]. We denote the index of an information bit by $i$ ($i = 0, 1, 2, \ldots, Kr - 1$). Then window $w$ ($w = 0, 1, 2, ..., r - 1$) contains information bits with indices $i$ satisfying $w = (i \mod r)$. Note that for convenience, we map bits in window $w$ to the same window $w$, and the

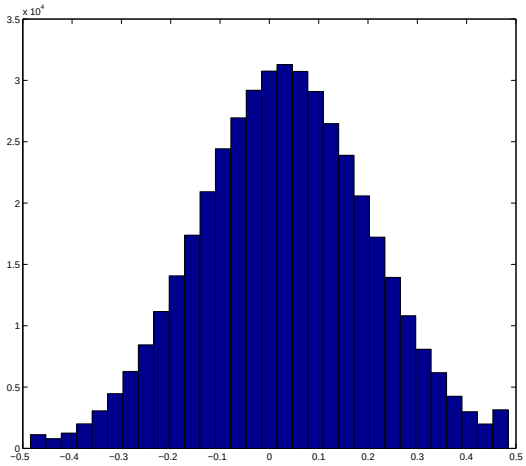


Figure 3.7: Histogram of the channel LLRs when "+1" are used as inputs.

interleaving process in different windows are different. Our simulation results show that there is no obvious performance degradation between FIWS random interleavers and traditional random interleavers.

### 3.1.3 Channel

The channel LLR $\boldsymbol{L}_k = \frac{2x_k}{\sigma^2}$ act as a soft input to the turbo Hadamard decoder. In a practical design, fixed-point values are used instead of floating-point values because of the limitation of the hardware resources. Since the code length of turbo Hadamard code is usually quite long (216450, 287235 and 358020 in our case), the storage of channel LLRs is a huge issue. We aim to use as few bits to quantize the channel LLRs as possible and to maintain a good error performance of the decoder at the same time.

We only consider the performance of turbo Hadamard codes under very low SNRs, i.e., $E_b/N_0 < 0$ dB. For a given code rate ($r_c$) and $E_b/N_0$, the noise variance is calculated using $\sigma^2 = 1/[(2r_c)(E_b/N_0)]$. We transmit a bit "0" as "+1" and bit "1" as "−1". Assuming a "+1" is transmitted to the channel with $E_b/N_0 = -1$ dB, the histogram of the channel LLRs is shown in Fig. 3.7. The figure shows that most of the LLR values falls within the range $(-0.5, 0.5)$, and so capping LLRs within $(-0.5, 0.5)$ should not lead to too much information loss. Computer simulations also show no obvious

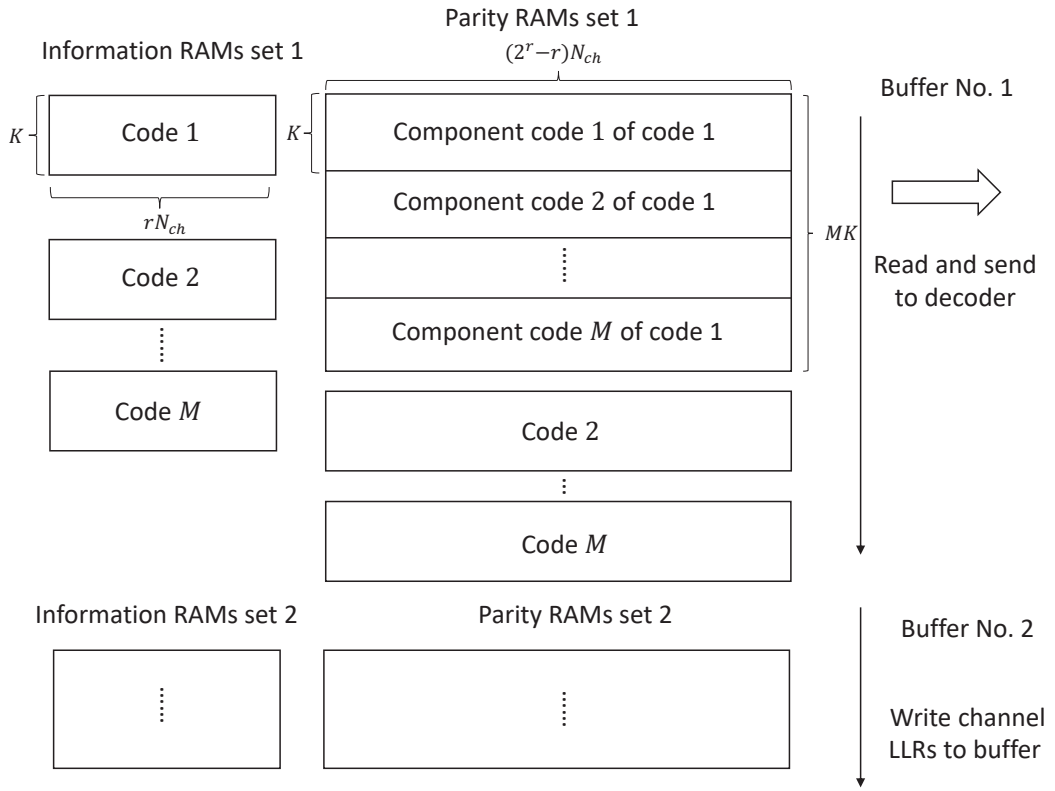


Figure 3.8: The structure of two buffers at the receiver, each of which stores the channel LLR values of $M$ turbo Hadamard codewords.

degradation in the overall error performance. Suppose a $N_{ch} = 6$-bit linear quantization is used. There are $2^6 - 1 = 63$ quantized values for LLRs and the quantization interval is $1/64$. Then the LLRs values are $-31/64, -30/64, ..., -1/64, 0, 1/64, ..., 30/64, 31/64$. Depending on the transmitted value being "+1" or "−1", we calculate the probability that a noisy signal (received signal) falls within the $2^{N_{ch}} - 1$ different intervals. We use sets of LFSRs to generate a random integer $f$. Based on the (i) code rate, (ii) $E_b/N_0$ and (iii) input bit, $f$ is mapped to the corresponding noisy channel output using a specific lookup table (see Appendix 3). In summary, the channel output ($N_{ch}$ LLR bits) for every code bit is based on the code rate, $E_b/N_0$ and the input code bit value.

### 3.1.4 Decoder

A receiver is responsible for receiving channel observations continuously. Similar to the transmitter, the receiver uses two sets of buffers to store channel observations. Referring to Fig. 3.8, the channel observations are stored in RAMs with a similar arrangement as in the transmitter side. However, instead of code bits, the store values are $N_{ch}$-bit LLR values. The decoder decodes $M$ turbo Hadamard codes from one buffer, while the receiver receives and stores the LLRs of the next $M$ turbo Hadamard codes into another buffer. Note that the receiver works continuously and the decoder must finish decoding before the next $M$ to-be-decoded turbo Hadamard codes are fully received and stored.

In our design, we build $M$ (component) sub-decoders when there are $M$ component codes. Moreover, the decoding process follows a pipeline nature and hence the $M$ sub-decoders operate on $M$ different codewords at the same time. For each THC sub-decoder, the following parameters are used.

- Channel LLRs are represented by $N_{ch}$ bits
- FHT core inputs are represented by $N_{FHT}$ bits
- Values in the BCJR algorithm are represented by $N_{BCJR}$ bits
- Dual FHT (DFHT) processor takes in $N_{DFHT}$-bit inputs and produces $N_{DFHT}$-bit outputs at each stage
- $N_{DFHT}$-bit LUTs are used in the DFHT computations

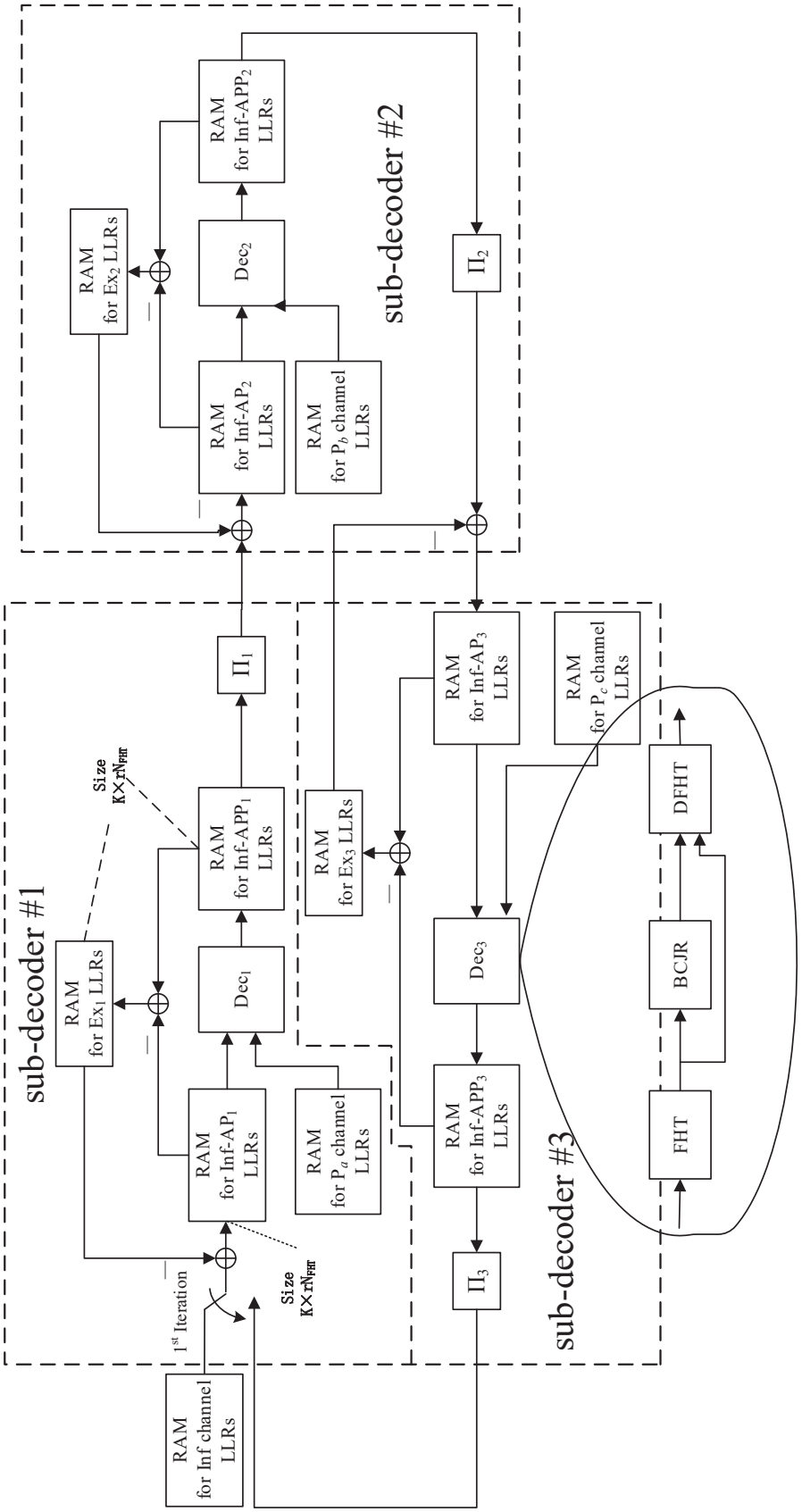


Figure 3.9: Overall Decoder Structure.

Referring to Fig. 3.9 where an $M = 3$ THC decoder is shown, the following operations are performed in each sub-decoder.

1. (i) The output LLRs of the message bits from the *previous* sub-decoder (after interleaving), (ii) the extrinsic LLRs of the message bits produced by the current sub-decoder in the previous iteration, and (iii) channel LLRs of the parity bits of the current sub-decoder (stored in RAM) are input to the sub-decoder. The extrinsic LLRs of the current sub-decoder are subtracted from the output LLRs from the previous sub-decoder. Then together with the channel LLRs of the parity bits, these signals are input in a pipeline manner to the FHT processor. The FHT processor contains a number of stages. Let $\boldsymbol{c}$ be the transmitted codeword and $\boldsymbol{x}$ be the channel observation. Let $\boldsymbol{c}_k$ be the $k$th section of a convolutional-Hadamard code and $\boldsymbol{x}_k$ be the corresponding $k$th channel observation. Then the FHT processor calculates $\gamma_k(\pm\boldsymbol{h}^j) = \Pr(\boldsymbol{x}_k|\boldsymbol{c}_k = \pm\boldsymbol{h}^j)$ where $\pm\boldsymbol{h}^j$ stands for all valid Hadamard codes with order $r$ (see (2.27)).

2. Sets of outputs are sent to the BCJR processor. The outputs of the FHT block $\gamma_k(\pm\boldsymbol{h}^j)$ are used to calculate the probabilities of a transition between the states in a convolutional code based on the trellis of the code. The BCJR decoder is responsible for calculating $\gamma_k(\boldsymbol{c}_h)\alpha(s_k)\beta(s_{k+1})$, where $\alpha(s_k)$ and $\beta(s_{k+1})$ are the forward and backward recursion of BCJR decoder [65].

3. The computed set of $\gamma_k(\boldsymbol{c}_h)\alpha(s_k)\beta(s_{k+1})$ together with outputs from the FHT block are passed to the DFHT processor to compute the new LLR values of the message bits.

4. Each set of output LLR values from the DFHT processor will be stored in appropriate RAM locations after block interleaving. The interleaved LLRs are then passed to the next sub-decoder.

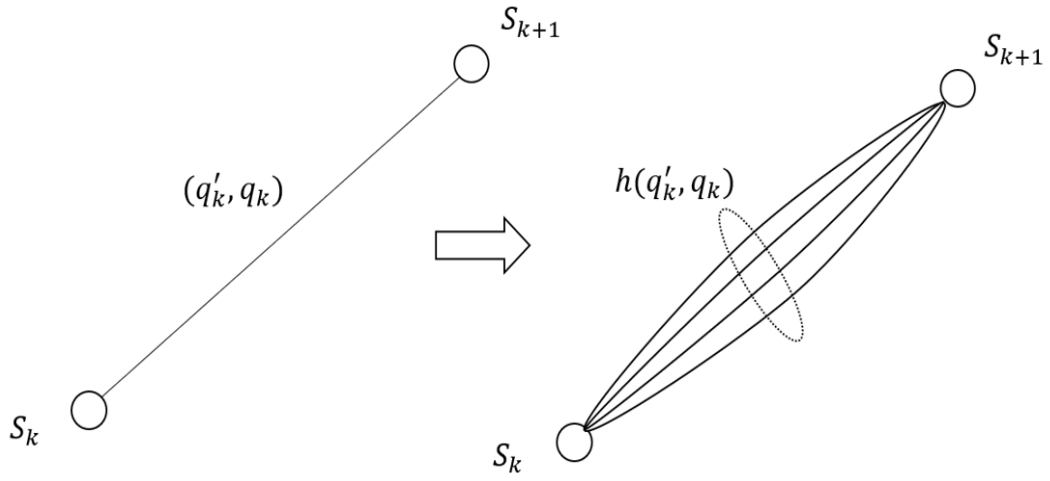


Figure 3.10: Trellis illustration of THC.

5. At the same time, the extrinsic LLRs of the message bits generated in this iteration are computed and stored.

Note that the BCJR computation in THC decoder is slightly different from that of the turbo decoder. Consider the state transition pair $(S_k, S_{k+1})$ which involves only one convolutional code $(q_k^{'}, q_k)$. Each of the convolutional code, however, corresponds to $2^{r-1}$ Hadamard codes $h(q_k^{'}, q_k)$. Thus, when calculating the transition probabilities of $(S_k, S_{k+1})$, all of the involved Hadamard codes $h(q_k^{'}, q_k)$ should be considered. The trellis of THC is illustrated in Fig. 3.10.

The BCJR computation of THC consists of the following stages.

1. Initialization. Calculate the transition probabilities

$$B(S_k, S_{k+1}) = \sum_{\text{all } \gamma_k \text{ in the branch}} \gamma_k \tag{3.3}$$

for all $k$.

2. Forward recursion. For $k = 1, 2, ..., K-1$, calculate

$$\alpha(S_k) = \sum_{\forall S_{k-1}} B(S_{k-1}, S_k)\alpha(S_{k-1}). \tag{3.4}$$

3. Backward recursion. For $k = K-2, K-1, ..., 0$, calculate

$$\beta(S_k) = \sum_{\forall S_{k+1}} B(S_k, S_{k+1})\alpha(S_{k+1}). \tag{3.5}$$

For the initialization stage, $2^{r-1}$ $\gamma_k$ are added in each of the branches. Thus it costs a lot of hardware resources and time efforts to perform the initialization stage. However, we find that in each of the state transition pair $(S_k, S_{k+1})$, the Hadamard codeword with the largest trellis probability $\gamma_k$ contributes the most to that trellis. The contribution of that single codeword is usually more than 95%. To simplify logics and to lower the decoding latency without obvious loss of performance, we replace Eq. (3.3) by

$$B(S_k, S_{k+1}) = \max_{\text{all } \gamma_k \text{ in the branch}} \gamma_k. \tag{3.6}$$

Note that the $\gamma_k$ in Eq. (3.6) equal to the exponential of the outputs of FHT. For further simplification, we use

$$B(S_k, S_{k+1}) = \max_{\text{all FHT}_{\text{out}} \text{ in the branch}} \text{FHT}_{\text{out}}. \tag{3.7}$$

The pipeline illustration of a sub-decoder is shown in Fig. 3.11. The solid lines represent valid input/output while the dash lines represent invalid input/output. For the first $K$ clock cycles, the FHT block performs FHT calculation to the incoming *a priori* information. At clock $r$, the FHT block outputs first valid trellis information to the BCJR block and the BCJR block starts performing forward recursion and calculating $\alpha(s_1), \alpha(s_2), ..., \alpha(s_K)$. At clock $r + K + 1$, the forward recursions for all $\alpha(s_k), k = 1, 2, ..., K$ complete and backward recursions for $\beta(s_{K+1}), \beta(s_K), ..., \beta(s_2)$ begin. The

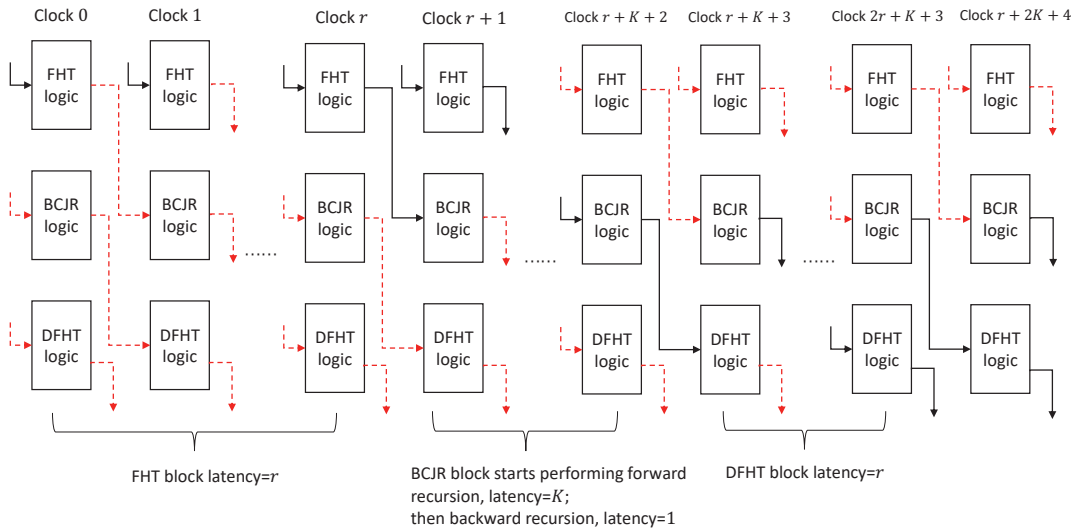


Figure 3.11: Pipeline structure within a sub-decoder. Solid line represents valid input/output, dash line represents invalid input/output.

BCJR block starts outputting $\gamma_k(\boldsymbol{c}_h)\alpha(s_k)\beta(s_{k+1})$ for $k = K, K-1, ..., 1$ to the DFHT block. Similar to the FHT block, the latency in the DFHT block is also $r$. So the total latency of a sub-decoder is $2r + K + 3$ clock cycles. Note that $K$ is usually much larger than $r$ and the latency can be approximated by $K$.

The inter-sub-decoder structure is also built in a pipeline manner and is shown in Fig. 3.12. The FIWS interleaver consists of $r$ depth-$K$ RAMs and $r$ depth-$K$ ROMs. In the first $K$ clock cycles, the $i$th sub-decoder ($i = 1, ..., M$) writes the APP information into the RAMs in natural order. Once all the APP information is stored in the RAMs, the RAMs start sending the information to the next sub-decoder in an interleaved order. The interleaving pattern is stored in the $r$ ROMs. The latency of the interleaver is thus $K$ clock cycles.

In a turbo Hadamard decoder, we need to quantize three variables: The LLR values in the FHT block, the trellis probabilities in the BCJR decoder and the probability values in the APP-FHT block.

- The FHT block requires $r$ stages of modulo-2 additions. To avoid data overflow, one bit is added to the quantized value after each stage. The input of FHT is $N_{FHT}$-bit quantized and thus the output of FHT is $(N_{FHT} + r)$-bit quantized. Note

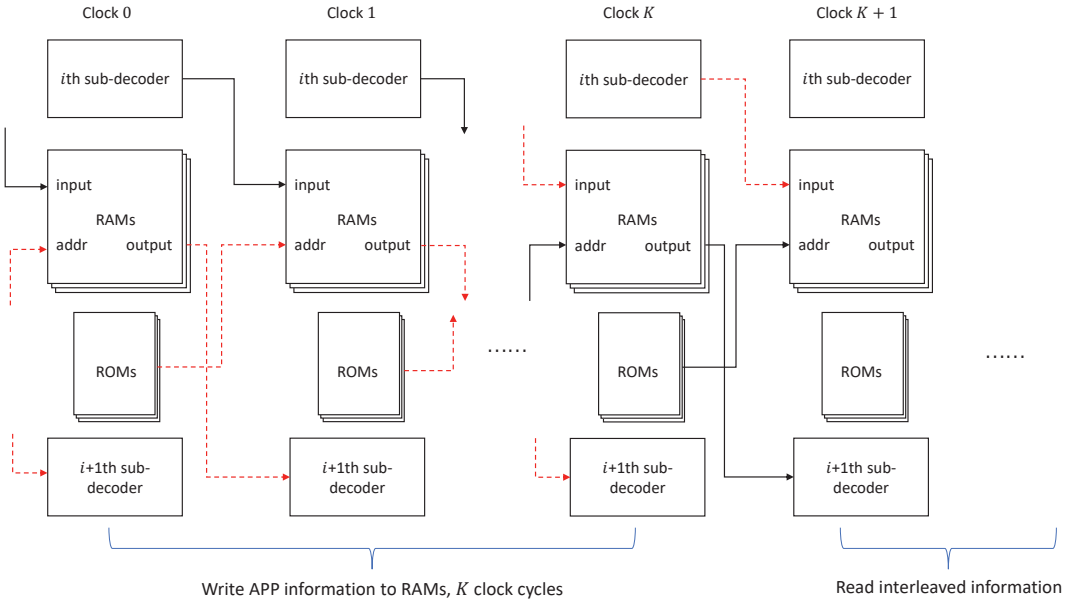


Figure 3.12: Pipeline structure between sub-decoders. Solid line represents valid input/output, dash line represents invalid input/output.

that all the quantized values have a sign bit.

- The BCJR decoder operates on probabilities, which are approximately equal to $e^{\mathrm{FHT}_{\mathrm{out}}}$. The dynamic range of those values is much higher and a lot of bits are needed to represent them. To reduce the number of quantization bits, logarithmic quantization is used. Moreover, the *a priori* value generated from the output of FHT can be transformed to the natural logarithms of trellis probabilities conveniently. After such transformations, multiplications become simple additions. However the original additions now become non-linear operations, and they are realized by look-up tables in our design. Since the values in the BCJR decoder are quantized into $N_{BCJR}$ bits, the size of the look-up table is $2^{N_{BCJR}} \times 2^{N_{BCJR}} = 2^{2N_{BCJR}}$, which is too large for a single BCJR stage even with a small $N_{BCJR}$. To reduce the hardware complexity, the Jacobian logarithm [76–78] is considered. Suppose the two inputs of a look-up table are $a, b$ and output is $c$. Then

$$c = \ln(e^a + e^b) = \max(a, b) + \ln(1 + e^{-|b-a|}). \tag{3.8}$$

Instead of a $(2N_{BCJR})$-bit input look-up table, the new table has an input width of $N_{BCJR}+1$, and the new table is only of size $2^{N_{BCJR}+1} \times 1$. Also note that the quantization of the FHT block is different from that of the BCJR block, so a look-up table is needed for the conversion of data.

- The APP-FHT block is similar to the FHT block except that the data being processed are supposed to be probabilities instead of LLRs. However, the dynamic range of the data being processed in the APP-FHT block is huge (like $10^{30}$).Moreover, the inputs of the APP-FHT block are directly connected to the output of the FHT block and the output of the BCJR block, which appear in the form of either LLRs or logarithm of trellis probabilities. To reduce the dynamic range of the data in the APP-FHT block and to reduce the complexity of the hardware, we use logarithmic quantization in the APP-FHT block. Thus the output of the previous blocks can be directly input to this block without data transformations. Similar to the BCJR block, look-up tables are used to realize the complex logarithm calculations. In the APP-FHT block, $N_{DFHT}$ bits are used to quantize the data and the sign of the data is always negative because the data represent probabilities that should always be in $[0, 1]$.

## 3.2 Hardware utilization rate

Fig. 3.13 shows the utilization of the components inside a sub-decoder. Denoting the hardware utilization rate of the FHT, BCJR, DFHT and interleaver as $U_{\mathrm{FHT}}$, $U_{\mathrm{BCJR}}$, $U_{\mathrm{DFHT}}$ and $U_{\pi}$, respectively, we have

$$\begin{aligned} U_{\mathrm{FHT}} = U_{\mathrm{DFHT}} = U_{\pi} &= \frac{K}{2K+2r} = \frac{1}{2}\frac{1}{1+\frac{r}{K}} \approx \frac{1}{2}; \\ U_{\mathrm{BCJR}} &= \frac{2K}{2K+2r} = \frac{1}{1+\frac{r}{K}} \approx 1. \end{aligned} \tag{3.9}$$

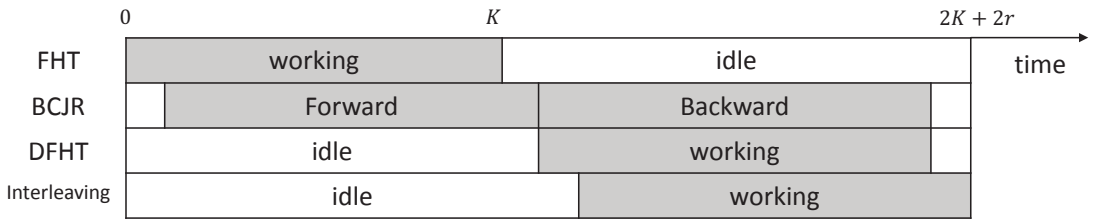


Figure 3.13: Component utilization of each sub-decoder

The BCJR unit works almost all the time during decoding while the other units operate approximately half of the time.

## 3.3 Throughput Calculation

Recall that the turbo Hadamard code has a trellis length of $K$ and the decoder described in last sub-section incurs a latency of about $2K$ clock cycles($K$ clock cycles from the BCJR decoder and another $K$ cycles from the interleaver). Since the number of component codes is $M$, the decoding latency for one iteration is therefore $2MK$. Assuming the number of iteration is $I$, we need $2IMK$ clock cycles to decode one turbo Hadamard code.

Also assuming the operating frequency of the FPGA decoder is $f_c$, the throughput $T$ of the decoder which decodes $M$ codewords in parallel is

$$T = M\frac{lf_c}{2IMK} = \frac{l}{2IK}f_c \tag{3.10}$$

where $l$ is the codeword length. The equation (3.10) shows that the throughput of the whole system is independent of the number of the component codes $M$, but is proportional to the code length and FPGA operating frequency; and is inverse proportional to the number of iterations and the length of the turbo trellis.

## 3.4 Implementation results and analysis

We consider a turbo Hadamard encoder/decoder system with the following parameters.

- Number of component codes $M = 3, 4, 5$
- Each message block $\boldsymbol{D}$ contains $L = 4095$ message bits
- Order of Hadamard code is $r = 7$
- Number of sub-blocks per message $K = \frac{L}{r} = 585$
- Number of convolutional code states $S = 2$
- Code length $l = 216450, 287235$ and $358020$
- Code rate $r_c \approx 0.0189, 0.0143$ and $0.0114$
- Channel LLRs are quantized into $N_{ch} = 6$ bits
- Inputs to FHT unit are quantized into $N_{FHT} = 10$ bits
- Data in BCJR unit are quantized into $N_{BCJR} = 7$ bits
- Data in DFHT unit are quantized into $N_{DFHT} = 10$ bits
- 10 iterations are used for decoding each codeword
- FPGA board Xilinx Virtex UltraScale+ VCU118

Based on parameters we used, the maximum operating frequency is found to be $f_c = 100$ MHz. Thus, the throughput $T \approx 1.92$ Gbps, 2.56 Gbps and 3.2 Gbps for $M = 3, 4$ and 5, respectively. Compared with computer simulations, the throughput of our design on FPGA is about $10,000$ times faster.

Table 3.1 shows the FPGA resources utilization for the turbo Hadamard encoder/decoder system with $M = 3, 4$ and 5. The look-up tables (LUT) used increase slightly from $M = 3$ to $M = 4$. For codes $M = 5$, the LUT usage increases a bit more because

| | Code Length | Look-up Table | Look-up Table RAM | Flip-Flop | Block RAM | IO | Global Clock Buffer |
|---|---|---|---|---|---|---|---|
| $M = 3$ | 217,620 | 413759 | 2741 | 146481 | 720.5 | 13 | 7 |
| $M = 4$ | 287,820 | 449648 | 3271 | 191756 | 1146.5 | 13 | 8 |
| $M = 5$ | 358,020 | 614993 | 3827 | 240610 | 1459.5 | 13 | 9 |
| System limit | N/A | 1182240 | 591840 | 2364480 | 2160 | 832 | 1800 |

Table 3.1: Resources utilization of THC with $M = 3, 4, 5$.

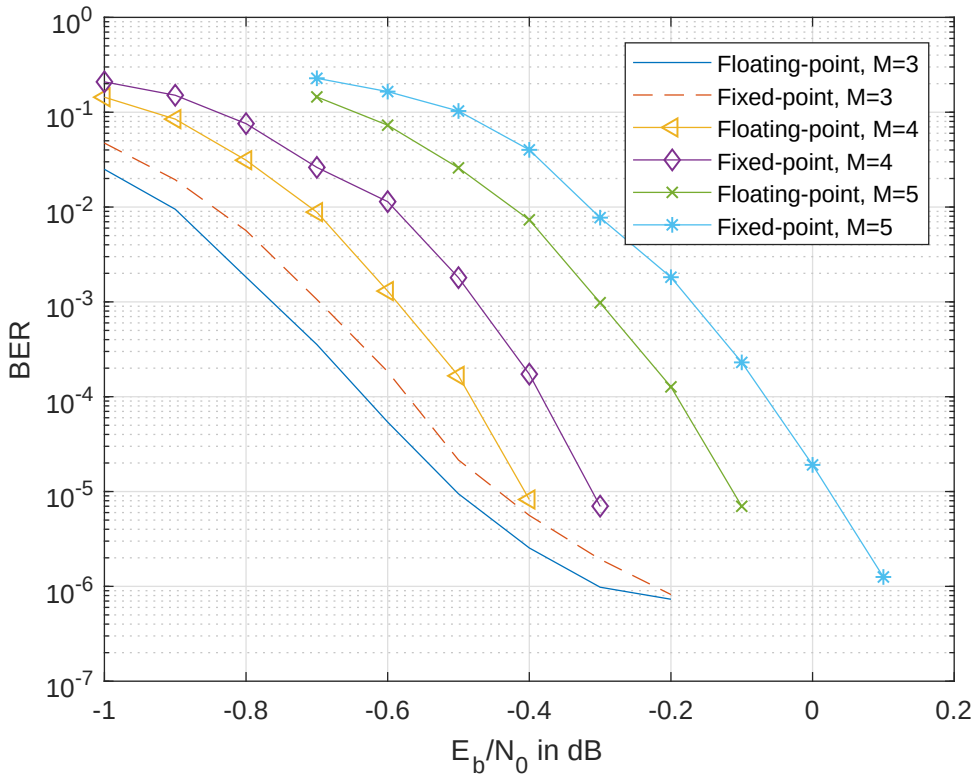


Figure 3.14: BER curves of the THC code with $M = 3, 4, 5$ component codes on computer simulations.

the design allows the transmission of punctured codes (see Chapter 5.1). The block RAMs (BRAM) used is almost proportional to the code length. Fig. 3.14 shows the bit-error-rate (BER) results of $M = 3$, $M = 4$ and $M = 5$ turbo Hadamard code under floating-point decoder and fixed-point decoder. All of the codes show a performance loss of no more than 0.15 dB at BER $= 10^{-5}$. For example, at a BER of $10^{-5}$, the fixed-point implementation of $M = 4$ THC requires an $E_b/N_0$ value of $-0.3$ dB, which is within 0.1 dB that of the floating-point result.

Fig. 3.15 plots BER curves of the THC for $M = 3, 4, 5$ implemented on the FPGA

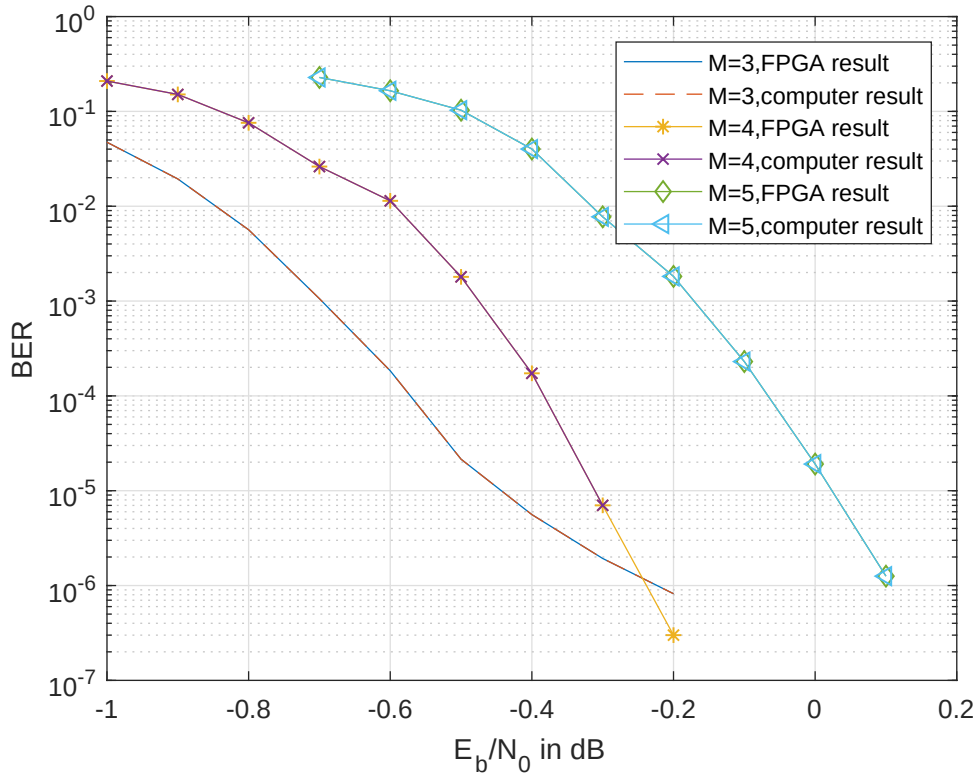


Figure 3.15: BER curves of the THC code with $M = 3, 4, 5$ component codes on FPGA compared to computer simulations.

board compared to the computer simulations. The results on two devices are identical and thus verify the correctness of the implementation. The result shows that THC with $M = 3$ achieves a better BER than $M = 4, 5$ in the low $E_b/N_0$ regime, but suffers from an error floor at a BER of $10^{-6}$.

## 3.5 Summary

In this chapter, an efficient design of an ultimate-Shannon-limit approaching encoder/decoder system has been explored using FPGA. The system implemented is based on turbo Hadamard codes with different lengths and can achieve throughputs up to 3.2 Gbps (when $M = 5$ and $r = 7$). The throughputs are about $10,000$ times faster than computer simulations. The BER results indicate that the FPGA experimental results are exactly the same as the fixed-point computer simulations, and are within 0.15 dB of

floating-point results. The operating frequency of our design is limited to 100 MHz. In the next chapter, we investigate the hardware design of another ultimate-Shannon-limit approaching code, namely concatenated zigzag Hadamard code. Since the concatenated zigzag Hadamard decoder does not require BCJR decoding, the corresponding decoder is simpler and can potentially operate at a higher frequency.

# Appendix 3: Generation of lookup tables for the channel block

Suppose the input of the channel is "+1", the code rate is $r_c$, the noise power is $\sigma^2$, and the noise is $n$. Also assuming the generated LLR falls in the interval $[a, b]$, we have

$$a \leqslant \frac{1+n}{\sigma^2} \leqslant b \tag{3.11}$$

$$a\sigma^2 - 1 \leqslant n \leqslant b\sigma^2 - 1 \tag{3.12}$$

Note that the noise $n$ follows normal distribution with zero mean and variation $\sigma^2$. Its probability density function (pdf) is given by

$$\rho(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.13}$$

and the probability of the LLR falling within the interval $[a, b]$ is

$$\Pr(a,b) = \int_{a\sigma^2-1}^{b\sigma^2-1} \rho(x)dx = 0.5\left[\operatorname{erf}\left(\frac{a\sigma^2-1}{\sigma\sqrt{2}}\right) - \operatorname{erf}\left(\frac{b\sigma^2-1}{\sigma\sqrt{2}}\right)\right] \tag{3.14}$$

where

$$\sigma = \sqrt{\frac{1}{2r_c E_b/N_0}}. \tag{3.15}$$

Based on the above results, the Gaussian noise tables are generated accordingly.

# Chapter 4

# Design of Concatenated Zigzag Hadamard Encoder/Decoder System

In the previous chapter, the hardware design of a turbo Hadamard encoder/decoder system has been investigated. Since Bahl-Cocke-Jelinek-Raviv (BCJR) decoding is required, the overall decoder design is relatively complex, limiting the operating frequency to 100 MHz. On the other hand, the concatenated zigzag Hadamard codes (CZHC) do not require BCJR decoding, potentially making the decoder simpler and operating at a higher frequency.

In this chapter, we propose a pipelined digital design of a concatenated zigzag Hadamard encoder/decoder system. Again we propose a multiple sub-decoder architecture to process multiple codes at the same time. However, different from the sub-decoders in the turbo Hadamard decoder system, each sub-decoder in the concatenated zigzag Hadamard decoder systems processes a set of multiple codes at the same time; and the same set of multiple codes is processed by different sub-decoders at different times. Such an arrangement aims to improve the throughput and also the hardware utilization rate. We overcome challenges similar to those occurring in the design of turbo Hadamard encoder/decoder systems. The final concatenated zigzag Hadamard encoder/decoder system is found to work with 50% increase in operating frequency

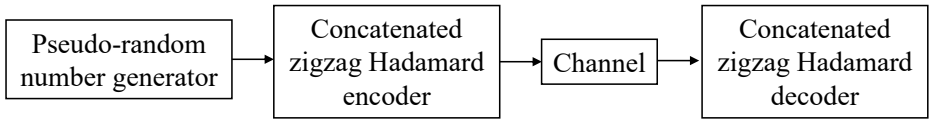


Figure 4.1: Data flow of the CZHC encoder/decoder system.

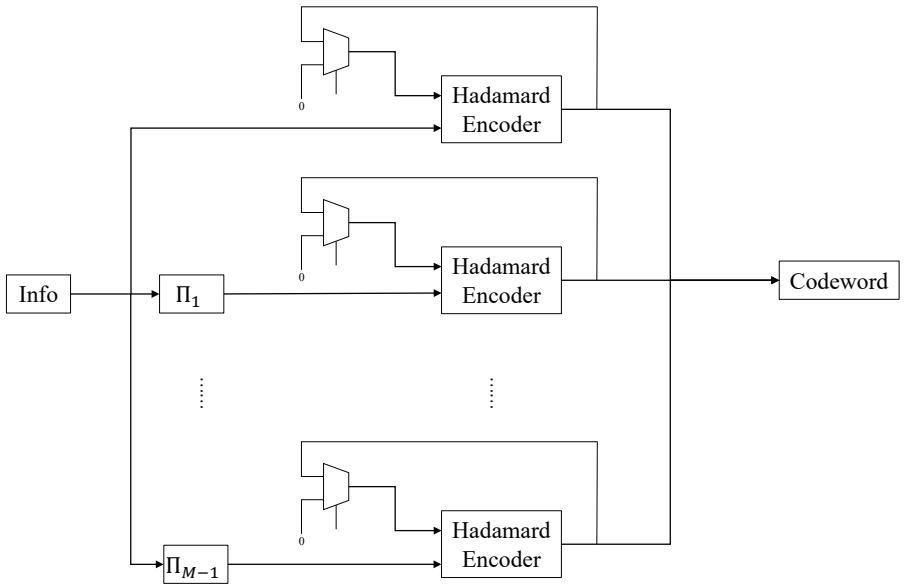


Figure 4.2: Structure of a concatenated zigzag Hadamard encoder with $M$ component codes.

compared with the turbo Hadamard encoder/decoder system and with similar error performance. The drawbacks of the concatenated zigzag Hadamard encoder/decoder system, however, are higher decoding latency and higher memory requirement.

## 4.1 Encoder Design

The data flow of our concatenated zigzag Hadamard encoder/decoder system is shown in Fig. 4.1. The structure of the CZHC encoder is shown in Fig. 4.2 where the $M$ ZHC components are encoded in parallel. To generate each CZHC codeword, the following steps are performed.

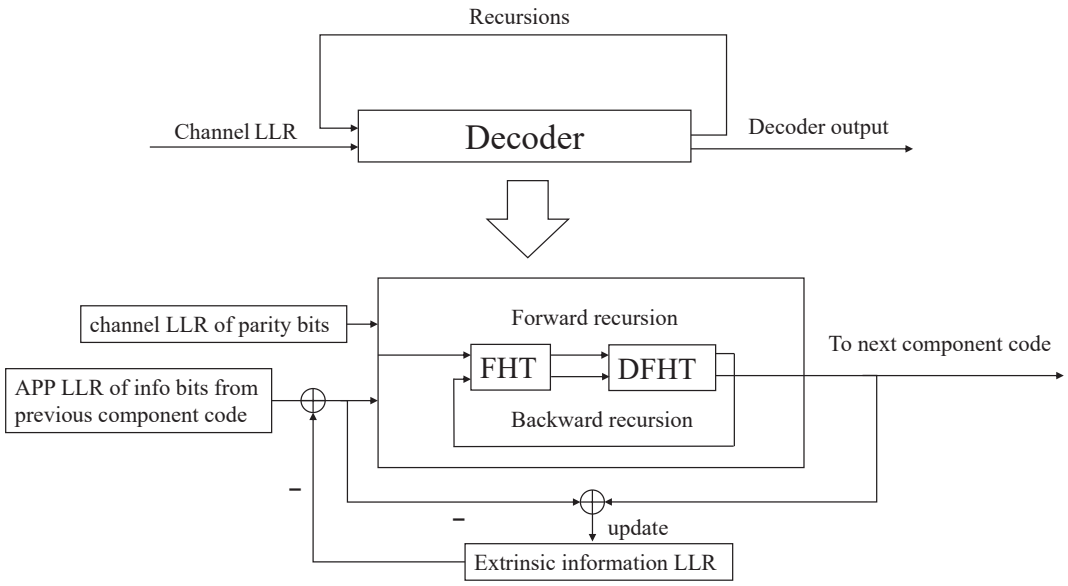


Figure 4.3: Overview of CZHC decoder.

1. Generate random information bits of length $rK$ using a pseudo random number generator (PRNG), which is realized by the use of linear feedback shift registers (LFSRs). Form the first component code using Step 2) below.

2. Divide the information bits into segments of length $r$. The $r$ information bits in each segment together with the common bit are then sent to the Hadamard encoder, producing Hadamard codewords. Note that the common bit is the feedback of the Hadamard encoder from the last segment. For the first segment, the common bit is set to 0 (see Fig. 2.10).

3. To generate the other $M-1$ component codes, send the original information bits to the corresponding interleaver denoted by $\prod_1$, $\prod_2$,..., $\prod_{M-1}$, respectively; and apply Step 2) above.

4. Send the original information bits together with all the parity-check bits generated from all component encoders to the channel.

## 4.2 Decoder Design

The structure of the decoder is illustrated in Fig. 4.3. The decoding process includes:

1. Preparation: (a) The *a priori* LLRs of the information bits, computed by subtracting the corresponding extrinsic LLRs of the information bits produced by the current decoder in the previous iteration from the APP LLRs of the information bits from the *previous* component (ZHC) decoder; and (b) channel LLRs of the parity bits of the current component ZHC code, are input to the decoder.

2. Forward recursion: The *a priori* LLRs are sent to the decoder to perform forward recursion. The forward recursion processor consists of an order-$r$ FHT block and an order-$r$ DFHT block. The *a priori* LLRs ($2^r$ for each segment of ZHC) are directly input to the FHT block where simple addition/subtraction operations are performed to produce $2^r$ outputs after $r$ stages. Exponential functions are performed to the $2^r$ outputs and their additive inverse (total $2^{r+1}$ data) before sending them to the DFHT block, which also produces $2^{r+1}$ outputs after $r$ stages. Then divisions are performed to the $2^{r+1}$ outputs to generate $2^r$ APP LLRs. Note that the exponential functions greatly increase the dynamic range of data in DFHT block and a large number of quantization bits is required to maintain the accuracy of decoding in DFHT block. To avoid the implementation of complicated exponential functions, we use logarithm quantization in the DFHT block. The benefits of the quantization include:

   - turning the exponential functions between two blocks into simple bitwise-NOT functions;
   - reducing the dynamic range of operations in DFHT block and hence the number of bits used to quantize the LLRs;
   - simplifying the decision block from division logics to subtraction logics.

An illustration of the proposed APP decoder for ZHC with order-2 is shown in Fig. 4.4.

3. Backward recursion: The backward recursion also consists of an order-$r$ FHT block and an order-$r$ DFHT block. Thus the APP decoding processors in the forward recursion can be reused. The backward recursion processor starts outputting the APP LLR continuously after $2r$ clocks delay. The outputs start from the $K$th segment and then all the way to the first segment.

4. The output data from the backward recursion processor are interleaved and passed to the next sub-decoder.

5. The extrinsic LLRs of the information bits in this iteration are generated and stored in the RAMs at the same time.

The FHT/DFHT blocks are implemented in the CZHC decoder to fast calculate the APP LLRs [79]. Each segment in both the forward and backward recursions must wait for the update from the previous (next) segment before continuing decoding. For each of the $K$ segments, the FHT and DFHT processors take a total of $2r$ clocks to complete the computations.

As shown in Fig. 4.5, only one of the $2r$ stages is working at any time. To better utilize the decoder hardware and to improve the throughput, we decode $2r$ CZHCs at the same time in our design. These $2r$ CZHCs are sent into the decoder segment-by-segment, i.e., first segment of the first code is sent to the decoder, followed by the first segment of the second code, and so on. After the first segments of all $2r$ CZHCs are sent, the second segments of the $2r$ CZHCs are sent. Note that the time when the DFHT processor finishes computing the APP LLR of the first segments of the $2r$ CZHCs, the *a priori* (AP) LLR of the second segments are arriving at the decoder. Both LLRs will then be sent to the FHT/DFHT processors to compute the forward recursion of the second segment. The operations of the FHT/DFHT processors for $2r$ CZHCs are

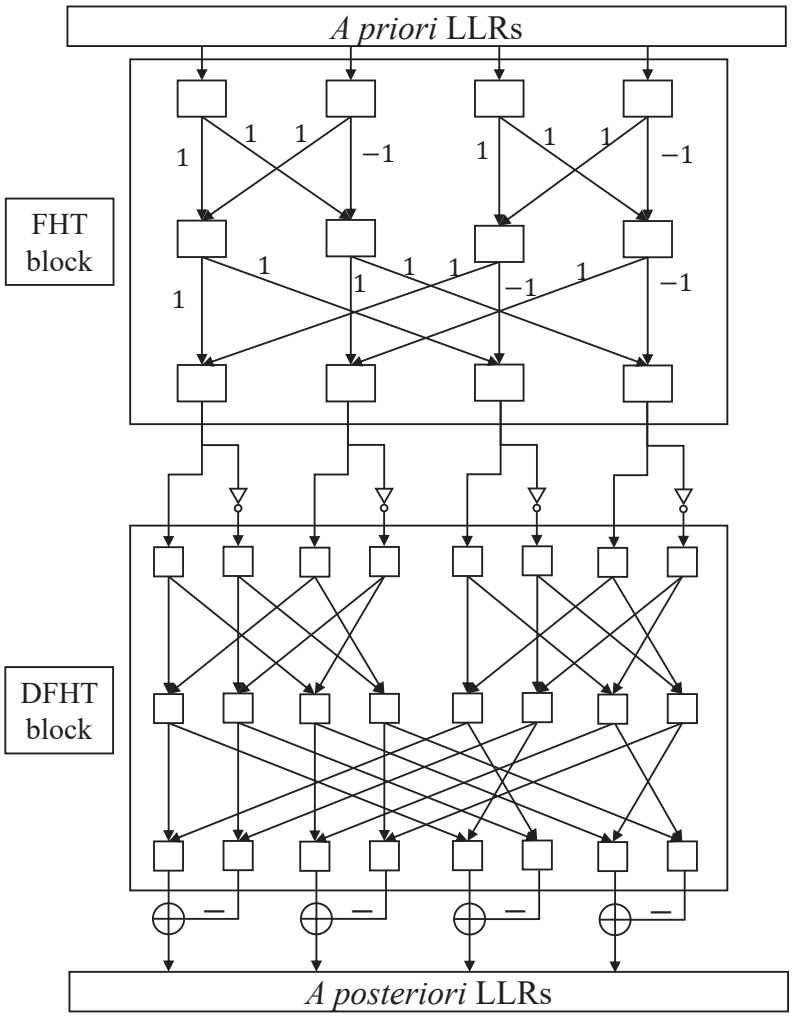


Figure 4.4: Detailed illustration of ZHC APP decoder with order-2.

illustrated in Fig. 4.6. The utilization rate of the FHT/DFHT processors are therefore greatly improved. Moreover, the latency (time between the last input code bit entering the decoder and the last decoded bit coming out of decoder) of each CZHC codeword is actually the same as that of decoding a single CZHC and the throughput of the decoder is increased by $2r$ times.

| Clock | 1 | 2 | 3 | ... | $r$ | $r+1$ | $r+2$ | ... | $2r$ | $2r+1$ | $2r+2$ | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Stage 1 of FHT | working | idle | idle | ... | idle | idle | idle | ... | idle | working | idle | ... |
| Stage 2 of FHT | idle | working | idle | ... | idle | idle | idle | ... | idle | idle | working | ... |
| Stage 3 of FHT | idle | idle | working | ... | idle | idle | idle | ... | idle | idle | idle | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| Stage $r$ of FHT | idle | idle | idle | ... | working | idle | idle | ... | idle | idle | idle | ... |
| Stage 1 of DFHT | idle | idle | idle | ... | idle | working | idle | ... | idle | idle | idle | ... |
| Stage 2 of DFHT | idle | idle | idle | ... | idle | idle | working | ... | idle | idle | idle | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| Stage $r$ of DFHT | idle | idle | idle | ... | idle | idle | idle | ... | working | idle | idle | ... |

Figure 4.5: Illustration of a forward recursion for a single CZHC code.

| Clock | 1 | 2 | 3 | ... | $r$ | $r+1$ | $r+2$ | ... | $2r$ | $2r+1$ | $2r+2$ | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Stage 1 of FHT | code 1 | code 2 | code 3 | ... | code $r$ | code $r+1$ | code $r+2$ | ... | code $2r$ | code 1 | code 2 | ... |
| Stage 2 of FHT | idle | code 1 | code 2 | ... | code $r-1$ | code $r$ | code $r+1$ | ... | code $2r-1$ | code $2r$ | code 1 | ... |
| Stage 3 of FHT | idle | idle | code 1 | ... | code $r-2$ | code $r-1$ | code $r$ | ... | code $2r-2$ | code $2r-1$ | code $2r$ | ... |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| Stage $r$ of FHT | idle | idle | idle | ... | code 1 | code 2 | code 3 | ... | code $r$ | code $r+1$ | code $r+2$ | ... |
| Stage 1 of DFHT | idle | idle | idle | ... | idle | code 1 | code 2 | ... | code $r-1$ | code $r$ | code $r+1$ | ... |
| Stage 2 of DFHT | idle | idle | idle | ... | idle | idle | code 1 | ... | code $r-2$ | code $r-1$ | code $r$ | ... |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| Stage $r$ of DFHT | idle | idle | idle | ... | idle | idle | idle | ... | code 1 | code 2 | code 3 | ... |

Figure 4.6: Forward recursion with $2r$ CZHC codes.

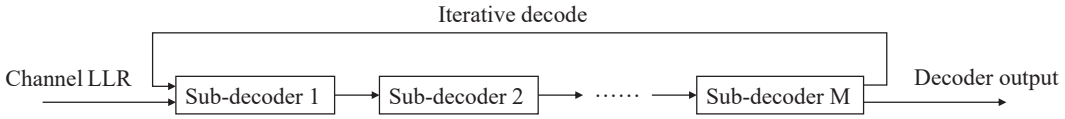


Figure 4.7: $M$ sub-decoders in one decoding system.

Note that the CZHC is a concatenated code with $M$ component codes. Each CZHC codeword needs to go through the decoding process in Fig. 4.3 $M$ times to complete one iteration. To simplify the control logic and to increase the throughput, we construct $M$ CZHC decoders (each called a sub-decoder) in our decoding system. Hence, $M$ times more CZHC codewords can be decoded simultaneously in a pipeline manner. The usage of control logic and block RAMs between consecutive component code decoders are reduced and the throughput of the decoding system is increased by another $M$ times, i.e., a total of $2rM$ times. Fig. 4.7 shows the decoding system that consists of $M$ sub-decoders. To decode $2rM$ CZHCs simultaneously, the decoder receives and stores the $2rM$ CZHCs in both the information RAMs and the parity RAMs which are shown in Fig. 4.8.

Between consecutive sub-decoders, interleavers (omitted in Fig. 4.7 for simplicity) are needed to shuffle the outputs of the current sub-decoder before inputting them to the next sub-decoder. We use fixed inter-window shuffle (FIWS) interleavers to enable parallel interleaving [72, 80, 81]. The size of the interleaver is $N = 2r \times L = 2Kr^2$ because we need to perform interleaving on the information bits of $2r$ CZHC codes at the same time. The interleaver is divided into $r$ sub-interleavers (also called windows) each with a window size of $2rK$. The windows are designed in such a way that memory contention is avoided when performing parallel interleaving. In other words, the first information bits of all $K$ segments in all $2r$ CZHCs are interleaved/deinterleaved in Window No. 1; the second information bits of all $K$ segments in all $2r$ CZHCs are interleaved/deinterleaved in Window No. 2; etc.

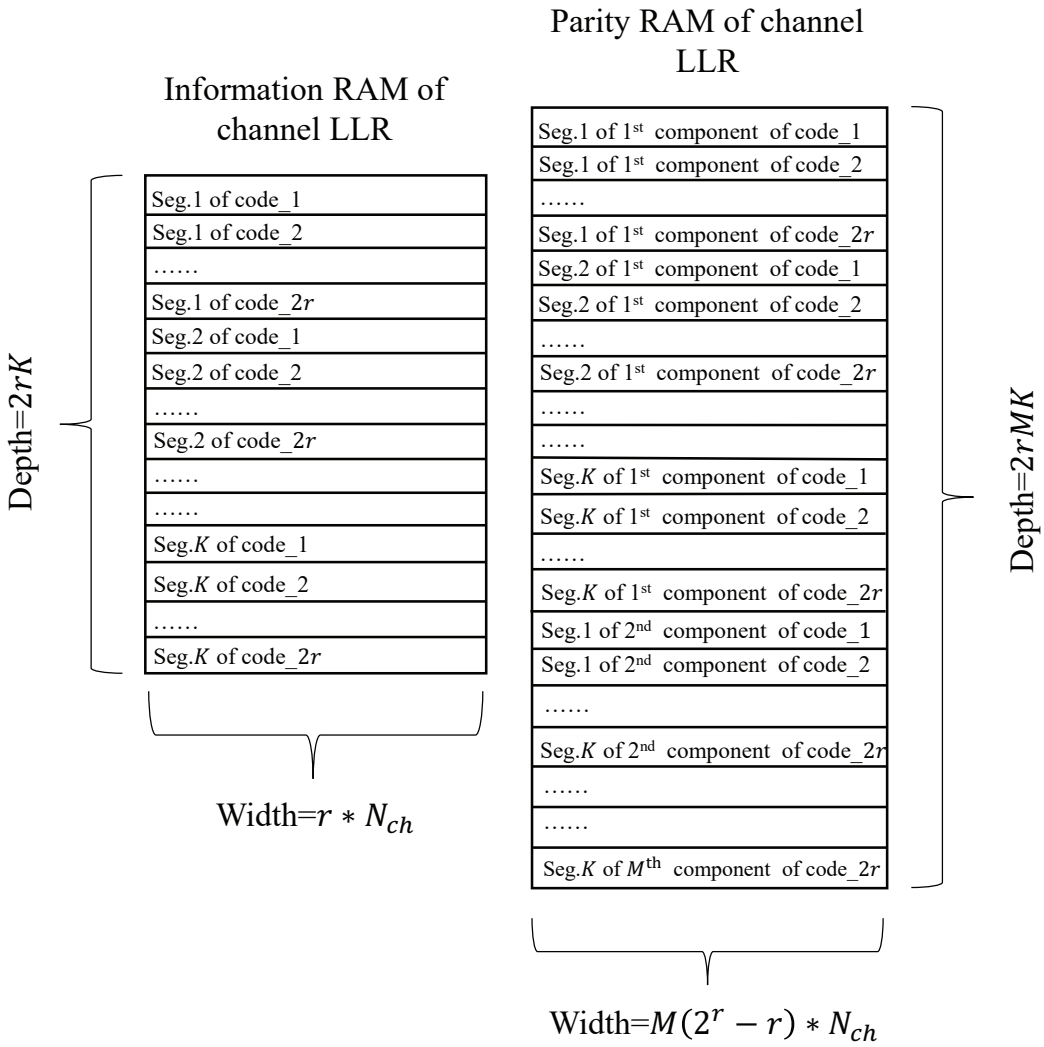


Figure 4.8: Storage order of channel LLRs.

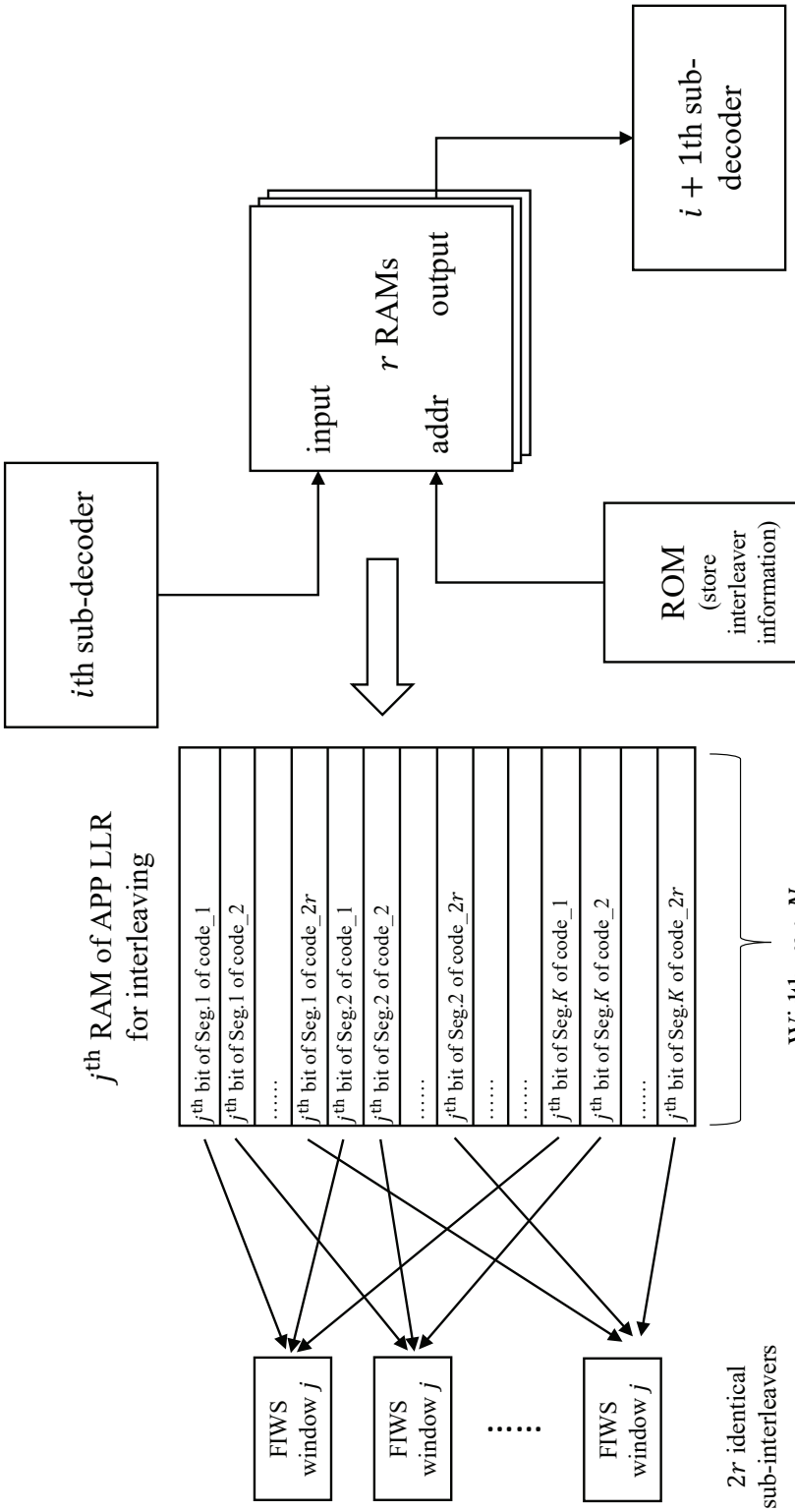


Figure 4.9: Illustration of the FIWS interleaver.

The FIWS interleaver is realized by (i) $r$ RAMs each of width-($r \times N_{FHT}$) and depth-$2rK$, and (ii) a depth-$K$ ROM. The operations are described below and illustrated in Fig. 4.9.

1. Store the output APP LLRs from the $i$th sub-decoder ($i = 1, 2, ..., M$) to the RAMs in the order that is shown in Fig. 4.9, i.e., the $j$th bit of all the $K$ segments of all the $2r$ codes are stored in the $j$th RAM ($j = 1, 2, ..., r$).

2. Read the interleaver patterns of the CZHC code from the ROM. Extract the interleaving information and evaluate the interleaver pattern for all the $2r$ CZHC codes in the $r$ RAMs correspondingly.

3. Read the interleaved APP LLRs from the $r$ different RAMs. Regroup the APP LLRs and send them to the next sub-decoder as the *a priori* LLRs.

## 4.3 Results and analysis

For an order-$r$ CZHC, the code length is $l = rK + MK(2^r - r)$ and the code rate is $r_c = \frac{rK}{rK+MK(2^r-r)} = \frac{r}{r+M(2^r-r)}$. A concatenated zigzag Hadamard encoder/decoder system with the following parameters is implemented.

- Number of component codes $M = 3$
- Each information block $\boldsymbol{D}$ contains $L = 3510$ message bits
- Hadamard order is $r = 6$ (Note that the order used here is different from that used in THC in the previous chapter. The reason is that $r$ needs to be even in order to perform systematic encoding in CZHC.)
- Number of segments per CZHC $K = \frac{L}{r} = 585$
- Number of CZHC codewords decoded in a sub-decoder $n = 2r = 12$

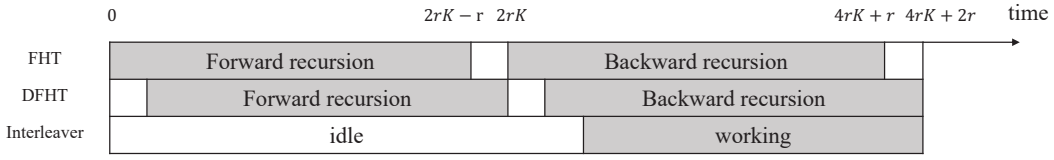


Figure 4.10: Component utilization of each sub-decoder.

- Code length of one CZHC $l = 105300$
- Code rate $r_c = 0.0333$
- Channel LLRs are quantized by $N_{ch} = 6$ bits
- Inputs to FHT unit are quantized by $N_{FHT} = 10$ bits
- Data in DFHT unit are quantized by $N_{DFHT} = 11$ bits
- $I = 10$ iterations used for decoding each codeword
- FPGA board Xilinx Virtex UltraScale+ VCU118

Based on the above parameters, the maximum operating frequency is found to be $f_c = 150$MHz. Note that a higher frequency will incur timing problem when routing.

Fig. 4.10 shows the hardware utilization of each unit inside the sub-decoder. The FHT and DFHT units are both utilized in forward and backward recursions. We denote the hardware utilization rate of the FHT, DFHT and interleaver as $U_{\text{FHT}}$, $U_{\text{DFHT}}$ and $U_\pi$, respectively. When $\frac{1}{K} \to 0$, we have

$$\begin{aligned} U_{\text{FHT}} = U_{\text{DFHT}} &= \frac{2rK - r + 2rK + r}{4rK + 2r} = \frac{1}{1 + \frac{1}{2K}} \approx 1; \\ U_\pi &= \frac{2rK}{4rK + 2r} = \frac{1}{2}\frac{1}{1 + \frac{1}{2K}} \approx \frac{1}{2}. \end{aligned} \tag{4.1}$$

The FHT/DFHT units work almost all the time during decoding while the interleavers operate approximately half of the time.

Referring to Fig. 4.10, it takes $4rK + 2r = 2r(1 + 2K)$ clocks to process one component code in each sub-decoder. Assuming the total number of iterations is $I$ and the

| Code type | FHT/DFHT | Interleaver utilization rate | BCJR unit utilization rate | Throughput | Approximate latency/sub-decoder |
|---|---|---|---|---|---|
| CZHC | 1 | 0.5 | N.A. | 1.44 Gbps | 9.76$\mu$s |
| THC | 0.5 | 0.5 | 1 | 0.96 Gbps | 1.22$\mu$s |

Table 4.1: Hardware utilization rate, throughput and latency of CZHC system and THC system.

number of component codes for each CZHC code is $M$, it takes $2rIM(1+2K)$ clocks to decode the $2r$ CZHC. Also assuming the operating frequency of the sub-decoder is $f_c$ and the CZHC code length is $l$, the sub-decoder can decode $\frac{2rlf_c}{2rIM(1+2K)}$ bits in one second. Moreover, the decoder consists of $M$ sub-decoders and can decodes $2rM$ CZHC in parallel. The throughput of the whole decoding system is approximated by

$$\begin{aligned} T &= \frac{M \times 2rlf_c}{2rIM(1+2K)} = \frac{[rK + MK(2^r - r)]f_c}{I(1+2K)} \\ &= \frac{[(1-M)r + 2^r M]f_c}{I(2+\frac{1}{K})} \approx \frac{2^{r-1}Mf_c}{I} \end{aligned} \tag{4.2}$$

where the approximation is made because $1/K \ll 1$ and $(M-1)r \ll 2^r M$.

The decoding path in the THC decoder involves going through the FHT block, the BCJR block and then the DFHT block with a latency of approximately $2K$. The decoding path of the CZHC decoder includes the FHT/DFHT block of the forward recursion and then the FHT/DFHT block of the backward recursion with a latency of approximately $4rK$. The latency of CZHC decoder is $2r$ times higher because the forward/backward recursions in CZHC decoder which cannot be performed in pipeline for one single code. However, it is possible to perform pipeline decoding of multiple codes, which is realized in our design.

In Table 4.1 and Table 4.2, we compare the FPGA implementation results of the encoder/decoder systems using CZHC and turbo Hadamard code (THC) under the same code rate and code length. Table 4.1 indicates that BCJR processor is not required in the CZHC decoder. The throughput of the CZHC system is 50% higher than that of the THC system. Table 4.2 shows that with the same code length, code rate and

the same number of channel quantization bits, the look-up tables (LUT), look-up table RAMs and flip-flops used in the CZHC system are, respectively, 73%, 41% and 85% of those in the THC system. The block RAM usage in CZHC system, however, is higher than that in the THC system. For the CZHC system with 4 bits channel LLRs quantization, the block RAMs usage is about $\frac{2}{3}$ of that with 6 bits quantization system. The other resources utilized are slightly less than the 6 bits quantization system.

| Code type | Code rate | Code length | channel quantization | Look-up Table count (normalized) | Look-up Table RAM count (normalized) | Flip-Flop count (normalized) | Block RAMs count (normalized) | Max. Operating frequency |
|---|---|---|---|---|---|---|---|---|
| THC | 0.0333 | 105300 | 6 bits | 1 | 1 | 1 | 1 | 100 MHz |
| CZHC | 0.0333 | 105300 | 6 bits | 0.737 | 0.417 | 0.856 | 5.180 | 150 MHz |
| CZHC | 0.0333 | 105300 | 4 bits | 0.683 | 0.408 | 0.805 | 3.788 | 150 MHz |

Table 4.2: Resources utilization of CZHC system compared to THC system. 6-bit quantized channel LLRs are used.

Fig. 4.11 shows the bit-error-rate (BER) results of CZHC and THC. Compared with floating-point CZHC decoder, fixed-point decoder shows a performance loss of about 0.1 dB at BER= $2 \times 10^{-5}$ when 5-bit or 6-bit quantized channel LLRs are used. For even smaller number of quantization bits, the BER performance is further degraded. Fig. 4.11 also shows that the BER performances of CZHC and THC are very close. Both codes can achieve BER= $1.5 \times 10^{-5}$ at $E_b/N_0 = -0.2$ dB.

Fig. 4.12 shows the fixed-point BER results under different number of decoding iterations. We observe that THC outperforms CZHC in low $E_b/N_0$ region, and performs similar in high $E_b/N_0$ region. Both codes show an error floor at a BER of $10^{-5}$. Fig. 4.13 shows the floating results over both AWGN channel and Rayleigh fading channel [82] for both THC and CZHC. A maximum iteration of 50 is performed. Again, the results show that the performance of THC and CZHC are very close, over both AWGN channel and Rayleigh fading channel.

## 4.4 Summary

In this chapter, we have designed and implemented a hardware structure for a concatenated zigzag Hadamard encoder/decoder system using FPGA. The system can achieve a throughput of 1.44 Gbps at a code rate of 0.0333 and BER= $1.5 \times 10^{-5}$ at $E_b/N_0 = -0.2$ dB. Compared to the turbo Hadamard system, the CZHC system achieves 1.5 times larger throughput with less complex hardware architecture but more block RAM usage. Another drawback of the CZHC system is a higher decoding latency. The BER performance of the CZHC is also outperformed by THC. In the next chapter, we continue our investigation on THC. In particular, we study punctured THC and also propose a way to lower the error floor of THC.

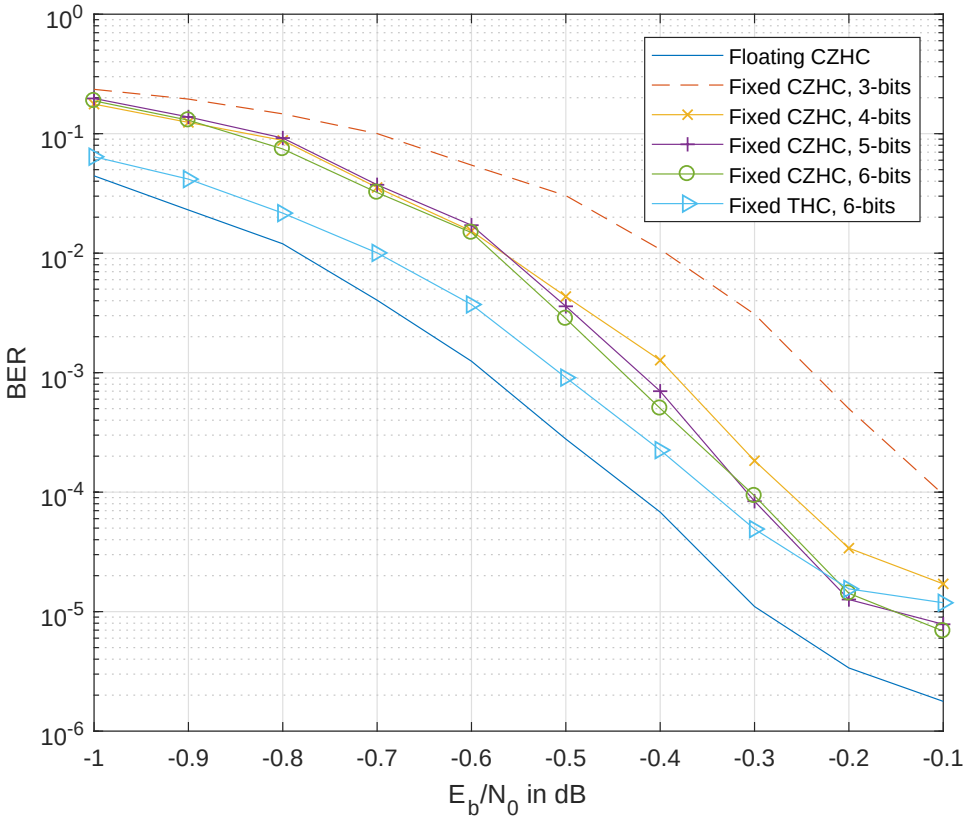


Figure 4.11: BER results of CZHC and THC under different quantization bits.

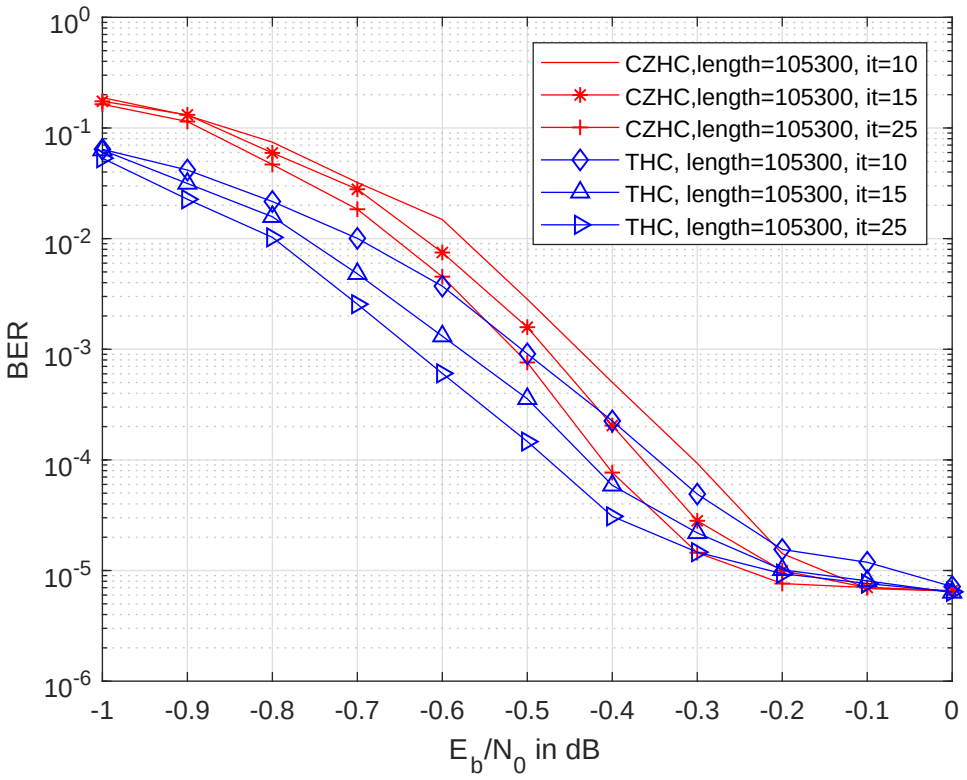


Figure 4.12: BER results of CZHC and THC under different number of iterations.

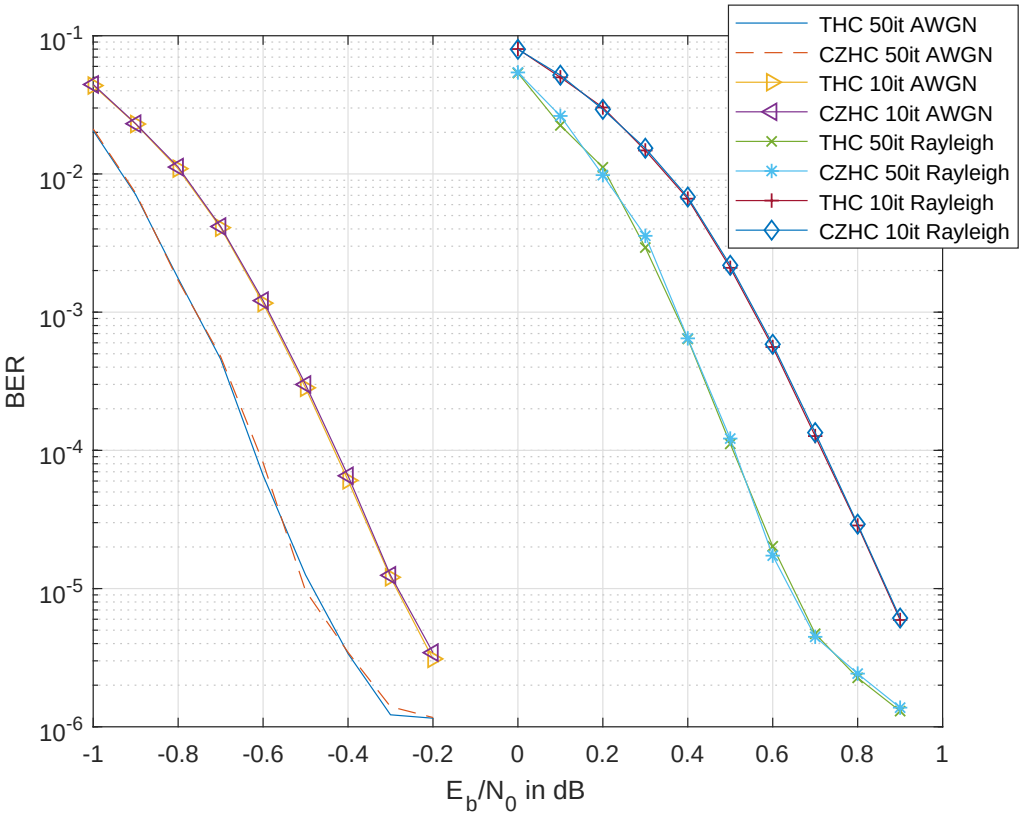


Figure 4.13: BER results of CZHC and THC over different channels.

# Chapter 5

# Optimization of Turbo Hadamard Codes

In the previous chapter, we have shown that a concatenated zigzag Hadamard encoder/decoder system can achieve a higher operating frequency and a higher throughput compared with a turbo Hadamard encoder/decoder system. The main drawbacks of the concatenated zigzag Hadamard encoder/decoder system are that it requires more memory storage, has a higher decoding latency, and a worse error performance. In this chapter, we continue our investigation on turbo Hadamard codes (THCs).

We propose ways to optimize the turbo Hadamard codes. Punctured Hadamard codes and punctured turbo Hadamard encoder/decoder systems are first investigated. By puncturing some of the code bits and not sending those bits through the channel, the rate of a code is improved and sometimes the bit error rate performance can be improved too. Here, two methods to select the punctured Hadamard code bits, or equivalently the puncturing patterns, are proposed. The first scheme aims to maximize the minimum Hamming distance of the punctured Hadamard codes. The upperbound of the minimum Hamming distance of punctured Hadamard codes is proved and then an algorithm is proposed to find punctured Hadamard codes achieving close to this bound. The second scheme aims to minimize the cross-correlations among

the punctured Hadamard codes. For punctured code sets having the same minimum cross-correlation, a new metric has been proposed to identify sets that can further enhance the reliability of decoding. For Hadamard codes, the two proposed puncturing schemes have shown error improvements over the use of random puncturing. Moreover, the scheme that minimizing the cross-correlations outperform that maximizing the minimum Hamming distance. By applying the more superior scheme to puncture Hadamard codes in a turbo Hadamard encoder/decoder system, the code rate is improved with little change in error performance. Another way to optimize the turbo Hadamard codes is to lower its error floor. At the high $E_b/N_0$ region, we observe that the error rate of the turbo Hadamard code may become flat. To tackle this issue, we investigate the overall turbo Hadamard code structure by looking into its parity-check matrix. By re-designing the interleavers and hence removing short cycles within the code, we observe that the error floor can be lowered.

## 5.1 Punctured Hadamard Codes/Turbo Hadamard Codes

Under the same signal-to-noise condition, punctured codes do not perform as well as the un-punctured ones due to the degradations of the minimum Hamming distance and cross-correlation functions. However, the degradation can be reduced by carefully designing the puncturing patterns. With appropriately designs, punctured Hadamard codes are expected to perform very closely to the un-punctured ones under the same bit-energy-to-noise-power-spectral-density ratio ($E_b/N_0$) in terms of BER. After finding good punctured Hadamard codes, we can apply them to puncture turbo Hadamard codes.

### 5.1.1 Punctured Hadamard Codes

Recall that an order-$r$ Hadamard matrix $\boldsymbol{H}_n$, where $n = 2^r$, can be constructed recursively using

$$\boldsymbol{H}_n = \begin{bmatrix} +\boldsymbol{H}_{n/2} & +\boldsymbol{H}_{n/2} \\ +\boldsymbol{H}_{n/2} & -\boldsymbol{H}_{n/2} \end{bmatrix} \tag{5.1}$$

with $\boldsymbol{H}_1 = [+1]$. For example, a Hadamard matrix of order $r = 3$ is given by

$$\boldsymbol{H}_8 = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{bmatrix}. \tag{5.2}$$

and $-\boldsymbol{H}_8$ is given by

$$-\boldsymbol{H}_8 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 \\ -1 & -1 & +1 & +1 & -1 & -1 & +1 & +1 \\ -1 & +1 & +1 & -1 & -1 & +1 & +1 & -1 \\ -1 & -1 & -1 & -1 & +1 & +1 & +1 & +1 \\ -1 & +1 & -1 & +1 & +1 & -1 & +1 & -1 \\ -1 & -1 & +1 & +1 & +1 & +1 & -1 & -1 \\ -1 & +1 & +1 & -1 & +1 & -1 & -1 & +1 \end{bmatrix}. \tag{5.3}$$

Moreover, each column in the matrix $\pm\boldsymbol{H}_n$ represents a Hadamard codeword.

The Hamming distance between an order-$r$ Hadamard codeword $\boldsymbol{c}$ and another

order-$r$ codeword $\boldsymbol{c}^{'}$ (i.e., number of locations with different entries), denoted as $d(\boldsymbol{c}, \boldsymbol{c}^{'})$, is given by

$$d(\boldsymbol{c}, \boldsymbol{c}^{'}) = \begin{cases} 0 & \boldsymbol{c}^{'} = \boldsymbol{c} \\ 2^{r} & \boldsymbol{c}^{'} = -\boldsymbol{c} \\ 2^{r-1} & \boldsymbol{c}^{'} \neq \boldsymbol{c} \end{cases}. \tag{5.4}$$

From (5.4), it can be easily seen that the minimum Hamming distance of an order-$r$ Hadamard code is $2^{r-1}$.

Denoting $\langle \boldsymbol{c}, \boldsymbol{c}^{'} \rangle$ as the inner product between $\boldsymbol{c}$ and $\boldsymbol{c}^{'}$, and $\psi(\boldsymbol{c}, \boldsymbol{c}^{'})$ as the correlation between $\boldsymbol{c}$ and $\boldsymbol{c}^{'}$; we have

$$\psi(\boldsymbol{c}, \boldsymbol{c}^{'}) = \frac{\langle \boldsymbol{c}, \boldsymbol{c}^{'} \rangle}{2^{r}} = \begin{cases} 1 & \boldsymbol{c}^{'} = \boldsymbol{c} \\ -1 & \boldsymbol{c}^{'} = -\boldsymbol{c} \\ 0 & \boldsymbol{c}^{'} \neq \boldsymbol{c} \end{cases}. \tag{5.5}$$

Note that the auto-correlation of each codeword is always 1 and the cross-correlation between codewords is either $-1$ or 0. The results are obvious because the codewords are either orthogonal or bi-orthogonal. Recall in (2.23) that the log-likelihood-ratio (LLR) of the $i$th bit in the Hadamard code is given by $L[i] = \ln \frac{\sum_{c[i]=+1} \exp\left(\frac{\langle \boldsymbol{c}, \boldsymbol{x} \rangle}{\sigma^2}\right)}{\sum_{c[i]=-1} \exp\left(\frac{\langle \boldsymbol{c}, \boldsymbol{x} \rangle}{\sigma^2}\right)}$. Since $\langle \boldsymbol{c}, \boldsymbol{x} \rangle$ is proportional to the correlation between the codeword $\boldsymbol{c}$ and the received observation $\boldsymbol{x}$, (2.23) shows that the correlations between the received observation $\boldsymbol{x}$ and all Hadamard codewords are involved in determining the value of the $i$-th bit. If the noise variance $\sigma$ is very small, the received observation $\boldsymbol{x}$ becomes very close to the transmitted codeword, say $\boldsymbol{c}^{*}$. According to (5.5) and (2.23), the correct codeword contributes $\exp\left(\frac{\langle \boldsymbol{c}^{*}, \boldsymbol{x} \rangle}{\sigma^2}\right) \approx \exp\left(\frac{2^r}{\sigma^2}\right) \to \infty$ to one of the two summations (e.g. the one in numerator) while the inverse codeword $-\boldsymbol{c}^{*}$ contributes $\exp(-\frac{2^r}{\sigma^2}) \to 0$ to the other summation (e.g. the one in denominator). The other uncorrelated codewords $\boldsymbol{c}$ ($\boldsymbol{c} \neq \pm \boldsymbol{c}^{*}$) only contribute $\exp(\frac{\langle \boldsymbol{c}, \boldsymbol{x} \rangle}{\sigma^2}) = \exp(\frac{0}{\sigma^2}) = 1$ to both summations. In this case,

the final LLR value can be used to estimate the transmitted bit correctly with a very high probability.

Suppose we puncture $p$ bits of an order-$r$ Hadamard code and denote the punctured codewords by $\boldsymbol{c}_p$. The equations (5.4) and (5.5) no longer hold and are modified to

$$d(\boldsymbol{c}_p, \boldsymbol{c}'_p) = \begin{cases} 0 & \boldsymbol{c}' = \boldsymbol{c} \\ 2^r - p & \boldsymbol{c}' = -\boldsymbol{c} \\ \in \{1, 2, \ldots, 2^{r-1}\} & \boldsymbol{c}' \neq \boldsymbol{c} \end{cases} \tag{5.6}$$

and

$$\psi_p(\boldsymbol{c}_p, \boldsymbol{c}'_p) = \frac{\langle \boldsymbol{c}_p, \boldsymbol{c}'_p \rangle}{2^r - p} \begin{cases} 1 & \boldsymbol{c}'_p = \boldsymbol{c}_p \\ -1 & \boldsymbol{c}'_p = -\boldsymbol{c}_p \\ \in (-1, 1) & \boldsymbol{c}'_p \neq \pm \boldsymbol{c}_p \end{cases}, \tag{5.7}$$

respectively. Both the minimum Hamming distance (MHD) and correlation of punctured Hadamard codes have a great impact on the performance of the codes [83]. Depending on the way how the codes are punctured, the cross-correlation values $\psi_p$ (when $\boldsymbol{c}'_p \neq \pm \boldsymbol{c}_p$) vary and so do their effect ($\exp(\frac{\langle \boldsymbol{c}_p, \boldsymbol{x}_p \rangle}{\sigma^2})$) on the bit decision expression in (2.22). To minimize the effect of such undesirable cross-correlations, the punctured Hadamard codes should be designed with care. In the next two sections, two algorithms are proposed for designing the puncturing patterns of the original Hadamard codes. Note that for a Hadamard code of order-$r$, the code rate after puncturing $p$ bits, denoted as $r'_{HC}$ is given by

$$r'_{HC} = \frac{r+1}{2^r - p}. \tag{5.8}$$

### 5.1.2 Maximizing minimum Hamming distance

The minimum Hamming distance (MHD) is commonly used as a metric to compare the error-correction capability of linear codes. A code with a larger MHD normally

provides a better BER than another with a smaller MHD. For punctured Hadamard codes, we provide its theoretical upper bound in Theorem 1.

**Theorem 1.** *Supposing we puncture $p$ bits from an order-$r$ Hadamard code, the MHD of the punctured Hadamard code is upper-bounded by $d \leq \left\lceil \frac{2^r-p}{2} \right\rceil$ where $\lceil y \rceil$ denotes the smallest integer larger than or equal to $y$.*

*Proof.* For simplicity, we replace "$-1$" with "0" in both the Hadamard codes and Hadamard matrices. Assume that the MHD after puncturing equals $d_p > \left\lceil \frac{2^r-p}{2} \right\rceil$. Denote the weight (i.e., number of 1's) of the $i$-th punctured codeword by $w_i$ ($i = 0, 1, 2, 3, \ldots, 2^{r+1} - 1$). Let $w_0$ be the weight of the all-zero codeword and $w_1$ be the weight of the all-one codeword. Then $w_0 = 0$ and $w_1 = 2^r - p$. Since the MHD of each codeword is assumed to be larger than $\left\lceil \frac{2^r-p}{2} \right\rceil$, we have $w_i > \left\lceil \frac{2^r-p}{2} \right\rceil$ ($i = 2, 3, 4, \ldots, 2^{r+1} - 1$). Therefore, the sum of the weights of all the codewords is given by

$$\begin{aligned} \sum_{i=0}^{2^{r+1}-1} w_i &= w_0 + w_1 + \sum_{i=2}^{2^{r+1}-1} w_i \\ &> 2^r - p + \left\lceil \frac{2^r - p}{2} \right\rceil (2^{r+1} - 2) \\ &\geqslant 2^r(2^r - p). \end{aligned} \tag{5.9}$$

On the other hand, the original Hadamard codes have a total weight of $2^r \times 2^r \times 2/2 = 2^{2r}$ before puncturing (equivalent to the number of 1's in $\pm \boldsymbol{H}_n$). Puncturing $p$ bits per codeword is equivalent to puncturing $p$ rows in the Hadamard matrices $\pm \boldsymbol{H}_n$. Subsequently, the number of 1's in the matrices and hence the total weight of the punctured codewords is reduced by $2^r p$. Thus, the total weight after puncturing equals $2^{2r} - 2^r p = 2^r(2^r - p)$ which is contradictory to (5.9). That completes the proof. □

Having derived the upper bound of the MHD, we propose the following method to find good punctured Hadamard codes. Again, we replace "$-1$" with "0" in both the Hadamard codes and Hadamard matrices.

1. Randomly generate 1,000,000 puncturing patterns, select the puncturing patterns with the largest minimum weight.

2. If there are more than one such pattern, select the patterns with the least number of minimum weights.

3. Run BER simulations of selected punctured codes, find the code with the best performance.

### 5.1.3 Minimizing correlation

From (2.23) and (5.7), we know that the cross-correlations (or inner product) of the punctured codewords should be minimized. For an order-$r$ Hadamard code with $p$ bits punctured, we calculate all $\binom{2^r}{2}$ cross-correlations. This step is repeated for different punctured codes. Then we look into the correlation values for each punctured code. We first determine the maximum cross-correlation (MaxCC) value in each punctured code set and only keep code sets with the minimum MaxCC (MinMaxCC) values. The reason is that the MinMaxCC values affect APP decoding most, as observed in (2.23). After this step, we will have a number of punctured code sets having the same MinMaxCC. However, the error performances of these punctured codes can be quite different.

Next, we consider the $i$-th bit of a candidate punctured Hadamard code set found above. We let $M(i)$ be the number of codeword pairs having the MaxCC within the code set and with the $i$-th bit being distinct. Such codeword pairs are harmful to the APP decoding and should be minimized. We also let $N(i)$ be the number of codeword pairs having the MaxCC within the code set and with the $i$th bit being identical. Such codeword pairs can enhance the reliability of decoding. For example, for a punctured Hadamard codes with $r = 5$ and $p = 10$, let MaxCC=10. (Note that from now on, we multiply the correlation value by the code length $2^r - p$ such that MaxCC becomes an integer.) We record $M(i), N(i)$ and the error rate of all information bits at $E_b/N_0 = 6$ dB

in Table 5.1. Our simulation uses APP decoding defined in (2.23), i.e.,

$$\begin{aligned} L[i] =& \ln \frac{\sum\limits_{c_p[i]=+1} \exp(-\frac{\|\boldsymbol{c}_p-\boldsymbol{x}_p\|^2}{2\sigma^2})}{\sum\limits_{c_p[i]=-1} \exp(-\frac{\|\boldsymbol{c}_p-\boldsymbol{x}_p\|^2}{2\sigma^2})} \\ =& \ln \frac{\sum\limits_{c_p[i]=+1} \exp(\frac{\langle \boldsymbol{c}_p,\boldsymbol{x}_p\rangle}{\sigma^2})}{\sum\limits_{c_p[i]=-1} \exp(\frac{\langle \boldsymbol{c}_p,\boldsymbol{x}_p\rangle}{\sigma^2})} \end{aligned} \tag{5.10}$$

where the subscript $p$ indicates that $p$ bits have been punctured from the codeword. The simulation stops after $10,000$ frame errors have occurred. The results in Table 5.1 show that the error rate of information bits with $\frac{M(i)}{N(i)} = 2$ are almost 10 times worse than unpunctured codes. For information bits with $\frac{M(i)}{N(i)} = 0.5$, the error rate only shows a slight degradation. Our second criterion when selecting punctured Hadamard code is therefore to minimize

$$\Omega = \sum_{i\in\{0,1,2,4,\ldots,2^{r-1}\}} \exp\left(\frac{M(i)}{N(i)}\right). \tag{5.11}$$

Note that we use an exponential function to enlarge the effect of $\frac{M(i)}{N(i)}$ because the error rate rises rapidly when $\frac{M(i)}{N(i)}$ becomes large.

Based on our findings, an algorithm to search for good punctured Hadamard codes is proposed and described below.

1. Arbitrarily puncture $p$ bits among the parity-check bits of the Hadamard code.

2. Measure the cross-correlation of all codewords pairs. The total number of cross-correlations is $\binom{2^r}{2}$. Find the MaxCC for the punctured code.

3. Repeat the above two steps a sufficient number of times, e.g. 100 times, until codes with small MaxCC are found.

4. Select the punctured codes with the minimum MaxCC (MinMaxCC) values.

5. For each selected code set, calculate the metric $\Omega$ given by (5.11).

Table 5.1: Bit error rate of the information bits $i$ before and after puncturing. Order of Hadamard code $r = 5$. Number of punctured bits $p = 10$. $E_b/N_0 = 6$ dB.

| $i$ | $M(i)$ | $N(i)$ | $\frac{M(i)}{N(i)}$ | Error rate | error rate of unpunctured codes |
|---|---|---|---|---|---|
| 0 | 32 | 16 | 2 | $1.67 \times 10^{-4}$ | $1.45 \times 10^{-5}$ |
| 1 | 16 | 32 | 0.5 | $2.59 \times 10^{-5}$ | $1.52 \times 10^{-5}$ |
| 2 | 32 | 16 | 2 | $1.67 \times 10^{-4}$ | $1.46 \times 10^{-5}$ |
| 4 | 32 | 16 | 2 | $1.71 \times 10^{-4}$ | $1.42 \times 10^{-5}$ |
| 8 | 16 | 32 | 0.5 | $2.61 \times 10^{-5}$ | $1.45 \times 10^{-5}$ |
| 16 | 16 | 32 | 0.5 | $2.49 \times 10^{-5}$ | $1.49 \times 10^{-5}$ |

6. Pick the punctured code set with the smallest $\Omega$.

### 5.1.4 Simulation results

We use an order-5 Hadamard code, and attempt to find good punctured codes by (i) maximizing minimum Hamming distance and (ii) minimizing correlation, separately. A total of $p = 15$ bits are punctured. Our results show that the MaxCC values for different punctured codes vary from 5 to 13. For punctured codes with MaxCC= 5, which is the MinMaxCC, we further evaluate the metric $\Omega$ given by (5.11).

We simulate the codes in an AWGN channel and we use APP decoding defined by (2.23) to decode the code bits. In Figure 5.1, we plot the BER performance of the following punctured Hadamard codes.

- Unpunctured code (shown as "Unpunctured code" in figure)
- Randomly punctured (shown as "Random puncture" in figure)
- Punctured while maximizing minimum Hamming distance ("MHD optimized")
- Punctured code with MaxCC = 13 ("MCC=13")
- Punctured code with MaxCC = 5, $\Omega = 26.46$ ("MCC=5, $\Omega = 26.46$")

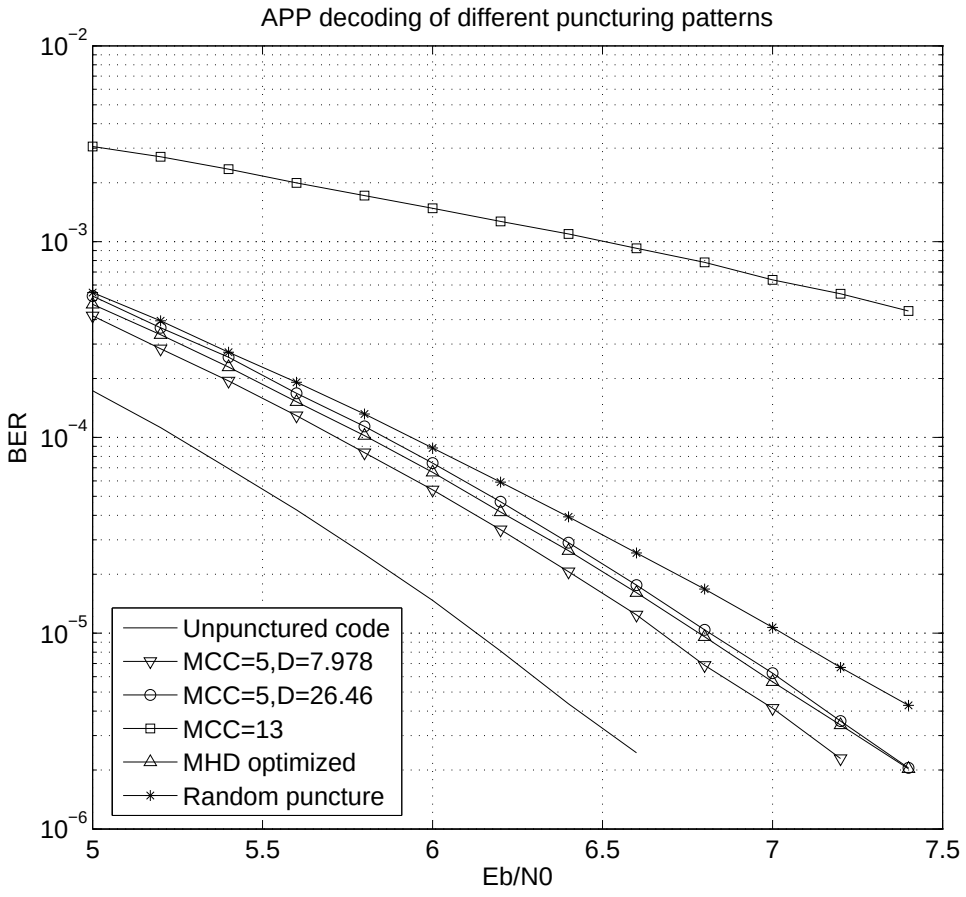


Figure 5.1: BER performance of different punctured Hadamard codes when APP decoding is used.

- Punctured code with MaxCC = 5, $\Omega$ = 7.978 ("MCC=5, $\Omega$ = 7.978")

It can be observed that the punctured Hadamard code with MinMaxCC and smallest $\Omega$ (MaxCC = 5, $\Omega$ = 7.978) gives the lowest error rate. At a BER of $10^{-5}$, it (i) outperforms the randomly punctured code by 0.35 dB, (ii) outperforms the one with the same MinMaxCC but a larger $\Omega$ (MaxCC = 5, $\Omega$ = 24.64) by 0.15 dB, and (iii) outperforms significantly the code with a large MaxCC (MaxCC = 13). The MHD optimized punctured Hadamard code slightly outperforms the one with the MaxCC =5 and $\Omega$ = 26.46. Finally, the punctured code with a large MaxCC performs the worst and should never be used.

We also perform hard decoding for the MHD optimized punctured Hadamard codes. In Figure 5.2, we plot the BER performance of unpunctured code, MHD optimized

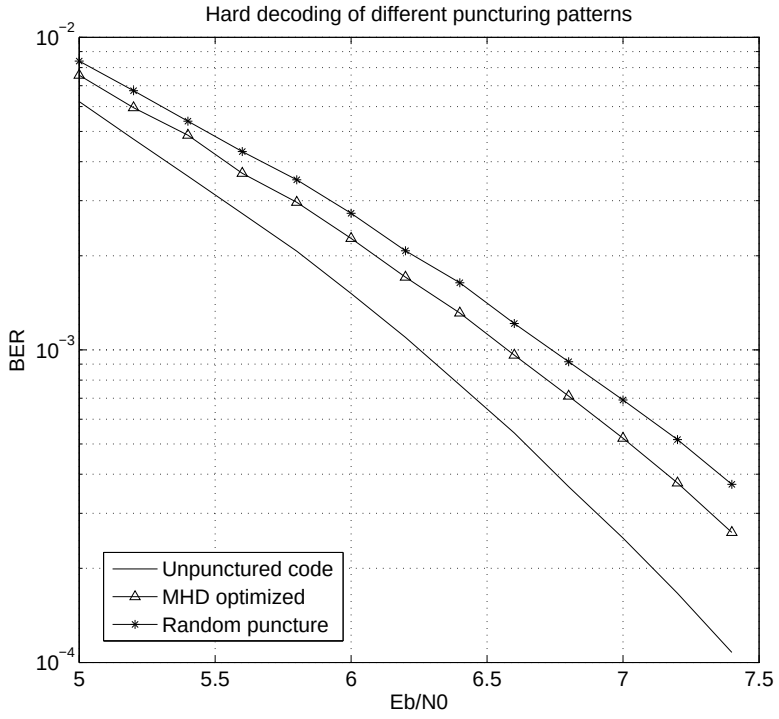


Figure 5.2: BER performance of different punctured Hadamard codes when hard decoding is used.

code and random punctured code. The results also show that the MHD optimized code outperforms the random punctured code in hard decoding.

### 5.1.5 Punctured Turbo Hadamard Codes

We further consider puncturing THC. Since only parity bits can be punctured, the puncturing of turbo Hadamard code is basically the puncturing of Hadamard code. Again the puncturing patterns can be designed by (i) maximizing the minimum Hamming distance or (ii) minimizing the correlations of all codewords. We use the latter method as it has shown better performance. We optimize the puncturing patterns for an $M = 5$ ($l = 358020$) turbo Hadamard code and generate codes with $l = 287820$ and 217620, respectively. The puncturing parameters are shown in Table 5.2 and the puncturing patterns are listed in Table 5.3. The punctured THC with $M = 5$ is also implemented on FPGA board (see Table 3.1). The BER results in Fig. 5.3 shows that the THCs af-

| Index | Code Rate $r_c$ | Message Length $N$ | Code Length | Punctured bits |
|---|---|---|---|---|
| 0 | 0.0114 | 4,095 | 358,020 | 0 bit |
| 1 | 0.0142 | 4,095 | 287,820 | 24 bits |
| 2 | 0.0188 | 4,095 | 217,620 | 48 bits |

Table 5.2: Number of punctured bits per Hadamard code.

| Index | Code Rate $r_c$ | Puncturing patterns (Numbers range from 0 to 127) |
|---|---|---|
| 0 | 0.0114 | NA |
| 1 | 0.0142 | 19 32 33 40 42 43 47 48<br>51 62 65 68 70 77 79 84<br>89 95 97 99 117 118 122 125 |
| 2 | 0.0188 | 5 8 9 10 12 13 14 19 20 22 23 25 32<br>37 40 44 45 46 47 49 51 53 54 56 58 61<br>62 66 68 75 78 82 84 86 88 90 93 94 104<br>107 111 114 118 120 122 123 124 125 |

Table 5.3: Punctured bits in a Hadamard code.

ter puncturing have very close performance compared with the original THC. In other words, the code rate of THC can be improved by puncturing and no error performance degradation is observed. Note that the channel noise power $\sigma^2 = 1/(2R(E_b/N_0))$. After puncturing, the code rate $R$ is increased. When $(E_b/N_0)$ is fixed, the noise power is decreased, making the decoders work in a less noisy environment. At the same time, the orthogonal property of Hadamard matrix is broken, which may lead to BER performance degradation. It is the combined effect of the two factors that makes the performance of punctured THC close to the unpunctured THC.

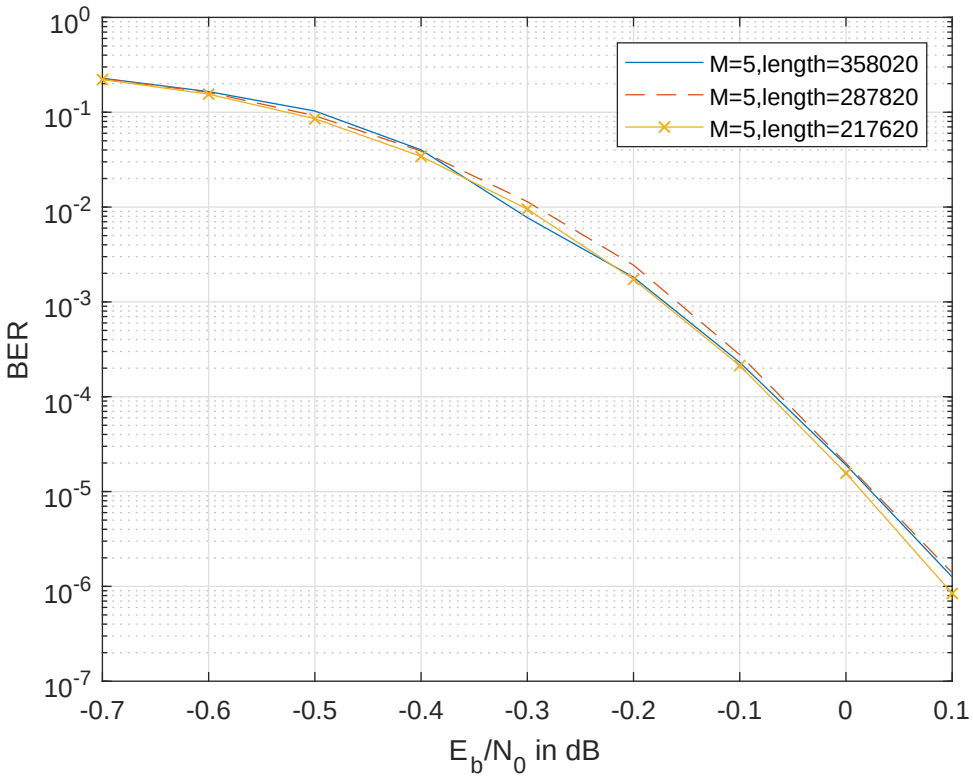


Figure 5.3: BER curves of the punctured and un-puctured THC codes with $M = 5$ component codes. The results are obtained by FPGA simulations.

## 5.2 Lowering the Error Floor

In Chapter 3, we have observed that an error floor occurs when there are $M = 3$ component codes in the THC. The existence of error floors is very common in turbo codes. An error floor occurs because of the relatively low free distance of the code [84, 85]. The problem is partially solved by the concatenation of the single-parity-check code and the recursive convolutional code [86].

Both turbo codes and low-density-parity-check (LDPC) codes are decoded iteratively. In [87], MacKay first related turbo codes to LDPC codes and showed that turbo codes can be treated as block codes. Subsequently, a unified factor graph representation was developed for these two iterative decoding codes [88] and the parity-check matrix of turbo codes was studied [89]. Similar to turbo codes, turbo Hadamard codes can be treated as block codes. In this section, we view THC as a block code and study its parity-check matrix. We optimize the parity-check matrix with an aim to lowering the error floor of the code.

### 5.2.1 Parity-check matrix

Referring to Fig. 5.4, the parity-check matrix relating the message bits $\boldsymbol{D}$ and the recursive convolutional encoder output $\boldsymbol{q}$ is given by

$$\boldsymbol{H}_0 = [\boldsymbol{H}_{D_0} \mid \boldsymbol{H}_q]$$

$$= \left[\begin{array}{cccccccccccccccc|ccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 1
\end{array}\right] \tag{5.12}$$

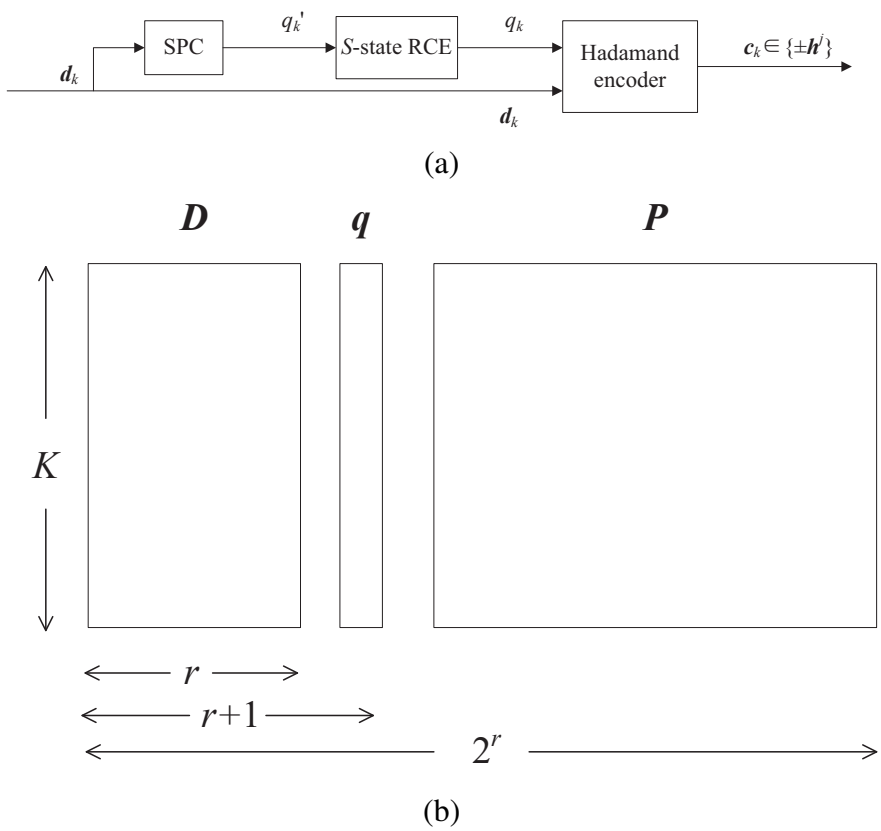


Figure 5.4: Convolutional-Hadamard code. (a) Encoder block diagram and (b) code structure. SPC: single-parity check; RCE: recursive convolutional encoder.

where

- $\boldsymbol{H}_{D_0}$ has a size of $K \times rK$ and each column represents the variable node of a message bit; and

- $\boldsymbol{H}_q$ has a size of $K \times K$ and the $k$-th column represents the recursive convolutional encoder (with generator polynomial $1/(1 + x)$) output $q_k$.

In the example shown in (5.12), we assume $r = 3$. Considering the $k$-th row in $\boldsymbol{H}_{D_0}$, the variable nodes (columns) corresponding to entries with 1's form $\boldsymbol{d}_k$. Then $\boldsymbol{d}_k$ and $q_k$ will be input to the Hadamard encoder to generate the Hadamard parity-check bits. Since the LLR information of the Hadamard parity-check bits is directly derived from the channel and will not be updated in the iterative decoding process, we do not need to consider the parity-check matrix of the Hadamard code here for simplicity.

Assume the number of component codes in the turbo Hadamard code is $M$. Denote the parity-check matrix relating the message bits $\boldsymbol{D}$ and the recursive convolutional encoder output $\boldsymbol{q}$ of the $m$th component code as $\boldsymbol{H}_m$ ($m = 0, 1, 2, ..., M-1$). Similar to $\boldsymbol{H}_0$, we have $\boldsymbol{H}_m = \left[\boldsymbol{H}_{D_m} \mid \boldsymbol{H}_q\right]$. Note that $\boldsymbol{H}_{D_m}$ is no longer in natural order as $\boldsymbol{H}_{D_0}$ because the message bits have been interleaved. Meanwhile the row weight and column weight of $\boldsymbol{H}_{D_m}$ remain the same as $\boldsymbol{H}_{D_0}$. The overall parity-check matrix $\boldsymbol{H}$ relating the message bits $\boldsymbol{D}$ and all $M$ recursive convolutional encoder outputs $\boldsymbol{q}$ is thus the concatenation of $\boldsymbol{H}_0, \boldsymbol{H}_1, ..., \boldsymbol{H}_{M-1}$, i.e.,

$$\boldsymbol{H} = \left[\begin{array}{c|cccc} \boldsymbol{H}_{D_0} & \boldsymbol{H}_q & \mathbf{0} & \cdots & \mathbf{0} \\ \hline \boldsymbol{H}_{D_1} & \mathbf{0} & \boldsymbol{H}_q & \cdots & \mathbf{0} \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{H}_{D_{M-1}} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{H}_q \end{array}\right] \equiv [\boldsymbol{H}_{\boldsymbol{D}} \mid \boldsymbol{H}_{\boldsymbol{Q}}]. \tag{5.13}$$

### 5.2.2 Maximizing the girth

A cycle can visualized as a path moving horizontally and vertically in an alternate manner along the "1"s in a parity-check matrix. Moreover, the path must be closed, that is to say, it starts and ends at the same "1" in the matrix. Obviously the length of a cycle must be an even number because the horizontal and vertical moves always appear in pairs. It is also easy to see that the minimum cycle length is 4. Fig. 5.5 and Fig. 5.6 illustrate cycles with length 4 and 6, respectively, in a parity-check matrix. The *girth* of a parity-check matrix is defined as the shortest cycle length found in the matrix.

Parity-check matrices with large girths is always preferred in the iterative decoding [90]. Moreover, cycle-4 must be avoided in all circumstances because such short cycles will significantly degrade the code performance. Based on (5.12) and (5.13), we can arrive at the following observations.

- $\boldsymbol{H}_q$ and $\boldsymbol{H}_Q$ are fixed and contain no cycle.

$$\left[\begin{array}{cccccccccccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & 1 & 1 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & 0 & 0
\end{array}\right]$$

Figure 5.5: Example of a cycle-4 between two $\boldsymbol{H}_{D_m}$ ($m = 0, 1, 2, ..., M-1$).

- Each of $\boldsymbol{H}_{D_m}$ ($m = 0, 1, 2, ..., M-1$) contains no cycle.
- Each of $[\boldsymbol{H}_{D_m} | \boldsymbol{H}_q]$ ($m = 0, 1, 2, ..., M-1$) contains no cycle.
- All cycles must involve two or more $\boldsymbol{H}_{D_m}$ ($m = 0, 1, 2, ..., M-1$) and may also involve $\boldsymbol{H}_q$.

We denote $\boldsymbol{H}_{D_m}[i, j]$ as the $(i, j)$ entry of $\boldsymbol{H}_{D_m}$; and $\boldsymbol{H}_q[i, j]$ as the $(i, j)$ entry of $\boldsymbol{H}_q$. Then we can obtain the following results.

**Cycle-4:** Cycle-4's only exist between $\boldsymbol{H}_{D_{m_1}}$ and $\boldsymbol{H}_{D_{m_2}}$ where $m_1, m_2 \in \{0, 1, 2, ..., M-1 : m_1 \neq m_2\}$. A cycle-4 exists if and only if $\boldsymbol{H}_{D_{m_1}}[i_1, j_1] = \boldsymbol{H}_{D_{m_1}}[i_1, j_2] = \boldsymbol{H}_{D_{m_2}}[i_2, j_2] = \boldsymbol{H}_{D_{m_2}}[i_1, j_2] = 1$. An example of cycle-4 is given in Fig. 5.5.

**Cycle-6:** A cycle-6 exists if $\boldsymbol{H}_{D_{m_1}}[i_1, j_1] = \boldsymbol{H}_{D_{m_1}}[i_1+1, j_2] = \boldsymbol{H}_{D_{m_2}}[i_2, j_2] = \boldsymbol{H}_{D_{m_2}}[i_1, j_2] = 1$, where $m_1, m_2 \in \{0, 1, 2, ..., M-1 : m_1 \neq m_2\}$ and $j_1 \neq j_2$. Two such examples of cycle-6 are given in Fig. 5.6 and Fig. 5.7.

**Cycle-6:** A cycle-6 exists if $\boldsymbol{H}_{D_{m_1}}[i_1, j_1] = \boldsymbol{H}_{D_{m_1}}[i_1, j_2] = \boldsymbol{H}_{D_{m_2}}[i_2, j_2] = \boldsymbol{H}_{D_{m_2}}[i_2, j_3] = \boldsymbol{H}_{D_{m_3}}[i_3, j_3] = \boldsymbol{H}_{D_{m_3}}[i_1, j_3] = 1$, where $m_1, m_2, m_3 \in \{0, 1, 2, ..., M-1 : m_1 \neq m_2 \neq m_3\}$ and $j_1 \neq j_2 \neq j_3$.

**Cycle-2*l*:** A cycle-2$l$ ($l \geq 4$) exists if $\boldsymbol{H}_{D_{m_1}}[i_1, j_1] = \boldsymbol{H}_{D_{m_1}}[i_1+l-2, j_2] = \boldsymbol{H}_{D_{m_2}}[i_2, j_2] = \boldsymbol{H}_{D_{m_2}}[i_1, j_2] = 1$, where $m_1, m_2 \in \{0, 1, 2, ..., M-1 : m_1 \neq m_2\}$ and $j_1 \neq j_2$.

Note that there are other possibilities for obtaining cycle-8 or above, but they are not listed here. In our study, we focus on avoiding cycle-4 and cycle-6.

The algorithm of eliminating cycle-4 and cycle-6 in the parity-check matrix include the following steps:

1. Randomly generate M-1 interleavers. Note that the first interleaver is in natural order and is fixed.

2. Starting from the second interleaver, check if there is any cycle-4 or cycle-6 associated with the entries of the interleaver. If a short cycle is found, randomly swap the associated entry with another entry of the interleaver.

3. Repeat Step 2 until there are no cycle-4 and cycle-6 existing in the parity-check matrix.

### 5.2.3 Performance evaluation

We simulate a THC with $M = 3$ and a length of 105300. We also design interleaving patterns for $\boldsymbol{H}_{D_1}$ and $\boldsymbol{H}_{D_2}$ such that there is no cycle-4 or cycle-6 in $\boldsymbol{H}$. The bit-error-rate (BER) performance of the optimized THC is shown in Figure. 5.8. We also plot the BER curve of the same code with randomized interleavers for every new set of message bits. The curve therefore represents the average BER over all types of interleavers. Compared with a THC with random interleavers, our optimized THC produces very similar error performance in the waterfall region (from $-1$ to $-0.4$ dB)

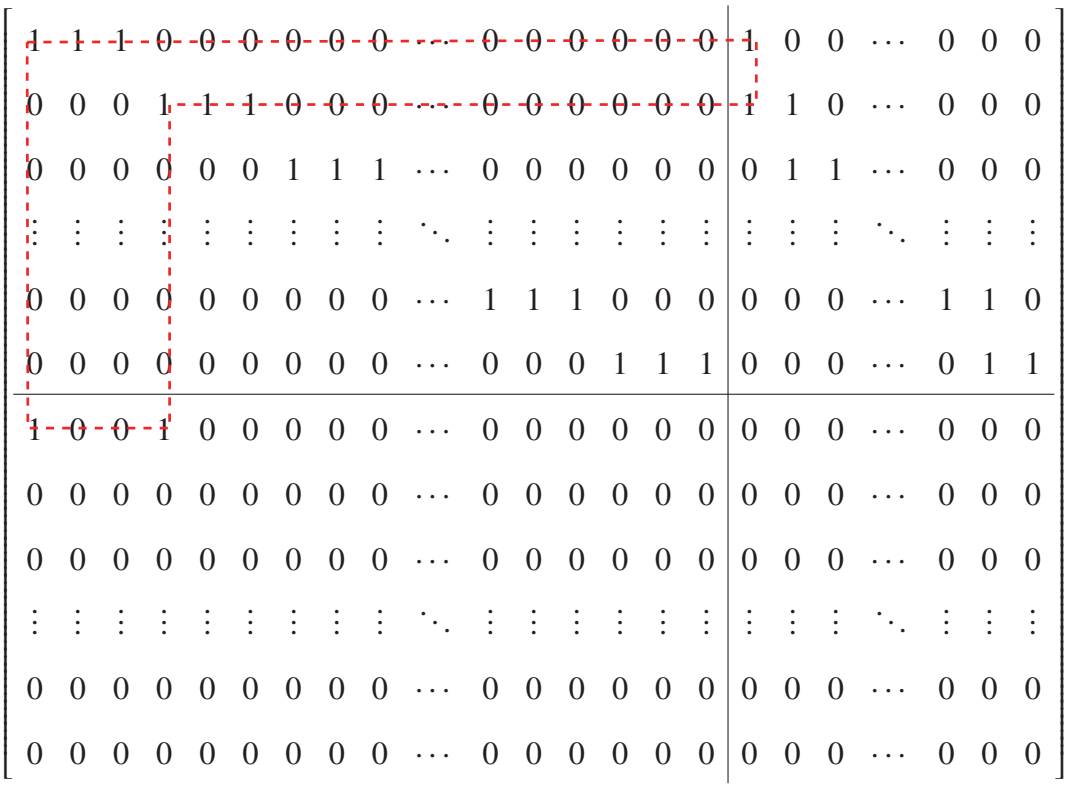

Figure 5.6: An example of cycle 6 involving two $\boldsymbol{H}_{D_m}$ ($m = 0, 1, 2, ..., M-1$) and one $\boldsymbol{H}_q$.

but much better error performance in the error floor region (beyond −0.4 dB). It shows that our criteria for designing interleavers can significantly lower the error floor of THCs.

## 5.3 Summary

In this chapter, we try to optimize turbo Hadamard codes from two aspects — improve the code rate and lower the error floor. Firstly, we try to increase the code rate of the turbo Hadamard codes by puncturing the parity-check bits. We have proposed two effective methods to find good punctured Hadamard codes. The first method is based on maximizing minimum Hamming distance while the second one is based on minimizing cross-correlation values. A second metric $\Omega$ is further used to differentiate codes achieving the same minimum cross-correlation values. Simulation results show that the punctured Hadamard code based on correlation optimization achieves the best BER

Figure 5.7: Another example of cycle 6 involving two $\boldsymbol{H}_{D_m}$ ($m = 0, 1, 2, ..., M-1$) and one $\boldsymbol{H}_q$.

performance in APP decoding, outperforming the one optimized by maximizing MHD by about 0.15 dB at a BER of $10^{-5}$. The punctured Hadamard encoder/decoder system is also realized onto an FPGA. Secondly, we view turbo Hadamard codes as block codes and consider the related parity-check matrices. We optimize the interleavers of the turbo Hadamard code in order to maximize the girth of the related parity-check matrix. The results show that the code using optimized interleavers can lower the error floor compared with that using randomized interleavers. Note that the two proposed optimization techniques can also be used in other hybrid concatenated Hadamard codes like zigzag Hadamard codes.

Evaluating the cycles in the parity-check matrix in this chapter can require quite substantial computational resources, especially when the matrix is large and a large number of cycles are involved. Finding the number of possible cycles (or closed path) for a given matrix can help estimating the amount of computational effort required. In the next chapter, we will propose a method for evaluating the number of closed paths in a matrix.

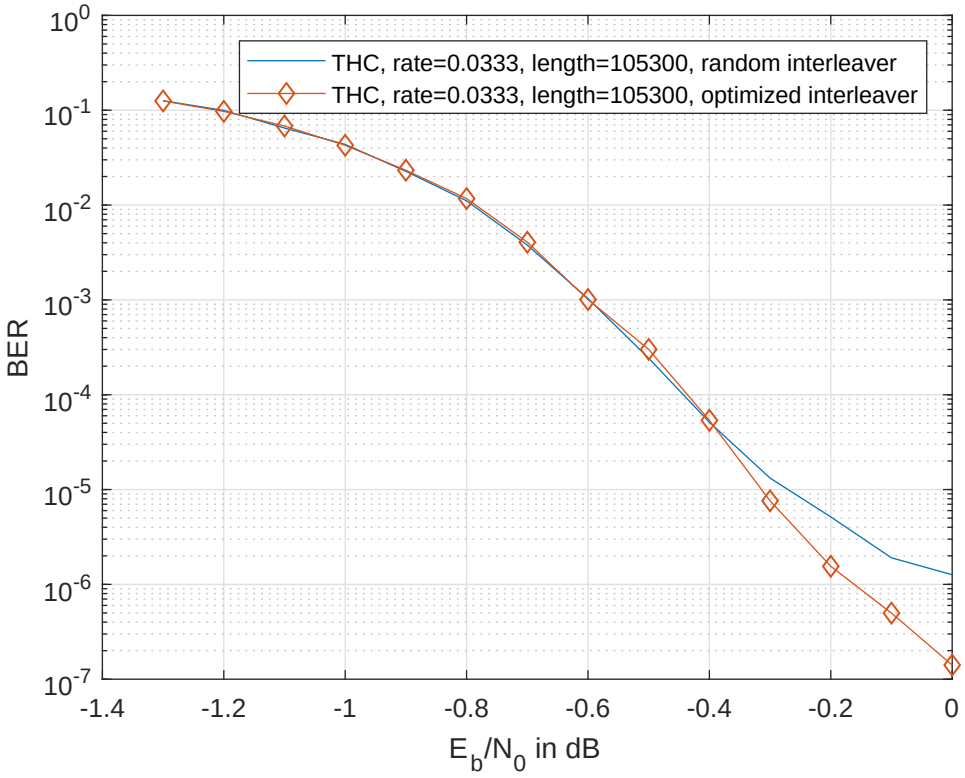


Figure 5.8: BER curves of a THC using optimized interleavers and random interleavers. $M = 3$ and length = 105300.

# Chapter 6

# Evaluating the Number of Closed Paths in an All-One Base Matrix

In the previous chapter, cycle-4 and cycle-6 have been eliminated in the parity-check matrix to improve the error floor of turbo Hadamard codes. Short cycles degrade error performance of not only turbo Hadamard codes, but also low-density parity-check (LDPC) codes. Thus the study of cycles is important in designing high-performing channel codes. In fact, much research has been conducted to construct high performance LDPC codes [11, 91–95] where the girth — minimum cycle length — plays an important role.

A cycle is a closed path in the Tanner graph which starts and ends in the same node. The path alternates between check and variable nodes [93, 94, 96]. The cycle can also be easily analyzed in the corresponding parity-check matrix because each check node in the Tanner graph corresponds to a row in the parity-check matrix, and each variable node corresponds to a column. If a variable node is connected to a check node, the corresponding element in the parity-check matrix will be "1". Similarly, a "0" in the matrix means the corresponding nodes are not connected. A cycle can thus be visualized as a path moving horizontally and vertically in an alternate manner along the "1"s in the matrix. Moreover, the path must be closed, that is to say, it starts and

ends at the same "1" in the matrix.

One way to form an LDPC code is to construct a parity-check matrix of a particular size by assigning "1"s randomly with a certain probability. Such a construction method is not so desirable because the code is unstructured, making the encoding and decoding processes rather complicated to implement in terms of hardware. Another approach is to form a small-size base matrix first and then to replace each non-zero element in the base matrix with a circulant permutation matrix (CPM) or a random permutation matrix (RPM) or a sum/mix of both types of matrices [97, 98].

In this chapter, we consider an all-one base matrix $\mathbf{B}$ of size $M \times N$. Moreover, we denote the "1" in the $(i, j)$-th position of the base matrix by $P_{i,j}$ $(1 \leq i \leq M, 1 \leq j \leq N)$, i.e.,

$$\mathbf{B} = \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} & \dots & P_{1,N} \\ P_{2,1} & P_{2,2} & P_{2,3} & \dots & P_{2,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P_{M,1} & P_{M,2} & P_{M,3} & \dots & P_{M,N} \end{bmatrix}. \tag{6.1}$$

A closed path of length $2l$ can therefore be described as $P_{i_1,j_1} \rightarrow P_{i_2,j_1} \rightarrow P_{i_2,j_2} \rightarrow P_{i_3,j_2} \rightarrow \cdots \rightarrow P_{i_l,j_l} \rightarrow P_{i_1,j_l} \rightarrow P_{i_1,j_1}$ where $i_1 \neq i_2 \neq i_3 \neq \cdots \neq i_{l-1} \neq i_l \neq i_1$ and $j_1 \neq j_2 \neq j_3 \neq \cdots \neq j_{l-1} \neq j_l \neq j_1$. In other words, the closed path starts from the element $P_{i_1,j_1}$, then moves vertically to $P_{i_2,j_1}$, then moves horizontally to $P_{i_2,j_2}$, ..., and finally goes back to the original element $P_{i_1,j_1}$. Note that each of such closed paths may give rise to one or more cycles in the corresponding LDPC code when each "1" in the base matrix is replaced with a circulant permutation matrix (CPM) [1] or a random permutation matrix (RPM). [2] However in practice, with careful selection of the CPMs or RPMs by code designers, many of such cycles in the LDPC code can be avoided. An example is given in Fig. 6.1, where a $2 \times 3$ all-one base matrix is shown. The base matrix has a closed path of length-4. By replacing the 1's in the base matrix with

[1] A circulant permutation matrix is an identity matrix or a cyclic-right-shift of the identity matrix.

[2] A random permutation matrix is a square matrix with exactly one 1 in each row and in each column.

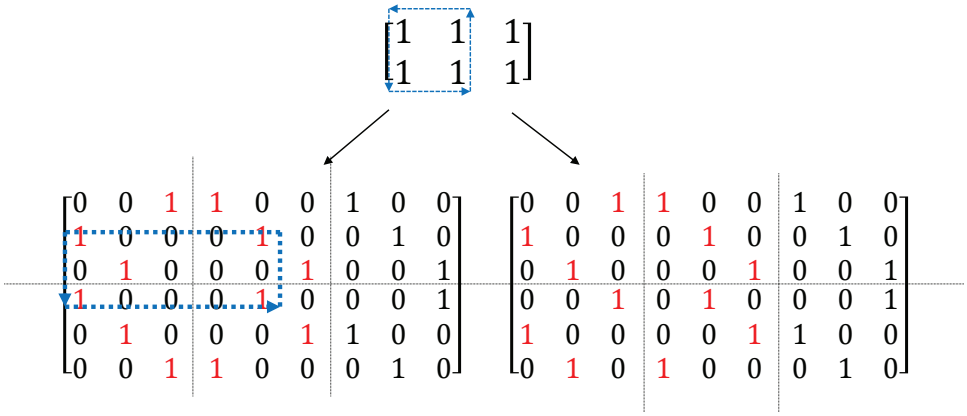

Figure 6.1: Illustration of closed path and cycle

different CPMs, the formed LDPC code may have cycle-4's or have no cycle-4.

One simple way to determine the number of closed paths in the all-one based matrix is to use the "tree method" [96]. The main idea of the tree method is to construct trees whose roots are variable nodes. The tree is extended based on the Tanner graph of the codes. We start from one certain variable node (i.e., column), and put the check nodes (i.e., rows) that connect to the root variable node (column) in the next layer. After that, we add one more variable-node layer where all variable nodes are connected to the check nodes in the previous layer. The procedure of adding a layer is actually a move in the Tanner graph. Once the tree is expanded further enough, closed cycles of different lengths that start and end at the same root node can be found. The main drawback of the tree method is that the trees are always too large to manipulate. It works well for short closed paths and small base matrices. However, it takes a huge amount of space to store the tree and it costs a lot of time to perform the exhaustive searching for when the base matrix becomes large. The searching results also contain duplicates, which can only be eliminated by exhaustive comparisons.

In this chapter, we propose a novel method to evaluate the total number of closed paths of different lengths in an all-one base matrix. Since LPDC codes with short cycles are known to give unsatisfactory error correction capability, the results in this chapter can be used to estimate the amount of effort required to evaluate the number

of potential cycles of an LDPC code or to optimize the code [99, 100]. The results are also verified by those found by the tree method.

## 6.1 Two preliminary functions

**Theorem 2.** *Suppose we have to assign $v$ different digits to $u$ consecutive slots where $u \geq v$ and $u, v \in \mathbb{Z}^+$. Moreover, consecutive slots must contain different digits and all $v$ digits should be used. Then the total number of combinations equals*

$$G(u, v) = v! \sum_{\mathbf{\Omega}} \prod_{j=2}^{v} (j-1)^{\alpha_j} \tag{6.2}$$

*where*

$$\mathbf{\Omega} = \{\{\alpha_j \in \mathbb{N} : j = 2, 3, \ldots, v\} : \sum_{j=2}^{v} \alpha_j = u - v\}. \tag{6.3}$$

*Proof.* Suppose we fill the slots one-by-one. Considering the first slot, i.e, slot $i = 1$, we arbitrarily pick one digit out of the $v$ digits. As consecutive slots must contain different digits, we can only pick another digit out of the $v - 1$ "unused" digits for the second slot, i.e, slot $i = 2$. (We call a digit unused if the digit never appears in previous slots.) For each of the remaining slots, i.e, slot $i = 3, 4, \ldots, u$, we always have two strategies: pick a digit we have used before or pick an unused digit. The only point we need to consider is to make sure that by the last slot, i.e, slot $i = u$, all $v$ digits are used at least once. For $i = 1, 2, 3, ..., u$,

- let $\eta_i$ represent the strategy used to select a digit for slot $i$: $\eta_i = 0$ means a used digit is selected and $\eta_i = 1$ indicates an unused digit is selected;

- $y_i$ denote the number of possible digits to pick for slot $i$ given $\eta_i$;

- $x_i$ denote the number of distinct digits used up to and including slot $i$.

Table 6.1: Two different digit selection sequences for $u = 5$ and $v = 4$.

(a) Digit selection sequence #1

| $i$ | $\eta_i$ | $y_i$ | $x_i$ |
|---|---|---|---|
| 1 | 1 | 4 | 1 |
| 2 | 1 | 3 | 2 |
| 3 | 0 | 1 | 2 |
| 4 | 1 | 2 | 3 |
| 5 | 1 | 1 | 4 |

(b) Digit selection sequence #2

| $i$ | $\eta_i$ | $y_i$ | $x_i$ |
|---|---|---|---|
| 1 | 1 | 4 | 1 |
| 2 | 1 | 3 | 2 |
| 3 | 1 | 2 | 3 |
| 4 | 1 | 1 | 4 |
| 5 | 0 | 3 | 4 |

Note that

$$\sum_{i=1}^{u} \eta_i = v \tag{6.4}$$

because all the $v$ digits must be used/selected at least once in the $u$ slots.

Using the above notations, $x_{i-1}$ distinct digits have been used up to slot $i-1$.

- Supposing $\eta_i = 0$ which means a used digit is to be picked for slot $i$, the number of possible digits to pick equals $y_i = x_{i-1} - 1$ because the digit must be different from the digit in slot $i-1$. Moreover, the number of distinct digits used remains the same and hence $x_i = x_{i-1}$.

- Supposing $\eta_i = 1$ which means an unused digit is to be picked for slot $i$, the number of possible digits to pick equals $y_i = v - x_{i-1}$ and the number of distinct digits used is increased by one, i.e., $x_i = x_{i-1} + 1$.

In both cases described above, we have $x_i = x_{i-1} + \eta_i$. The digit picking procedures are described with a tree shown in Fig. 6.2. Further, two different digit selection sequences for $u = 5$ and $v = 4$ are shown in Table 6.1. The difference between the two selection sequences is the slot number that a used digit is selected. In sequences #1 and #2, a used digit is selected in slot 3 and slot 5, respectively.

Supposing $\eta_i = 0$, then $y_i = x_{i-1} - 1 = x_i - 1 = j - 1$ for some $j \in \{2, \ldots, v\}$.

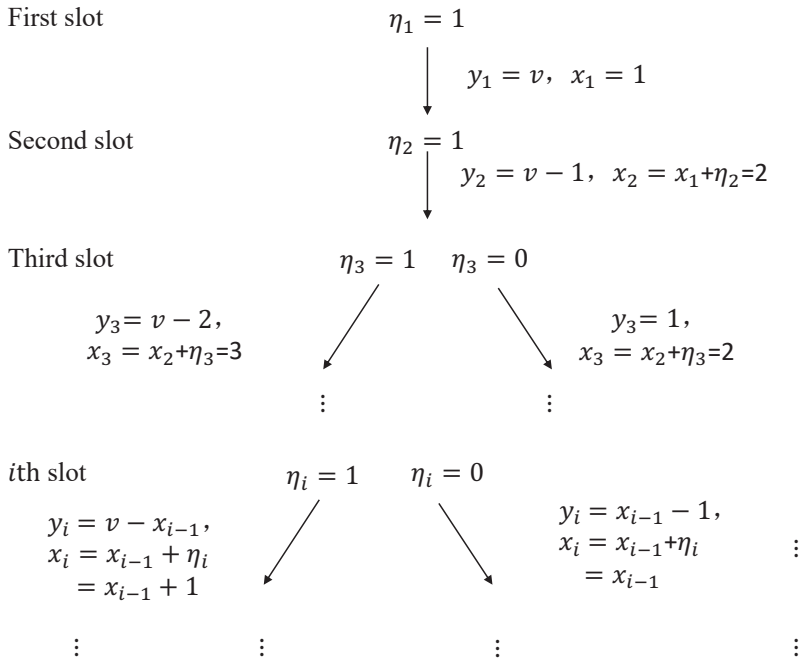


Figure 6.2: Illustration of the digit picking procedures.

Moreover, there are $u$ slots and $v$ digits and hence reused digits must be selected $u - v$ times, i.e., $\eta_i = 0$ must occur $u - v$ times. Denote $\alpha_j$ $(j = 2, \ldots, v)$ as the number of occurrences of the event $\{\eta_i = 0, x_i = j\}$, i.e., $\eta_i = 0$ and $x_i = j$, among all $i = 1, 2, \ldots, u$. Combining the above, we have

$$\sum_{j=2}^{v} \alpha_j = u - v. \tag{6.5}$$

Note that the solution set $\{\alpha_j : j = 2, \ldots, v\}$ for (6.5) is usually not unique. In the digit selection sequence #1 shown in Table 6.1, $\eta_i = 0$ only when $i = 3$. Hence, $x_3 = 2$ and subsequently $\alpha_2 = 1, \alpha_3 = \alpha_4 = 0$. In sequence #2, $\eta_i = 0$ only when $i = 5$. Hence, $x_5 = 4$ and subsequently $\alpha_4 = 1, \alpha_2 = \alpha_3 = 0$. In both cases, we have $\sum_{j=2}^{v} \alpha_j = u - v = 1$.

For a given set $\Theta = \{\alpha_j : j = 2, \ldots, v\}$ that satisfies (6.5), it can be readily shown that the total number of possible digit-sequence selections for the $u - v$ slots where

used digits are selected equals

$$\prod_{\{i\in\{1,2,\ldots,u\}:\eta_i=0\}} y_i = \prod_{j=2}^{v} (j-1)^{\alpha_j}. \tag{6.6}$$

For the remaining $v$ slots where unused digits are selected ($\eta_i = 1$), the number of choices is decreased by one every time and hence the total number of possible digit-sequence selections equals

$$\prod_{\{i\in\{1,2,\ldots,u\}:\eta_i=1\}} y_i = v(v-1)(v-2)\ldots\times 2\times 1 = v!. \tag{6.7}$$

Combining all the above results, the total number of combinations is hence given by

$$\begin{aligned} G(u,v) &= \sum_{\Theta\in\mathbf{\Omega}} \prod_{i=1}^{u} y_i \\ &= \sum_{\Theta\in\mathbf{\Omega}} \left[ \left( \prod_{\{i\in\{1,2,\ldots,u\}:\eta_i=1\}} y_i \right) \left( \prod_{\{i\in\{1,2,\ldots,u\}:\eta_i=0\}} y_i \right) \right] \\ &= v! \sum_{\Theta\in\mathbf{\Omega}} \prod_{j=2}^{v} (j-1)^{\alpha_j} \end{aligned} \tag{6.8}$$

where $\mathbf{\Omega} = \{\{\alpha_j \in \mathbb{N} : j = 2,3,v\} : \sum_{j=2}^{v} \alpha_j = u - v\}$ denotes the solution sets of $\{\alpha_j\}$. □

**Theorem 3.** *If we impose an additional condition on Theorem 2 that the last slot must contain a different digit from the first one, the total number of combinations becomes*

$$G'(u,v) = v! \sum_{\mathbf{\Omega}} \frac{(v-1)^{\alpha_v+1} + (-1)^{\alpha_v}}{v} \prod_{j=2}^{v-1} (j-1)^{\alpha_j} \tag{6.9}$$

*where*

$$\mathbf{\Omega} = \{\{\alpha_j \in \mathbb{N} : j = 2,3,v\} : \sum_{j=2}^{v} \alpha_j = u - v\}. \tag{6.10}$$

*Proof.* We use the same notations as in the previous proof. Most of the proof of Theorem 2 is similar to that of Theorem 1 except when dealing with the last slot, in which the digit must now be different from the digit in the first slot.

Among the $u$ slots, there are $v$ slots where $\eta_i = 1$ because all the $v$ distinct digits must be used at least once. Let $\gamma$ denote the position of the last slot where $\eta_i = 1$. Then, $\eta_\gamma = 1$ and for all subsequent slots, i.e., slots $i = \gamma + 1, \gamma + 2, \ldots, u$ (altogether $u - \gamma$ slots), we always have $\eta_i = 0$ and $x_i = v$. Recall that $\alpha_j$ ($j = 2, \ldots, v$) denotes the number of occurrences of the event $\{\eta_i = 0, x_i = j\}$. Thus, in the case of $j = v$, we have $\alpha_v = u - \gamma$. Also, the last $\alpha_v$ slots all contain used digits.

Denote $f(\alpha_v)$ as the total number of choices for the last $\alpha_v$ consecutive slots. Thus

$$f(\alpha_v) = \prod_{\{i \in \{\gamma+1,\ldots,u\}: \eta_i = 0\}} y_i = \prod_{i \in \{\gamma+1,\ldots,u\}} y_i. \tag{6.11}$$

(Note that in Theorem 1, $f(\alpha_v) = (v-1)^{\alpha_v}$ and equals 1 when $\alpha_v = 0$.)

- If $\alpha_v = 1$, the $(u-1)$-th slot contains an unused digit and the last (i.e., $u$-th) slot contains a used digit. Since the $(u-1)$th slot contains an unused digit, this digit is different from the digit in the first (i.e., $i = 1$) slot. As the digit in the last slot must be different from that in the $(u-1)$th slot and first slot, it has $v - 2$ choices and hence

$$f(\alpha_v = 1) = v - 2. \tag{6.12}$$

- If $\alpha_v = 2$, the $(u-2)$-th slot contains an unused digit and the last two slots (i.e., $(u-1)$-th and $u$-th slots) contain used digits. We use similar arguments as above. Since the $(u-2)$th slot contains an unused digit, this digit is different from the digit in the first (i.e., $i = 1$) slot. We then consider the $(u-1)$-th slot. If it contains the same digit as the one in the first slot (only one choice), the $u$-th slot has $v - 1$ choices; otherwise the digit in the $(u-1)$-th slot must be different from that in the first slot and the $u$-th slot ($v - 2$ choices), and then the $u$-th slot will have $v - 2$ choices. The total number of choices for the last $\alpha_v = 2$ consecutive slots

therefore equals

$$f(\alpha_v = 2) = 1 \times (v-1) + (v-2) \times (v-2) = v^2 - 3v + 3. \tag{6.13}$$

- We consider the general case where $\alpha_v \geq 3$. The $\gamma$-th (i.e, $(u - \alpha_v)$-th) is the last slot where an unused digit is selected.

  1. If the $(\gamma+1)$-th slot is the same as the first slot (1 choice), the $(\gamma+2)$-th slot must be different from the first slot ($v-1$ choices) and from the $(\gamma+3)$-th to $u$-th slots, the number of choices is given by $f(\alpha_v - 2)$.
  2. If the $(\gamma+1)$-th slot is different from the first slot ($v-2$ choices), the number of choices from the $(\gamma+2)$-th to $u$-th slots is given by $f(\alpha_v - 1)$.

  The total number of choices thus equals

$$\begin{aligned} f(\alpha_v) &= 1 \times (v-1) \times f(\alpha_v - 2) + (v-2) \times f(\alpha_v - 1) \\ &= (v-1) \times f(\alpha_v - 2) + (v-2) \times f(\alpha_v - 1). \end{aligned} \tag{6.14}$$

Using (6.12), (6.13) and (6.14), it can be readily shown that

$$f(\alpha_v) = \frac{(v-1)^{\alpha_v+1} + (-1)^{\alpha_v}}{v} \;\forall\; \alpha_v = 1, 2, \ldots \tag{6.15}$$

and hence (6.11) can be written as

$$\prod_{\{i \in \{\gamma+1,\ldots,u\}:\eta_i=0\}} y_i = \frac{(v-1)^{\alpha_v+1} + (-1)^{\alpha_v}}{v} \;\forall\; \alpha_v = 1, 2, \ldots \tag{6.16}$$

where $\gamma = u - \alpha_v$.

For a given set $\Theta = \{\alpha_j : j = 2, \ldots, v\}$ that satisfies (6.5) and the given conditions (particularly that the last slot must contain a different digit from the first one), the total

number of possible digit-sequence selections for the $u - v$ slots where used digits are selected equals

$$\begin{aligned} & \prod_{\{i\in\{1,2,\ldots,u\}:\eta_i=0\}} y_i \\ = & \left(\prod_{\{i\in\{1,2,\ldots,\gamma\}:\eta_i=0\}} y_i\right)\times\left(\prod_{\{i\in\{\gamma+1,\ldots,u\}:\eta_i=0\}} y_i\right) \\ = & \left(\prod_{j=2}^{v-1}(j-1)^{\alpha_j}\right)\times\left(\frac{(v-1)^{\alpha_v+1}+(-1)^{\alpha_v}}{v}\right). \end{aligned} \tag{6.17}$$

Combining the above results with (6.7), the total number of combinations is hence given by

$$\begin{aligned} G'(u,v) &= \sum_{\Theta\in\mathbf{\Omega}}\prod_{i=1}^{u} y_i \\ &= v!\sum_{\mathbf{\Omega}}\left(\frac{(v-1)^{\alpha_v+1}+(-1)^{\alpha_v}}{v}\prod_{j=2}^{v-1}(j-1)^{\alpha_j}\right). \end{aligned} \tag{6.18}$$

(Note also that (6.18) still holds even when $\alpha_v = 0$. It is because according to (6.15), $f(\alpha_v) = 1$ when $\alpha_v = 0$.) □

*Remark:* Using the inclusion-exclusion principle, it can also be proved that

$$G(u,v) = \sum_{i=0}^{v-2}(-1)^i\binom{v}{i}(v-i)(v-i-1)^{u-1}, \quad v \le u; \tag{6.19}$$

and

$$G'(u,v) = G(u,v) - G'(u-1,v) \tag{6.20}$$

with $G'(v,v) = G(v,v)$.

Table 6.2 and 6.3 show the values of $G(u,v)$ and $G'(u,v)$, respectively, for $2 \le v \le u \le l = 7$. These values are sufficient when considering cycles with length no longer

Table 6.2: $G(u, v)$ function

| $G$ | $v = 2$ | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $u = 2$ | 2 | | | | | |
| 3 | 2 | 6 | | | | |
| 4 | 2 | 18 | 24 | | | |
| 5 | 2 | 42 | 144 | 120 | | |
| 6 | 2 | 90 | 600 | 1,200 | 720 | |
| 7 | 2 | 186 | 2,160 | 7,800 | 10,800 | 5,040 |

Table 6.3: $G'$ function

| $G'$ | $v = 2$ | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $u = 2$ | 2 | | | | | |
| 3 | 0 | 6 | | | | |
| 4 | 2 | 12 | 24 | | | |
| 5 | 0 | 30 | 120 | 120 | | |
| 6 | 2 | 60 | 480 | 1,080 | 720 | |
| 7 | 0 | 126 | 1,680 | 6,720 | 10,080 | 5,040 |

than 14. Note also that a quasi-cyclic LDPC code constructed from an all-one base matrix has a girth bounded by 12. For a fixed $v \geq 3$, we can see that both $G(u, v)$ and $G'(u, v)$ increase exponentially with $u$.

## 6.2 Configurations of Closed Paths

Referring to Fig. 6.3, we extract an $m \times n$ sub-matrix from the $M \times N$ all-one base matrix and represent the $(i, j)$-th element in the sub-matrix by $Q_{i,j}$ $(1 \leq i \leq m, 1 \leq j \leq n)$. We define a closed path (CP) as an $(m, n)$-CP if it passes $m$ distinct rows and $n$ distinct columns of an all-one matrix. Denoting an $(m, n)$-CP by $Q_{i_1,j_1} \rightarrow Q_{i_2,j_1} \rightarrow Q_{i_2,j_2} \rightarrow Q_{i_3,j_2} \rightarrow \cdots \rightarrow Q_{i_l,j_l} \rightarrow Q_{i_1,j_l} \rightarrow Q_{i_1,j_1}$ where $i_1 \neq i_2 \neq i_3 \neq \cdots \neq i_{l-1} \neq i_l \neq i_1$

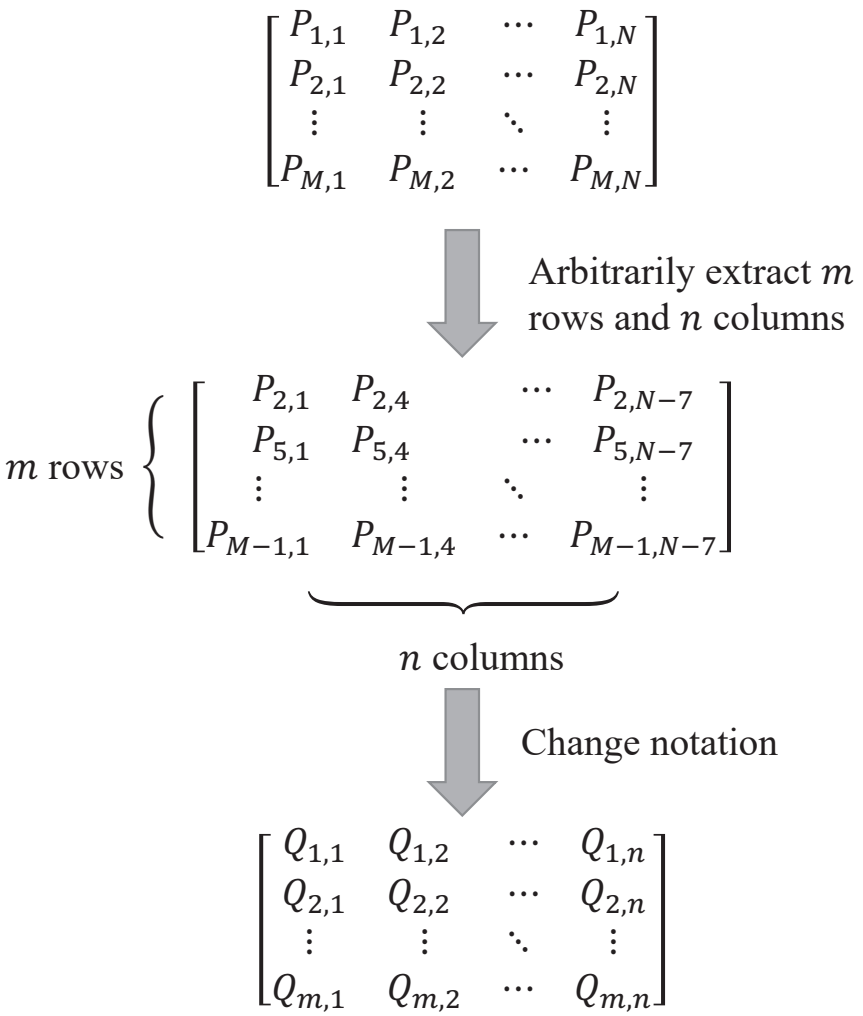


Figure 6.3: Extracting an $m \times n$ sub-matrix from an all-one $M \times N$ base matrix.

and $j_1 \neq j_2 \neq j_3 \neq \cdots \neq j_{l-1} \neq j_l \neq j_1$, we represent the $l$-tuple row-index vector and column-index vector of the $(m, n)$-CP by, respectively, $\mathbf{I} = (i_1, i_2, \ldots, i_{l-1}, i_l)$ and $\mathbf{J} = (j_1, j_2, \ldots, j_{l-1}, j_l)$. Since an $(m, n)$-CP passes $m$ rows and $n$ columns, there are exactly $m$ distinct values in the row-index vector $\mathbf{I}$ and $n$ distinct values in the column-index vector $\mathbf{J}$.

For a given $M \times N$ all-one base matrix and a given cycle length $2l$, the values of $m$

and $n$ are constrained by

$$2 \leq m \leq \min(M, l) \tag{6.21}$$

$$2 \leq n \leq \min(N, l). \tag{6.22}$$

For every $(m, n)$ satisfying (6.21) and (6.22), the numbers of row-index and column-index combinations can be calculated by using the $G'$ function in (6.9). Denoting $R(l, m)$ as the number of row-index combinations and $C(l, n)$ as the number of column-index combinations for an $(m, n)$-CP configuration with length $2l$, we have

$$R(l, m) = G'(l, m) \tag{6.23}$$

$$C(l, n) = G'(l, n). \tag{6.24}$$

The total number of $(m, n)$-CP configurations with length $2l$ therefore equals $R(l, m)C(l, n) = G'(l, m)G'(l, n)$ when duplicated CPs such as those shown in Fig. 6.4 are not eliminated.

## 6.3 Duplicated Closed Paths

Given two different row-index/column-index vector pairs $(\mathbf{I}, \mathbf{J})$ and $(\mathbf{I}', \mathbf{J}')$, they may represent the same CP. Fig. 6.4 shows an example where the same CP of length 6 is formed with different starting element and hence different row-index/column-index vector pairs. In the following, we will remove such duplicates in our computation of the number of CPs.

### 6.3.1 Eliminate CPs that do not start from the first row

First, we eliminate CPs that do not start from the first row (e.g., eliminate CPs that start with elements $Q_{2,2}, Q_{3,3}, Q_{3,2}$ or $Q_{2,1}$ in Fig. 6.4). We therefore add a constraint that every CP must start from the first row of the $m \times n$ sub-matrix. That is to say, we

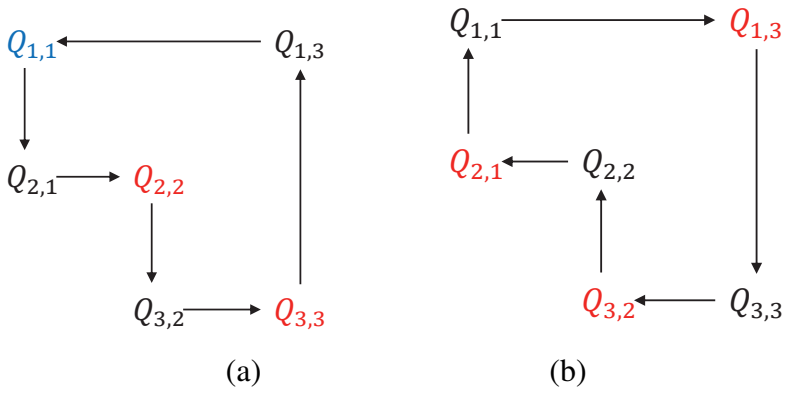


Figure 6.4: Illustration of duplicated closed-paths of length 6. CPs starting from $Q_{2,2}, Q_{3,3}, Q_{1,3}, Q_{3,2}$ and $Q_{2,1}$ are duplicates of that starting from $Q_{1,1}$.

require $i_1 = 1$ for every CP configuration.

Denote $R'(l, m, t)$ as the number of row-index combinations with the new constraint where $t$ represents the number of paths along the first row of the CP configuration. In fact, $t$ also represents the number of "1"s in the row-index vector $\mathbf{I}$. Since consecutive elements in $\mathbf{I}$ must differ, there are at most $\left\lfloor \frac{l}{2} \right\rfloor$ "1"s in $\mathbf{I}$ and thus $1 \le t \le \left\lfloor \frac{l}{2} \right\rfloor$. Moreover, $t$ is restricted by $t \le l - m + 1$. To summarize, we have

$$t \le \min\left(\left\lfloor \frac{l}{2} \right\rfloor, l - m + 1\right). \tag{6.25}$$

Figs. 6.4 and 6.5 illustrate, respectively, the cases where $t = 1$ and $t = 2$. In this chapter we only consider CPs with length $2l$ ranging from 4 to 10. Thus $2 \le l \le 5$ and $1 \le t \le 2$. For larger values of $l$ and $t$, the guiding principles are the same and the results can be derived in a similar manner.

Since $i_1 = 1$, we only need to assign values to $i_2, i_3, \ldots, i_l$.

1. When $t = 1$, we assign all $m - 1$ digits to the $l - 1$ slots: $i_2, i_3, \ldots, i_l$ and make sure $i_2 \ne i_3 \ne i_4 \ne \cdots \ne i_l$. According to Theorem 2, we have

$$R'(l, m, 1) = G(l - 1, m - 1). \tag{6.26}$$

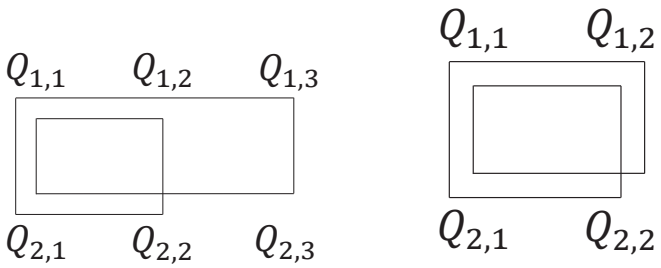


Figure 6.5: Two different CP configurations each of length 8. (a) No transform-exact pair exists, and (b) transform-exact pairs exist.

2. When $t = 2$, one and only one element of $i_3, \ldots, i_{l-1}$ equals 1. Assuming $i_\xi = 1$, the row-index vector becomes $\mathbf{I} = (1, i_2, i_3, \ldots, i_{\xi-1}, 1, i_{\xi+1}, \ldots, i_l)$.

   - If $i_{\xi-1} \neq i_{\xi+1}$, it is equivalent to assigning $m-1$ different digits to $i_2, i_3, \ldots, i_{\xi-1}, i_{\xi+1}, i_{\xi+2}, \ldots$ with different consecutive elements. According to Theorem 2, there are $G(l-2, m-1)$ combinations.
   - If $i_{\xi-1} = i_{\xi+1}$ then it is equivalent to assign $m-1$ different digits to $i_2, i_3, \ldots, i_{\xi-1}, i_{\xi+2}, i_{\xi+3}, \ldots$ with different consecutive elements. According to Theorem 2, there are $G(l-3, m-1)$ combinations. In such a situation, $m < l-1$.

   Combining the above two scenarios, we have

$$R'(l,m,2) = \begin{cases} 0 & \text{if } m > l-1 \\ G(l-2,m-1) & \text{if } m = l-1 \\ G(l-2,m-1) + G(l-3,m-1) & \text{if } m < l-1 \end{cases} \cdot \tag{6.27}$$

### 6.3.2 Eliminate duplicated CPs that start from the first row

After the new constraint in the previous section has been applied, most of the duplicated CPs are eliminated. Some duplicates still remain due to the fact that more than

one elements in the first row can act as the starting element. For example in Fig. 6.4(b), the CP $Q_{1,3} \to Q_{3,3} \to Q_{3,2} \to Q_{2,2} \to Q_{2,1} \to Q_{1,1}$ is a duplicated version of $Q_{1,1} \to Q_{2,1} \to Q_{2,2} \to Q_{3,2} \to Q_{3,3} \to Q_{1,3}$ while both start from the first row.

Assume that we have an $(\mathbf{I}, \mathbf{J})$ pair denoted by

$$\begin{cases} \mathbf{I} = (i_1, i_2, \ldots, i_{l-1}, i_l) \\ \mathbf{J} = (j_1, j_2, \ldots, j_{l-1}, j_l) \end{cases}. \tag{6.28}$$

Then the "reversed" version of the $(\mathbf{I}, \mathbf{J})$ pair will always correspond to the same CP because it is equivalent to indexing the same CP from the last element to the first one. Denoting the reversed pair as $(\mathbf{I}', \mathbf{J}')$, we have

$$\begin{cases} \mathbf{I}' = (i_1, i_l, i_{l-1}, \ldots, i_3, i_2) \\ \mathbf{J}' = (j_l, j_{l-1}, \ldots, j_2, j_1) \end{cases}. \tag{6.29}$$

Moreover, an $(m, n)$-CP of length $2l$ can start from any of the $2l$ elements. If $\mathbf{I}$ and $\mathbf{J}$ are cyclically shifted with same number of positions, the CP that the new pair represents will be identical to the original one. In the following, we define an operator that performs transformation on the $(\mathbf{I}, \mathbf{J})$ pair.

**Definition 1.** *For any* $(\mathbf{I}, \mathbf{J})$ *pair with the form of* (6.28)*, we define* $F(\mathbf{I}, \mathbf{J}, k)$ *as the transformation of the* $(\mathbf{I}, \mathbf{J})$ *pair such as all elements in* $\mathbf{I}$ *and* $\mathbf{J}$ *are cyclically shifted to the left by k positions (*$k = 1, 2, \ldots, l-1$*). Let* $(\mathbf{I}_k, \mathbf{J}_k) = F(\mathbf{I}, \mathbf{J}, k)$*. Then*

$$\begin{cases} \mathbf{I}_k = (i_{k+1}, i_{k+2}, \ldots, i_l, i_1, i_2, \ldots, i_k) \\ \mathbf{J}_k = (j_{k+1}, j_{k+2}, \ldots, j_l, j_1, j_2, \ldots, j_k) \end{cases}. \tag{6.30}$$

It can be easily shown that all the transformed pairs represent the same CP as the original one. Similarly, we have $(\mathbf{I}'_k, \mathbf{J}'_k) = F(\mathbf{I}', \mathbf{J}', k)$ for $k = 1, 2, \ldots, l-1$. All these $l$ pairs also represent the same CP as the $(\mathbf{I}, \mathbf{J})$ pair. In summary, for a given $(\mathbf{I}, \mathbf{J})$ pair,

there are another $2l-1$ $(\mathbf{I}'', \mathbf{J}'')$ transformed pairs that give rise to the same CP.

In the previous section, we have already eliminated duplicates that do not start from the first row. So we only need to consider transformed pairs whose CPs start from the first row. To analyze such cases, we make use of the following theorem.

**Theorem 4.** *For an* $(\mathbf{I}, \mathbf{J})$ *pair with* $t$ *"1"s in* $\mathbf{I}$*, there are* $2t-1$ *transformed pairs whose CPs start from the first row.*

*Proof.* Obviously $(\mathbf{I}', \mathbf{J}')$ is a transformed pair that starts from the first row. Assume that the $t$ "1"s are located at the 1st, $n_2$-th, $n_3$-th, ..., $n_t$-th positions in $\mathbf{I}$. Then $(\mathbf{I}_{n_k-1}, \mathbf{J}_{n_k-1})$ and $(\mathbf{I}'_{l-n_k+1}, \mathbf{J}'_{l-n_k+1})$ for $k = 2, 3, \ldots, t$ are all transformed pairs that starts from the first row. So altogether there are $2t-1$ such transformed pairs. □

When all the transformed pairs are different from the original pair, we call the original pair $(\mathbf{I}, \mathbf{J})$ "transform-equivalent". According to Theorem 4, there are exact $2t-1$ duplicates for each transform-equivalent pair. In some occasions, some of the $2t-1$ transformed pairs are exactly the same as the original $(\mathbf{I}, \mathbf{J})$ pair. In such cases, we call the original $(\mathbf{I}, \mathbf{J})$ "transform-exact". To find out the duplicates, we have to consider each transform-exact pair separately. Fig. 6.5 illustrates the CP configurations with and without transform-exact pair.

Considering the CP configuration shown in Fig. 6.5(a), we denote

$$\begin{cases} \mathbf{I} = (1,2,1,2) \\ \mathbf{J} = (1,3,1,2) \end{cases}. \tag{6.31}$$

Hence, the 3 transformed pairs are given by

$$\begin{cases} \mathbf{I}' = (1,2,1,2) \\ \mathbf{J}' = (2,1,3,1) \end{cases} \quad \begin{cases} \mathbf{I}_2 = (1,2,1,2) \\ \mathbf{J}_2 = (1,2,1,3) \end{cases} \quad \begin{cases} \mathbf{I}'_2 = (1,2,1,2) \\ \mathbf{J}'_2 = (3,1,2,1) \end{cases}. \tag{6.32}$$

Since all 4 column-index vectors $\mathbf{J}, \mathbf{J}', \mathbf{J}_2, \mathbf{J}_2'$ are different, $(\mathbf{I}, \mathbf{J})$ is a transform-equivalent pair and has 3 duplicates.

For the CP configuration shown in Fig. 6.5(b), we denote

$$\begin{cases} \mathbf{I} = (1, 2, 1, 2) \\ \mathbf{J} = (1, 2, 1, 2) \end{cases} . \tag{6.33}$$

The 3 transformed pairs are therefore given by

$$\begin{cases} \mathbf{I}' = (1, 2, 1, 2) \\ \mathbf{J}' = (2, 1, 2, 1) \end{cases} \quad \begin{cases} \mathbf{I}_2 = (1, 2, 1, 2) \\ \mathbf{J}_2 = (1, 2, 1, 2) \end{cases} \quad \begin{cases} \mathbf{I}_2' = (1, 2, 1, 2) \\ \mathbf{J}_2' = (2, 1, 2, 1) \end{cases} . \tag{6.34}$$

Since $(\mathbf{I}, \mathbf{J}) = (\mathbf{I}_2, \mathbf{J}_2)$ and $(\mathbf{I}', \mathbf{J}') = (\mathbf{I}_2', \mathbf{J}_2')$, $(\mathbf{I}, \mathbf{J})$ is a transform-exact pair and has only 1 duplicate.

We denote $D_{ex}(l, m, n, t)$ and $D_{eq}(l, m, n, t)$, respectively, as the number of transform-exact and transform-equivalent pairs for an $(m, n)$-CP with length $2l$. We also denote $D_{sum}(l, m, n, t)$ as the total number of distinct (i.e., no duplicates) $(m, n)$-CP with length $2l$ and $t$ "1"s in $\mathbf{I}$. Then for $t = 1$ and 2, we have the following.

**t=1**

We denote the index vectors as

$$\begin{cases} \mathbf{I} = (1, i_2, \ldots, i_{l-1}, i_l) \\ \mathbf{J} = (j_1, j_2, \ldots, j_{l-1}, j_l) \end{cases} . \tag{6.35}$$

According to Theorem 4, there is 1 (= $2t-1$) transformed pair that may be duplicated. Obviously, the transformed pair is given by

$$\begin{cases} \mathbf{I}' = (1, i_l, i_{l-1}, \ldots, i_3, i_2) \\ \mathbf{J}' = (j_l, j_{l-1}, j_{l-2} \ldots, j_2, j_1) \end{cases}. \tag{6.36}$$

If $l \equiv 0 \pmod 2$, then

$$\mathbf{J} = (j_1, j_2, \ldots, j_{l/2}, j_{(l/2)+1}, \ldots, j_{l-1}, j_l) \tag{6.37}$$

$$\mathbf{J}' = (j_l, j_{l-1}, \ldots, j_{(l/2)+1}, j_{l/2}, \ldots, j_2, j_1). \tag{6.38}$$

The $(l/2)$-th elements of $\mathbf{J}$ and $\mathbf{J}'$ are $j_{l/2}$ and $j_{(l/2)+1}$ respectively. Since $j_{l/2} \neq j_{(l/2)+1}$, $\mathbf{J} \neq \mathbf{J}'$ and $(\mathbf{I}, \mathbf{J}) \neq (\mathbf{I}', \mathbf{J}')$. If $l \equiv 1 \pmod 2$, it can proved in a similar way that $\mathbf{I} \neq \mathbf{I}'$ and $(\mathbf{I}, \mathbf{J}) \neq (\mathbf{I}', \mathbf{J}')$.

To sum up, when $t = 1$, all $(\mathbf{I}, \mathbf{J})$ pairs are transform-equivalent and there is 1 (= $2t-1$) duplicate for each $(\mathbf{I}, \mathbf{J})$ pair. As a result,

$$D_{eq}(l, m, n, 1) = R'(l, m, 1)C(l, n) \tag{6.39}$$

$$D_{ex}(l, m, n, 1) = 0 \tag{6.40}$$

$$D_{sum}(l, m, n, 1) = \frac{D_{eq}(l, m, n, 1)}{2}. \tag{6.41}$$

**t=2**

Besides the first element in $\mathbf{I}$, another element among $i_3, \ldots, i_{l-1}$ equals 1. Assuming $i_\xi = 1$, we denote the index vectors by

$$\begin{cases} \mathbf{I} = (1, i_2, i_3, \ldots, i_{\xi-1}, 1, i_{\xi+1}, \ldots, i_l) \\ \mathbf{J} = (j_1, j_2, \ldots, j_l) \end{cases}. \tag{6.42}$$

According to Theorem 4, there are 3 ($= 2t - 1$) transformed pairs that may be duplicated. These pairs are given by

$$\begin{cases} \mathbf{I}' = (1, i_l, i_{l-1}, \ldots, i_{\xi+1}, 1, i_{\xi-1}, \ldots, i_3, i_2) \\ \mathbf{J}' = (j_l, j_{l-1}, \ldots, j_2, j_1) \end{cases} \tag{6.43}$$

$$\begin{cases} \mathbf{I}_{\xi-1} = (1, i_{\xi+1}, \ldots, i_l, 1, i_2, i_3, \ldots, i_{\xi-1}) \\ \mathbf{J}_{\xi-1} = (j_\xi, j_{\xi+1}, \ldots, j_l, j_1, j_2, j_3 \ldots, j_{\xi-1}) \end{cases} \tag{6.44}$$

$$\begin{cases} \mathbf{I}'_{\xi-1} = (1, i_{\xi-1}, i_{\xi-2}, \ldots, i_3, i_2, 1, i_l, i_{l-1}, \ldots, i_{\xi+2}, i_{\xi+1}) \\ \mathbf{J}'_{\xi-1} = (j_{\xi-1}, j_{\xi-2}, \ldots, j_2, j_1, j_l, j_{l-1}, \ldots, j_{\xi+1}, j_\xi) \end{cases} . \tag{6.45}$$

1. Using a similar proof as in the case $t = 1$, it can be shown that $(\mathbf{I}, \mathbf{J}) \neq (\mathbf{I}', \mathbf{J}')$.

2. Next we consider $(\mathbf{I}, \mathbf{J})$ and $(\mathbf{I}'_{\xi-1}, \mathbf{J}'_{\xi-1})$. If we assume $\mathbf{I} = \mathbf{I}'_{\xi-1}$, then $(i_2, i_3, \ldots, i_{\xi-2}, i_{\xi-1}) = (i_{\xi-1}, i_{\xi-2}, \ldots, i_3, i_2)$. Since consecutive elements in the index vectors must be different, the number of elements in $(i_2, i_3, \ldots, i_{\xi-2}, i_{\xi-1})$ must be odd. Thus $\xi - 2$ is an odd number and so is $\xi$. If we further assume $\mathbf{J} = \mathbf{J}'_{\xi-1}$, then $(j_1, j_2, \ldots, j_{\xi-2}, j_{\xi-1}) = (j_{\xi-1}, j_{\xi-2}, \ldots, j_2, j_1)$. As a result, $\xi - 1$ should be an odd number and $\xi$ should be even. This contradicts with the above requirement that $\xi$ should be odd. Therefore, $\mathbf{I} = \mathbf{I}'_{\xi-1}$ and $\mathbf{J} = \mathbf{J}'_{\xi-1}$ cannot be true simultaneously and $(\mathbf{I}, \mathbf{J}) \neq (\mathbf{I}'_{\xi-1}, \mathbf{J}'_{\xi-1})$.

3. Finally we consider the pairs $(\mathbf{I}, \mathbf{J})$ and $(\mathbf{I}_{\xi-1}, \mathbf{J}_{\xi-1})$. If $(\mathbf{I}, \mathbf{J}) = (\mathbf{I}_{\xi-1}, \mathbf{J}_{\xi-1})$, it can be easily seen that $(\mathbf{I}', \mathbf{J}') = (\mathbf{I}'_{\xi-1}, \mathbf{J}'_{\xi-1})$. Under this circumstance, there can only be one duplicate.

In the following, we evaluate the exact numbers of CP-configurations that belong to the transform-exact category and transform-equivalent category, respectively.

**Transform-exact category:** Suppose $\mathbf{I} = \mathbf{I}_{\xi-1}$, we have $i_2 = i_{\xi+1}$, $i_3 = i_{\xi+2}, \cdots$,

and $i_{\xi-1} = i_l$. Obviously $\xi - 1 - 2 = l - (\xi + 1)$ and hence $\xi = \frac{l+2}{2}$, which means the positions of "1"s in the row-index vector $\mathbf{I}$ are fixed. Since there are $m$ distinct digits in $\mathbf{I}$, there are $m - 1$ distinct digits in the vector $(i_2, i_3, \ldots, i_{\xi-1})$. The number of elements in the vector equals $\xi - 2 = \frac{l-2}{2}$. According to Theorem 2, we have $G(\frac{l-2}{2}, m - 1)$ such row-index vectors. If $\mathbf{J} = \mathbf{J}_{\xi-1}$, we further have $j_1 = j_\xi$, $j_2 = j_{\xi+1}$, $\cdots$, $j_{\xi-1} = j_l \neq j_1$. We need to assign $n$ distinct digits to the column-index vector $(j_1, j_2, \ldots, j_{\xi-1})$ which has $\xi - 1 = \frac{l}{2}$ elements. We also need to make sure $j_1 \neq j_{\xi-1}$. According to Theorem 3, we have $G'(\frac{l}{2}, n)$ such column-index vectors. Hence,

$$D_{ex}(l, m, n, 2) = G\left(\frac{l-2}{2}, m-1\right) G'\left(\frac{l}{2}, n\right). \tag{6.46}$$

For the transform-exact pairs with the above row-index vectors and column-index vectors, each pair has only one duplicate. Therefore, the number of CPs without duplicates equals $\frac{D_{ex}(l,m,n,2)}{2}$.

**Transform-equivalent category:** Since the total number of transform-exact pairs and transform-equivalent pairs equals $R'(l, m, 2)C(l, n)$, the number of transform-equivalent pairs equals

$$D_{eq}(l, m, n, 2) = R'(l, m, 2)C(l, n) - D_{ex}(l, m, n, 2). \tag{6.47}$$

For the transform-equivalent pairs with the above row-index vectors and column-index vectors, each pair has 3 duplicates. Therefore, the number of such CPs without duplicates equals $\frac{D_{eq}(l,m,n,2)}{4}$.

To summarize, when $t = 2$, the total number of distinct CPs equals

$$D_{sum}(l, m, n, 2) = \frac{D_{eq}(l, m, n, 2)}{4} + \frac{D_{ex}(l, m, n, 2)}{2}. \tag{6.48}$$

### 6.3.3 Overall Results

When $t = 1, 2$, the total number of distinct $(m, n)$-CP with length $2l$ is given by

$$\begin{aligned} D_{sum}(l, m, n, t) &= \frac{D_{eq}(l, m, n, t)}{2t} + \frac{D_{ex}(l, m, n, t)}{t} \\ &= \frac{R'(l, m, t)C(l, n) + D_{ex}(l, m, n, t)}{2t} \end{aligned} \tag{6.49}$$

where

$$D_{ex}(l, m, n, 1) = 0 \tag{6.50}$$

$$D_{ex}(l, m, n, 2) = \begin{cases} 0 & \text{if } l \equiv 1 \pmod 2 \\ G(\frac{l-2}{2}, m-1)G'(\frac{l}{2}, n) & \text{if } l \equiv 0 \pmod 2 \end{cases} \tag{6.51}$$

and other expressions are defined in (6.2), (6.9), (6.24), (6.26) and (6.27).

## 6.4 Number of Closed Cycles with Different Lengths

In this section, we will evaluate the number of CPs with length ranging from 4 to 10. For an $M \times N$ all-one base matrix, we denote $S_{dup}(l, M, N)$ as the number of length-$2l$ CPs with duplicates and $S(l, M, N)$ as the number of distinct length-$2l$ CPs. Then we

have

$$S_{dup}(l, M, N) = \sum_{m=2}^{\min(M,l)} \sum_{n=2}^{\min(N,l)} C(l, n)R(l, m)C_M^m C_N^n \tag{6.52}$$

$$S(l, M, N) = \sum_{m=2}^{\min(M,l)} \sum_{n=2}^{\min(N,l)} \sum_{t=1}^{\min(\lfloor \frac{l}{2} \rfloor, l-m+1)} D_{sum}(l, m, n, t)C_M^m C_N^n \tag{6.53}$$

where $C_K^k = \frac{K!}{(K-k)!k!}$.

### 6.4.1 Length-4 CPs

For CPs with length 4, we have $l = 2$, $m = n = 2$, and $t = 1$. Thus for $N \geq M \geq 2$,

$$S_{dup}(2, M, N) = 4C_M^2 C_N^2 \tag{6.54}$$

$$S(2, M, N) = C_M^2 C_N^2. \tag{6.55}$$

### 6.4.2 Length-6 CPs

For CPs with length 6, we have $l = 3$, $2 \leq m, n \leq 3$, and $t = 1$. Thus for $N \geq M \geq 3$,

$$S_{dup}(3, M, N) = 36C_M^3 C_N^3 \tag{6.56}$$

$$S(3, M, N) = 6C_M^3 C_N^3. \tag{6.57}$$

### 6.4.3 Length-8 CPs

For CPs with length 8, we have $l = 4$, $2 \le m, n \le 4$, $t = 1, 2$. Thus for $N \ge M \ge 4$,

$$\begin{aligned} S_{dup}(4, M, N) &= C_M^2(4C_N^2 + 24C_N^3 + 48C_N^4) \\ &\quad + C_M^3(24C_N^2 + 144C_N^3 + 288C_N^4) \\ &\quad + C_M^4(48C_N^2 + 288C_N^3 + 576C_N^4) \end{aligned} \tag{6.58}$$

$$\begin{aligned} S(4, M, N) &= C_M^2(C_N^2 + 3C_N^3 + 6C_N^4) \\ &\quad + C_M^3(3C_N^2 + 18C_N^3 + 36C_N^4) \\ &\quad + C_M^4(6C_N^2 + 36C_N^3 + 72C_N^4). \end{aligned} \tag{6.59}$$

### 6.4.4 Length-10 CPs

For CPs with length 10, we have $l = 5$, $2 \le m, n \le 5$, $t = 1, 2$. Thus for $N \ge M \ge 5$,

$$\begin{aligned} S_{dup}(5, M, N) &= C_M^3(900C_N^3 + 3600C_N^4 + 3600C_N^5) \\ &\quad + C_M^4(3600C_N^3 + 14400C_N^4 + 14400C_N^5) \\ &\quad + C_M^5(3600C_N^3 + 14400C_N^4 + 14400C_N^5) \end{aligned} \tag{6.60}$$

$$\begin{aligned} S(5, M, N) &= C_M^3(90C_N^3 + 360C_N^4 + 360C_N^5) \\ &\quad + C_M^4(360C_N^3 + 1440C_N^4 + 1440C_N^5) \\ &\quad + C_M^5(360C_N^3 + 1440C_N^4 + 1440C_N^5). \end{aligned} \tag{6.61}$$

In the above equations, we set $C_K^k = 0$ for $k > K$ which occurs when the base matrix is not large enough to generate the CPs. The above equations also provide the number of different CP configurations of a given length under a specific sub-matrix size. For

example in (6.61), the term $360C_M^4C_N^3$ implies that for a sub-matrix with size 4×3, there are 360 different CP configurations with length 10. Similarly, the term $1440C_M^5C_N^4$ indicates that for a sub-matrix with size 5×4, there are 1440 different CP configurations with length 10.

Table 6.4: Number of CPs of different lengths under different base-matrix sizes.

| CP length | Base matrix size ($M \times N$) | No. of CPs obtained by the proposed method with (without) duplicates removed | No. of CPs obtained by the "Tree Method" |
|---|---|---|---|
| 4 | $2 \times 3$ | 3 (12) | 3 |
| 6 | $3 \times 4$ | 24 (144) | 24 |
| 8 | $4 \times 5$ | 2,760 (21,840) | 2,760 |
| 10 | $5 \times 5$ | 104,040 (1,040,400) | 104,040 |
| 10 | $4 \times 24$ | 154,471,680 (1,544,716,800) | NA |
| 10 | $5 \times 20$ | 252,560,160 (2,525,601,600) | NA |

Table 6.4 shows the number of CPs evaluated based on our proposed method for different path lengths under different base-matrix sizes. For example, with a base matrix of size $5 \times 5$, the number of CPs of length 10 calculated using our method equals $104,040$ ($= S(5,5,5)$ in (6.61)). The results are compared with those obtained by the tree method. It can be seen that our method produces the exact numbers of CPs as the tree method.

## 6.5 Conclusion

In this chapter, we present a new method of evaluating the number of closed paths in a base matrix. Although we only give results up to length 10, results for longer paths can be readily derived and computed using similar principles. Compared with the traditional "tree method" which uses exhaustive searching, our method reveals the principle of closed paths and their duplicates and derives expressions for computing the number of closed paths. The results are useful when estimating the time resources required in optimizing and constructing high-performing channel codes such as LDPC codes.

# Chapter 7

# Conclusions and Suggestions for Future Work

In this chapter, we re-iterate the main contributions of the thesis and discuss some potential topics for future research.

## 7.1 Main Contributions of the Thesis

As the number of wireless IoT devices keeps increasing, more and more devices need to access the base station and get connected to the Internet. As a result, the multiple access environment may become harsher. In other words, the same frequency spectrum needs to be shared among more devices and the receivers need to operate with a lower signal-to-noise ratio (or $E_b/N_0$). Another application where the receiver needs to operate at a very low SNR is space communications. In this thesis, we investigate two ultimate-Shannon-limit-approaching channel codes, namely turbo Hadamard codes and concatenated zigzag Hadamard codes. We also propose a method for evaluating the number of cycles in an all-one base matrix. The main contributions of the thesis are summarized as follows.

1. We have designed a turbo Hadamard encoding/decoding system and have im-

plemented it onto an FPGA. We have proposed a modified fixed inter-window shuffle interleaver to avoid contentions in interleaving process, and have made use of a multiple sub-decoder system design to enhance the throughput. Moreover, each component of the encoding/decoding system is fully optimized to achieve high hardware-utilization rate. Throughputs up to 3.2 Gbps have been achieved.

2. We have designed a concatenated zigzag Hadamard encoding/decoding system and have implemented it onto an FPGA. Again a multiple sub-decoder system design is used to improve the throughput, which shows a 50% improvement compared with turbo Hadamard encoding/decoding system. The drawbacks are that the decoding latency is larger and more memories are required.

3. We have proposed two effective methods to find good punctured Hadamard codes. The first one aims at maximizing the minimum Hamming distance of the punctured code; and the second aims at minimizing the cross-correlations of the punctured code. The punctured code found by latter method slightly outperforms that found by the former one; while codes found by both methods have shown good error performance. We have applied the puncturing patterns to turbo Hadamard codes and have shown that the code rate can be improved without error performance loss.

4. We have investigated the cycles of the parity-check matrix related to the turbo Hadamard codes. By designing interleavers that avoid both cycle-4 and cycle-6 in the matrix, we have shown that the error floor of the code is lower compared with using random interleavers.

5. We have proposed a new method of evaluating the number of closed paths in an all-one base matrix. Results for cycles up to 10 have been shown. Compared with the traditional “tree method” which uses exhaustive searching; our proposed

method reveals the principle of closed paths and their duplicates, and derives expressions for computing the number of closed paths. The evaluations are useful when estimating the time resources required in optimizing and constructing high performing channel codes such as LDPC codes.

## 7.2 Suggestions for Future Work

### 7.2.1 Enhancing concatenated zigzag Hadamard code

1. In Chapter 5, we have proposed two methods to design punctured Hadamard codes. These punctured Hadamard codes have been applied to turbo Hadamard codes. They can further be applied to concatenated zigzag Hadamard codes.

2. In Chapter 5, we have eliminated cycle-4 and cycle-6 in the parity-check matrix related to turbo Hadamard codes and have designed a code with a lower error floor. Since error floors are also observed for concatenated zigzag Hadamard codes, a similar technique can be applied with an aim to lowering the error floors. The major task is to find the parity-check matrix related to the concatenated zigzag Hadamard code.

3. The latency of a concatenated zigzag Hadamard decoder is relatively high. A potential solution to reduce the latency is to break the Markov chain of the decoding process into several segments, and to perform parallel decoding on these segments. The forward/backward decoding of consecutive segments should be overlapped in order to minimize the performance degradation.

### 7.2.2 Design of low-density-parity-check Hadamard encoder/decoder system

The design of turbo/concatenated zigzag Hadamard code can be further extended to the design of low-density-parity-check (LDPC) Hadamard code. The FHT/APP-FHT processors can be readily used. Yet, the parallel structure of LDPC Hadamard decoder may require a lot of FHT/APP-FHT processors to work simultaneously. New hardware architecture should be proposed to reduce the hardware complexity.

# Bibliography

[1] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[2] D. N. Rowitch and L. B. Milstein, "On the performance of hybrid FEC/ARQ systems using rate compatible punctured turbo (RCPT) codes," *IEEE Transactions on Communications*, vol. 48, no. 6, pp. 948–959, 2000.

[3] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 389–400, 1988.

[4] W. Ryan and S. Lin, "Channel codes classical and modern," *Channel Codes: Classical and Modern*, 01 2009.

[5] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding," in *IEEE International Conference on Communications*, vol. 2, May 1993, pp. 1064–1070.

[6] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, 1996.

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

[8] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, 1997.

[9] A. K. Pradhan, A. Thangaraj, and A. Subramanian, "Construction of near-capacity protograph LDPC code sequences with block-error thresholds," *IEEE Transactions on Communications*, vol. 64, no. 1, pp. 27–37, Jan 2016.

[10] Z. Wang and Z. Cui, "Low-complexity high-speed decoder design for quasi-cyclic LDPC codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 1, pp. 104–114, Jan 2007.

[11] Q. Lu, J. Fan, C. W. Sham, W. M. Tam, and F. C. M. Lau, "A 3.0 Gb/s throughput hardware-efficient decoder for cyclically-coupled QC-LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 1, pp. 134–145, Jan 2016.

[12] V. Guruswami and P. Xia, "Polar codes: Speed of polarization and polynomial gap to capacity," *IEEE Transactions on Information Theory*, vol. 61, no. 1, pp. 3–16, Jan 2015.

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

[14] P. C. Massey and D. J. Costello, "New low-complexity turbo-like codes," in *Proceedings 2001 IEEE Information Theory Workshop (Cat. No.01EX494)*, 2001, pp. 70–72.

[15] D. J. Costello, A. Banerjee, Ching He, and P. C. Massey, "A comparison of low complexity turbo-like codes," in *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers, 2002.*, vol. 1, 2002, pp. 16–20 vol.1.

[16] M. El-Khamy, J. Hou, and N. Bhushan, "Design of rate-compatible structured LDPC codes for hybrid ARQ applications," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 965–973, August 2009.

[17] S. Kallel and D. Haccoun, "Generalized type II hybrid ARQ scheme using punctured convolutional coding," *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 1938–1946, 1990.

[18] L. Ping, W. K. Leung, and K. Y. Wu, "Low-rate turbo-Hadamard codes," *IEEE Transactions on Information Theory*, vol. 49, no. 12, pp. 3213–3224, Dec 2003.

[19] G. Yue, L. Ping, and X. Wang, "Generalized low-density parity-check codes based on Hadamard constraints," *IEEE Transactions on Information Theory*, vol. 53, no. 3, pp. 1058–1079, March 2007.

[20] W. K. R. Leung, G. Yue, L. Ping, and X. Wang, "Concatenated zigzag Hadamard codes," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1711–1723, April 2006.

[21] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.

[22] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Transactions on Communications*, vol. 49, no. 10, pp. 1727–1737, 2001.

[23] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Transactions on Communications*, vol. 52, no. 4, pp. 670–678, 2004.

[24] M. P. C. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[25] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 2966–2984, 2004.

[26] G. Liva and M. Chiani, "Protograph LDPC codes design based on EXIT analysis," in *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, 2007, pp. 3250–3254.

[27] C. E. Shannon, "A mathematical theory of communication." *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948. [Online]. Available: http://dblp.uni-trier.de/db/journals/bstj/bstj27.html#Shannon48

[28] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. USA: Wiley-Interscience, 2006.

[29] A. Goldsmith, *Wireless Communications*. USA: Cambridge University Press, 2005.

[30] E. Elsheikh, K.-K. Wong, Y. Zhang, and T. Cui, "Chapter 10 - user cooperative communications," in *Cognitive Radio Communications and Networks*, A. M. Wyglinski, M. Nekovee, and Y. T. Hou, Eds. Oxford: Academic Press, 2010, pp. 261 – 305. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123747150000101

[31] T. Cover and A. E. Gamal, "Capacity theorems for the relay channel," *IEEE Transactions on Information Theory*, vol. 25, no. 5, pp. 572–584, 1979.

[32] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of product codes," in *1994 IEEE GLOBECOM. Communications: The Global Bridge*, 1994, pp. 339–343 vol.1.

[33] R. M. Pyndiah, "Near-optimum decoding of product codes: block turbo codes," *IEEE Transactions on Communications*, vol. 46, no. 8, pp. 1003–1010, 1998.

[34] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 409–428, 1996.

[35] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Transactions on Information Theory*, vol. 44, no. 3, pp. 909–926, 1998.

[36] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," *Telecommunications and Data Acquisition Progress Report*, vol. 42, 07 1996.

[37] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *Telecommunications and Data Acquisition Progress Report*, 04 1995.

[38] D. Divsalar and R. J. McEliece, "Effective free distance of turbo codes," *Electronics Letters*, vol. 32, no. 5, pp. 445–, 1996.

[39] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and G. Nebe, "Interleaver design for turbo codes," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 5, pp. 831–837, 2001.

[40] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, 1996.

[41] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 585–598, 2001.

[42] A. Kavcic, Xiao Ma, and M. Mitzenmacher, "Binary intersymbol interference channels: Gallager codes, density evolution, and code performance bounds," *IEEE Transactions on Information Theory*, vol. 49, no. 7, pp. 1636–1652, 2003.

[43] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[44] Sae-Young Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Communications Letters*, vol. 5, no. 2, pp. 58–60, 2001.

[45] *Standard for Information Technology-Local and Metropolitan Area Networks-Specific Requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY)*, IEEE 802.11n-2009, 2009.

[46] *Standard for Local and Metropolitan Area Networks-Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, IEEE 802.16-2004, 2004.

[47] ETSI, *ETSI EN 302 307 v1.3.1 Digital Video Broadcasting (DVB); Second Generation [Online].*, Available: https://www.dvb.org/ standards/dvb-s2.

[48] CCSDS, "Ccsds 131.0-b-2 recommendation for space data system standards; TM synchronization and channel coding," 2011.

[49] V. Oksman and S. Galli, "G.hn: The new ITU-T home networking standard," *IEEE Communications Magazine*, vol. 47, no. 10, pp. 138–145, 2009.

[50] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[51] N. Miladinovic and M. P. C. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1594–1606, 2005.

[52] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1610–1614, 2010.

[53] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for LDPC codes," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3385–3400, 2014.

[54] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 638–656, 2001.

[55] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, 1999.

[56] Jinghu Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and Xiao-Yu Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[57] Xiao-Yu Hu, E. Eleftheriou, D. . Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *GLOBE-*

*COM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, vol. 2, 2001, pp. 1036–1036E vol.2.

[58] S. Johnson, "Introducing low-density parity-check codes," 05 2010.

[59] Li Ping, Lihai Liu, Keying Wu, and W. K. Leung, "Interleave division multiple-access," *IEEE Transactions on Wireless Communications*, vol. 5, no. 4, pp. 938–947, April 2006.

[60] X. Wu, Z. Yang, J. Yan, and J. Cui, "Low-rate turbo-Hadamard coding approach for narrow-band interference suppression," in *2014 IEEE International Conference on Communications (ICC)*, 2014, pp. 2130–2134.

[61] S. Che, P. Wang, and X. Wang, "Concatenated twist Hadamard codes," *IEEE Communications Letters*, vol. 11, no. 12, pp. 992–994, 2007.

[62] S. Che and X. Wang, "Parallel concatenated tree Hadamard codes," *IEEE Communications Letters*, vol. 15, no. 7, pp. 743–745, 2011.

[63] W. K. Pratt, J. Kane, and H. C. Andrews, "Hadamard transform image coding," *Proceedings of the IEEE*, vol. 57, no. 1, pp. 58–68, 1969.

[64] M. O. Damen, K. Abed-Meraim, and J. . Belfiore, "Diagonal algebraic space-time block codes," *IEEE Transactions on Information Theory*, vol. 48, no. 3, pp. 628–636, 2002.

[65] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (Corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[66] Yao-Jun Wu and Li Ping, "On the limiting performance of turbo-Hadamard codes," *IEEE Communications Letters*, vol. 8, no. 7, pp. 449–451, 2004.

[67] Li Ping and N. Phamdo, "Zigzag codes and concatenated zigzag codes," in *1999 Information Theory and Networking Workshop (Cat. No.99EX371)*, 1999, pp. 70–.

[68] K. Li, X. Wang, and A. Ashikhmin, "Exit functions of Hadamard components in repeat-zigzag-Hadamard (RZH) codes with parallel decoding," *IEEE Transactions on Information Theory*, vol. 54, no. 4, pp. 1773–1785, 2008.

[69] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[70] P. Martin, "An analysis of random number generators for a hardware implementation of genetic programming using FPGAs and Handel-C." 06 2002, pp. 837–844.

[71] Jing Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 101–119, 2005.

[72] A. Nimbalker, T. K. Blankenship, B. Classon, T. E. Fuja, and D. J. Costello, "Contention-free interleavers for high-throughput turbo decoding," *IEEE Transactions on Communications*, vol. 56, no. 8, pp. 1258–1267, August 2008.

[73] C. Wong, M. Lai, C. Lin, H. Chang, and C. Lee, "Turbo decoder using contention-free interleaver and parallel architecture," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 2, pp. 422–432, 2010.

[74] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 1249–1253, 2006.

[75] T. Blankenship, B. Classon, and V. Desai, "High-throughput turbo decoding techniques for 4G," *Proc. Int. Conf. 30 Wireless and Beyond*, pp. 137–142, 01 2002.

[76] M. Sybis and P. Tyczka, "Reduced complexity Log-MAP algorithm with Jensen inequality based non-recursive max operator for turbo TCM decoding," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, p. 238, 09 2013.

[77] P. Kumar, S. Sutha, V. K. Mourthy, and M. Student, "Mathematical implementation of MAX LOG MAP algorithm for low power applications in turbo decoders," 2014.

[78] M. Torun, O. Yilmaz, and A. Akansu, "FPGA, GPU, and CPU implementations of Jacobi algorithm for eigenanalysis," *Journal of Parallel and Distributed Computing*, vol. 96, 05 2016.

[79] Li Ping and S. Chan, "Iterative decoding of concatenated Hadamard codes," in *ICC '98. 1998 IEEE International Conference on Communications. Conference Record. Affiliated with SUPERCOMM'98 (Cat. No.98CH36220)*, vol. 1, 1998, pp. 136–140 vol.1.

[80] S. Jiang, P. W. Zhang, F. C. M. Lau, C. . Sham, and K. Huang, "A turbo-Hadamard encoder/decoder system with hundreds of Mbps throughput," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, 2018, pp. 1–5.

[81] S. Jiang, P. W. Zhang, F. C. M. Lau, and C. . Sham, "An ultimate-Shannon-limit-approaching Gbps throughput encoder/decoder system," *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–1, 2019.

[82] B. Sklar, "Rayleigh fading channels in mobile digital communication systems .i. characterization," *IEEE Communications Magazine*, vol. 35, no. 7, pp. 90–100, 1997.

[83] K. J. Hole, "Punctured convolutional codes for the 1-D partial-response channel," *IEEE Transactions on Information Theory*, vol. 37, no. 3, pp. 808–817, 1991.

[84] J. Hagenauer and L. Papke, "Decoding "turbo"-codes with the soft output Viterbi algorithm (SOVA)," in *Proceedings of 1994 IEEE International Symposium on Information Theory*, 1994, pp. 164–.

[85] L. C. Perez, J. Seghers, and D. J. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1698–1709, 1996.

[86] D. Divsalar, S. Dolinar, and F. Pollara, "Iterative turbo decoder analysis based on density evolution," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 5, pp. 891–907, 2001.

[87] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[88] N. Wiberg, H. . Loeliger, and R. Kotter, "Codes and iterative decoding on general graphs," in *Proceedings of 1995 IEEE International Symposium on Information Theory*, 1995, pp. 468–.

[89] F. Jiang, E. Psota, and L. C. Perez, "The generator and parity-check matrices of turbo codes," in *2006 40th Annual Conference on Information Sciences and Systems*, 2006, pp. 1451–1454.

[90] S. J. Johnson and S. R. Weller, "Codes for iterative decoding from partial geometries," *IEEE Transactions on Communications*, vol. 52, no. 2, pp. 236–243, 2004.

[91] Y. Wang, S. C. Draper, and J. S. Yedidia, "Hierarchical and high-girth QC-LDPC codes," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4553–4583, July 2013.

[92] D. Mitchell, R. Smarandache, and D. Costello, "Quasi-cyclic LDPC codes based on pre-lifted protographs," in *IEEE Information Theory Workshop (ITW)*, Oct 2011, pp. 350–354.

[93] G. Zhang, "Type-II quasi-cyclic low-density parity-check codes from Sidon sequences," *Electronics Letters*, vol. 52, pp. 367–369(2), March 2016. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/el.2015.2634

[94] M. Gholami and M. Alinia, "High-performance binary and non-binary low-density parity-check codes based on affine permutation matrices," *IET Communications*, vol. 9, pp. 2114–2123(9), November 2015.

[95] C.-W. Sham, X. Chen, F. C. M. Lau, Y. Zhao, and W. M. Tam, "A 2.0 Gb/s throughput decoder for QC-LDPC convolutional codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 7, pp. 1857–1869, July 2013.

[96] X. Zheng, F. C. M. Lau, and C. K. Tse, "Constructing short-length irregular LDPC codes with low error floor," *IEEE Transactions on Communications*, vol. 58, no. 10, pp. 2823–2834, Oct. 2010.

[97] W. M. Tam, F. C. M. Lau, and C. K. Tse, "A class of QC-LDPC codes with low encoding complexity and good error performance," *IEEE Communications Letters*, vol. 14, no. 2, pp. 169–171, 2010.

[98] Y. Fang, G. Bi, Y. L. Guan, and F. C. M. Lau, "A survey on protograph LDPC codes and their applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 1989–2016, Fourthquarter 2015.

[99] Y. Wang and J. S. Yedidia, "Construction of high-girth QC-LDPC codes," in *5th International Symposium on Turbo Codes and Related Topics*, Sept 2008, pp. 180–185.

[100] F. C. M. Lau and W. M. Tam, "A fast searching method for the construction of QC-LDPC codes with large girth," in *IEEE Symposium on Computers and Communications (ISCC)*, July 2012, pp. 125–128.