INVESTIGATION OF LOW-DENSITY PARITY-CHECK CODES AND DECODERS

FANLU MO
M.Phil
The Hong Kong Polytechnic University
2018

The Hong Kong Polytechnic University
Department of Electronic and Information Engineering

Investigation of low-density parity-check codes and decoders

Fanlu Mo

A thesis submitted in partial fulfilment

of the requirements for the

Degree of Master of Philosophy

August 2017

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)

Fanlu Mo (Name of student)

To my family

Abstract

Low-density parity-check (LDPC) codes have been proved to have theoretical limits approaching the channel capacity. Quasi-cyclic LDPC (QC-LDPC) codes are a particularly important class of LDPC codes. QC-LDPC codes have attracted much attention because of their special advantages, flexible design and ease of implementation. Recently, a novel type of QC-LDPC block codes called cyclically-coupled quasi-cyclic LDPC block codes (CC-QC-LDPC codes) have been proposed. CC-QC-LDPC code have been shown to achieve outstanding bit error performance with extremely low error floor.

Our first study is a novel construction method of CC-QC-LDPC codes which shortens the code length of CC-QC-LDPC codes and at the same time achieves a larger girth and a better error performance. We show that even with a shorter code length, the new CC-QC-LDPC codes can outperform the traditional CC-QC-LDPC ones.

In our second study, we modify the CC-QC-LDPC code construction by using random permutation matrices instead of circulant permutation matrices, forming random-permutation-matrix-based CC-LDPC (RP-CC-LDPC) codes. Compared with CC-QC-LDPC codes, regular and irregular QC-LDPC codes, the BER performance of the RP-CC-LDPC code is comparable to and can possibly exceed that of other codes. However, decoder complexity and throughput are the major issues of the RP-CC-LDPC code.

With an aim to solving the above problem, our third study proposes tree permutation matrices (TPM) and uses them to construct new types of LDPC codes. This kind of LDPC codes is named as tree-permutation-matrix LDPC

(TPM-LDPC) codes. We also realize parallel decoding of TPM-LDPC codes using field-programmable gate-array (FPGA). Compared with random-permutation-matrix-based LDPC codes, the TPM-LDPC codes can achieve a higher throughput. Compared with QC-LDPC codes, the TPM-LDPC codes achieve a slightly better BER but requires a bit more resources in hardware implementation.

Publications

1. F. C. M. Lau, Fanlu Mo, Qing Lu, Wai M. Tam and Chiu-Wing Sham, "Novel Types of Cyclically-Coupled Quasi-Cyclic LDPC Block Codes," *In Proc., 2016 International Conference on Advanced Technologies for Communications (ATC 2016)*, October 2016, Hochiminh City, Vietnam, pp. 27–31.

2. F. C. M. Lau, Fanlu Mo, Wai M. Tam and Chiu-Wing Sham, "Random-Permutation-Matrix-Based Cyclically-Coupled LDPC Codes," *In Proc., International Conference on Advanced Communication Technology*, February 2017, Korea, pp. 280–284. (Received Outstanding Paper Award)

3. Sheng Jiang, Fanlu Mo, F. C. M. Lau and Chiu-W. Sham, "Tree-Permutation-Matrix-Based LDPC Codes," to appear in *IEEE Trans. on Circuits and Systems II*.

Acknowledgements

I would like to take this opportunity to thank my supervisor Prof. F.C.M. Lau and other colleagues in the laboratory for their advice, assistance and patience during my study at The Hong Kong Polytechnic University.

Table of contents

List of Tables

List of Figures

Chapter 1

Introduction

As one of the two known classes of Shannon limit-approaching codes, low-density parity-check (LDPC) codes are wildly used and studied because of their better error performance especially for high code rate and their structure that is strongly preferred in hardware implementation. The study of LDPC codes can be traced back to mid-1990's. Up to now, studies presented by a great number of researchers covered areas such as fast encoding, construction of LDPC codes with large girth and hardware implementation of decoders. This thesis is related to the construction of LDPC codes with good error performance and the implementation of decoders on field-programmable gate-array (FPGA). This chapter presents the background of LDPC codes, the scope of our work, and the organization of this thesis.

1.1 Background

Low-density parity-check (LDPC) codes introduced by Gallager in 1962 [1] have been widely applied and studied in the past two decades. LDPC codes have been proved to have theoretical limits approaching the channel capacity. Moreover, extremely long codes (length larger than 10^6) can perform very close to their theoretical limits because the probability of having short cycles in their graph representations is very small [2]. However, such long codes are not practical in most applications because of the high hardware complexity and high latency [3, 4]. Codes of length ranging from 500 to 100,000 are more practical because they can be applied to more communication systems. It is also well-known that error performance of LDPC codes with finite lengths may degrade significantly from their theoretical limits if there exist many short cycles in their associated Tanner graphs. Such short cycles will also form structures such as stopping sets [5, 6], elementary/dominant/detrimental trapping sets [7–9] and absorbing sets [9, 10] that give rise to the error floor in the high signal-to-noise-ratio (SNR) region. One effective way to maximize the girth (shortest cycle length) of an LDPC block code is to connect the variable nodes and the check nodes based on the progressive-edge growth (PEG) method [11] or its modifications [12, 13]. Yet, the codes constructed using these algorithms may still possess error floors in the high signal-to-noise-ratio (SNR) region.

To lower the error floor by reducing the occurrences of stopping sets and/or elementary/detrimental trapping sets and/or absorbing sets, various construction methods that (i) avoid small-size stopping sets [6]; (ii) avoid short cycles with approximate cycle extrinsic message degree (ACE) below a given value [14]; (iii) combine PEG and ACE [12]; (iv) combine PEG and Approximate-minimum-Cycle-Set-

Extrinsic message- degree (ACSE) [9]; and (v) control the absorbing set spectrum [15, 16]; have been proposed [6, 9, 12, 14–16]. These methods in general can be applied to the construction of random LDPC block codes as well as structured LDPC block codes such as quasi-cyclic LDPC (QC-LDPC) block codes.

Random construction and algebraic construction are two main set of methods to construct LDPC codes. Random constructions can produce LDPC codes that closely approach the Shannon capacity [17], [18], [19], [2]. Some irregular LDPC codes have shown better error performance than regular LDPC codes. On the other side, algebraic constructions have also attracted much interest because algebraic constructions yield structures that are strongly preferred in hardware implementations. Quasi-cyclic LDPC (QC-LDPC) codes are a particularly important class of algebraic-construction-based LDPC codes. QC-LDPC codes have been widely used because of their flexible design and ease of implementation. We can use different structured codes to construct QC-LDPC codes, such as finite geometry codes [20] and circulant-permutation-matrix (CPM) based codes [21],[22],[23]. In order to optimize the performance of QC-LDPC codes, short cycles should be removed during code constructions. However, structured codes usually cannot guarantee a large girth (shortest cycle length) with a relatively short code length. Hence error floors can still exist in the high SNR region for codes constructed under the method like PEG.

Some efficient computer searching methods have been proposed to remove the short cycles in QC-LDPC codes. Since there is an enormous number of possible cycle combinations, it is extremely time-consuming and infeasible to conduct an exhaustive computer search. The hill-climbing algorithm [24] is an effective searching

method that modifies the elements of the base matrix iteratively. In [25], a fast hill-climbing method has been further proposed to reduce the time complexity and make the searching method more efficient. However, whatever construction methods we use, a regular QC-LDPC code cannot have a girth larger than 12 because these cycles are inevitable [26],[9].

Using LDPC convolutional codes (LDPCCCs) and QC-LDPC convolutional codes (QC-LDPCCCs) is another way to improve the error performance and lower the error floor [27],[28],[29]. However, the decoder of LDPCCC has a high hardware complexity. Recently, a novel type of LDPC block code called cyclically-coupled quasi-cyclic LDPC block codes (CCQC-LDPC code) has been proposed [30]. It is formed by spatially-coupling a number of identical QC-LDPC block codes partially in a cyclic manner. Furthermore, this property allows a simple design of a CC-QC-LDPC decoder. To verify the feasibility of the idea that coupling the QC-LDPC block codes partially can improve the overall error performance, the authors in [30] construct a CC-QC-LDPC code with a rate of 5/6 and length around 98000. The only criterion in the design is that the girth (minimum cycle length) equals 8. No other optimization technique such as minimum distance analysis has been applied. An experimental decoder has also been implemented on a field-programmable-gate-array (FPGA) device. Under the condition that 10 decoding iterations are performed for each codeword, the decoder achieves a throughput of 3.0 Gb/s . Moreover, the CC-QC-LDPC decoder has a much lower complexity requirement and 50% higher throughput compared to a LDPCCC decoder that achieves similar BER performance [29]. For this rate-5/6 CC-QC-LDPC code, no error floor has been observed above a bit error rate (BER) of 10^{-14} and a BER below 10^{-15} is expected at a bit-

energy-to-noise-power-spectral-density ratio ($Eb/N0$) of 3.5 dB. A net coding gain of 11.5 dB is therefore achieved. Furthermore, having taken the difference in code rates into consideration, the CC-QCLDPC code has still demonstrated substantial improvement over the un-coupled QC-LDPC block codes.

1.2 Objective and Scope

It is well-known that error performance of LDPC codes with finite lengths may degrade significantly from their theoretical limits if there exists many short cycles in their associated Tanner graphs. How to construct LDPC codes with an outstanding error performance has always been a problem. Various construction methods also have been proposed to optimize the error performance of LDPC codes. Recently, a new class of QC-LDPC codes which is called cyclically-coupled quasi-cyclic LDPC (CC-QC-LDPC) block codes have been proposed and shown to achieve outstanding bit error performance with extremely low error floor.

In this thesis, our first study is focused on methods that shorten the code length of CC-QC-LDPC codes and at the same time, allow the code achieving a larger girth and a better error performance. With an aim to improving the girth, we propose two special types of CC-QC-LDPC codes. We convert some sub-matrices in a full CC-QC-LDPC code into zero matrices. Because of the conversion, cycles passing through these removed sub-matrices can be eliminated. As a result, when we construct the CC-QC-LDPC codes with a smaller sub-matrix size, we can still obtain girth 8. The structures and error performance of the two new types of CC-QCLDPC codes is given in our study.

Our second study is focused on achieving a better error performance of CC-QC-LDPC codes with the same code length. We modify the CC-QC-LDPC code and propose a new type of CC-LDPC codes. Instead of using circulant permutation matrices, we make use of random permutation matrices to form the basic building blocks of CC-LDPC codes. The proposed code is called random-permutation-matrix-based CC-LDPC (RP-CC-LDPC) code. The decoder of the proposed RC-CC-LDPC codes has been implemented using FPGA. We also compare the bit error rate (BER) results and decoder complexity of our codes with those of regular and irregular QC-LDPC codes under the same code length and code rate. Results show that the BER performance of the RP-CC-LDPC code is comparable to and can possibly exceed that of other codes. However, decoder complexity and throughput are major issues of the RP-CC-LDPC code which makes use of relatively more complex random permutation matrices instead of simple circulant permutation matrices.

Our third task is therefore to design CC-LDPC codes with a lower complexity than RP-CC-LDPC codes and good BER performance. In order to reduce the resource utilization of FPGA and improve the throughput of the decoder, we modify the QC-LDPC code construction by using tree-permutation-matrices (TPM) instead of circulant permutation matrices. A TPM can be obtained by repeatedly replacing a '1' in a matrix with a 2×2 permutation matrix. Moreover, parallel decoding can be realized for TPM-based LDPC codes. As a result, compared with the RP-CC-LDPC codes, a higher throughput can be achieved and the complexity of the decoder is reduced. Compared with QC-LDPC codes, although TPM-based LDPC codes require more resources in implementation, they provide a better BER performance. Compared with CC-QC-LDPC codes, TPM-LDPC codes also use more resources in

implementation and they have a lower error floor when the code length is relatively short.

1.3 Organization

Chapter 2 gives an introduction to LDPC codes, including the definition of LDPC codes and their representations based on parity-check matrices and Tanner graphs (bipartite graphs). Then we review the belief propagation (BP) algorithm which is used to decode LDPC codes. After that an important class of LDPC codes, namely quasi-cyclic (QC) LDPC codes, is presented. We also introduce the hill-climbing algorithm and the fast hill-climbing algorithms, both of which are used to search for high-girth QC-LDPC codes. At the end of Chapter 2, the definition and construction of CC-QC-LDPC codes will be briefly reviewed. We also introduce the overall FPGA architecture of its decoder as the background of our study.

In chapter 3, we introduce two new types of CC-QC-LDPC codes which possess better girth properties. We construct the codes and describe the structures of the two new types of CC-QCLDPC codes. We also present the error performance of the new CC-QC-LDPC codes. We find that even with a shorter code length, the new CC-QC-LDPC codes can achieve a better error performance compared with the original CC-QC-LDPC ones. In the same chapter, we further propose a new type of CC-LDPC codes. Instead of using circulant permutation matrices, we make use of random permutation matrices to form the basic building blocks of CC-LDPC codes. The proposed code is called a random-permutation-matrix-based CC-LDPC (RP-CC-LDPC) code. The decoder of the proposed RC-CC-LDPC codes has been

implemented using FPGA. The error performance and decoder complexity of the new code are evaluated and compared with other codes. We will describe the structure of the proposed RP-CC-LDPC codes and introduce the memory arrangement of the hardware implementation. Finally, we show the simulation results and compare the results with other LDPC codes.

In chapter 4, we modify the LDPC code construction by using tree-permutation-matrices (TPM) instead of circulant permutation matrices or random permutation matrices. TPM-based LDPC codes can allow parallel decoding. We can achieve a higher throughput and at the same time reduce the complexity of FPGA decoder compared with RP-CC-LDPC code. We will first introduce TPM and TPM-LDPC codes. After that we propose an effective method to find cycles in TPM-LDPC codes and an efficient way to construct a high-girth TPM-LDPC code systematically. Then we explain how to avoid the RAM access conflicts in the decoder design. Finally, we show the simulation results of our proposed TPM-LDPC codes and compare the results and decoder complexity with other regular and irregular LDPC codes. Throughout our investigation, 10 belief propagation decoding iterations are used and 4-bit quantization is applied in FPGA decoding.

In chapter 5, we summarize the work performed in this thesis and present some future work.

Chapter 2

Low-Density Parity-Check Codes

This chapter gives a brief introduction of low-density parity-check (LDPC) codes. The definition of an LDPC code and its representations based on parity-check matrix and Tanner graph (bipartite graph) are reviewed. Then the belief propagation (BP) algorithm which is used to decode LDPC codes is presented. After that an important class of LDPC codes called quasi-cyclic (QC) LDPC codes will be reviewed and the hill-climbing methods for constructing QC-LDPC codes with large girth are introduced. Finally, the definition of cyclically-coupled QC-LDPC (CC-QC-LDPC) codes and the decoder architecture will be briefly reviewed.

2.1 Definition of LDPC Codes

2.1.1 Matrix Representation of LDPC Codes

A parity-check code of length N can be defined by an $M \times N$ parity-check matrix $\mathbf{H}$, whose M rows specify each of the M constraints. For example, if the first constraint designates that the 3rd bit and 5th bit should be equal, then there must be two 1's in the first row of $\mathbf{H}$, one at column three and another one at column five, and 0's elsewhere. Let $\mathbf{C}$ be a set of binary vectors and $\mathbf{H}$ be a matrix containing only 0's and 1's. If the set of binary vectors $\mathcal{C}$ satisfies all the constraints, i.e., $\mathcal{C} = \{\mathbf{c} : \mathbf{c}\mathbf{H}^T = \mathbf{0}\}$, $\mathcal{C}$ is defined as a parity-check code and $\mathbf{H}$ defined as the parity-check matrix of the code. Denoting the rank of $\mathbf{H}$ by $r = \text{rank}(\mathbf{H}) \leq M$, the code dimension K is given by $K = N - r$ and it represents the number of information bits in each codeword $\mathbf{c}$. Since each codeword has length N and contains K information bits, its code rate is K/N.

A "low-density" parity-check (LDPC) code is defined by a parity-check matrix which is sparse [1]. This parity-check matrix should contain mostly 0's and relatively few 1's. The parity-check matrix of a regular LDPC code is an $M \times N$ binary matrix containing j ones in each column and l ones in each row, where $j < l < N$ and $Ml = Nj$. Since $Ml = Nj$, the parameters N, M, j and l cannot be chosen independently. For example, N, j and l should be chosen in such a way that Nj/l is an integer. The code rate $R = 1 - M/N$ (when $\mathbf{H}$ is full rank) of a regular LDPC code equals $R = 1 - j/l$. For an irregular LDPC code, the parity-check matrix is still sparse but not all rows and columns have the same number of 1's. In (2.1) and (2.2), we show the parity-check matrix of an $(N, j, l) = (10, 2, 5)$ regular LDPC

code and an irregular LDPC code, respectively. In (2.1) we can see that there are 10 columns. Moreover, each column contains two 1's and each row contains five 1's. The code rate is $R = 1 - j/l = 3/5$.

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.1}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.2}$$

2.1.2 Graph Representation of LDPC Codes

The graph representation of a parity-check matrix was first introduced by Gallager [1] in 1963. He used a tree to connect codeword symbols and their corresponding parity-check equations. In 1981, Tanner presented the concept of bipartite graph (Tanner graph), and then Tanner graph became the most useful graph representation of LDPC codes, especially in explaining belief propagation decoding.

A Tanner graph, which can be used to specify an LDPC code, is essentially a visual representation of a parity-check matrix $\mathbf{H}$. Recall that an $M \times N$ parity-check matrix $\mathbf{H}$ can represent an LDPC code whose code length is N and each codeword satisfies a set of M parity-check constraints. The Tanner graph of $\mathbf{H}$ contains N

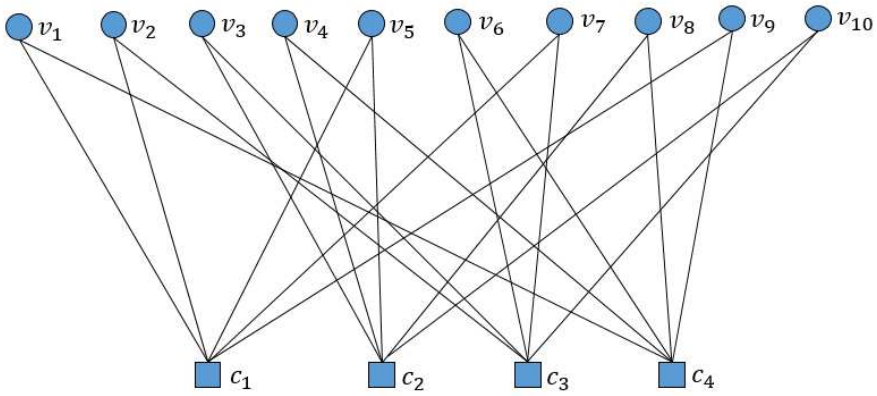


Figure 2.1: The Tanner graph of a $(10, 2, 5)$ regular LDPC code

variable nodes (bit nodes) corresponding to the elements of the codeword and M check nodes corresponding to the set of parity-check constraints. We use circles to represent the variable nodes and squares to represent the check nodes. If the (m, n)-th entry in $\mathbf{H}$ equals 1, i.e., $\mathbf{H}_{m,n} = 1$, the n-th variable node participates in the m-th check node and the n-th variable node will be connected to the m-th check node. The Tanner graph is also called a bipartite graph because it contains two distinct type of nodes — check nodes and variable nodes — and direct connections cannot exist between any two variable nodes or two check nodes.

The Tanner graph associated with the (10,2,5) LDPC code in (2.1) is shown in Figure 2.1. The 10 variable nodes are shown as circles at the top of the figure and the 4 check nodes are shown as squares at the bottom. We can see that each variable node has two branches (i.e., degree 2) connecting to 2 different check nodes because each column in $\mathbf{H}$ contains two 1's. Similarly, each row in $\mathbf{H}$ has five 1's and hence the number of branches connected to each check node is always five (i.e.,

degree 5) .

An ensemble of LDPC codes bearing the same degree distributions can be characterized by the degree-distribution polynomials

$$\lambda(x) = \sum_{i=2}^{j_{\max}} \lambda_i x^{i-1} \tag{2.3}$$

and

$$\rho(x) = \sum_{i=2}^{l_{\max}} \rho_i x^{i-1} \tag{2.4}$$

where

- $\lambda(x)$ denotes the variable-node degree distribution;
- λ_i is the fraction of edges connected to variable nodes of degree i;
- $j_{\max}$ is the maximum variable-node degree;
- $\rho(x)$ denotes the check-node degree distribution;
- ρ_i is the fraction of edges connected to check nodes of degree i;
- $l_{\max}$ is the maximum check-node degree.

Moreover, the code rate R is given by

$$R = 1 - \frac{\int_0^1 \rho(x)\,dx}{\int_0^1 \lambda(x)\,dx}. \tag{2.5}$$

In the case of the irregular LDPC code in (2.2), the degree-distribution polynomials are given by

$$\lambda(x) = \sum_{i=2}^{j_{max}} \lambda_i x^{i-1} = \frac{2}{5}x + \frac{2}{5}x^2 + \frac{1}{5}x^3,$$

$$\rho(x) = \sum_{i=2}^{l_{max}} \rho_i x^{i-1} = \frac{4}{9}x^4 + \frac{5}{9}x^5,$$

and the code rate equals

$$R = 1 - \frac{\int_0^1 \rho(x)\,dx}{\int_0^1 \lambda(x)\,dx} = 0.6.$$

2.2 LDPC Decoding

In this section, the belief propagation (BP) algorithm, which is also called the sum-product algorithm (SPA), is introduced. The BP algorithm is the most widely used decoding algorithm and has demonstrated successes in many applications including LDPC decoding and turbo decoding.

The BP algorithm carries out iterative message-passing processes to achieve convergence for all constrained variable nodes. We use

- $\mathbf{c} = (c_1, c_2, c_3, \ldots, c_N)$ to represent the codeword;
- $\mathbf{t} = ((-1)^{c_1+1}, (-1)^{c_2+1}, (-1)^{c_3+1}, \ldots, (-1)^{c_N+1})$ to represent the transmitted vector; and
- $\mathbf{r} = (r_1, r_2, r_3, ...r_N)$ to represent the received vector.

Given the received vector $\mathbf{r}$, the log-likelihood ratio of the n-th bit is given by

$$\lambda_n = \log \frac{Pr[c_n = 1|\mathbf{r}]}{Pr[c_n = 0|\mathbf{r}]} \tag{2.6}$$

where $Pr[x]$ denotes the probability of x. According to Bayes rule,

$$\begin{aligned} Pr[c_n = 1|\mathbf{r}] &= Pr[c_n = 1|r_n, \{r_{i\neq n}\}] \\ &= \frac{Pr(r_n, c_n = 1, \{r_{i\neq n}\})}{Pr(r_n, \{r_{i\neq n}\})} \\ &= \frac{Pr(r_n|c_n = 1, \{r_{i\neq n}\})Pr(c_n = 1, \{r_{i\neq n}\})}{Pr(r_n|\{r_{i\neq n}\})Pr(\{r_{i\neq n}\})} \\ &= \frac{Pr(r_n|c_n = 1)Pr(c_n = 1, \{r_{i\neq n}\})}{Pr(r_n|\{r_{i\neq n}\})Pr(\{r_{i\neq n}\})} \end{aligned} \tag{2.7}$$

Similarly,

$$Pr[c_n = 0|\mathbf{r}] = \frac{Pr(r_n|c_n = 0)Pr(c_n = 0, \{r_{i\neq n}\})}{Pr(r_n|\{r_{i\neq n}\})Pr(\{r_{i\neq n}\})} \tag{2.8}$$

and hence λ_n can be re-written as

$$\begin{aligned} \lambda_n &= \log \frac{Pr(r_n|c_n = 1)Pr(c_n = 1, \{r_{i\neq n}\})}{Pr(r_n|c_n = 0)Pr(c_n = 0, \{r_{i\neq n}\})} \\ &= \log \frac{Pr(r_n|c_n = 1)}{Pr(r_n|c_n = 0)} + \log \frac{Pr(c_n = 1, \{r_{i\neq n}\})}{Pr(c_n = 0, \{r_{i\neq n}\})}. \end{aligned} \tag{2.9}$$

In (2.9), the first item denotes the "direct" contribution derived from the received signal r_n. Assuming an additive white Gaussian noise (AWGN) channel with noise power σ^2, we have

$$Pr(r_n|c_n) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left(\frac{-(r_n - 2c_n + 1)^2}{2\sigma^2}\right) \tag{2.10}$$

and

$$\log \frac{Pr(r_n|c_n = 1)}{Pr(r_n|c_n = 0)} = \frac{2}{\sigma^2} r_n. \tag{2.11}$$

In (2.9), the second item denotes the "indirect" contribution coming from other received signals $\{r_{i\neq n}\}$ and can be derived from the BP iterative algorithm.

2.2.1 Belief Propagation Iterative Algorithm

Considering the parity-check matrix in (2.1) and its corresponding Tanner graph in Figure 2.1, each BP iteration can be divided into two parts. In the first part of the iteration, the check-to-variable (C2V) message u_{mn} sent from check node m to variable node n is computed by

$$u_{mn} = 2\tanh^{-1}\left(\prod_{n'\in N(m),n'\neq n}\tanh\left(\frac{\lambda_{mn'}}{2}\right)\right) \quad \forall\, m = 1,2,\ldots,M \text{ and } n \in N(m) \tag{2.12}$$

where

- $N(m)$ denotes the set of variable nodes connected to check node m; and
- $\lambda_{mn'}$ is the variable-to-check (V2C) message sent from variable node n' to check node m in the previous iteration.

Basically, check node m computes the C2V message to variable node n based on all incoming V2C messages $\lambda_{mn'}$ except the one from variable node n. For example, in Figure 2.2, check node c_1 uses the information $\lambda_{12}, \lambda_{15}, \lambda_{17}, \lambda_{19}$ from v_2, v_5, v_7, v_9, respectively, and computes the C2V message u_{11} to v_1.

In the second part of the iteration, the overall LLR of variable node n is computed using

$$\lambda_n = \frac{2}{\sigma^2}r_n + \sum_{m'\in M(n)} u_{m'n} \quad \forall\, n = 1,2,\ldots,N; \tag{2.13}$$

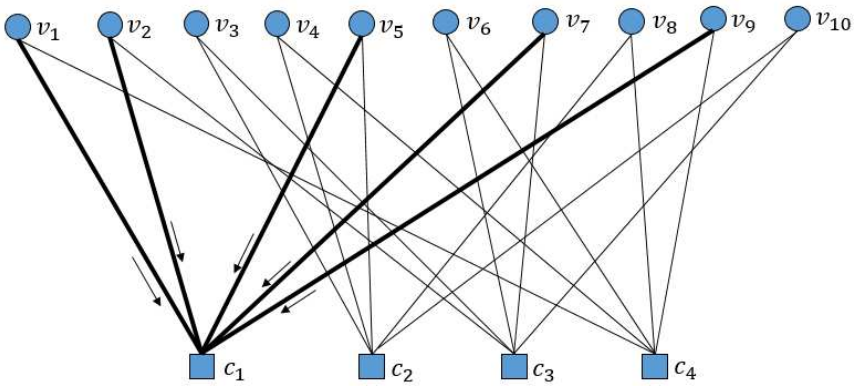


Figure 2.2: Variable-to-check messages

and the V2C message λ_{mn} sent from variable node n to check node m is computed using

$$\lambda_{mn} = \frac{2}{\sigma^2} r_n + \sum_{m' \in M(n), m' \neq m} u_{m'n} = \lambda_n - u_{mn} \quad \forall\, n = 1, 2, \ldots, N \text{ and } m \in M(n) \tag{2.14}$$

where $M(n)$ denotes the set of check nodes connected to variable node n. For example in Figure 2.3, variable node v_1 receives the incoming C2V messages u_{11} and u_{41} from c_1 and c_4, respectively. Then v_1 updates the LLR λ_1, and computes the V2C messages λ_{11} and λ_{41} and send them to c_1 and c_4, respectively.

The procedures of the BP algorithm for a $M \times N$ parity-check matrix $\mathbf{H}$ are summarized in Algorithm 1.

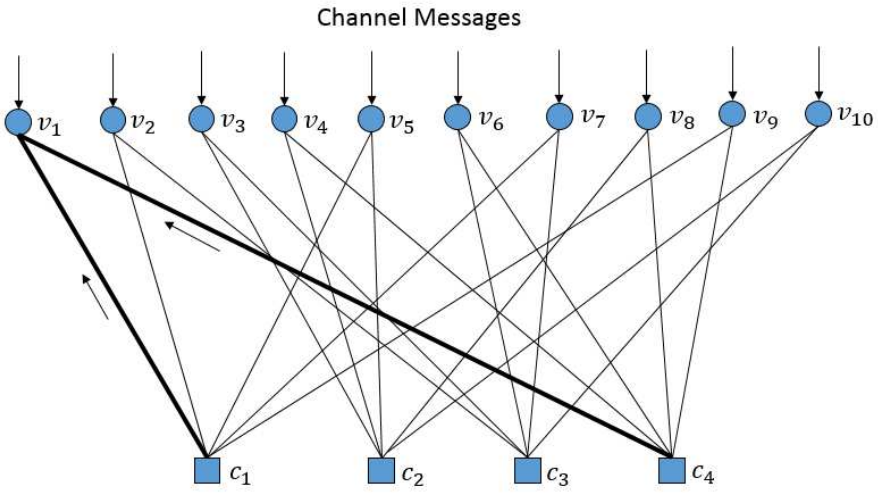


Figure 2.3: Check-to-variable messages

Algorithm 1 Procedures of the BP algorithm

1: **Initialization**: Set $\lambda_{mn}^{(0)} = \frac{2}{\sigma^2} r_n \; \forall m \; \in \{1, 2, \ldots, M\}$ and $n \in \{1, 2, \ldots, N\}$.
2: **Begin iteration**
3: **for** iteration $i = 1, 2, \ldots, i_{\max}$ **do**
4: Check-node update: for $m \in \{1, 2, \cdots, M\}$ and $n \in N(m)$, compute

$$u_{m,n}^{(i)} = 2 \tanh^{-1} \left(\prod_{n' \in N(m), n' \neq n} \tanh \left(\frac{\lambda_{mn'}^{(i-1)}}{2} \right) \right) \tag{2.15}$$

5: Variable node update: for $n \in \{1, 2, \ldots, N\}$ and $m \in \{1, 2, \cdots, M\}$, compute

$$\begin{aligned} \lambda_n^{(i)} &= \frac{2}{\sigma^2} r_n + \sum_{m' \in M(n)} u_{m'n}^{(i)} \\ \lambda_{mn}^{(i)} &= \lambda_n^{(i)} - u_{mn}^{(i)} \end{aligned} \tag{2.16}$$

end
6: **Decision**: If $\lambda_n^{(i_{\max})} > 0, c_n = 1$; else $c_n = 0$.

2.3 Quasi-Cyclic LDPC Codes

Quasi-cyclic LDPC (QC-LDPC) codes are a particularly important class of algebraically constructed LDPC codes. The parity-check matrix of a QC-LDPC code

can be divided into sub-matrices which have a circulant structure. QC-LDPC code is also a widely used LDPC code and has been featured in many communication system standards, e.g., IEEE 802.16e, DVB-S2, and 801.11 [31, 32].

A (J, L) regular QC-LDPC code can be described by a parity-check matrix $\mathbf{H}$

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{(p_{0,0})} & \mathbf{I}_{(p_{0,1})} & \cdots & \mathbf{I}_{(p_{0,L-1})} \\ \mathbf{I}_{(p_{1,0})} & \mathbf{I}_{(p_{1,1})} & \cdots & \mathbf{I}_{(p_{1,L-1})} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{I}_{(p_{J-1,0})} & \mathbf{I}_{(p_{J-1,1})} & \cdots & \mathbf{I}_{(p_{J-1,L-1})} \end{bmatrix} \tag{2.17}$$

where $\mathbf{I}_{(p_{j,l})}$ represents a $z \times z$ circulant permutation matrix (CPM) obtained by cyclically right-shifting the $z \times z$ identity matrix $\mathbf{I}$ by $p_{j,l}$ positions ($0 \leq j \leq J-1, 0 \leq l \leq L-1, 0 \leq p(i,j) \leq z-1$). $\mathbf{I}_{(p_{j,l})}$ represents the $z \times z$ identical matrix $\mathbf{I}$ when $p_{j,l} = 0$. Therefore, QC-LDPC codes can also be defined by a $J \times L$ base matrix $\mathbf{H}_b$ where

$$\mathbf{H}_b = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,L-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,L-1} \\ \cdots & \cdots & & \cdots \\ p_{J-1,0} & p_{J-1,1} & \cdots & p_{J-1,L-1} \end{bmatrix}. \tag{2.18}$$

A cycle is defined as a path that begins and ends at the same variable node. For a QC-LDPC code, an arbitrary cycle of length $2i$ has to pass through $2i$ elements of the base matrix. The sequence of the elements are denoted by $(j_0, l_0), (j_0, l_1), (j_1, l_1), \ldots, (j_{i-1}, l_{i-1}), (j_{i-1}, l_0)$, where $j_s \neq j_{s+1}$ and $l_s \neq l_{s+1}$ for $0 \leq s-1 \leq i$. Then there

exists a cycle of length $2i$ if [22]

$$\sum_{s=0}^{i-1}(p_{j_s,l_s} - p_{j_s,l_{s+1}}) = 0 \pmod{z}. \tag{2.19}$$

Figure 2.4 shows the expanded parity-check matrix $\mathbf{H}_b$ for the base matrix

$$\mathbf{H}_b = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 2 & 1 & 1 & 1 \end{bmatrix} \tag{2.20}$$

with $z = 3$. It can be seen that cycle-4 exists and involves $p_{0,0}, p_{0,1}, p_{1,1}$ and $p_{1,0}$. Moreover, it can be easily found that $p_{0,0} - p_{1,0} + p_{1,1} - p_{0,1} = 0 \mod 3$. For an LDPC code, the girth is defined as its minimum cycle length. The error performance of LDPC codes with finite lengths may degrade significantly from their theoretical limits if many short cycles exist. Therefore short cycles should be eliminated when constructing LDPC codes.

2.3.1 Hill-Climbing Algorithm

The hill-climbing algorithm [24] is an effective computer searching method to find QC-LDPC codes with large girth. The algorithm starts with a base matrix $\mathbf{H}_b$ of a QC-LDPC code where the elements $p_{j,l}$ are assigned values randomly among $\{0, 1, 2, \ldots, z-1\}$. Using the base matrix $\mathbf{H}_b$, the cost vector $\mathbf{c}_{j,l} = \{c_{j,l,0}, c_{j,l,1}, \ldots, c_{j,l,z-1}\}$ for each element $p_{j,l}$ is computed where $c_{j,l,k}$ denotes the cost when $p_{j,l} = k$. The cost function is related to the numbers of cycles whose lengths are less than a desired girth and some weight parameters. In the cost function, cycles with a shorter length are always weighted more costly than cycles with a longer length. A cost matrix $\mathbf{C}$

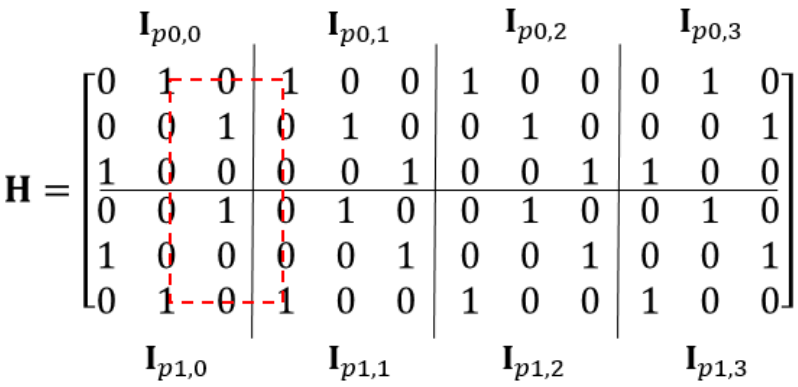


Figure 2.4: Illustration of a cycle-4 in a QC-LDPC code

can be defined to track the cost, i.e.,

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_{0,0} & \mathbf{c}_{0,1} & \cdots & \mathbf{c}_{0,L-1} \\ \mathbf{c}_{1,0} & \mathbf{c}_{1,1} & \cdots & \mathbf{c}_{1,L-1} \\ \cdots & \cdots & & \cdots \\ \mathbf{c}_{J-1,0} & \mathbf{c}_{J-1,1} & \cdots & \mathbf{c}_{J-1,L-1} \end{bmatrix}. \tag{2.21}$$

In each iteration, a change that can reduce the cost most is made. The base matrix and cost matrix are modified iteratively until the cost cannot be further reduced.

2.3.2 Fast Hill-Climbing Algorithm

In the original hill-climbing algorithm, the computational complexity is relatively large because in each iteration, all the cycles with length shorter than the desired girth should be counted for all elements. It is necessary to reduce the computational

complexity of the original hill-climbing algorithm when the base matrix is large and/or the sub-matrix size $z \times z$ is large. In [25], a fast searching method for the construction of QC-LDPC codes with large girth has been proposed.

This fast searching algorithm also begins with a randomly constructed base matrix. Then the elements $p_{j,l}$ are selected and modified one-by-one in a sequential way, as shown in Fig. 2.5. However, the evaluation criterion (cost) is changed adaptively, as illustrated in Fig. 2.6. To begin with, the evaluation criterion is the number of cycles with length 4. Therefore for each element $p_{j,l}$, a particular value is selected such that the smallest number of cycle-4 is formed by this element. After all elements have been considered once, the first element will be re-considered again. The process repeats until a base matrix without cycle-4 is found. Then the evaluation criterion is changed to (i) no cycle-4 AND (ii) the number of cycles with length 6. When each element $p_{j,l}$ is modified, the new value should ensure that no cycle-4 will be formed and the number of cycle-6 will not be reduced. Otherwise, its value should not be changed. Again the process repeats for all elements iteratively until a base matrix without cycle-4 nor cycle-6 is found. Then the evaluation criterion is changed to (i) no cycle-4 AND (ii) no cycle-6 AND (iii) the number of cycles with length 8. The iterative procedure continues until the desired girth with a minimum number of cycles is found or no further changes can be made. Note that when an element is being considered, only cycles passing through this element and length no longer than the current girth will be evaluated. For example, if the evaluation criterion is (i) no cycle-4 AND (ii) the number of cycles with length 6, cycles larger than 6 are not evaluated. Thus the computational intensity can be reduced distinctly compared with the original hill-climbing method and at the same

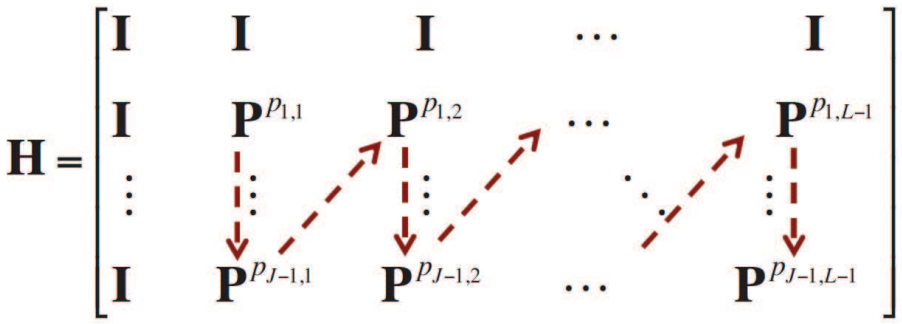


Figure 2.5: Selecting and modifying each element of the base matrix one-by-one.

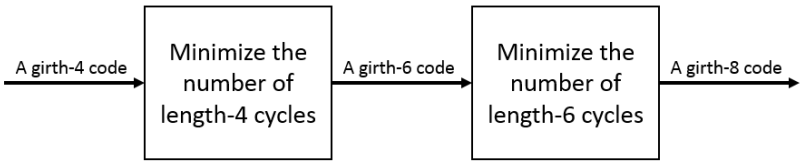


Figure 2.6: A fast searching method for the construction of QC-LDPC codes with large girth.

time a desired girth can be obtained.

2.4 Cyclically-Coupled QC-LDPC Codes

2.4.1 Parity-check Matrix Representation

Define a QC-LDPC subcode $\mathbf{H_c}$ as

$$\mathbf{H_c} = \begin{bmatrix} \mathbf{H}_l & \mathbf{H}_m & \mathbf{H}_r \end{bmatrix} \tag{2.22}$$

where the sub-matrices $\mathbf{H}_l$, $\mathbf{H}_r$ and $\mathbf{H}_m$ consist of circulant permutation matrices (CPMs); $\mathbf{H}_l$ and $\mathbf{H}_r$ are of the same size; and $\mathbf{H}_m$ has the same number of block rows but not necessarily the same number of block columns as the other two sub-matrices. In other words, the base matrices of $\mathbf{H}_l$ and $\mathbf{H}_r$ are of the same size whereas the base matrices of $\mathbf{H}_l$, $\mathbf{H}_r$ and $\mathbf{H}_m$ have the same number of rows. Then a cyclically-coupled QC-LDPC code (CC-QC-LDPC) code can be constructed with k QC-LDPC subcodes and by shifting and coupling $\mathbf{H}_l$ and $\mathbf{H}_r$ from two consecutive subcodes $\mathbf{H_c}$ in a cyclic manner. The parity-check matrix of a CC-QC-LDPC code, denoted by $\mathbf{H}_{cc,k}$, can therefore be given by [30]

$$\mathbf{H}_{cc,k} = \begin{bmatrix} \mathbf{H}_l & \mathbf{H}_m & \mathbf{H}_r & & & & & & \\ & & \mathbf{H}_l & \mathbf{H}_m & \mathbf{H}_r & & & & \\ & & & & \mathbf{H}_l & \cdots & & & \\ & & & & & \ddots & & & \\ & & & & & \cdots & \mathbf{H}_r & & \\ \mathbf{H}_r & & & & & & \mathbf{H}_l & \mathbf{H}_m \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \end{matrix} k \quad \text{rows} \tag{2.23}$$

In general, we assume that in the QC-LDPC subcode $\mathbf{H_c}$

- there are $J \times L$ CPMs of size $z \times z$;
- both $\mathbf{H}_l$ and $\mathbf{H}_r$ contain $J \times W$ sub-matrices;
- $\mathbf{H}_m$ contains $J \times (L - 2W)$ sub-matrices;

where W is defined as the "coupling degree". For example in [30], the parameters $J = 4, L = 28, W = 4$ are used, meaning that $\mathbf{H}_l$ and $\mathbf{H}_r$ consist of 4 block rows and 4 block columns, while $\mathbf{H}_m$ has 4 block rows and 20 block columns.

2.4.2 Architecture of Hardware Decoder

Figure 2.7 shows the overall hardware architecture of a CC-QC-LDPC code decoder (more details can be found in [30]). For every QC-LDPC sub-code $\mathbf{H}_c$, there will be a corresponding QC-LDPC sub-decoder within the CC-QC-LDPC decoder. In the example given in (2.23), there will be k sub-decoders connected in a cyclic way, i.e., the first sub-decoder connected to the second and k-th sub-decoders; the second sub-decoder connected to the first and third sub-decoders; etc. Each sub-decoder is responsible for (i) the decoding of the corresponding sub-code and (ii) exchanging check-to-variable (C2V) and variable-to-check (V2C) messages with its adjacent sub-decoders. With such a decoder design, each sub-decoder can remain relatively simple. The structure also allows parallel operation of the sub-decoders, and the overall throughput is approximately the sum of the throughputs of the sub-decoders.

Considering the structure of a CC-QC-LDPC code, one QC-LDPC sub-code includes three parts — $\mathbf{H}_l$, $\mathbf{H}_m$ and $\mathbf{H}_r$. When updating the V2C messages of $\mathbf{H}_l$ and $\mathbf{H}_r$, the C2V messages of the adjacent sub-codes are also needed. As a result, the V2C messages and C2V messages of $\mathbf{H}_r$ and the C2V messages of $\mathbf{H}_l$ are updated in the current sub-decoder; while the V2C messages of $\mathbf{H}_l$ are updated in the preceding sub-decoder.

In each QC-LDPC sub-decoder, there are two processors, namely variable-node processor (VNP) and check-node processor (CNP), to update the V2C messages and C2V messages, respectively. The architecture also includes a number of random access memories (RAMs) to store the C2V messages, V2C messages and channel messages. These RAMs are connected to the corresponding VNP and/or CNP so that updated messages can be read and written during iterations. Each sub-decoder

includes four types of RAMs and they are described as follows.

Type I RAMs are used to store the messages corresponding to $\mathbf{H}_m$ and $\mathbf{H}_r$. Both C2V and V2C messages of $\mathbf{H}_m$ and $\mathbf{H}_r$ are updated in the current sub-decoder. Moreover during the decoding, C2V and V2C messages will not be required at the same time and hence only one memory location is needed for each check-variable edge. When a V2C message is used by CNP to update C2V message, its memory location will then overwritten by the updated C2V message of the same check-variable edge.

Type II RAMs provide storage for C2V messages of $\mathbf{H}_l$. These C2V messages will be sent to the VNP of the preceding sub-decoder to update the V2C messages of $\mathbf{H}_l$. In other words, this kind of RAMs is read by the preceding sub-decoder but written by the current sub-decoder.

Type III RAMs are used to store the V2C messages corresponding to $\mathbf{H}_l$. The V2C messages updated by the preceding sub-decoder will be written into this kind of RAMs. Contrary to Type II RAMs, this kind of RAMs is read by the current sub-decoder but written by the preceding sub-decoder. To ensure that there is no conflict occurring during the decoding process, two types of RAMs are needed to store the messages of $\mathbf{H}_l$.

The last type of RAMs stores the channel messages. During decoding the same codeword, messages in this kind of RAMs are kept unchanged.

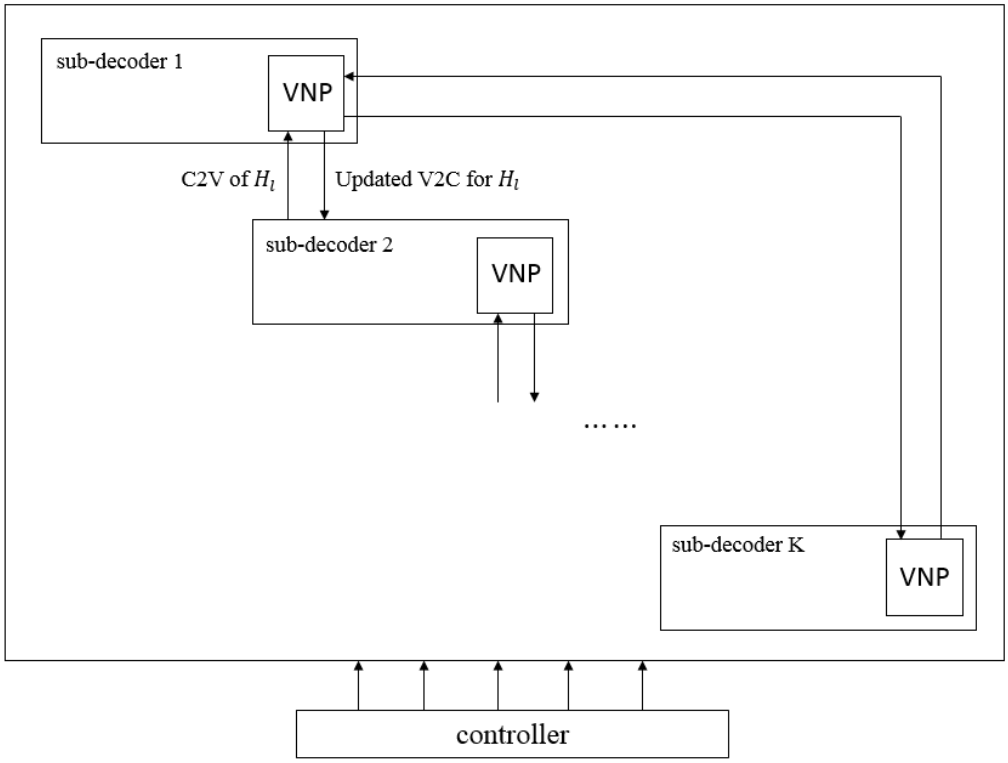


Figure 2.7: Overall architecture of a CC-QC-LDPC hardware decoder

2.5 Summary

This chapter reviews the fundamental concepts of low-density parity-check (LDPC) codes. The definition of LDPC codes have been first introduced and two methods to represent the LDPC codes, namely parity-check matrices and Tanner graphs (bipartite graphs), have been presented. The belief propagation (BP) algorithm which is used to decode LDPC codes has been briefly described. An important class of LDPC codes, namely quasi-cyclic LDPC (QC-LDPC) codes, has been introduced and two methods for constructing QC-LDPC codes with large girth — hill-climbing and fast hill-climbing — are introduced. Finally, the structure of cyclically-coupled

QC-LDPC (CC-QC-LDPC) codes and the hardware decoder architecture are shown. In the next chapter, we will begin presenting our own works.

Chapter 3

New Types of Cyclically-Coupled LDPC Block Codes

In this chapter, we investigate the performance of CC-QC-LDPC codes when the subcode structure is changed. Recall that a CC-QC-LDPC code can be represented by [30]

$$\mathbf{H}_{cc,k} = \begin{bmatrix} \mathbf{H}_l & \mathbf{H}_m & \mathbf{H}_r & & & & & & \\ & & \mathbf{H}_l & \mathbf{H}_m & \mathbf{H}_r & & & & \\ & & & & \mathbf{H}_l & \cdots & & & \\ & & & & & \ddots & & & \\ & & & & & \cdots & \mathbf{H}_r & & \\ \mathbf{H}_r & & & & & & \mathbf{H}_l & \mathbf{H}_m \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ k \quad \text{rows} \end{matrix} \tag{3.1}$$

and $\mathbf{H}_c = [\mathbf{H}_l \quad \mathbf{H}_m \quad \mathbf{H}_r]$ denotes a QC-LDPC subcode. Moreover, in the QC-LDPC subcode $\mathbf{H_c}$,

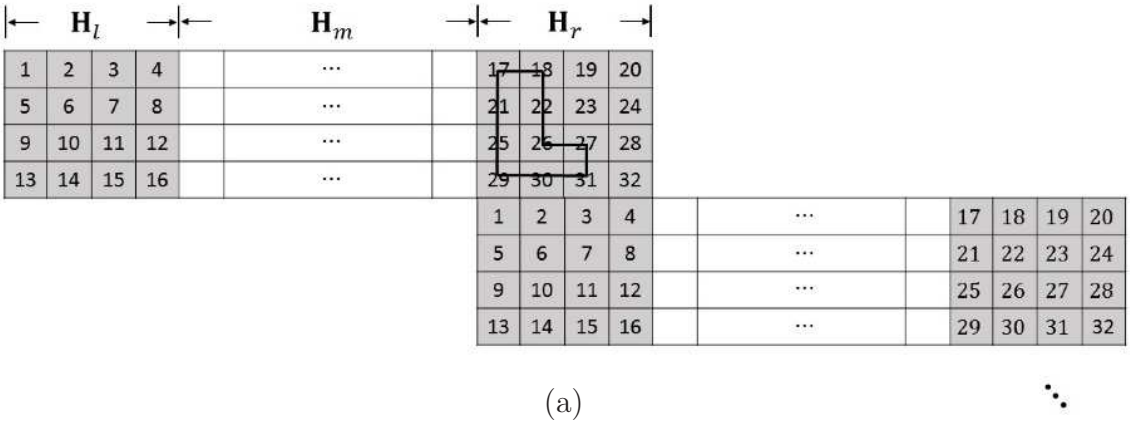
- there are $J \times L$ circulant permutation matrices (CPMs) each of size $z \times z$;

- both $\mathbf{H}_l$ and $\mathbf{H}_r$ contain $J \times W$ CPMs; and
- $\mathbf{H}_m$ contains $J \times (L - 2W)$ CPMs.

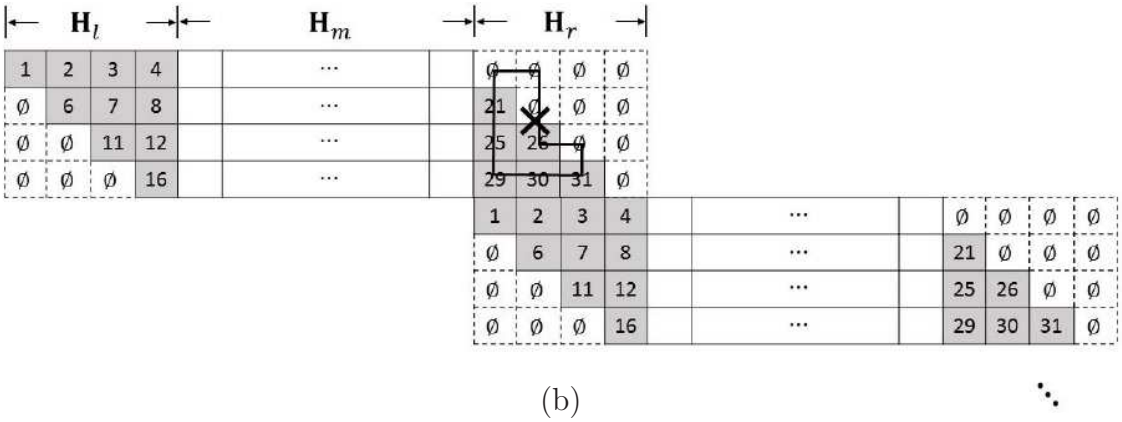
In our first study, we replace the regular QC-LDPC subcodes of a CC-QC-LDPC code with irregular ones by making some of the circulant permutation matrices (CPMs) to be zero matrices [33]. In particular, we make the components $\mathbf{H}_l$ and $\mathbf{H}_r$ to be triangular, as shown in Fig. 3.1(b) and (c). With such a modification, it will become easier to remove short cycles and to construct codes with a larger girth. In our second study, we replace the circulant permutation matrices (CPMs) in the regular QC-LDPC subcodes with random-permutation matrices, forming the random-permutation-matrix-based CC-LDPC codes. In both cases, we will evaluate not only the bit error rate performance of the resultant CC-LDPC codes, but also the decoder complexity.

3.1 CC-QC-LDPC Codes Based on Irregular QC-LDPC Subcodes

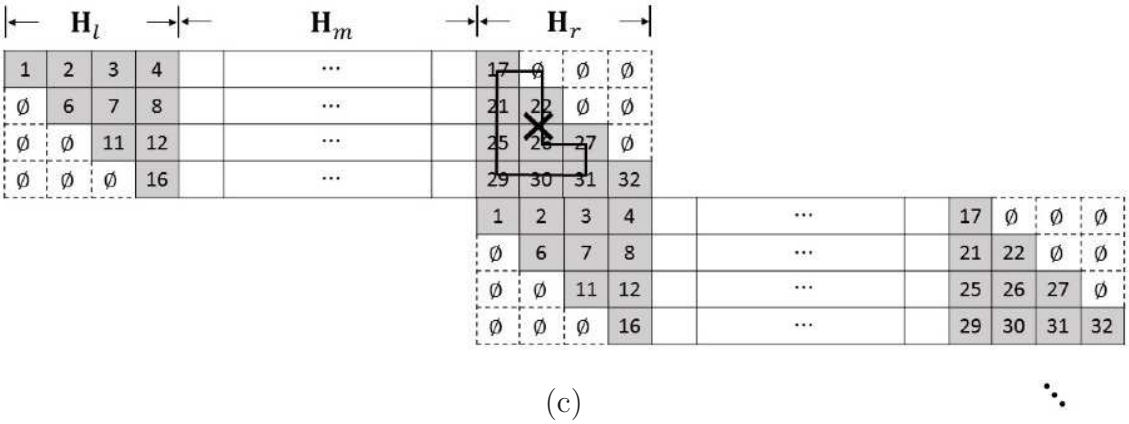
Assuming $J = 4$ and $W = 4$, Fig. 3.1(a) illustrates a *full* CC-QC-LDPC code. Here all the sub-matrices are non-zero CPMs and the fast hill-climbing described in chapter 2.3.2 [30] can be applied to select the CPMs such that a desired girth can be achieved. Note that if we are to achieve a girth of 8, only cycles formed within two subcodes need to be considered. When $L = 28$ and $k = 4$, a girth-8 CC-QC-LDPC code can be found with $z = 1024$. When z is reduced to 512, 256 and 128, however, we can only find CC-QC-LDPC codes with a girth 6 due to substantially smaller



(a)



(b)



(c)

Figure 3.1: CC-QC-LDPC codes. (a) Full (b) Type-I and (c) Type-II

search spaces.

3.1.1 Type-I and Type-II CC-QC-LDPC Codes

With an aim to improving the girth, we propose two special types of CC-QC-LDPC codes which contain zero sub-matrices [33]. They are shown in Figs. 3.1(b) and 3.1(c), respectively. Type-I CC-QC-LDPC codes are characterized with $\mathbf{H}_l$ being upper triangular and $\mathbf{H}_r$ being *strictly* lower triangular; and Type-II CC-QC-LDPC codes are characterized with $\mathbf{H}_l$ being upper triangular and $\mathbf{H}_r$ being lower triangular. By converting some sub-matrices in Fig. 3.1(a) to zero submatrices, we form the Type-I and Types-II CC-QC-LDPC codes shown in Figures 3.1(b) and (c). Because of the conversion, cycles passing through these removed submatrices can be eliminated. One such removal of cycle-6 has been demonstrated in Fig. 3.1. As a result, CC-QC-LDPC codes with girth 8 can be easily designed with a smaller value of z, i.e., shorter codelength. Note that by converting the submatrices in $\mathbf{H}_l$ and $\mathbf{H}_r$ instead of $\mathbf{H}_m$ to zero submatrices, we can maintain column weights of at least 4.

3.1.2 Simulation Results

We compare CC-QC-LDPC codes with the following parameters: $J = 4, L = 28, W = 4, k = 4$ and a code rate of 5/6. For all cases, we optimize the girth of the CC-QC-LDPC codes using the fast hill-climbing method [25], the block diagram of which has been shown in Fig. 2.6. Note that in the optimizing process, it is adequate to consider cycles formed by two consecutive subcodes only, i.e., cycles found in Fig. 3.1. Moreover, all decoders have been implemented on an Altera Stratix

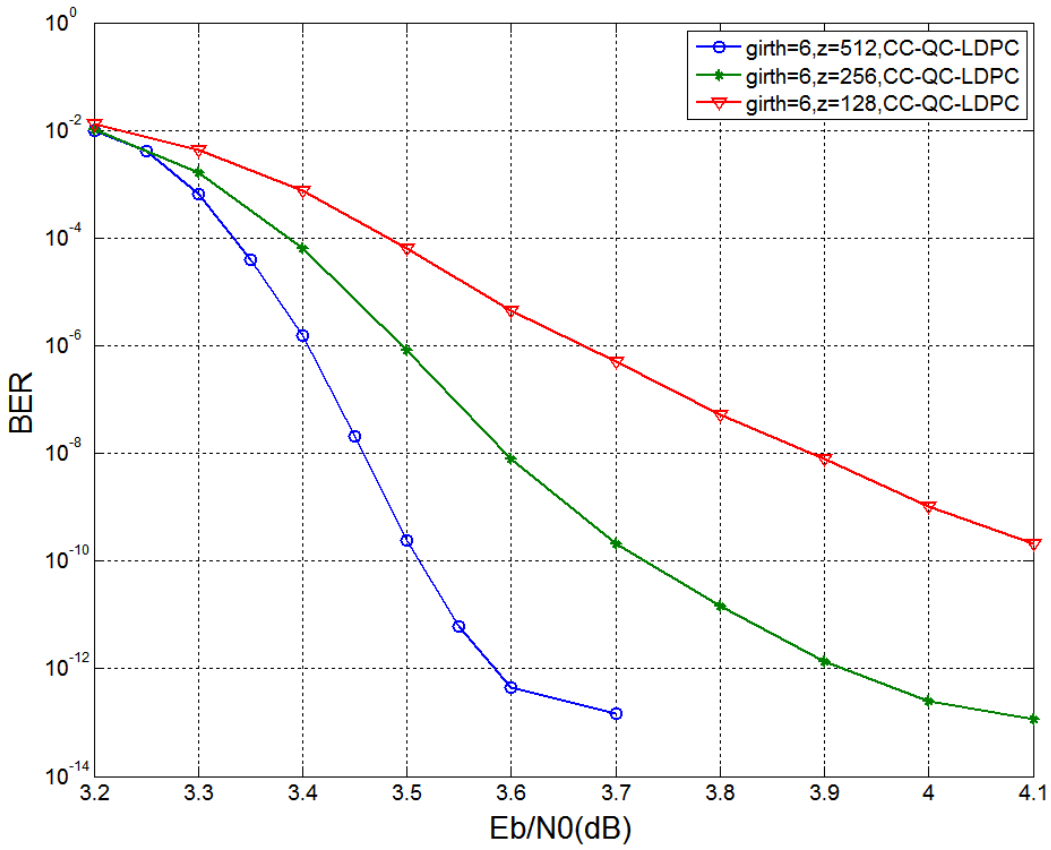


Figure 3.2: Bit error rate of *full* CC-QC-LDPC codes with $z = 512, 256$ and 128.

IV EP4SE530H35C2 FPGA. Also 4-bit quantization and 10 decoding iterations are used.

First, we consider the *full* CC-QC-LDPC codes with $z = 512, 256$ and 128. The optimized codes all have the same girth of 6. In Fig. 3.2, we present the bit error rate (BER) results found by FPGA implementation. We can observe that as the code length increases, the BER performance improves. However, error floor exists in all cases.

Next, we attempt to construct Type-I and Type-II CC-QC-LDPC codes with girth 8. The minimum values of z that achieve girth 8 are 293 and 331, respectively,

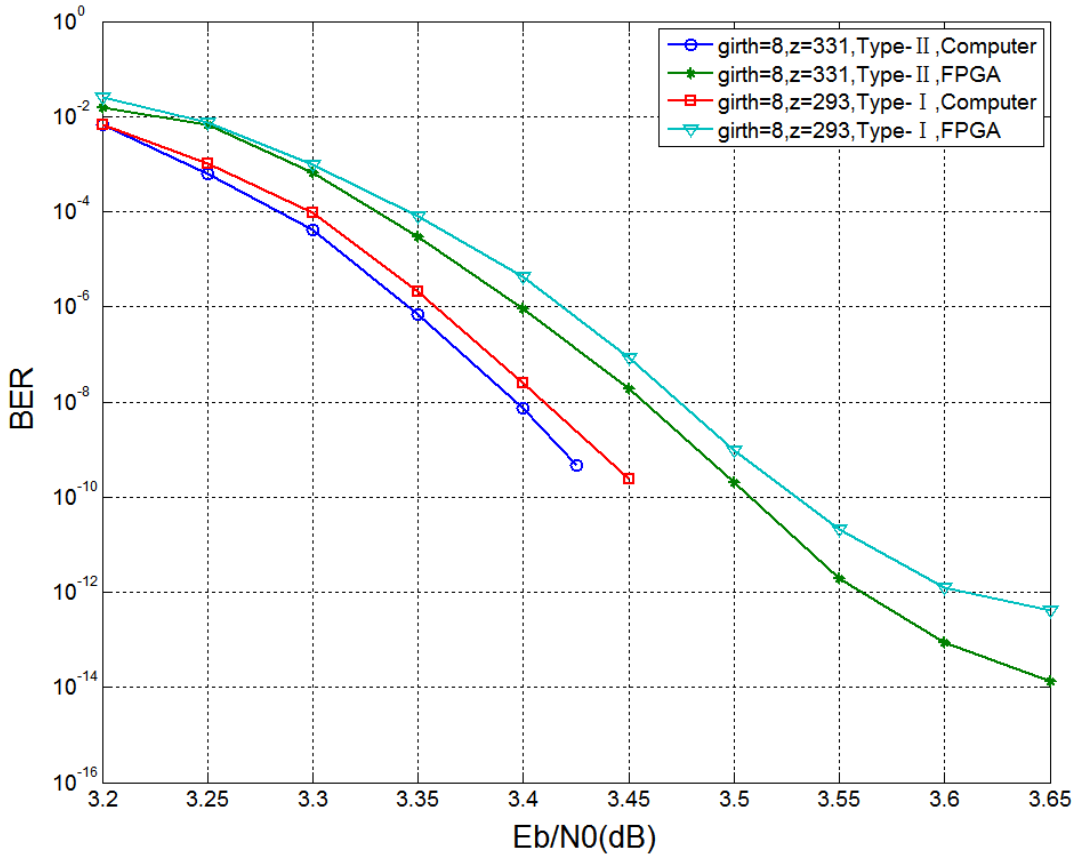


Figure 3.3: Bit error rates of Type-I and Type-II CC-QC-LDPC codes. $z = 293$ and 331, respectively, for Type-I and Type-II codes.

for Type-I and Type-II codes. In Figure 3.3, we further present the BER results found by computer simulation and FPGA implementation. Comparing the computer simulation and the FPGA implementation results, we observe that the BER degrades by about 0.05 dB. The main reason is that computer simulation uses floating-point computations while the FPGA implementation makes use of fixed-point calculations. Moreover, Type-II CC-QC-LDPC code slightly outperforms Type-I code in terms of BER and has a lower error floor. One possible explanation is that the column weights of Type-I code are all 4 but the column weights of Type-II code can be 4 or

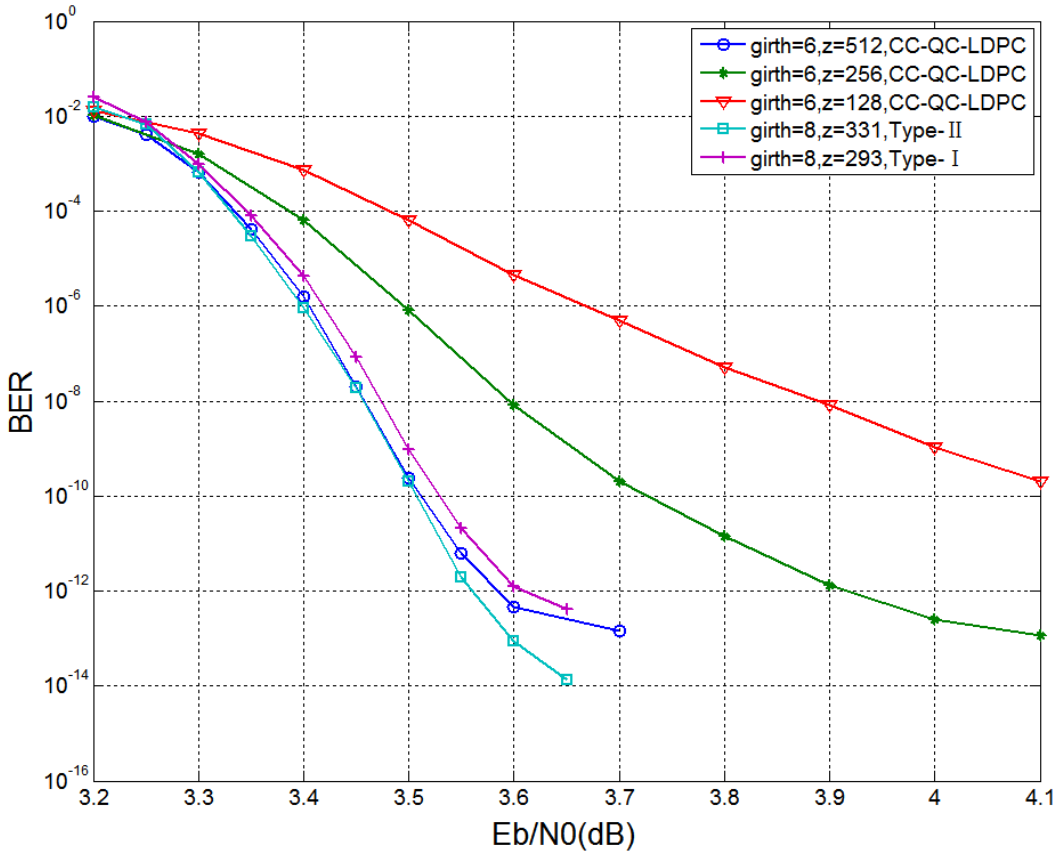


Figure 3.4: Bit error rate of all three types of CC-QC-LDPC codes.

5.

Finally, we compare the FPGA experimental BERs of all three different types of CC-QC-LDPC codes in Figure 3.4. We find that Type-I and Type-II CC-QC-LDPC codes outperform full CC-QC-LDPC codes with $z = 128$ and 256, and have similar BER performances as full CC-QC-LDPC codes with $z = 512$. In particular, Type-I code (with $z = 293$) though having a shorter codelength can achieve very similar BER performance compared with the full CC-QC-LDPC code with $z = 512$. Type-II code (with $z = 331$) can even accomplish a lower error floor compared with the full CC-QC-LDPC code with $z = 512$. We also look into the complexity of the

different decoders. The overall architectures of the three CC-QC-LDPC decoders are similar except that RAM blocks can be saved when CPMs are replaced by all-zero matrices. Thus the complexity becomes lower when more sub-matrices in a full CC-QC-LDPC code are converted to zero sub-matrices. Therefore the decoder complexity of the Type-I CC-QC-LDPC code is the lowest, and that of the Type-II CC-QC-LDPC code is lower than that of the full CC-QC-LDPC code.

3.2 Random-Permutation-Matrix-Based Cyclically-Coupled LDPC Codes

3.2.1 Code Structure and Construction

We again consider the same sub-code $\mathbf{H}_s$ of CC-QC-LDPC codes in the previous section. In $\mathbf{H}_s$, both $\mathbf{H}_l$ and $\mathbf{H}_r$ contain 4×4 sub-matrices and $\mathbf{H}_m$ contains 4×20 sub-matrices. However, instead of CPMs, random permutation matrices are used in the sub-matrices [34]. We can also view the modified $\mathbf{H}_s$ as a protograph-based LDPC code [35] with a base matrix $\mathbf{B}_s$ given by

$$\mathbf{B}_s = [\mathbf{1}_{4\times4} \;\; \mathbf{1}_{4\times20} \;\; \mathbf{1}_{4\times4}] \tag{3.2}$$

where $\mathbf{1}_{4\times4}$ and $\mathbf{1}_{4\times20}$ represent the 4×4 and 4×20 all-one matrices, respectively. The corresponding CC-LDPC codes constructed from $\mathbf{B}_s$, which we called random-permutation-matrix-based CC-LDPC (RP-CC-LDPC) codes, are therefore represented by the base matrix $\mathbf{B}_{cc}$

$$\mathbf{B}_{cc} = \begin{bmatrix} \mathbf{1}_{4\times4} & \mathbf{1}_{4\times20} & \mathbf{1}_{4\times4} & \mathbf{0} & \mathbf{0} & ... & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1}_{4\times4} & \mathbf{1}_{4\times20} & \mathbf{1}_{4\times4} & ... & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{1}_{4\times4} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & ... & \mathbf{1}_{4\times4} & \mathbf{1}_{4\times20} \end{bmatrix}. \tag{3.3}$$

By replacing each 1 in $\mathbf{B}_s$ by a random permutation matrix of size $z \times z$ and then substituting the results into $\mathbf{B}_{cc}$, RP-CC-LDPC codes are obtained.

3.2.2 Accessing the V2C and C2V Messages Stored in RAMs

For CC-QC-LDPC codes, the sub-matrices are all circulant permutation matrices, which means the data corresponding to consecutive columns/rows in the CPM are stored next to each other in the RAMs and can be read/written in a successive manner. Figure 3.5 shows the method of specifying the addresses of RAM spaces when decoding the CC-QC-LDPC codes. Here we use counters to generate sets of consecutive numbers as the addresses of RAM. At clk1, the counter provides a signal of '1', this signal is directly connected to the interface which receives address information of a RAM. At the next clock, the signal change to '2' and then the signal will point to the next location in RAM. For $z \times z$ sub-matrices, we need z counters since the start address of a sub-matrix can be any number between 1 and z.

When implementing the decoder of RC-CC-LDPC codes on FPGA, specifying the addresses of RAMs which store the check-to-variable (C2V) and variable-to-check (V2C) messages becomes more complicated. It is because the sub-matrices are all randomly generated permutation matrices. The positions of the '1's in these

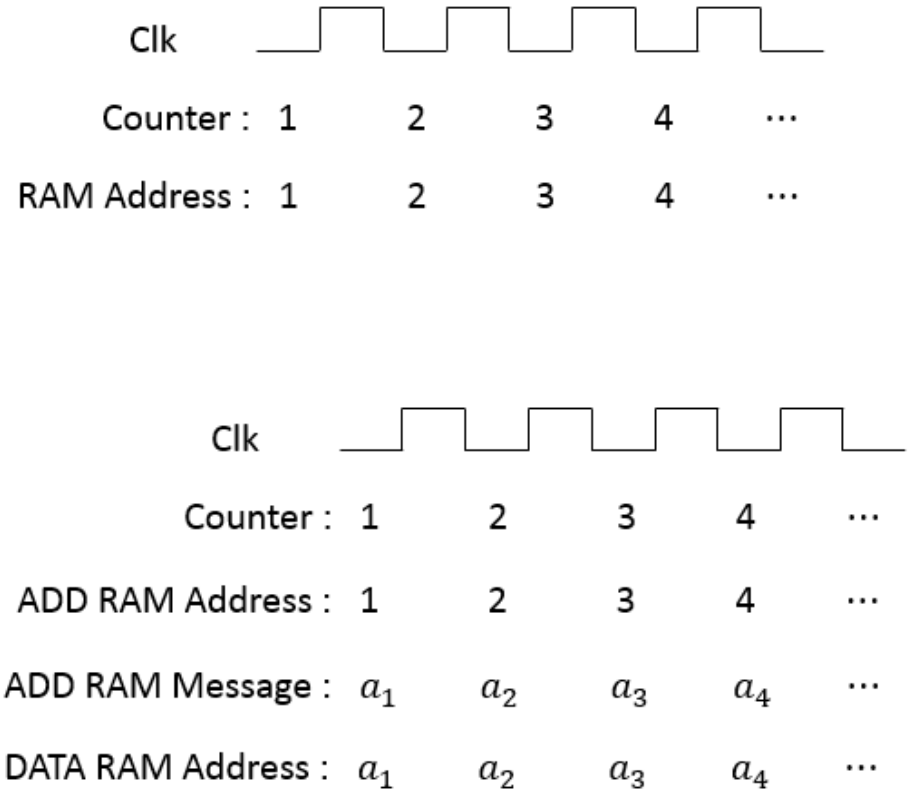


Figure 3.5: Access the memory locations of the RAMs. Top: CC-QC-LDPC code; Bottom: RP-CC-LDPC code.

sub-matrices do not possess the regularity of CPMs. As a result, we need to store the positions of the '1's and use them to locate the addresses in a RAM where we should read or write messages. Fig. 3.5 also shows the method of specifying the addresses of RAM locations when decoding RC-CC-LDPC codes. In Fig. 3.5, 'ADD

RAM' is used to store position information and 'DATA RAM' is used to store the C2V and V2C messages. We use counters to point to consecutive addresses of the ADD RAM and read the associated contents in consecutive clock cycles. Based on the contents which in fact indicate the addresses of DATA RAM, we can locate where the C2V and V2C messages should be read/written. For example at clk1, the counter provides a signal of '1' and hence the content a_1 in the first location of the ADD RAM is read. Here $a_1 \in \{1, 2, \ldots, z\}$ denotes the column position of the '1' in the first row of the random permutation sub-matrix. According to a_1, the correct location in the DATA RAM can be accessed.

Suppose the degree of parallelism is G, which means G rows of messages in the parity-check matrix are updated at the same time. Since we can only write to or read from one location of each RAM in each clock cycle, we will have to use G RAMs to store the messages for each sub-matrix. (When the size of each sub-matrix is $z \times z$, the depth of each RAM is $D = z/G$.) In the case of CC-QC-LDPC codes, due to their regular structure, we can ensure that the G messages are coming from different RAMs when the messages are properly arranged. However, it is not possible to ensure conflict of RAM access will not occur in the case RP-CC-LDPC codes.

Figure 3.6 shows an example illustrating the conflict of RAM access, where $z = 9$, $G = 3$ and $D = z/G = 3$. Each '1' in the matrices represents an edge between a variable node and a check node, and there are two kinds of messages in each edge — C2V and V2C message. Since we only use one memory location to store both kinds of messages for most of the matrices, we use $C_j \leftrightarrow V_l$ to define the message in a RAM location. Supposing we need to update the messages of

		v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
Matrix 1	c_1		1							
	c_2				1					
	c_3			1						
	c_4							1		
	c_5									1
	c_6						1			
	c_7	1								
	c_8					1				
	c_9								1	
Matrix 2	c_{10}						1			
	c_{11}	1								
	c_{12}								1	
	c_{13}		1							
	c_{14}					1				
	c_{15}			1						
	c_{16}				1					
	c_{17}									1
	c_{18}							1		

Matrix 1

RAM 1	RAM 2	RAM 3
$c_1 \leftrightarrow v_2$	$c_2 \leftrightarrow v_4$	$c_3 \leftrightarrow v_3$
$c_4 \leftrightarrow v_7$	$c_5 \leftrightarrow v_9$	$c_6 \leftrightarrow v_6$
$c_7 \leftrightarrow v_1$	$c_8 \leftrightarrow v_5$	$c_9 \leftrightarrow v_8$

Matrix 2

RAM 4	RAM 5	RAM 6
$c_{10} \leftrightarrow v_6$	$c_{11} \leftrightarrow v_1$	$c_{12} \leftrightarrow v_8$
$c_{13} \leftrightarrow v_2$	$c_{14} \leftrightarrow v_5$	$c_{15} \leftrightarrow v_3$
$c_{16} \leftrightarrow v_4$	$c_{17} \leftrightarrow v_9$	$c_{18} \leftrightarrow v_7$

Figure 3.6: Illustration of conflict of RAM access

the first 3 rows corresponding to C_1, C_2 and C_3, we need to access not only the memory locations $C_1 \leftrightarrow V_2$, $C_2 \leftrightarrow V_4$ and $C_3 \leftrightarrow V_3$ at the same time, but also all memory locations that corresponding to the related columns V_2, V_4 and V_3. In other words, the memory locations $C_{13} \leftrightarrow V_2$, $C_{16} \leftrightarrow V_4$ and $C_{15} \leftrightarrow V_3$ have to be accessed simultaneously as we access $C_1 \leftrightarrow V_2$, $C_2 \leftrightarrow V_4$ and $C_3 \leftrightarrow V_3$. However, it is obvious that the locations $C_{13} \leftrightarrow V_2$ and $C_{16} \leftrightarrow V_4$ are in the same RAM, i.e., RAM 4. Thus a conflict of RAM access will occur. Note that we are using only two sub-matrices as an example in Fig. 3.6. In practice, tens of sub-matrices will be involved depending on the row weight and column weight, i.e., the numbers of '1's in each row and each column. Thus conflict of RAM access is almost guaranteed to occur unless the sub-matrices have a regular structure. In other words, *for each sub-code in a RP-CC-LDPC code, we can either implement (i) no parallelism* $G = 1$*, i.e., process one row (check node) at a time for each sub-code or (ii) full parallelism* $G = z$*, i.e., process* z *rows (check nodes) at a time for each sub-code.* In the latter case, a lot of hardware resources will be required and is not desirable.

3.2.3 Results

We consider a RP-CC-LDPC code with $J = 4$, $L = 28$, $W = 4$, $K = 4$ and $z = 1024$. As mentioned earlier, permutation matrices are randomly generated for the RP-CC-LDPC code. We compare the bit error performance and hardware complexity of our proposed RP-CC-LDPC code with other LDPC codes. All codes have the same code rate 5/6 and code length 98304, and are simulated under an additive white Gaussian noise (AWGN) channel. All our results are generated by FPGA implementation of the corresponding decoders. Moreover, 10 decoding iterations and 4-bit quantization

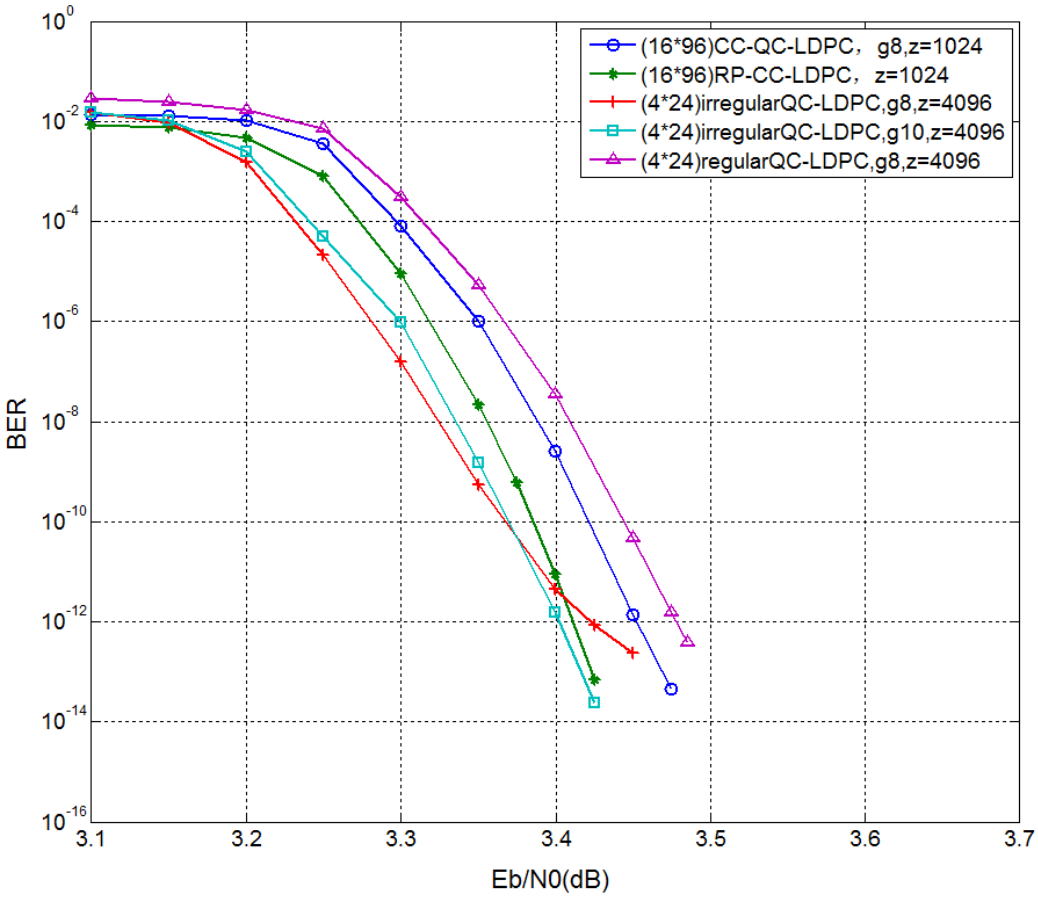


Figure 3.7: The bit error rate (BER) comparison of different codes. All the results are obtained from FPGA simulations under an AWGN channel and 4-bit quantization.

are used.

3.2.3.1 Comparison with CC-QC-LDPC Code

First, we compare our RP-CC-LDPC code with the CC-QC-LDPC code reported in [30]. Fig. 3.7 shows that the proposed RP-CC-LDPC code outperforms the CC-QC-LDPC code in terms of bit error rate (BER). In particular, RP-CC-LDPC code

Table 3.1: Hardware Information of the Decoder Implementations. Code A: 4×24 regular QC-LDPC code, girth=8, $z = 4096$; Code B: 4×24 irregular QC-LDPC code, girth=8, $z = 4096$; Code C: 4×24 irregular QC-LDPC code, girth=10, $z = 4096$; Code D:16×96 CC-QC-LDPC code, girth=8, $z = 1024$; Code E: 16×96 RP-CC-LDPC code, $z = 1024$.

Code	A	B	C	D	E
Parallelism degree	32	32	32	32	4
ALUTs	65,178	56,137	55,990	70,342	31,660
Registers	45,336	41,740	41,740	43,801	13,564
Memory bits	1,769,372	1,474,560	1,474,560	2,359,296	4,297,289
Clock	100 MHz	100 MHz	100 MHz	100 MHz	100 MHz
Throughput	1.55 Gbps	1.55 Gbps	1.55 Gbps	1.55 Gbps	0.182 Gbps

outperforms CC-QC-LDPC code by 0.05 dB at a BER of 10^{-13}. No error floor is also observed above this BER level.

Table 3.1 compares the complexity and throughput of the two decoders. The CC-QC-LDPC code and RP-CC-LDPC code are represented by Code D and Code E, respectively. As we can observe, the degree of parallelism of the RP-CC-LDPC code is reduced by 8 times compared with that of the CC-QC-LDPC code. (Note that the degree of parallelism of the RP-CC-LDPC code equals 4 because $k = 4$ subcodes are processed at the same time.) The main reason is the random permutation matrices being used in the RP-CC-LDPC code. Consequently, the same ratio in terms of decoder throughput reduction is observed. We can also see a substantial increase of memory required in the RP-CC-LDPC decoder. They are used to store the entries of the permutation matrices which are random in general.

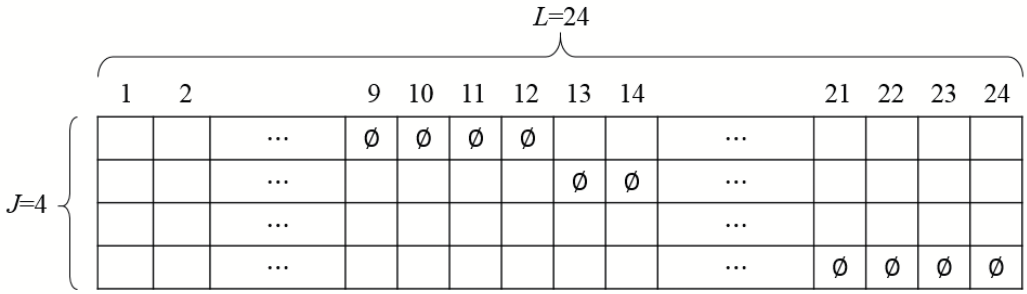


Figure 3.8: Structure of the irregular QC-LDPC code.

3.2.3.2 Comparison with Regular and Irregular QC-LDPC Codes

We construct a 4×24 QC-LDPC code with $z = 4096$. Using the fast hill-climbing algorithm in [25] and illustrated in Fig. 2.6, we can eliminate both cycle-4 and cycle-6 but not cycle-8. The largest girth that we can obtain is therefore 8. The resultant 4×24 QC-LDPC code is denoted as Code A in Table 3.1. In order to achieve a larger girth, we assign 4 all-zero sub-matrices for each row in a 4×24 QC-LDPC code. Thus, all block rows have a weight of 20, 16 block columns have a weight of 3, and 8 block columns have a weight of 4. The structure of the resultant irregular QC-LDPC code is shown in Fig. 3.8. Each ϕ in Fig. 3.8 represents an all-zero sub-matrix. Again we apply the fast hill-climbing algorithm and we are able to eliminate all cycle-4, cycle-6 and cycle-8. A girth-10 irregular QC-LDPC code is hence obtained and is denoted as Code C in Table 3.1. A girth-8 irregular QC-LDPC code, denoted as Code B in Table 3.1, is also generated and used for comparison.

Referring to Fig. 3.7, the BER curves show that the regular QC-LDPC code performs the worst among all codes while the irregular ones can outperform the

proposed RP-CC-LDPC code in the low E_b/N_0 region. However, an error floor occurs at around 10^{-13} BER for the irregular QC-LDPC code with girth 8. At $E_b/N_0 = 3.425$ dB, the BER performances of irregular QC-LDPC code with girth 10 and RP-CC-LDPC code are very similar. The BER curve trends seem to forecast that the irregular QC-LDPC code will be outperformed by RP-CC-LDPC code beyond $E_b/N_0 = 3.45$ dB. That needs to be verified though. Table 3.1 compares the complexity and throughput of the decoders. Again, the RP-CC-LDPC code is at a disadvantageous position in terms of decoder complexity and throughput.

3.3 Summary

In this chapter, we have firstly proposed two new types of CC-QC-LDPC codes, Type-I and Type-II, which have a shorter code length and achieve a larger girth compared with the full CC-QC-LDPC codes. We also have demonstrated their superior BER performance and low error floors even with short code lengths. Their decoder complexities are also lower. We have also proposed a random-permutation-matrix-based cyclically-coupled LDPC (RP-CC-LDPC) code. We have compared its performance with the CC-QC-LDPC code, regular and irregular QC-LDPC codes using FPGA simulations. Results show that the BER performance of the RP-CC-LDPC code is comparable to and can possibly exceed that of other codes. However, decoder complexity and throughput are major issues of the RP-CC-LDPC code, which makes use of relatively more complex random permutation matrices instead of simple circulant permutation matrices. In the next chapter, we will propose a new type of LDPC code which has a "semi-regular" structure. We will investigate its BER performance as well as decoder complexity.

Chapter 4

Tree-Permutation-Matrix Based LDPC Codes

In the previous chapter, we have shown that when random permutation matrices are used as parity-check sub-matrices of LDPC codes, decoder complexity and throughput become the major issues. In the other extreme, using circulant permutation matrices (CPMs) as sub-matrices allows a simple and high-throughput decoder structure to be implemented. In this chapter, we propose a new type of matrices called tree-permutation-matrices (TPM) and apply them in the construction of LDPC codes [36]. TPMs are not as regular as CPMs but also not as random as random permutation matrices. Thus we call TPMs as *semi-regular* matrices. Like QC-LDPC codes, TPM-based LDPC codes allow decoding operations to be conducted in parallel and can achieve a high decoding throughput. Moreover, there are more choices of TPMs compared with CPMs for a given matrix size. Hence, a larger girth can potentially be achieved for TPM-based LDPC codes.

4.1 Tree Permutation Matrices

We first make use of 2×2 permutation matrices to illustrate how to construct TPMs. There are only two different 2×2 permutation matrices — the identity 2×2 matrix (denoted by $\mathbf{I}_{2\times 2}$) and the anti-diagonal 2×2 matrix (denoted by $\tilde{\mathbf{I}}_{2\times 2}$). To construct TPMs, we refer to Fig. 4.1 and start with a matrix that contains only a single element "1" as shown at the top.

1. To construct Layer-1 TPMs, we replace the '1' at the top with either $\mathbf{I}_{2\times 2}$ ($a_{1,0} = 0$) or $\tilde{\mathbf{I}}_{2\times 2}$ ($a_{1,0} = 1$). There are only two possibilities and hence $N_1 = 2$ possible Layer-1 TPMs, the size of which are 2×2.

2. To construct Layer-2 TPMs, the 1's in each Layer-1 TPMs are replaced with either $\mathbf{I}_{2\times 2}$ or $\tilde{\mathbf{I}}_{2\times 2}$. For each Layer-1 TPM, there are $2^2 = 4$ possible choices ($a_{2,0}a_{2,1} = 00, 10, 01, 11$). Hence the total number of Layer-2 TPMs equals $N_2 = 2^2 \times N_1 = 2^3 = 8$. The size of each Layer-2 TPM is $2^2 \times 2^2$.

3. To construct Layer-3 TPMs, the 1's in each Layer-2 TPMs are replaced with either $\mathbf{I}_{2\times 2}$ or $\tilde{\mathbf{I}}_{2\times 2}$. For each Layer-2 TPM, there are $2^{2^2} = 16$ possible choices. Hence the total number of Layer-3 TPMs equals $N_3 = 2^{2^2} \times N_2 = 2^7 = 128$. The size of each Layer-3 TPM is $2^3 \times 2^3$.

4. TPMs in subsequent layers are generated in a similar manner. It can be easily shown that at Layer M, there are 2^{2^M-1} different TPMs, the size of which are $2^M \times 2^M$. The tree expansion is illustrated in Fig. 4.2. We also denote P_2^M as a Layer-M TPM matrix formed by $\mathbf{I}_{2\times 2}$ or $\tilde{\mathbf{I}}_{2\times 2}$.

In our study, we consider only TPMs formed by $\mathbf{I}_{2\times 2}$ or $\tilde{\mathbf{I}}_{2\times 2}$. In general, TPMs can be constructed by $Z \times Z$ permutation matrices. In Figure 4.3, we further show the expansion tree if every '1' in the previous layer is replaced by a $Z \times Z$ permutation matrix. At Layer M, each TPM will have a size of $Z^M \times Z^M$.

Referring to Fig. 4.2, a Layer-M TPM formed by expanding 2×2 permutation matrices repeatedly can be defined by the "tree" vector

$$\mathbf{V} = (a_{1,0}, a_{2,0}, a_{2,1}, a_{3,0}, a_{3,1}, a_{3,2}, a_{3,3}, \ldots, a_{M,0}, a_{M,1}, \ldots, a_{M,2^{2^M-1}-1})$$

in which each element is either "0" or "1". When the element assumes the value "0" and "1", it represents an expansion with $\mathbf{I}_{2\times 2}$ and $\tilde{\mathbf{I}}_{2\times 2}$. respectively. It can be readily shown that

- *a* $2^M \times 2^M$ *identity matrix will be associated with the all-zero tree vector with* $2^M - 1$ *elements; and*
- *a* $2^M \times 2^M$ *TPM has a fixed column*[1] *if and only if at least one branch transversing from the top to the bottom of the tree assumes all "0" values, e.g., the branch* $a_{1,0} = a_{2,0} = a_{3,0} = \cdots = a_{M,0} = 0$.

4.1.1 Multiplication of TPMs

It can be easily verified that TPMs formed by 2×2 permutation matrices are closed under multiplication. One way is to consider all possible multiplications and then build a look-up table to simplify the computation. The table can be large depending

[1] A permutation matrix has a fixed column (or row) if and only if it overlaps with the identity matrix in at least one column (or row) [1].

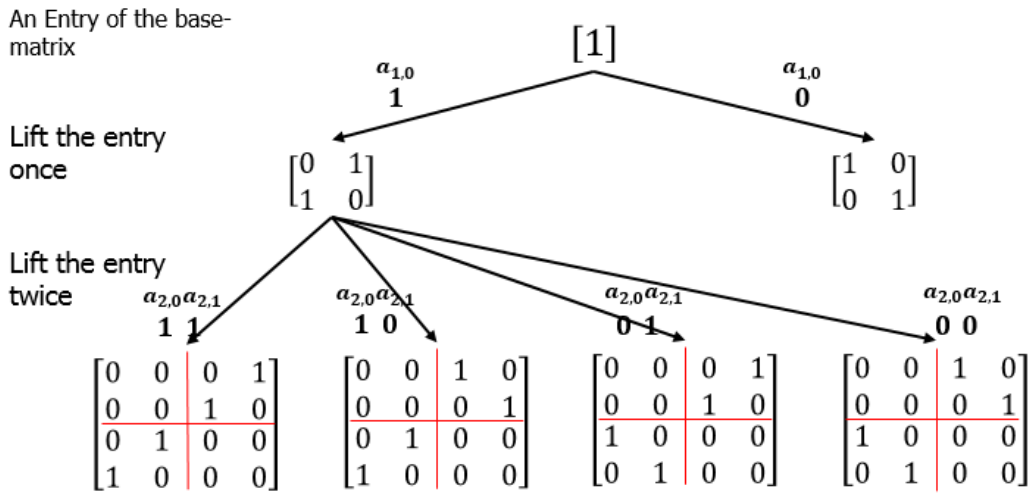


Figure 4.1: Forming tree-permutation matrices by replacing each '1' in the upper layer with a 2×2 permutation matrix.

on the value of M. On the other hand, each TPM is represented by a tree vector. To find the product of two TPMs, we can make use of their corresponding tree vectors, perform appropriate module-2 additions, and arrive at a new tree vector that represents the TPMs' product. The procedures to compute the product of two TPMs with size $2^M \times 2^M$ is shown in Algorithm 2. In Algorithm 2, $\mathbf{P}_A$ and $\mathbf{P}_B$ denote the tree vectors of the two TPMs to be multiplied. $\mathbf{V}$ is a vector used to locate the corresponding elements in the two tree vectors. $\mathbf{P}_M$ represents the tree vector of the product of the two TPMs.

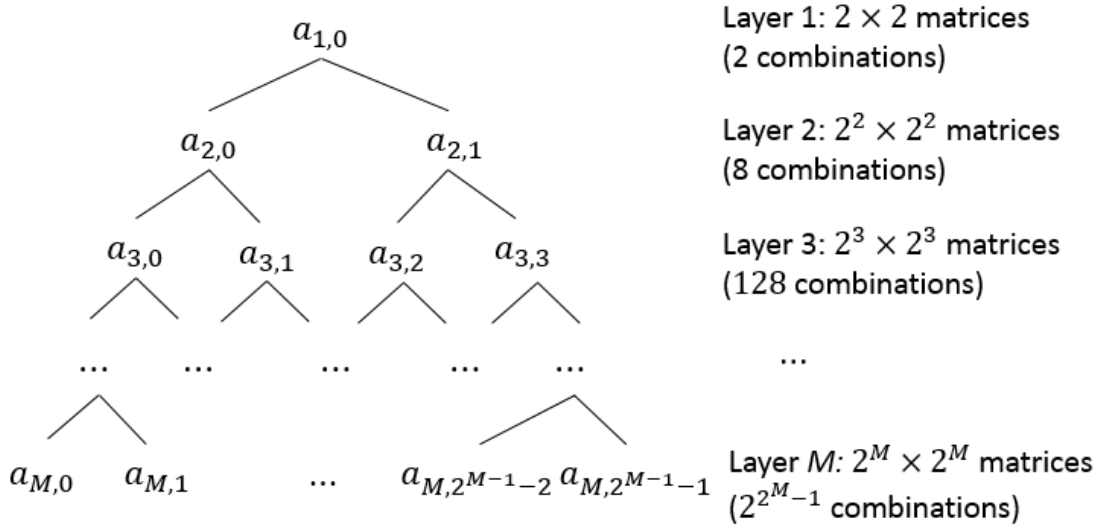


Figure 4.2: The full tree representation of P_2^M.

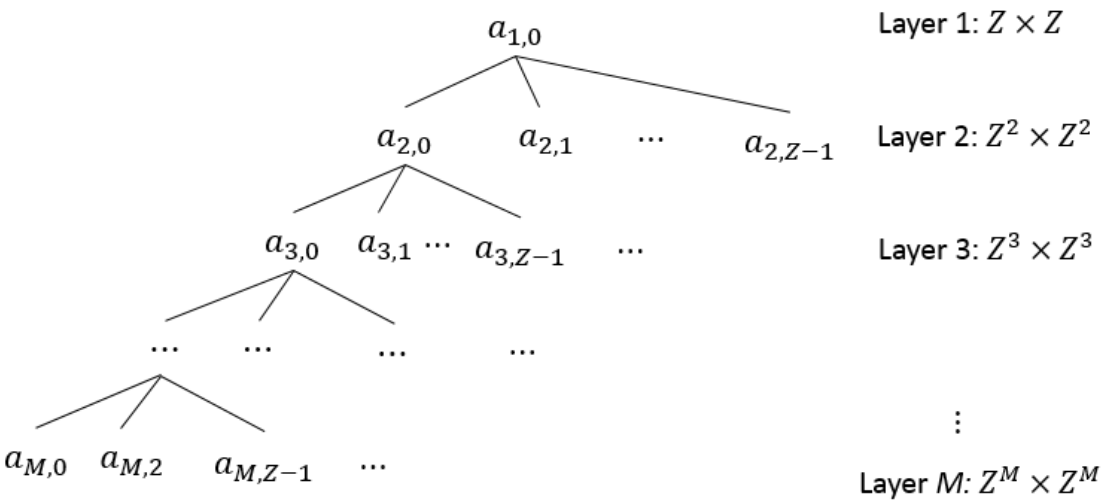


Figure 4.3: The general tree representation

Algorithm 2 Multiplication of two P_2^M TPMs

1: $\mathbf{P}_M[0] \leftarrow \mathbf{P}_A[0] \oplus \mathbf{P}_B[0], \mathbf{V}[0] \leftarrow 0$
2: **for** $node = 1; node < 2^M - 2; node++$ **do**
3: $\quad parent \leftarrow (node - 1)/2$
4: $\quad$ **if** $node$ is odd **then**
5: $\quad\quad \mathbf{V}[node] \leftarrow \mathbf{V}[parent] \times 2 + \mathbf{P}_A[parent]$
6: $\quad$ **else**
7: $\quad\quad \mathbf{V}[node] \leftarrow \mathbf{V}[parent] \times 2 - \mathbf{P}_A[parent]$
8: $\quad \mathbf{P}_M[node] \leftarrow \mathbf{P}_A[node] \oplus \mathbf{P}_B[node + \mathbf{V}[node]]$

Algorithm 3 Transpose of a P_2^M TPM

1: $\mathbf{P}^T[0] \leftarrow \mathbf{P}[0], \mathbf{V}[0] \leftarrow 0$
2: **for** $node = 1; node < 2^M - 2; node++$ **do**
3: $\quad parent \leftarrow (node - 1)/2$
4: $\quad$ **if** $node$ is odd **then**
5: $\quad\quad \mathbf{V}[node] \leftarrow \mathbf{V}[parent] \times 2 + \mathbf{P}[parent]$
6: $\quad$ **else**
7: $\quad\quad \mathbf{V}[node] \leftarrow \mathbf{V}[parent] \times 2 - \mathbf{P}[parent]$
8: $\quad \mathbf{P}^T[node + \mathbf{V}[node]] \leftarrow \mathbf{P}[node]$

4.1.2 Transpose of a TPM

Similar to multiplication, the transpose operation on a TPM can be conducted effectively based on its tree vector. The transpose of a TPM with size $2^M \times 2^M$ is computed with the method shown in Algorithm 3. In Algorithm 3, $\mathbf{P}$ and $\mathbf{P}^T$ represent the tree vectors of a TPM and its transpose, respectively. $\mathbf{V}$ is a vector used to locate the corresponding elements in the two tree vectors.

4.2 TPM-based LDPC codes

We define a TPM-based parity-check matrix $\mathbf{H}_{\text{TPM}}$ as

$$\mathbf{H}_{\text{TPM}} = \begin{bmatrix} \mathbf{T}_{0,0} & \mathbf{T}_{0,1} & ... & \mathbf{T}_{0,L-1} \\ \mathbf{T}_{1,0} & \mathbf{T}_{1,1} & ... & \mathbf{T}_{1,L-1} \\ ... & ... & & ... \\ \mathbf{T}_{J-1,0} & \mathbf{T}_{J-1,1} & ... & \mathbf{T}_{J-1,L-1} \end{bmatrix} \tag{4.1}$$

where each $\mathbf{T}_{i,j}$ indicates a TPM matrix, and J and L correspond to the number of rows and columns of the parity-check base matrix. We further define the corresponding LDPC code of the TPM-based parity-check matrix as a TPM-based LDPC code.

4.2.1 Cycle Evaluation

To evaluate the cycles of a TPM-based LDPC code, we can apply the following theorem (Details can be found in [37]).

Theorem 4.1. *Let $\mathcal{C}$ be a code which can be described by a parity-check matrix $\mathbf{H} = (\mathbf{P}_{i,j})$, where the (i,j)-th entry $\mathbf{P}_{i,j}$ represents a $p \times p$ permutation matrix. If there exists a cycle of length $2l$ which including the indices $i_0, i_1, ..., i_{l-1}$ and $j_0, j_1, ..., j_{l-1}$ ($i_s \neq i_{s+1}$, $j_s \neq j_{s+1}$, for all $s \in 0, 1, ..., l-1$), then the product of the matrices*

$$P_{i_0,j_0} P_{i_1,j_0}^T P_{i_1,j_1} P_{i_1,j_1}^T ... P_{i_{l-1},j_{l-1}} P_{i_0,j_{l-1}}^T \tag{4.2}$$

has a fixed column.

Moreover, in the previous section, we have stated that a TPM has a fixed column if and only if there exists an all-zero branch in its tree representation. Therefore, a cycle exists if the tree vector corresponding to (4.2) contains an all-zero branch. Furthermore, the tree vector can be readily evaluated based on the tree vectors of the component TPMs in (4.2) using Algorithms 2 and 3.

4.2.2 Code Construction

Recall that a $2^M \times 2^M$ TPM has 2^{2^M-1} different combinations. In the design of a TPM-based parity-check matrix $\mathbf{H}_{\text{TPM}}$ shown in (4.1), there will be a tremendous number of possible combinations to consider if M is large. Fortunately, it can be readily shown that if a $\mathbf{H}_{\text{TPM}}$ achieves a girth of g, expanding each '1' in $\mathbf{H}_{\text{TPM}}$ with $\mathbf{I}_{2\times 2}$ and $\tilde{\mathbf{I}}_{2\times 2}$ will result in a new $\mathbf{H}'_{\text{TPM}}$ with girth no less than g.

Hence, we can start our code construction using small-size TPMs and consider small girth first. Then we expand the TPMs and try to achieve a higher girth. Since the expansion will not lead to new cycles whose length is smaller than the current girth, we do not need to consider the previous portion (i.e., original tree vector) of a TPM when searching for a higher girth, but only need to focus on the expanded part (i.e., new elements in the tree vector that define the expansion). Because of this advantage, the fast hill-climbing algorithm can be easily used in searching for high-girth TPM-LDPC codes. Recall the fast hill-climbing algorithm introduced in Chapter 2. The process is divided into several steps and in each step, each element is optimized with an adaptive cost. For TPMs, this adaptive cost can be simplified because after we have achieved a girth-$2l$ TPM-LDPC code, we do not need to consider cycles shorter than $2l$ after expanding the code.

In order to reduce the computational intensity when searching for high-girth TPM-LDPC codes using fast hill-climbing algorithm, we first focus on small-size TPMs. When the size of TPMs is small, the conditions of each sub-matrix we need to consider is relatively small. Then we expand the TPMs to try to achieve a larger girth. For example, the construction of a girth-8 TPM-LDPC code with 3×10 TPMs is presented as follows. We replace each '1' in the 3×10 base matrix with a randomly chosen $2^3 \times 2^3$ TPM. Note that there are $2^{2^3-1} = 128$ different $2^3 \times 2^3$ TPMs. Then we minimize the number of cycle-4 by varying the TPMs one-by-one. If all cycle-4s are eliminated and a girth-6 TPM-LDPC code is found, we expand the TPMs to the size of $2^4 \times 2^4$ and attempt to minimize the number of cycle-6. At this step, the initial part of the matrix (or tree vector) is fixed and we only need to consider the expanded part. In addition, all the choices of expanded part will not lead to new length-4 cycles so the calculation only includes the paths of potential length-6 cycle. Then the conditions to traverse will be reduced distinctly. If all cycle-6 cannot be eliminated after a number of iterations, we expand each TPM again to become $2^5 \times 2^5$ and repeat the operation.

When the size of a TPM becomes large, there will be an enormous number of possible combinations after each expansion. It is extremely time-consuming to try all the possible combinations. Therefore, we will randomly pick some of these combinations and select the one with the minimum cost. In this way, the TPM-LDPC code can be further "optimized" with the fast hill-climbing algorithm even when the size is large. With this method, we obtain a girth-8 TPM-LDPC code with a base matrix of size 4×24, and a girth-10 TPM-LDPC code with a base matrix of size 3×10.

4.2.3 Parallel Decoding and Message Storage

In chapter 3.2.2, we have shown that conflicts of RAM access will occur when random permutation matrices are used in constructing LDPC codes. In the case of TPM-based LDPC codes, such conflicts can be avoided when the messages are properly stored and the number of parallel processors are chosen with care.

For TPM-LDPC codes, a P_2^M TPM has a size $2^M \times 2^M$. It can also be characterized by 2^{M_1} smaller TPMs each of size $2^{M_2} \times 2^{M_2}$ where $M_1 + M_2 = M$. For example, each of the $2^3 \times 2^3$ TPMs shown in Fig. 4.4 can be characterized by (i) $2^1(= 2)$ smaller $2^2 \times 2^2 (= 4 \times 4)$ TPMs, or (ii) $2^2(= 4)$ smaller $2^1 \times 2^1 (= 2 \times 2)$ TPMs. To avoid conflicts of RAM access, the degree of parallelism G must equal 2^{M_2} for some M_2. Moreover, the 2^{M_2} $C \leftrightarrow V$ messages corresponding to each $2^{M_2} \times 2^{M_2}$ small TPM must be assigned to $G = 2^{M_2}$ different RAMs. Then, the condition that two $C \leftrightarrow V$ messages in the same RAM are needed at the same time will never happen. From Figure 4.4, we can visualize why parallel decoding can be performed without RAM access conflicts when $G = 2$. In fact, we can also see that parallel decoding can be performed when $G = 4$.

In addition, we need to use a group of RAMs (called *address RAMs*) to store the address information of the RAMs storing $C \leftrightarrow V$ messages. Due to the possibility of parallel decoding, the number and size of such kind of RAMs in a TPM-LDPC decoder are reduced compared with those needed in random-permutation-matrix LDPC decoders.

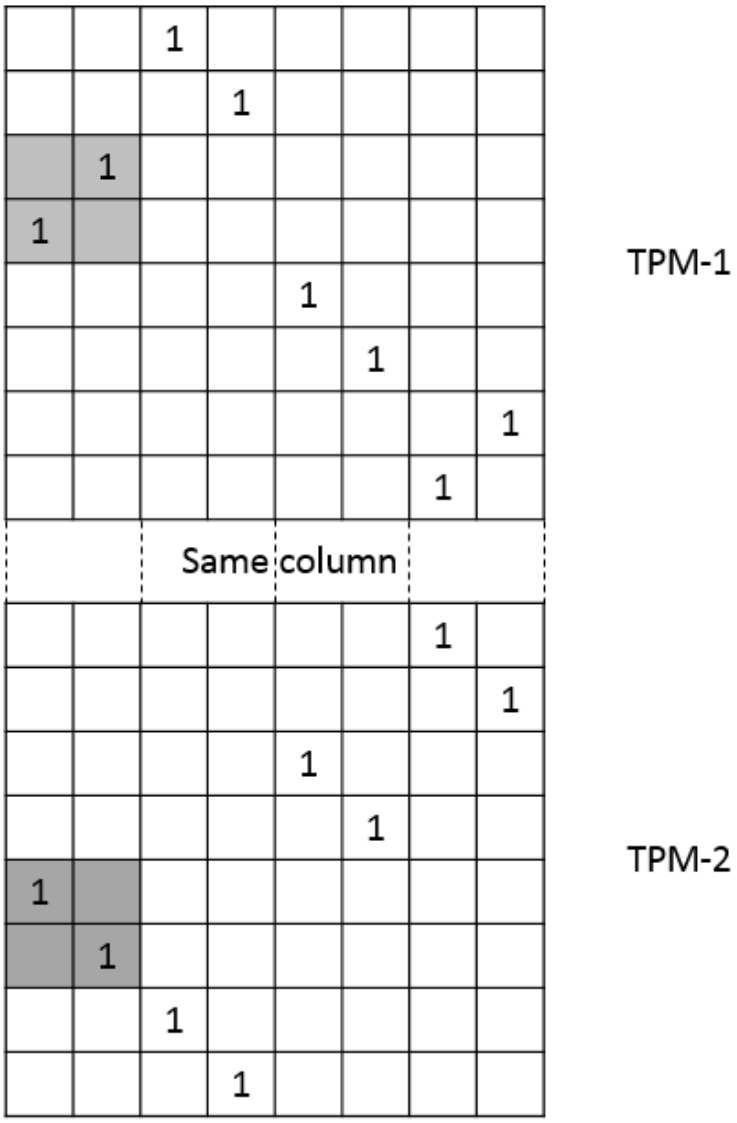


Figure 4.4: Illustration of possibility of parallel decoding of TPM-based LDPC code.

4.3 Simulation Results

We first construct and simulate TPM-LDPC codes with the following parameters.

- $J = 4, L = 24, z = 4096, g = 8$, a code rate of 5/6 and a code length of 98304
- $J = 4, L = 24, z = 2048, g = 6$, a code rate of 5/6 and a code length of 49152

For both cases, we optimize the girth of the LDPC codes using the fast hill-climbing method [25]. The decoder is implemented on an Altera Stratix IV EP4SE530H35C2 FPGA. The BER results are plotted in Fig. 4.5. We also plot the BER of the following codes for comparison.

- Regular 4×24 QC-LDPC codes with $z = 4096$, $g = 8$ and length 98304
- 16×96 CC-QC-LDPC code with $z = 1024$, and $g = 8$ and length 98304
- 16×96 CC-QC-LDPC code with $z = 512$, $g = 6$ and length 49152

Comparing codes with length 98304, the CC-QC-LDPC code accomplishes the best BER, outperforming our proposed TPM-based LDPC code by about 0.015 dB and QC-LDPC code by about 0.03 dB. Comparing codes with length 49152, the CC-QC-LDPC code outperforms our proposed TPM-based LDPC code by about 0.02 dB. However, the CC-QC-LDPC code reaches an error floor at around 10^{-13} whereas our proposed TPM-based LDPC code does not show an error floor below 10^{-13}.

Next, we construct and simulate a TPM-based LDPC code with the following parameters:$J = 3, L = 10, z = 4096, g = 10$, code rate of 7/10, and a code length 40960. The BER curve is plotted in Fig. 4.6. We also show the BER of a 3×10 regular QC-LDPC code with $z = 4096$ and $g = 10$ in the same figure. Both codes cannot achieve a girth of 12 at $z = 4096$ using the fast hill-climbing algorithm. The

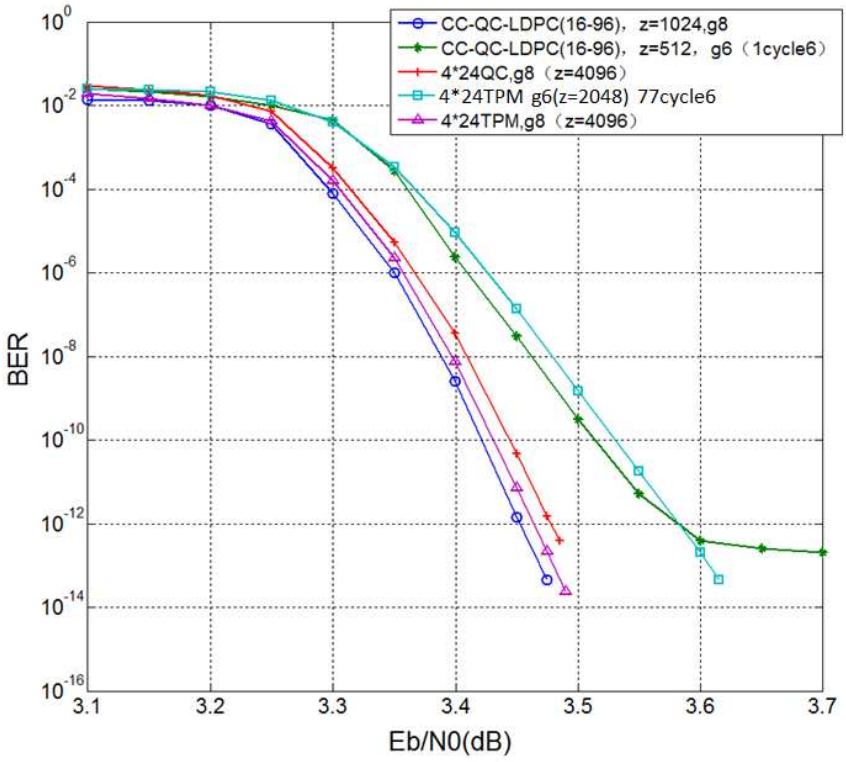


Figure 4.5: Bit error rates of different codes

results show that the proposed TPM-LDPC code not only outperforms the regular QC-LDPC code, but also shows a lower error floor.

In Table 4.1, we show the hardware complexity of the different decoders. In general, the complexity becomes lower when fewer address RAMs are used during the decoding. The decoder complexity of the regular QC-LDPC code (Code A) is the lowest. The CC-QC-LDPC decoder (Code D) is more complex than the regular QC-LDPC decoder but is simpler than the TPM-LDPC decoder (Code B) and the RP-CC-LDPC decoder (Code C). The decoder complexity of the TPM-LDPC codes is lower than that of the RP-CC-LDPC codes. In addition, the degree of parallelism has a great impact on the throughput of the decoder. The regular QC-LDPC code

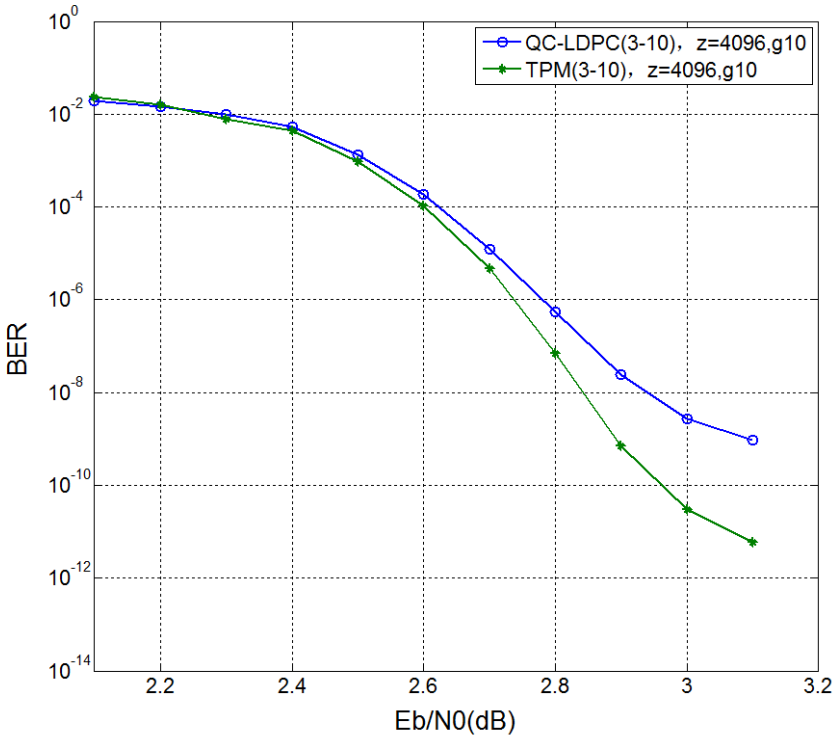


Figure 4.6: Bit error rate of QC-LDPC code and TPM-based LDPC code with 3×10 sub-matrices.

has the same throughput as the TPM-LDPC code and the CC-QC-LDPC code. The throughput is much higher than that of the RP-CC-LDPC code.

4.4 Summary

In this chapter, a new type of LDPC code called tree-permutation-matrix (TPM)-LDPC code has been proposed. We introduce the construction and properties of TPMs and we define TPM-LDPC codes. We have also described simple methods for computing the product and transpose of TPMs. According to these basic

Table 4.1: Hardware Information of the Decoder Implementations. Code A: 4×24 regular QC-LDPC code, girth=8, $z = 4096$; Code B: 4×24 TPM-LDPC code, girth=8, $z = 4096$; Code C:16×96 RP-CC-LDPC code, $z = 1024$; Code D:16×96 CC-QC-LDPC code, $z = 1024$.

Code	A	B	C	D
Parallelism degree	32	32	4	32
ALUTs	65,178	87,479	31,660	70,342
Registers	45,336	49,322	13,564	43,801
Memory bits	1,796,372	2,589,891	4,297,289	2,395,296
Clock	100 MHz	100 MHz	100 MHz	100 MHz
Throughput	1.55 Gbps	1.55 Gbps	0.182 Gbps	1.55 Gbps

characteristics of TPMs, we have proposed a systematic way of finding large-girth TPM-LDPC codes based on fast hill-climbing algorithm. Using this method, we obtain a girth-8 TPM-LDPC code with a base matrix of size 4×24 and a girth-10 TPM-LDPC code with a base matrix of size 3×10. We have shown how RAM access conflicts can be avoided when designing parallel TPM-LDPC decoders.

We have shown our simulation results of three TPM-LDPC codes with different parameters. We have also compared the BER results and decoder complexity of the proposed TPM-LPDC codes with other LDPC codes under the same code length and code rate. Simulation results have shown that TPM-LDPC codes make use of fewer memory resources and have a higher throughput compared with RP-CC-LDPC codes. Compared with QC-LDPC codes, TPM-LDPC codes require more resources in implementation but can provide a slightly better BER performance. While the CC-QC-LDPC codes require less resource in implementation, TPM-LDPC codes have a lower error floor when the code length is relatively short.

Chapter 5

Conclusions and Future Directions

5.1 Conclusions

In this thesis, we have studied/proposed different kinds of LDPC codes and simulated their error performance based on computer and/or FPGA. Moreover, the girth of the codes, whenever possible, is optimized by using the fast hill-climbing algorithm.

Firstly, with an aim to achieving a shorter code length and a larger girth, we have proposed two new types of CC-QC-LDPC codes. We eliminate some cycle possibilities by converting some sub-matrices in a full CC-QC-LDPC code into zero matrices. Simulation results have demonstrated that compared with the full CC-QC-LDPC ones, our new types of CC-QC-LDPC codes can achieve a better performance and have lower error floors even with shorter code lengths. The decoder complexities of our new CC-QC-LDPC codes are also lower because fewer RAMs are required. We have also modified the CC-QC-LDPC code and propose a new type of CC-LDPC

codes. We have replaced the circulant permutation matrices in a CC-QC-LDPC code with random permutation matrices, forming the random-permutation-matrix CC-LDPC (RP-CC-LDPC) code. We have further implemented the decoder of the proposed RP-CC-LDPC codes using FPGA. We have compared the error performance of the RP-CC-LDPC code with the CC-QC-LDPC code, the regular and irregular QC-LDPC code. Simulation results have shown that RP-CC-LDPC code can outperform CC-QC-LDPC code and regular QC-LDPC code in terms of BER. However, RP-CC-LDPC code does not perform as well as irregular QC-LDPC code. The RP-CC-LDPC code also suffers from the high complexity and low throughput issues.

We have also modified the QC-LDPC code construction by using tree-permutation-matrices (TPM) instead of circulant permutation matrices, and have proposed a new type of LDPC codes called TPM-LDPC codes. We have introduced the TPM and its expansion method. We define TPM-LDPC codes and provide a systematic way of reducing the number of cycles in TPM-LDPC codes based on the fast hill-climbing algorithm. We have obtained a girth-8 TPM-LDPC code with a base matrix of size 4×24 and a girth-10 TPM-LDPC code with a base matrix of size 3×10. We have described how to avoid memory access conflict and to achieve parallel TPM-LDPC decoding. The simulated BER results and the decoder complexity of TPM-LDPC codes have been shown and compared with other regular and irregular LDPC codes under the same code length and code rate. The TPM-LDPC codes outperform the CC-QC-LDPC codes and regular QC-LDPC codes in terms of BER but is outperformed by the same two codes in terms of decoder complexity.

5.2 Future Work

Firstly, the CC-QC-LDPC codes can be modified by using TPMs as their sub-matrices. According to the simulation results in chapter 4, we find that using TPMs instead of CPMs can improve the BER performance and lower the error floor of the LDPC codes. TPMs also provide the opportunity of parallel operation which can reduce the complexity and improve the throughput of FPGA design of the decoder.

Secondly, in Chapter 4, the size of the permutation matrices used to construct the TPMs is 2×2. Matrices of larger sizes can be used to construct TPMs and hence TPM-LDPC codes. The error performance of such codes should be investigated.

Finally, searching for large-girth LDPC codes with large-size TPMs using computers becomes very time-consuming. FPGAs which can perform parallel and fast computations can be considered for such purposes.

Bibliography

[1] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[2] S. Y. 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 Commun. Letters*, vol. 5, no. 2, pp. 58–60, 2001.

[3] M. M. Mansour and N. P. Shanbhag, "Low-power VLSI decoder architectures for LDPC codes," in *Proc. Int. Symp. Low Power Electronics and Design*. New York, USA, 2002, pp. 284–289.

[4] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 976–996, 2003.

[5] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, no. 6, pp. 1570–1579, 2002.

[6] T. Tian, C. R. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242–1247, 2004.

[7] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annu. Allerton Conf. on Communication, Control, and Computing*. Urbana-Champaign, USA, 2003, pp. 1426–1435.

[8] D. J. C. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, pp. 97–104, 2003.

[9] 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.

[10] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "Analysis of Absorbing Sets and Fully Absorbing Sets of Array-based LDPC Codes," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 181–201, 2010.

[11] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 386–398, 2005.

[12] H. Xiao and A. H. Banihashemi, "Improved progressive-edge-growth (PEG) construction of irregular LDPC codes," *IEEE Communications Letters*, vol. 8, no. 12, pp. 715–717, 2004.

[13] F. C. M. Lau, W. M. Tam, and C. K. Tse, "Increasing the local girth of irregular low-density parity-check codes based on degree-spectrum analysis," *IET Communications*, vol. 5, no. 11, pp. 1506–1511, 2011. [Online]. Available: http://link.aip.org/link/?COM/5/1506/1

[14] G. Richter and A. Hof, "On a construction method of irregular LDPC codes without small stopping sets," in *Proc. IEEE Int. Conf. Commun.* Istanbul, Turkey, 2006, pp. 1119–1124.

[15] L. Dolecek, J. Wang, and Z. Zhang, "Towards improved LDPC code designs using absorbing set spectrum properties," in *Turbo Codes and Iterative Information Processing (ISTC), 2010 6th International Symposium on*, Sept 2010, pp. 477–481.

[16] J. Wang, L. Dolecek, Z. Zhang, and R. Wesel, "Absorbing set spectrum approach for practical code design," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, July 2011, pp. 2726–2730.

[17] D. J. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[18] 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.

[19] 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.

[20] Y. Kou, S. Lin, and M. P. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2711–2736, 2001.

[21] J. L. Fan, "Array codes as LDPC codes," in *Constrained Coding and Soft Iterative Decoding*. Springer, 2001, pp. 195–203.

[22] O. Milenkovic, N. Kashyap, and D. Leyba, "Shortened Array Codes of Large Girth," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3707–3722, 2006.

[23] M. P. Fossorier, "Quasi-Cyclic Low-Density Parity-Check Codes from Circulant Permutation Matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[24] Y. Wang, J. S. Yedidia, and S. C. Draper, "Construction of high-girth QC-LDPC codes," in *5th International Symposium on Turbo Codes and Related Topics*, vol. 185. Lausanne, Switzerland, 2008.

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

[26] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386–398, 2005.

[27] A. J. Felstrom and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 2181–2191, 1999.

[28] A. Sridharan, D. Truhachev, M. Lentmaier, D. J. Costello, and K. S. Zigangirov, "Distance bounds for an ensemble of LDPC convolutional codes," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4537–4555, 2007.

[29] 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," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 60, no. 7, pp. 1857–1869, July 2013.

[30] 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, 2016.

[31] T. Cooklev, *Air Interface for Fixed Broadband Wireless Access Systems.* John Wiley & Sons, Inc., 2011.

[32] B. Bangerter, E. Jacobsen, M. Ho, A. Stephens, A. Maltsev, A. Rubtsov, and A. Sadri, "High-Throughput Wireless LAN Air Interface," *Intel Technology Journal*, no. 3, 2003.

[33] F. C. M. Lau, F. Mo, Q. Lu, W. M. Tam, and C. W. Sham, "Novel types of cyclically-coupled quasi-cyclic LDPC block codes," in *2016 International Conference on Advanced Technologies for Communications (ATC)*, Oct 2016, pp. 27–31.

[34] F. C. M. Lau, F. Mo, W. M. Tam, and C. W. Sham, "Random-permutation-matrix-based cyclically-coupled LDPC codes," in *2017 19th International Conference on Advanced Communication Technology (ICACT)*, Feb 2017, pp. 497–500.

[35] 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.

[36] S. Jiang, F. Mo, F. C. M. Lau, and C.-W. Sham, "Tree-Permutation-Matrix-Based LDPC Codes," *to appear in IEEE Trans. on Circuits and Systems II.*

[37] R. Smarandache, D. G. M. Mitchell, and D. J. Costello, "Partially Quasi-Cyclic Protograph-Based LDPC Codes," in *2011 IEEE International Conference on Communications (ICC)*, June 2011, pp. 1–5.