A FLEXIBLE DECODER AND PERFORMANCE EVALUATION

FOR

ARRAY-STRUCTURED LDPC CODES

Dissertation by

SUNG CHUL HAN

Submitted in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL AND COMPUTER ENGINEERING

Commite members:

Prof. Rohit Negi (Advisor)

Prof. Markus Püschel (Co-advisor)

Prof. B.V.K. Vijaya Kumar

Prof. James Hoe

Dr. Hongwei Song (Marvell)

CARNEGIE INSTITUTE OF TECHNOLOGY

CARNEGIE MELLON UNIVERSITY

Pittsburgh, Pennsylvania

September, 2007

ABSTRACT

The low density parity check (LDPC) codes designed by a pseudorandom construction, as proposed in Gallager's original work, have been shown to perform very close to the Shannon limit (when constructed as very long codes); however, the lack of structure in such codes makes them unsuitable for practical applications due to high encoding complexity and costly decoder implementations. These difficulties have lead to numerous works on the structured LDPC codes, especially array-structured codes with quasi-cyclic property.

Among the array-structured codes, those with an array of cyclic permutation matrices have been of particular interest due to the balanced edge partitioning inherent in the structure that simplifies the implementation of highly parallel decoders. While many construction methods have been proposed for this circulant permutation array (CPA) structure, the performance of the codes has been reported to a very limited extent. Especially, the effect on the performance by the explicit control of graph parameters has not been provided despite the fact their importance is emphasized in the construction process.

In the decoder design for quasi-cyclic LDPC codes, the primary concern is to exploit the array structure for efficient implementation. Fast hardware-based decoders on a medium-capacity FPGA are often faster than the software implementation by at least one or two orders of magnitude, and thus important for both actual deployment in practical systems and evaluation of error performance. As a large number of high-throughput decoders in the literature are designed for specific array dimensions and the bus and memory connections are simplified using the array structure of the code, the degree of parallelism in the decoders is dependent on the code parameters, making it difficult to parameterize the hardware to use a desired amount of hardware resource. Furthermore, such architectures cannot support a large class of array-structured codes with very different array dimensions.

In this thesis, we present a generalized hardware decoder that supports any kind of quasi-cyclic

LDPC codes including CPA-structured codes. The decoder has been designed with a priority on high flexibility. In the synthesis step, the degree of parallelism can be chosen independently from the code parameters. Thus, for FPGA implementation, the decoder can be parameterized to fully utilize a given amount of hardware resource. Also, it supports run-time reconfiguration of code parameters, i.e., different codes can be supported by changing register contents without a new synthesis. In wireless applications, such flexibility makes it possible to choose a channel code based on the varying channel condition. When used for performance evaluation purposes for a large set of codes, it saves a considerable amount of time by eliminating the need for re-synthesis for each code.

Using the FPGA implementation of the proposed decoder, we characterize the performance of array-structured LDPC codes, with a primary focus on pseudorandomly constructed CPA-structured codes. Based on the obtained simulation results, we show the effect of combinatorial parameters (girth, diameter and column weight) on the error performance. The pseudorandom construction is also compared with algebraic construction, and with the codes specified in the IEEE 802.16e standards.

ACKNOWLEDGMENTS

Having arrived at the end of five-year-long graduate student life, I realize that I have lived one of the happiest periods in life. It was never easy in the beginning, especially after spending many years at industry, to get used to the pressure of taking the full control of my own research, take the responsibilities of every decision I should make, and strive to make meaningful contributions. Any accomplishments I achieved would have been impossible without the support of the excellent faculty here at Carnegie Mellon University. I'd like to thank my advisors Prof. Rohit Negi and Prof Markus Püschel for their insightful directions and invaluable guidance. It was my greatest joy to take Prof. Negi's excellent lectures and see how he extracts the essence out of seemingly intricate subjects and provides insightful views. Prof Püschel's mathematical insight shaped my research and his positive outlook on the thesis topic always gave me the courage to carry on. I'd like to express my sincere thanks to Prof. Vijaya Kumar, Prof. James Hoe, and Dr. Hongwei Song for all the helps they provided during the course of my research.

My colleagues Xun, Yaron, Arjun, Gyouhwan, Satashu, Soowoong, and Murali that I worked with have been my best friends as well as my best teachers. The discussions they were eager to take part in were of great help to my research. The time spent together for various non-academic activities was always a pleasant experience for me. I'd also like to give my special thanks to Jingu for all the things he did that I could possibly expect from a friend.

My wife Dahee, whom I cannot thank enough for the support and care, has been an indispensable part of my life here at Pittsburgh. I'd also like to thank my parents for their endless love and encouragements to accomplish my goals.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

One of the key underlying technologies in our increasingly connected world is the method for efficiently communicating discretized information over a physical medium such as telephone lines, optical cables, radio links, or magnetic storages. Channel coding plays an integral role in providing a reliable communication method that can overcome signal degradation in practical channels. Turbo codes, invented by Berrou, Glavieux and Thitimajshim in 1993, are the first known capacity-approaching error correction code that provides a powerful error correction capability when decoded by an iterative decoding algorithm. More recently, research efforts toward searching for lower complexity codes and iterative decoding led to the rediscovery of low density parity check (LDPC) code, which was originally proposed by Gallager in 1960 and was later generalized as MacKay-Neal code [1]. The LDPC codes have been shown to achieve near-optimal performance in additive white Gaussian noise channels when decoded with the sum-product (SP) algorithm [2].

LDPC codes have several advantages over turbo codes. While it is difficult to apply parallelism in the decoding of turbo code due to the sequential nature of the decoding algorithm, LDPC decoding can be performed with a high degree of parallelism to achieve a very high decoding throughput. LDPC codes do not require a long interleaver, which causes a large delay in turbo codes. LDPC codes can be directly designed for a desired code rate while turbo codes, that are based on convolutional codes, require other techniques such as puncturing to get the desired rate.

While LDPC codes designed by a pseudorandom construction, as proposed in Gallager's original work, have been shown to perform very close to the Shannon limit (when constructed as very long codes), the lack of structure leads to very costly decoder implementations. Worse yet, the encoding complexity that is quadratic in the codeword length puts LDPC codes at a serious disadvantage over turbo codes that has linear encoding complexity.

To overcome the difficulty in practical implementations, various methods have been proposed to introduce structure in the parity check matrix. Kou, Lin and Fossorier introduced algebraic constructions based on finite geometries in [3, 4]. Other constructions based on combinatorics include disjoint difference sets (DDS) codes by Johnson and Weller [5] and Song, Liu and Kumar [6] and balanced incomplete block design (BIBD) codes by Ammar, Honary, Kou, Xu and Lin [7] and Vasic and Milenkovic [8]. Many of the codes designed by algebraic or combinatorial construction have certain desirable characteristics such as cyclic or quasi-cyclic properties that lead to linear-time encoding and a guarantee that two rows in the parity check have less than two 1's in the common bit positions to help the sum-product algorithm perform better.

The parity check matrix $\mathbf{H}$ of a code with quasi-cyclic property can be put into an array of circulant matrices with column and row rearrangements. The LDPC codes with the quasi-cyclic property are called QC-LDPC codes. Because of its practical importance, most of the LDPC decoder architectures in the literature have been designed for QC-LDPC codes or its subclass, in which the array structure can be exploited for more efficient implementation.

Among various array-structured LDPC codes, a class of codes of which the $\mathbf{H}$-matrix is an array of cyclic permutation matrices have been of particular interest since the balanced partitioning of 1's into submatrices facilitates the design of a highly parallelized decoders. In the rest of this chapter, we give a brief overview on the structure and the code construction methods for the LDPC codes based on the cyclic permutation array, and state the goals of this thesis.

1.1 CYCLIC PERMUTATION ARRAY (CPA) STRUCTURE

Initially, Fan proposed in [9] to use a binary *array code* to construct an LDPC code. An *array code* in this context is a general class of algebraic codes defined as a two-dimensional array of code symbols that lie in Galois rings, and should not be confused with the array structure in parity check representation. These original *array codes* are discussed in [10, 11]. He showed that the parity check matrix $\mathbf{H}$ of an *array code* is a two-dimensional array of $P \times P$ submatrices that form a particular series of powers of a single cyclic permutation matrix, i.e.,

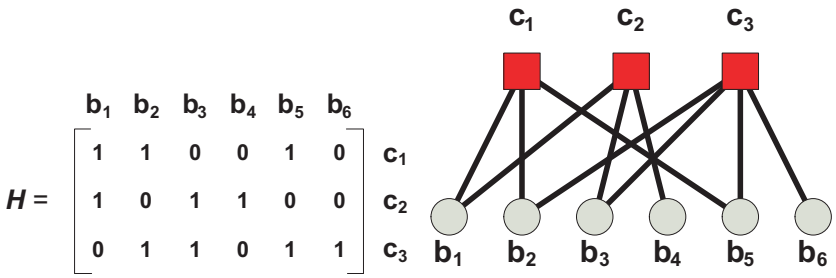
$$\mathbf{H} = \begin{pmatrix} \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} \\ \mathbf{I} & \mathbf{C} & \dots & \mathbf{C}^{P-1} \\ \vdots & \vdots & & \vdots \\ \mathbf{I} & \mathbf{C}^{r-1} & \dots & \mathbf{C}^{(r-1)(P-1)} \end{pmatrix}, \tag{1.1}$$

where

$$\mathbf{C} = \begin{pmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{pmatrix}, \tag{1.2}$$

and it was shown that the code is cyclic with an odd prime P and $r \leq P$. Henceforth, this construction was also termed "array code", which is in fact a special case of the *array code* in the original context of [10, 11]. It was also named "lattice code" in [8]. This structure will be denoted as array(r, P) in Chapter 5.

A more general notion of an array code was given in [8, 9] where the submatrices can be any permutation matrices, to facilitate the cycle analysis of the array code rather than to introduce a new family of codes. Tanner in [12] and Milenkovic in [13] also generalized the array code by considering other series of powers, but still based on algebraic construction. More recently, Lu, Moura and Niesen proposed pseudorandom construction of the array code where the submatrices can be any cyclic permutations [14]. Later, Lu extended the idea by allowing all-zero submatrices



(a) Parity check matrix representation (b) Tanner graph representation

Figure 1.1: A parity check matrix and a Tanner graph

in the array in his Ph.D. thesis [15]. This structure has also been chosen as the structure of the LDPC codes in the IEEE 802.16e (Mobile WiMAX) standards [16]. To avoid confusion with the array codes used in the narrower sense to denote a particular progression in the powers, we will refer to the more general structure consisting of any circulant permutation matrices as the circulant permutation array (CPA) structure in the rest of this thesis.

1.2 CONSTRUCTION OF THE CPA-STRUCTURED LDPC CODES

A parity check matrix can be graphically represented by a Tanner graph, which is a bipartite graph with a set of check nodes on top and a set of bit nodes on bottom. When a check equation in the $\mathbf{H}$-matrix checks a particular code bit, there lies an edge between the corresponding check node and bit node. Figure 1.1 shows a simple parity check matrix and the corresponding Tanner graph.

The performance of the SP algorithm is known to degrade if there exist short cycles in the Tanner graph [17]. One intuitive explanation was given by Gallager in [18] by showing that the number of independent decoding iterations m is determined by $m = \lceil g/4 \rceil - 1$, where *girth* g is defined as the length of the shortest cycle. In addition, Tanner derived a lower bound on the minimum distance of a regular LDPC code as an increasing function of g in [19], i.e.,

$$d_{\min} \geq \begin{cases} \frac{2\left((j-1)^{g/4}-1\right)}{j-2}, & g/2 \ : \ \text{even}, \\ \frac{j(j-1)^{\lfloor g/4 \rfloor}-2}{j-2}, & g/2 \ : \ \text{odd}, \end{cases} \tag{1.3}$$

where j is the column weight of the parity check matrix. For these reasons, it has been of primary

concern in LDPC code construction to avoid short cycles in the Tanner graph.

Most of the codes based on algebraic or combinatorial construction are designed to avoid 4-cycles (i.e., cycles of length 4), which is the shortest possible length for bipartite graphs. While none of them have direct methods to achieve a girth greater than 6, there have been indirect methods to enlarge the girth by selectively eliminating columns in the parity check matrix. For pseudorandom constructions, there have been explicit efforts to find codes with a large girth. A heuristic method called "bit-filling" tries to construct the parity check matrix column by column while maintaining the target girth and the predefined column weight [20]. In another heuristic approach, they search for a good LDPC code based on the average girth, where the girth is redefined as a node girth, i.e., the length of the shortest cycle that passes through each node [21]. Due to the arbitrary constructions, a hardware-based decoding for these codes can be prohibitively costly. On the other hand, Lu showed that a pseudorandom search on the CPA structure based on the girth finds good codes while allowing a simple decoder architecture [15].

1.3 THESIS GOAL

As indicated above, general CPA-structured LDPC codes encompass a broad class of algebraic and pseudorandom construction methods. The performance of some pseudorandomly constructed codes was given in [15], but a direct comparison with other algebraic constructions were not provided. In addition, since most of the results in the literature are obtained from software simulation, the performance of CPA-structured codes have been usually explored only down to the bit error rate (BER) range between 10^{-6} and 10^{-8}. The performance at very low BER cannot be simply extrapolated from that of the medium BER range due to the error floor, which is a phenomenon in which a code exhibits a sudden saturation in the BER curve at sufficiently high SNR. For some applications that require extremely reliable signalling schemes such as magnetic storage and satellite communication systems, it is important to choose a code that does not have an error floor at very low BERs (10^{-11} to 10^{-15}).

A fast hardware-based simulation environment can extend the simulation's reach far beyond the limit of a software-based simulation. A large number of high-throughput hardware-based decoders in the literature are designed for specific array dimensions, and the bus and memory connections are simplified using the array structure of the code. As a result, the degree of parallelism in the decoders is dependent on the code parameters, making it difficult to parameterize the hardware to use a desired amount of hardware resource. Furthermore, such architectures cannot support a large class of array-structured codes with very different array dimensions.

In this thesis, we pursue broadening our understanding of the array-structured LDPC codes with a primary focus on pseudorandomly constructed CPA-structured codes. The main objectives include the design of a highly flexible hardware-based decoder and performance evaluation by means of simulation using this flexible decoder.

Hardware simulation environment: We present a hardware simulation environment including a Gaussian noise generator and a general decoder for array-structured LDPC codes. The decoder has been designed to serve the purpose of this research; it supports any code in the family of QC-LDPC codes including the subclass of the CPA structures, and exploits their structural property to simplify the implementation. Unlike most of the decoder architectures in the literature that are designed for high throughput, the proposed decoder has been designed with a priority on flexibility. The degree of parallelism in the decoder is independent of the code parameters, making it possible to fully utilize the given hardware resource regardless of the column and row weights or the size of the permutation matrix. Moreover, the architecture supports run-time reconfigurability, i.e., the parameter of the code to be decoded can be changed simply by modifying the parameter values stored in registers without a new synthesis of the decoder. In wireless applications, this architecture makes it possible to flexibly choose a channel code under varying channel conditions. When hardware decoders implemented in field programmable gate array (FPGA) are used for performance evaluation purposes, this run-time flexibility can save a considerable amount of time by eliminating the need for re-synthesis for each code to be tested.

Investigation of the effect of code parameters and combinatorial metrics: With the hard-

ware simulator, we will investigate the effect of code parameters (column weight, code length, and code rate) on the performance, and evaluate the influence of combinatorial metrics (girth and diameter) on performance, towards a better design methodology of CPA-structured codes. We will also compare pseudorandom construction methods and the algebraic construction of CPA-structured LDPC codes. Besides the CPA structure, the error performance of selected finite-geometry codes will be measured and compared with the CPA-structured codes.

Study of extension of CPA structure: We also introduce a more general array structure to find codes with larger girths than the CPA-structured codes. Although Lu showed in [15] that an array-structured code can be constructed to have an arbitrary girth by recursively replacing the 1's in the **H**-matrix of CPA-structured codes by permutation matrices and the 0's by all-zero matrices, this procedure tends to result in very long codes, and hence it is not suitable for the construction of the codes with moderate lengths. Our extension will take a different approach in that we allow non-cyclic permutations as submatrices, which will be denoted as group permutation array (GPA) structure.

1.4 THESIS ORGANIZATION

The rest of this thesis is organized as follows. Chapter 2 provides notations and definitions to be used in this thesis, reviews previous works on LDPC code construction and decoding algorithm, and introduces a nonparametric hypothesis test that will be used in Chapter 5. Chapter 3 presents new methods to find the girth and the diameter of QC-LDPC codes. Chapter 4 provides the details of the proposed flexible decoder for QC-LDPC codes and a comparison with a selection of other decoder architectures. In Chapter 5, we present the performance simulation results obtained using the FPGA-based simulator and study the effect of structural parameters on the performance of CPA-structured codes. Finally, Chapter 6 summarizes the thesis and discusses future work.

CHAPTER 2

BACKGROUND

2.1 A BRIEF HISTORY OF LDPC CODES

Any information sent over a practical channel is subject to errors that are caused by various physical impairments such as thermal noise, attenuation, multi-path wave reflections, distortion from previously transmitted signals, interference from other transmitters or imperfect timing recovery at the receiver. Until the late 1940's, a commonly held belief was that there is a rate-reliability tradeoff in communication over noisy channels, i.e., to reduce the errors in the channel, either the transmission power has to be increased or the message information has to be sent repeatedly. Shannon's work in 1948 disproved this belief by showing that there is no rate-reliability tradeoff if the information rate is below an information-theoretic limit called *channel capacity*, which is the maximum average number of information bits that can be reliably (i.e., with arbitrary small error) transmitted per second for a given transmission bandwidth and signal-to-noise ratio (SNR).

Although Shannon showed the existence of good codes that achieve channel capacity by using a "random code", such code is not realizable due to its intractable encoding decoding complexity. As a result, research efforts have been focused on finding codes with simplifying structure to ensure a good minimum distance property rather than more random-like codes envisioned by Shannon. For 45 years, researchers developed many new coding techniques with an aim to find codes that are good (close to Shannon's limit) and also practical, but none of them were successful in approaching capacity, with the best practical code being away from the Shannon's limit by 3 to 5 dB..

In 1993, a significant breakthrough was achieved by turbo codes [22], which reached the BER of 10^{-5} at SNRs within 1 dB of the Shannon's limit. While many of the coding techniques developed earlier were based upon delicate algebraic structures, the interleaver used in turbo code incorporated a certain degree of "random-like" nature in the code construction. Although there was no mathematical justification for the performance of turbo codes when they were first presented, many researchers investigated the underlying principles of turbo coding and validated the significance of the invention.

LDPC codes form another class of "random-like" codes that approaches the Shannon's limit, and date back to Gallager's doctoral dissertation in 1961 [18], which was much earlier than turbo codes. In 1981, Tanner generalized LDPC codes with a graphical representation, now called a Tanner graph [19]. In spite of the excellent performance of LDPC codes, these previous works had not drawn much attention probably because the encoding and decoding were computationally intensive with the hardware technology at that time. LDPC codes began to be recognized as a strong competitor to turbo codes in late 1990's when MacKay, Luby, and others rediscovered LDPC codes and showed that they have excellent theoretical performance [2, 23].

The asymptotic performance of LDPC codes, when the codeword length tends to infinity, has been studied using analytical techniques called density evolution or Gaussian approximation [24–26]. Very recently, it has been also shown that the error performance in the waterfall region under binary input memoryless symmetric channels can be predicted by analytical methods [27].

2.2 LDPC CODE STRUCTURE

In this section, we provide the formal definition of general LDPC codes and define several array structures to be discussed in the thesis. We confine the discussion to binary linear codes.

2.2.1 Definition of an LDPC Code and Other Notations

An (N, K) linear block code is defined as a set of codewords that forms a K-dimensional subspace in an N-dimensional vector space over GF(2). The usual convention to represent a linear

block code is either as the row space of a generator matrix, i.e.,

$$\mathcal{C} = \{\mathbf{x} : \mathbf{x} = \mathbf{u}G, \forall \mathbf{u} \in \text{GF(2)}^M\}, \tag{2.1}$$

or as the null space of a parity check matrix, i.e.,

$$\mathcal{C} = \{\mathbf{x} : H\mathbf{x} = 0\}. \tag{2.2}$$

A *(j, k)-regular LDPC code* is defined as the null space of a $M \times N$ parity check matrix $\mathbf{H}$ that has the following structural properties:

1. Each row contains k 1's.
2. Each column contains j 1's.
3. Both j and k are small compared to the number of columns and the number of rows in the $\mathbf{H}$-matrix.
4. The number of 1's common to any two rows is 0 or 1.

This definition is sometimes referred to as "LDPC code in narrow sense" while the one without the last condition is called "LDPC code in wide sense". The last condition, which is sometimes called row-column (RC) constraint, ensures that there are no 4-cycles in the corresponding Tanner graph. The column weight j and row weight k denote the number of 1's in each column and row, respectively. The length of the codewords is N, and there are M parity check equations. If the rank of the $\mathbf{H}$-matrix is r, $K = N - r$ message bits can be transmitted per codeword. Accordingly, the code rate is given by

$$R = \frac{K}{N} \geq \frac{N - M}{N},$$

where the equality holds when all M rows are linearly independent.

For regular LDPC codes as defined above, the total number of 1's in the $\mathbf{H}$ matrix is $jN = kM$. If the column weight or row weight is not uniform, it is called an irregular LDPC code.

2.2.2 LDPC Codes with Array Structure

There are particular properties of codes that result in useful array structures in the parity check matrix. This section will provide formal definitions of the structures.

Cyclic codes: A linear code $\mathcal{C}$ is called *cyclic* if every cyclic shift of a codeword in $\mathcal{C}$ is also in $\mathcal{C}$. The parity check matrix of a cyclic code can be put into the form of a square circulant matrix. It is easy to check that the null space of a circulant $\mathbf{H}$ forms a cyclic code. Suppose $\mathbf{H}$ is a $N \times N$ circulant matrix with redundant rows to have a nonzero code rate. Let $\mathbf{x}$ be a codeword in $\mathcal{C}$ defined as the null space of $\mathbf{H}$, i.e., $\mathbf{Hx} = 0$. Denoting a single downward cyclic shift of $\mathbf{x}$ as $\tilde{\mathbf{x}}$, it can be expressed as $\tilde{\mathbf{x}} = \mathbf{Cx}$, where $\mathbf{C}$ is a $N \times N$ single cyclic permutation matrix as defined in eq. (1.2). Since $\mathbf{C}$ and $\mathbf{H}$ are both circulant, $\mathbf{HC}$ = $\mathbf{CH}$. Multiplying $\tilde{\mathbf{x}}$ with $\mathbf{H}$ yields

$$\mathbf{H}\tilde{\mathbf{x}} = \mathbf{HCx} = \mathbf{CHx} = 0, \tag{2.3}$$

which shows that $\tilde{\mathbf{x}}$ is still in the null space of $\mathbf{H}$. By applying this property multiple times, it is readily noticed that shifting by any number of symbol positions yields a valid codeword.

Cyclic codes form an important class of linear codes since the encoding complexity is linear in codeword length and can be implemented with a simple linear shift register.

Quasi-cyclic codes: A linear code $\mathcal{C}$ is called *quasi-cyclic* if a codeword in $\mathcal{C}$ cyclically shifted by a fixed number s of bit positions is also in $\mathcal{C}$. In other words, if $\mathbf{x}^{\mathrm{T}} = (x_0, x_1, \ldots, x_{N-1})$ is a codeword, $\tilde{\mathbf{x}}^{\mathrm{T}} = (x_{N-s}, x_{N-s-1}, \ldots, x_{N-1}, x_0, x_1, \ldots, x_{N-s-1})$ is another valid codeword. If we rearrange the bit indices in $\mathbf{x}$ by repositioning x_{sm+i} at the index $Pi + m$, where $0 \leq m \leq P-1$, $0 \leq i \leq s-1$ and $P = N/s$, the $\mathbf{x}$ before and after the shift take the following form:

$$\begin{aligned}
\mathbf{x}^{\mathrm{T}} &= (x_0, x_s, \ldots, x_{(P-1)s} | x_1, x_{s+1}, \ldots, x_{(P-1)s+1} | \ldots | x_{s-1}, x_{2s-1}, \ldots, x_{(P-1)s+s-1}), \\
\tilde{\mathbf{x}}^{\mathrm{T}} &= (x_{(P-1)s}, x_0, \ldots, x_{(P-2)s} | x_{(P-1)s+1}, x_1, \ldots, x_{(P-2)s+1} | \ldots \\
&\quad | x_{(P-1)s+s-1}, x_{s-1}, \ldots, x_{(P-2)s+s-1}),
\end{aligned} \tag{2.4}$$

which indicates that a cyclic shift by s positions in the original indices is equivalent to a single shift in each of the s subvectors in the rearranged indices.

The parity check matrix of a quasi-cyclic code can be rewritten as an array of circulant matrices. To check that the null space of such parity check matrix forms a quasi-cyclic code, consider a code whose parity check matrix is an $N_c \times N_b$ array of $P \times P$ submatrices, i.e.,

$$\begin{pmatrix} \mathbf{H}_{0,0} & \mathbf{H}_{0,1} & \dots & \mathbf{H}_{0,N_b-1} \\ \mathbf{H}_{1,0} & \mathbf{H}_{1,1} & \dots & \mathbf{H}_{1,N_b-1} \\ \vdots & \vdots & & \vdots \\ \mathbf{H}_{N_c-1,0} & \mathbf{H}_{N_c-1,1} & \dots & \mathbf{H}_{N_c-1,N_b-1} \end{pmatrix}, \tag{2.5}$$

where each submatrix $\mathbf{H}_{i,j}$ is a circulant matrix. If we represent a codeword in the form

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{N_b-1} \end{pmatrix},$$

where $\mathbf{x}_i$ denotes the i-th subvector of the $\mathbf{x}$, a new codeword $\tilde{\mathbf{x}}$ made by a single downward cyclic shift in each subvector of $\mathbf{x}$ can be represented as

$$\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{C}\mathbf{x}_0 \\ \mathbf{C}\mathbf{x}_1 \\ \vdots \\ \mathbf{C}\mathbf{x}_{N_b-1} \end{pmatrix}.$$

Multiplying $\tilde{\mathbf{x}}$ by $\mathbf{H}$ yields

$$\begin{aligned}
\mathbf{H}\tilde{\mathbf{x}} &= \mathbf{H}\begin{pmatrix} \mathbf{C} & & & \\ & \mathbf{C} & & \\ & & \ddots & \\ & & & \mathbf{C} \end{pmatrix}\begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{N_b-1} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{H}_{0,0}\mathbf{C} & \mathbf{H}_{0,1}\mathbf{C} & \dots & \mathbf{H}_{0,N_b-1}\mathbf{C} \\ \mathbf{H}_{1,0}\mathbf{C} & \mathbf{H}_{1,1}\mathbf{C} & \dots & \mathbf{H}_{1,N_b-1}\mathbf{C} \\ \vdots & \vdots & & \vdots \\ \mathbf{H}_{N_c-1,0}\mathbf{C} & \mathbf{H}_{N_c-1,1}\mathbf{C} & \dots & \mathbf{H}_{N_c-1,N_b-1}\mathbf{C} \end{pmatrix}\begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{N_b-1} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{C} & & & \\ & \mathbf{C} & & \\ & & \ddots & \\ & & & \mathbf{C} \end{pmatrix}\mathbf{H}\mathbf{x} \\
&= 0,
\end{aligned}$$

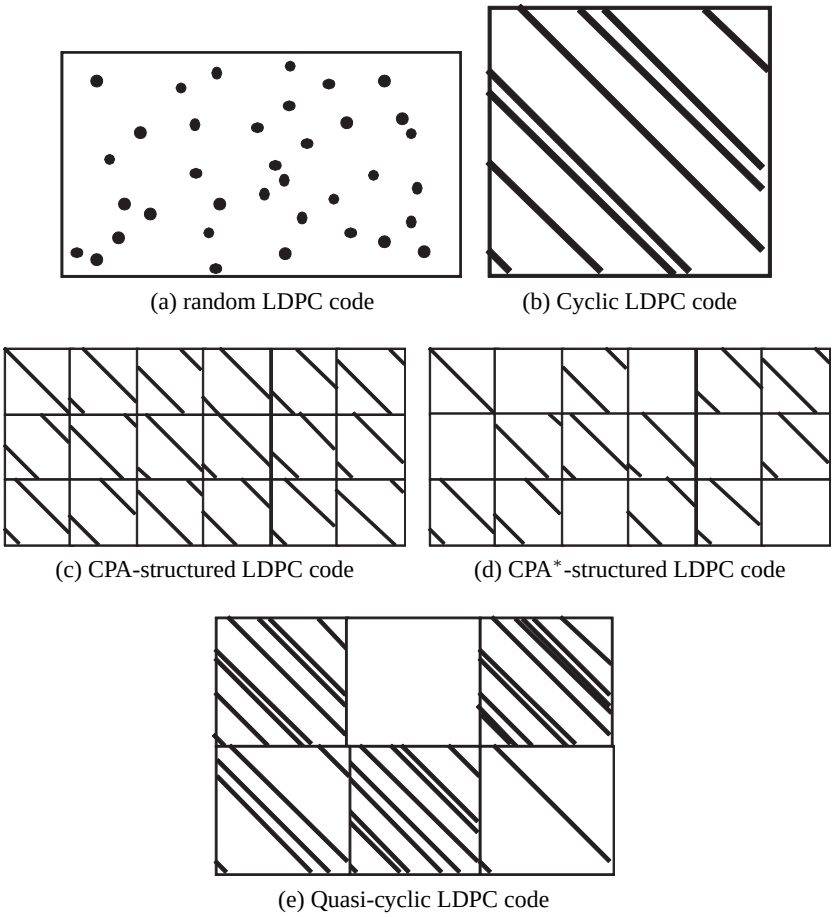
which shows that $\tilde{\mathbf{x}}$ is also in the null space of $\mathbf{H}$.

As in the case of cyclic codes, quasi-cyclic codes are of particular interest in practical systems since there exist linear-time encoders for quasi-cyclic codes that can be implemented with simple shift registers [28, 29].

CPA-structured codes: A circulant permutation array (CPA) structure is a special subclass of QC-LDPC codes and also a generalization of the array codes introduced in Chapter 1. The parity check matrix of the CPA-structured codes takes the following form:

$$\mathbf{H} = \begin{pmatrix} \mathbf{C}^{s_{0,0}} & \mathbf{C}^{s_{0,1}} & \dots & \mathbf{C}^{s_{0,N_b-1}} \\ \mathbf{C}^{s_{1,0}} & \mathbf{C}^{s_{1,1}} & \dots & \mathbf{C}^{s_{1,N_b-1}} \\ \vdots & \vdots & & \vdots \\ \mathbf{C}^{s_{N_c-1,0}} & \mathbf{C}^{s_{N_c-1,1}} & \dots & \mathbf{C}^{s_{N_c-1,N_b-1}} \end{pmatrix},$$

where $\mathbf{C}$ is a $P \times P$ single downward circulant permutation matrix and each shift value $s_{i,j}$ is in



(a) random LDPC code (b) Cyclic LDPC code

(c) CPA-structured LDPC code (d) CPA*-structured LDPC code

(e) Quasi-cyclic LDPC code

Figure 2.1: Examples of unstructured and structured parity check matrices

$\{ 0 , 1 , \ldots , P - 1 \}$. The CPA-structured code is regular, and the column weight j and row weight k are equal to N_c and N_b, respectively. We will use the notation CPA(N_c, N_b, P) to denote such code in the rest of this thesis.

CPA*-structured codes: A CPA*structure is a generalization of CPA structure by allowing $P \times P$ all-zero matrices in place of any of the submatrices in the CPA structure.

Figure 2.1 shows some example codes with the structures defined in this section, where white space represents 0's while black points or lines represent 1's in the parity check matrix. The set diagram in Figure 2.2 illustrates the relationship among these classes of codes.

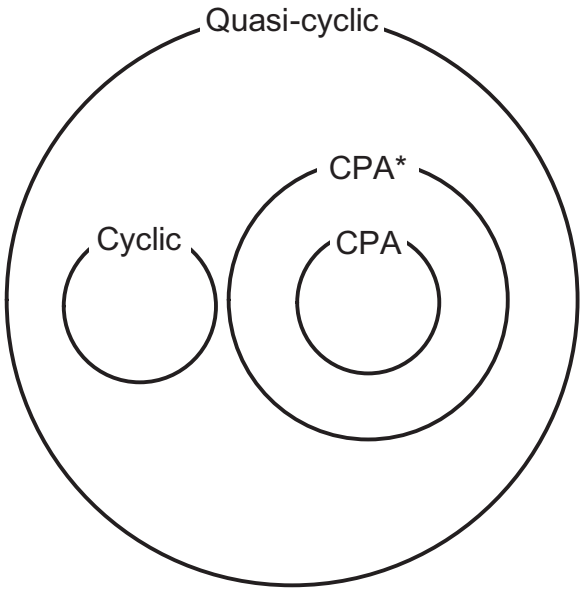


Figure 2.2: The relationship among the structured-codes

2.3 ITERATIVE DECODING ALGORITHM

2.3.1 Overview on the Sum-Product Algorithm

The sum-product algorithm is an approximation to an exact marginalization of a joint probability density function [30]. We will use a simple example to show how they are related.

Suppose there is a multi-variable function composed of factors shown as

$$g(x_1, \ldots, x_6) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5), \tag{2.6}$$

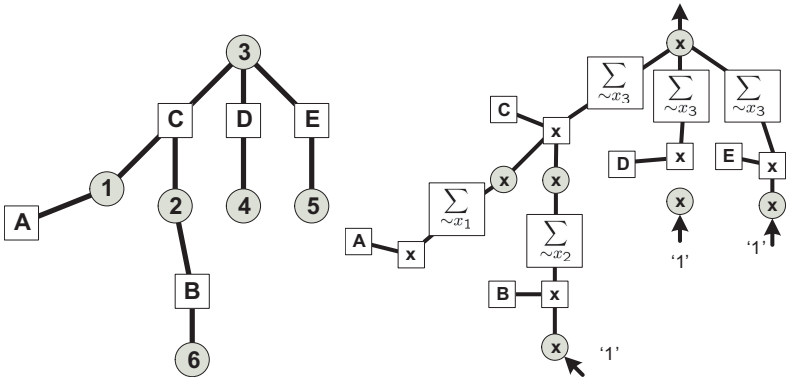
and we want to compute the marginal function $g_3(x_3)$ for x_3 defined as

$$g_3(x_3) = \sum_{x_1} \sum_{x_2} \sum_{x_4} \sum_{x_5} \sum_{x_6} g(x_1, \ldots, x_6) \tag{2.7}$$

$$= \sum_{\sim x_3} g(x_1, \ldots, x_6), \tag{2.8}$$

where the "not-sum" operator '$\sim$' indicates the variables being excluded in the summation.

By using the fact that each factor involves only some of the variables, we can manipulate



(a) Factor graph (b) Converted graph

Figure 2.3: Example of a factor graph and the corresponding conversion

eq. (2.7) to rewrite it as

$$g_3(x_3) = \left(\sum_{x_4} f_D(x_3, x_4)\right) \cdot \left(\sum_{x_5} f_E(x_3, x_5)\right) \cdot \left(\sum_{x_1, x_2} f_A(x_1) f_C(x_1, x_2, x_3) \left(\sum_{x_6} f_B(x_2, x_6)\right)\right),$$

or equivalently,

$$g_3(x_3) = \left(\sum_{\sim x_3} f_D(x_3, x_4)\right) \cdot \left(\sum_{\sim x_3} f_E(x_3, x_5)\right) \cdot \left(\sum_{\sim x_3} f_A(x_1) f_C(x_1, x_2, x_3) \left(\sum_{\sim x_2} f_B(x_2, x_6)\right)\right).$$

The expression in eq. (2.9) shows the order of computation that can reduce the total number of operations. In [30], it has been shown that such an expression can be obtained from a graph representation of the function $g(x_1, \ldots, x_6)$ in a straight-forward manner. The graph representation is called a *factor graph*, and it is a bipartite graph where the variables are mapped to variable nodes, the factors are mapped to factor nodes, and an edge between x_i and $f_j(\cdot)$ indicates that $f_j(\cdot)$ has x_i as its argument. The factor graph for eq. (2.7) is shown Figure 2.3a.

Regarding the variable x_3 as the root of a tree, the order of the computation can be obtained by traversing the graph from the leaf nodes to the root node. Each variable node corresponds to the multiplication of all incoming messages, and each factor node corresponds to the multiplication

of all incoming messages and the local function (factor) followed by a not-sum operation over the parent variable node. A message can be regarded as the evaluation of a local function for all possible values in the alphabet. The factor graph after this mapping is shown in Figure 2.3b.

In summary, the variable-to-factor message that is generated at each variable node can be computed as

$$\pi_{v\to f}(x) = \prod_{f'\in\mathcal{N}(v)\setminus f} \pi_{f'\to v}, \tag{2.9}$$

where $\mathcal{N}(v)$ denotes the neighbor nodes of the variable node v and '$\setminus$' denotes exclusion. Likewise, the factor-to-variable message generated at each factor node can be computed as

$$\pi_{f\to v}(x) = \sum_{\sim x} \left(f(\mathcal{N}(f)) \prod_{v'\in\mathcal{N}(f)\setminus v} \pi_{v'\to f} \right), \tag{2.10}$$

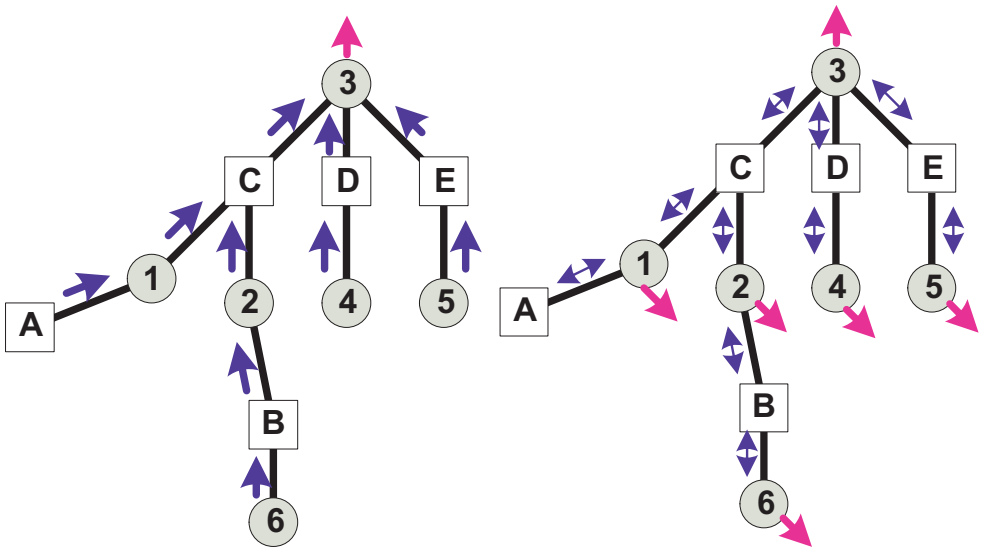
where $\mathcal{N}(f)$ denotes the neighbor nodes of the variable node f.

As previously stated, the marginalization for a single variable can be done by setting it as the root, and propagating messages from the leaves to the root, as shown in Figure 2.4a. Interestingly, the marginalization for all variables can be performed simultaneously by propagating messages from all leaves to all other leaves as shown in Figure 2.4b. In this manner, intermediate results can be reused.

When the factor graph is cycle-free, as in the previous example, the sum-product algorithm computes the exact marginal function, and the algorithm stops when all messages have propagated from all leaf nodes to all other leaf nodes, i.e., when all edges have delivered messages in both directions. On the other hand, with cycles in the graph, there is no clear condition for stopping the algorithm, and the result is not the exact marginal function. However, when the sum-product algorithm is applied to the decoding of channel codes, it has been empirically shown that, with a sufficiently large number of iterations, the results can closely approximate the true marginal functions [1].

Let $\mathbf{x}$ be the codeword vector and $\mathbf{y}$ be the observed channel output. If we apply the symbol-by-symbol maximum a posteriori (MAP) detection, the optimal decoding is defined as

$$\hat{x}_i = \arg \max_{x_i\in\{0,1\}} p(x_i|\mathbf{y}), \tag{2.11}$$



(a) Marginalization for a single variable (b) Marginalization for all variables

Figure 2.4: Propagation of messages in sum-product algorithm

where

$$p(x_i|\mathbf{y}) = \sum_{\sim x_i} p(\mathbf{x}|\mathbf{y}) \tag{2.12}$$

$$= \sum_{\sim x_i} \frac{p(\mathbf{x})p(\mathbf{y}|\mathbf{x})}{p(\mathbf{y})}. \tag{2.13}$$

Under the assumption of equiprobable codewords and memoryless channel, we can make the following substitutions:

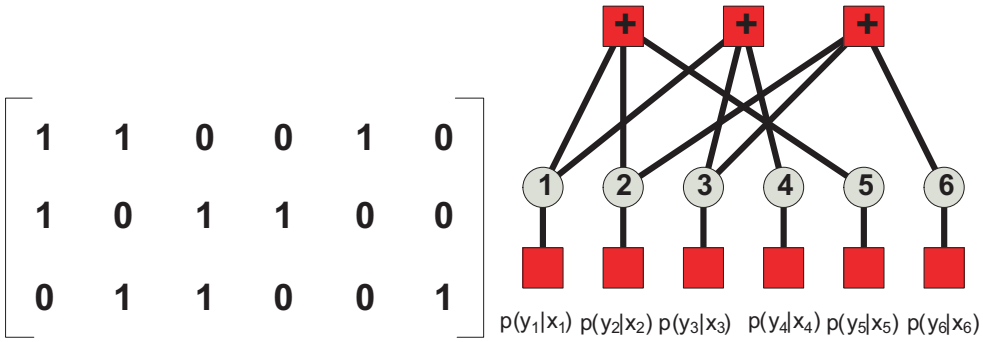
$$p(\mathbf{x}) = \frac{1}{2^k} I(\mathbf{x} \in \mathcal{C}); \tag{2.14}$$

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} p(y_i|x_i), \tag{2.15}$$

where $I(t)$ is an indicator function which gives 1 if the argument t is true and 0 if t is false. Accordingly, $I(\mathbf{x} \in \mathcal{C})$ indicates the membership of a vector $\mathbf{x}$ in a set of valid codewords $\mathcal{C}$.

Now, the function to be marginalized in eq. (2.12) is

$$g(\mathbf{x}) = I(\mathbf{x} \in \mathcal{C}) \prod_{i=1}^{n} p(y_i|x_i), \tag{2.16}$$



(a) Parity check matrix (b) Factor graph

Figure 2.5: Example of a code and the corresponding factor graph

where the scaling factors $1/2^k$ and $p(\mathbf{y})$ have been dropped since they are constant for a given observation $\mathbf{y}$.

To show the application of the sum-product algorithm to the decoding of binary linear codes, consider another example in Figure 2.5. From the parity check matrix in Figure 2.5a, the indicator function can be represented as the product of factors, each of which corresponds to a check equation, i.e.,

$$I(\mathbf{x} \in \mathcal{C}) = I(x_1 + x_2 + x_5 = 0) \cdot I(x_1 + x_3 + x_4 = 0) \cdot I(x_2 + x_3 + x_6), \tag{2.17}$$

where the additions are in GF(2). The corresponding factor graph is shown in Figure 2.5b. Note that the Tanner graph is equivalent to the factor graph except that the factor nodes for the conditional probabilities $p(y_i|x_i)$'s are explicitly shown.

For the case of binary codes, the variable nodes are called bit nodes and the factor nodes are called check nodes. For this special case, the alphabet of the variables are in GF(2), and the messages can be represented as the ratio of the value at $x = 1$ to the value $x = 0$. With this simplification, the bit(variable) node computation in eq. (2.9) can be written as

$$\Lambda_{b \to c} = \rho_b \prod_{c' \in \mathcal{N}(b) \setminus c} \Lambda_{c' \to b}, \tag{2.18}$$

where the input likelihood ratio ρ_b is defined as

$$\rho_b \quad = \quad \frac{p(y_i|x_i=1)}{p(y_i|x_i=0)}.$$

Likewise, the check (factor) node operation in eq. (2.10) can be simplified as

$$\Lambda_{c\to b} \quad = \quad \bigoplus_{b'\in\mathcal{N}(c)\backslash b} \Lambda_{b'\to c}, \tag{2.19}$$

where $\bigoplus$ denotes a repetitive application of the special binary operator $\oplus$, which is defined for eq. (2.19) as

$$a \oplus b \quad = \quad \frac{a+b}{1+ab}$$

With a further simplification of representing messages in log-likelihood domain, i.e., $\lambda = \log\Lambda$ and $\mu = \log\rho$, the bit node operation can be represented as

$$\lambda_{b\to c} \quad = \quad \mu_b + \sum_{c'\in\mathcal{N}(b)\backslash c} \lambda_{c'\to b}, \tag{2.20}$$

and the check node operation can be expressed as

$$\lambda_{c\to b} \quad = \quad F\Big(\bigoplus_{b'\in\mathcal{N}(c)\backslash b} F(\lambda_{b'\to c})\Big), \tag{2.21}$$

where $\bigoplus$ denotes a repetitive application of the operator $\oplus$, which is *redefined* for eq. (2.21) as

$$a \oplus b \quad = \quad \mathrm{sgn}(a)\mathrm{sgn}(b)(|a|+|b|), \tag{2.22}$$

and the $\mathrm{sgn}(x)$ is the sign of x as 1 or -1. The function $F(\lambda)$ is defined as

$$F(\lambda) \quad = \quad \mathrm{sgn}(\lambda)\log\frac{e^{|\lambda|}+1}{e^{|\lambda|}-1},$$

and is shown in Figure 2.6.

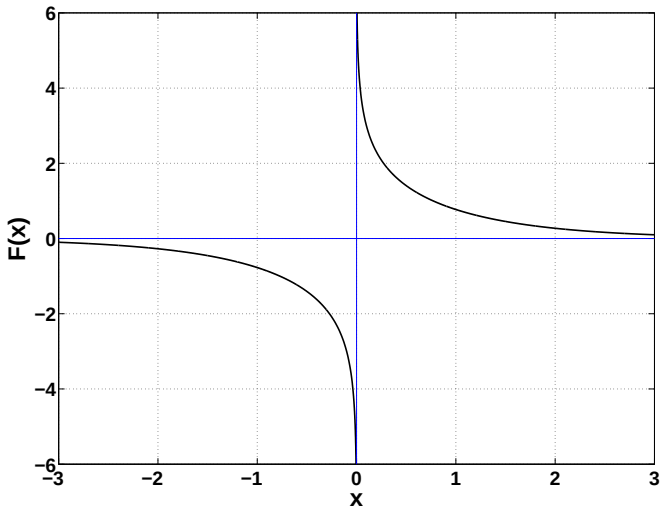


Figure 2.6: The function $F(x)$

2.3.2 Min-Sum Algorithm

The min-sum (MS) algorithm in the log-likelihood ratio (LLR) domain is an approximation of the SP algorithm [30]. First, note that $F(x)$ has the following properties:

$$|F(x)| = F(|x|);$$

$$F\left(F(x)\right) = x.$$

For check node computation, if we assume one of terms in the summation in eq. (2.21) is dominantly large, the absolute value of the LHS can be approximated as follows:

$$\begin{aligned}
|\lambda_{c\to b}| &= F\Big(\Big|\bigoplus_{b'\in\mathcal{N}(c)\backslash b} F(\lambda_{b'\to c})\Big|\Big) \\
&\approx F\left(\max_{b'\in\mathcal{N}(c)\backslash b} |F(\lambda_{b'\to c})|\right) \\
&= F\left(\max_{b'\in\mathcal{N}(c)\backslash b} F(|\lambda_{b'\to c}|)\right) \\
&= F\left(F(\min_{b'\in\mathcal{N}(c)\backslash b} |\lambda_{b'\to c}|)\right) \\
&= \min_{b'\in\mathcal{N}(c)\backslash b} |\lambda_{b'\to c}|
\end{aligned}$$

Accordingly, the MS algorithm does not require the $F(\cdot)$ function, and the bit and check node operation can be written as

$$\lambda_{b\to c} \;=\; \mu_b + \sum_{c'\in\mathcal{N}(b)\backslash c} \lambda_{c'\to b} \tag{2.23}$$

and

$$\lambda_{c\to b} \;=\; \bigodot_{b'\in\mathcal{N}(c)\backslash b} \lambda_{b'\to c}, \tag{2.24}$$

respectively, where $\bigodot$ denotes a repetitive application of the special binary operation $\odot$ defined as

$$a \odot b \;=\; \mathrm{sgn}(a)\mathrm{sgn}(b)\min(|a|,|b|).$$

The MS algorithm is known to require fewer quantization levels than the SP algorithm. It is also known that the performance of the MS algorithm can be improved by scaling the soft information after each iteration [31]. This variant is called modified min-sum (MMS) algorithm, and it has been shown that it can perform as well as the SP algorithm with 6-bit quantization [32]. In [31], a scaling factor of 0.8 was found to be optimal for $(j = 3, k = 6)$ codes. Since the latest FPGAs contain a large number of dedicated multipliers, most of which are not utilized, we used multipliers to do the scaling. In a situation where multipliers are costly resource, an approximated value can be used to replace the multipliers with adders with a little performance degradation, e.g., 0.75 instead of 0.8.

2.4 LDPC CODE CONSTRUCTION

2.4.1 Original Gallager Construction

When Gallager proposed LDPC codes, he used a pseudorandom construction method based on random permutation of predefined columns. For a given j, the parity check matrix of size $pj \times pk$ is constructed with j vertically stacked $p \times pk$ matrices. The submatrix on the top has 1's fixed at certain positions by the following rule: The i-th row, $1 \le i \le p$, has ones at the columns $(ik-k+1, ik-k+2, \ldots, ik)$, making the submatrix $(1, k)$-regular. For the rest of the submatrices, the columns of the top submatrix are copied and randomly permuted, making the parity check matrix

$$\left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&1&1&0\\
\hline
1&0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0\\
0&0&0&1&0&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&1&0&0&1
\end{array}\right]$$

Figure 2.7: An example of Gallager construction

(j, k)-regular. An example of Gallager construction is shown in Figure 2.7.

In his work, Gallager used the ensemble of randomly permuted codes to show that such a construction provides capacity-approaching codes as $N \rightarrow \infty$ [18]. Such a construction by itself does not provide a mechanism to prevent 4-cycles and the actual code construction should rely on computer search to choose suitable permutations to avoid 4-cycles. Later, very long codes were constructed and shown to closely approach the Shannon limit [1, 24].

2.4.2 Finite-Geometry Codes

There are a class of codes that are algebraically constructed based upon the underlying structure of finite geometries. Examples of such codes include Euclidean-geometry (EG) codes and Projection-geometry (PG) codes [3, 4]. In this section, we describe the EG code.

Consider an m-dimensional vector space V over GF(2^s). It contains all m-tuples of the elements in GF(2^s). Accordingly, there are 2^{ms} vectors in V. The space V is known to form an m-dimensional finite Euclidean geometry, which is called EG($m, 2^s$). Each m-tuple in the vector space is regarded as a point in the geometry. A *line* is defined as the set of points given as

$$\{\mathbf{x}_0 + a\mathbf{x} \,|a \in \text{GF}(2^s)\}. \tag{2.25}$$

The line passes through the origin if and only if $\mathbf{x}_0$ and $\mathbf{x}$ are linearly dependent. Two different lines either do not meet or intersect on exactly one point.

As a multi-dimensional generalization of lines, a μ-*flat* is defined as the set of points

$$\{\mathbf{x}_0 + a_1\mathbf{x}_1 + \cdots + a_\mu\mathbf{x}_\mu | a_1, \ldots, a_\mu \in \text{GF}(2^s)\}, \tag{2.26}$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_\mu$ are linearly independent. If the $\mu+1$ vectors, including $\mathbf{x}_0$, are linearly dependent, the μ-flat contains the origin. As in the case of lines, two different μ flats either do not meet or intersect on exactly one $(\mu - 1)$-flat.

The elements of EG$(m, 2^s)$ are also known to form finite field GF(2^{ms}). There is a one-to-one mapping between an element in the geometry and an element in the field. For example, we can define a mapping T from G=EG$(2, 2^s)$ to F=GF(2^{2s}) as

$$T : G \rightarrow F \tag{2.27}$$

$$\mathbf{x} \rightarrow T(\mathbf{x}), \tag{2.28}$$

and define T as

$$T((x_0, x_1)) = x_0 + \alpha x_1, \tag{2.29}$$

where α is the primitive element of F. The vector components x_0 and x_1 are in GF(2^s) and they are also in F since GF(2^s) is a subfield of GF(2^{2s}). All of the points in G, or equivalently all of the elements in F can be expressed as 0 or the power of α, i.e., the set of all points is

$$\{0, \alpha^0, \alpha^1, \ldots, \alpha^{N-2}\}, \tag{2.30}$$

where $N = 2^{ms}$ is the total number of points in G.

Based upon the underlying geometry EG$(m, 2^s)$, we can form an incidence vector

$$\mathbf{h}_i = (h_{i,0}, h_{i,1}, \ldots, h_{i,N-1}), \tag{2.31}$$

where $h_{i,j}$ indicates that a line i is incident on the j-th point in eq. (2.30). By collecting all of the incident vectors $\mathbf{h}_i, i = 0, \ldots, N-1$, we can form a parity check matrix $\mathbf{H}$. More generally, if we

use the incident vectors of $(\mu+1)$-flats, $\mu \geq 0$, it is called a (μ, s)-th order EG code of length 2^{ms}. An interesting property in this structure is that, if we remove the origin and all lines passing through it, the $\mathbf{H}$-matrix becomes circulant. Thus, the corresponding code becomes cyclic.

If we choose $\mu = 0$, the number of 1's in any two rows is either 0 or 1, which prevents 4-cycles in the parity check matrix. By choosing $m \geq 2$ and $s \geq 2$, the density of 1's in the codes can be lowered. Such codes are called a $(0, s)$-th order cyclic LDPC code of length $2^{2s}-1$, and known to have the following properties:

$$N = 2^{ms} - 1, \tag{2.32}$$

$$M = \frac{(2^{(m-1)s}-1)(2^{ms}-1)}{2^s - 1}, \tag{2.33}$$

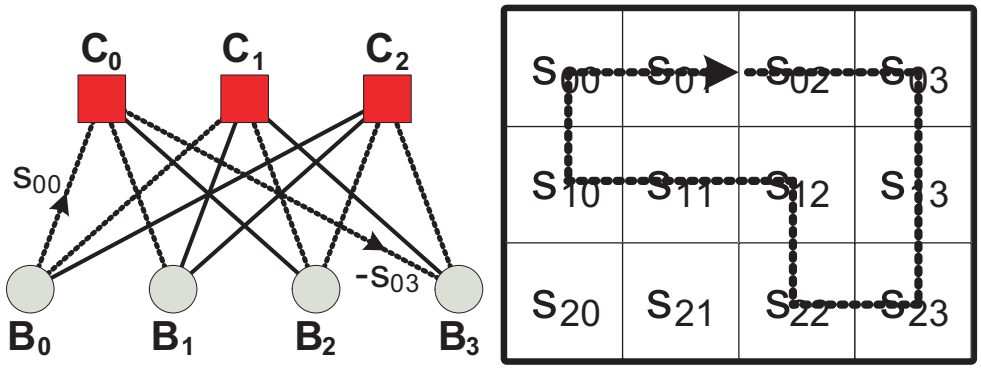
$$j = \frac{2^{ms}-1}{2^s-1}, \tag{2.34}$$

$$k = 2s. \tag{2.35}$$

For the special case of $m = 2$, the minimum Hamming distance d_{min} is exactly $j+1$, while d_{min} is lower bounded by $j+1$ in general for regular LDPC codes with girth 6 or greater. Furthermore, when $\mu = m-2$, the number of information bits K is given by

$$K = 2^{ms} - \binom{m+1}{m}^s. \tag{2.36}$$

The class of finite-geometry LDPC codes encompass a relatively large number of codes corresponding to the parameters m, s and μ, and they are known to give reasonably good or very good performance [3, 4]. Also, there are methods to extend or shorten existing finite-geometry codes to generate new parity check matrices using the techniques known as column or row splitting [4]. Some of the long extended codes have been shown to perform within a few tenths of decibel away from the Shannon limit. The performance comparison of (0,s)-th cyclic EG-LDPC codes of length $2^{2s}-1$ with CPA-structured codes will be provided in Section 5.5.



(a) Compact Tanner graph (b) **S**-matrix

Figure 2.8: Example: A 6-cycle on the compact Tanner graph and **S**-matrix

2.4.3 Finding the girth of CPA-Structured Codes

We can use a compact representation for the parity check matrix of the CPA-structured codes in eq. (2.6) by using an $N_c \times N_b$ matrix that contains the powers of $\mathbf{C}$. This matrix will be denoted as **S**-matrix. Since the pseudorandom generation of CPA-structured codes is essentially a computer search for high girth codes, it relies on a fast method for finding girth, which works on the **S**-matrix. Fan showed in [9] that, for an array structure with (not necessarily circulant) permutation submatrices, the girth can be found by examining the product of $\mathbf{H}_{ij}$ submatrices along a given cycle in the block matrix form of $\mathbf{H}$, and for circular permutation submatrices, by checking the sum of the entries in the **S**-matrix along a cycle in the **S**-matrix.

We will show by an example how the girth can be found in a CPA(N_c, N_b, P)-structured code. In the Tanner graph G, the check nodes are divided into N_c check partitions $C_i = (c_{iP}, c_{iP+1}, \ldots, c_{iP+P-1})$, $i = 0, \ldots, N_c-1$, and the bit nodes are divided into N_b partitions $B_j = (b_{jP}, b_{jP+1}, \ldots, b_{jP+P-1})$, $j = 0, \ldots, N_b - 1$. We can visualize the connections between partitions using a compact Tanner graph G' in Figure 2.8a. In G', each bit (check) supernode corresponds to a bit (check) partition, and an edge between a check supernode and a bit supernode in G' represents P edges in G. For a bit node b in B_j, denote the index of the bit node b within B_j as $\psi(b)$. Likewise, the index of c in

C_i is denoted as $\psi(c)$. Then, the indices of b and c are related by the following formulae:

$$\psi(c) = \psi(b) + S_{ij}; \tag{2.37}$$

$$\psi(b) = \psi(c) - S_{ij}, \tag{2.38}$$

where the operations '+' and '-' are defined as modulo-P addition and subtraction, respectively.

Now, consider a cycle of length 6 in G', traversing the dotted line in Figure 2.8a, which visits the partitions in the following sequence :

$$B_0 \rightarrow C_0 \rightarrow B_3 \rightarrow C_2 \rightarrow B_2 \rightarrow C_1 \rightarrow B_0. \tag{2.39}$$

If we arbitrarily choose a bit node $b_{(0)}$ in B_0 and follow the partition sequence in eq. (2.39), the 6-cycle in G' maps to a path of length 7 in G, going through exactly one node in each partition. If we label the nodes as $b_{(0)}, c_{(1)}, \ldots, b_{(6)}$ in the order in the path, we obtain the following path:

$$b_{(0)} \rightarrow c_{(1)} \rightarrow b_{(2)} \rightarrow c_{(3)} \rightarrow b_{(4)} \rightarrow c_{(5)} \rightarrow b_{(6)}. \tag{2.40}$$

This path is a cycle in G if and only if $\psi(b_{(0)}) = \psi(b_{(6)})$. This condition can be checked by calculating $\psi(b(6))$ using the formulae in eq. (2.37) and (2.38), i.e.,

$$\psi(b_{(6)}) = \psi(b_{(0)}) + s_{00} - s_{03} + s_{23} - s_{22} + s_{12} - s_{10}.$$

Accordingly, the path in eq. (2.40) is a cycle in G if and only if the cumulative shift value Δ_S is 0, i.e.,

$$\Delta_S = s_{00} - s_{03} + s_{23} - s_{22} + s_{12} - s_{10} = 0. \tag{2.41}$$

Thus, the existence of a cycle of length g in G can be determined by examining all cycles of length g in G' (or equivalently examining all cycles of length g in **S**-matrix composed of alternating horizontal and vertical edges), and checking the condition in eq. (2.41).

Tanner showed that an array structure with $N_c = j \geq 2$ and $N_b = k \geq 3$ with a group of submatrices homomorphic to a cyclic group cannot have a girth larger than 12 [12]. Consider the 12-cycle shown in Figure 2.9 which involves 6 shift values (a, b, c, d, e, f). The corresponding

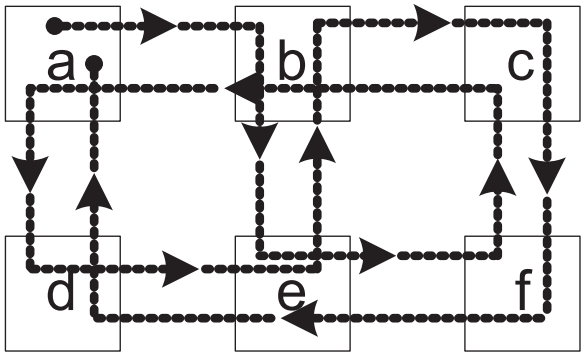


Figure 2.9: A cycle of length 12 in **S**-matrix

cumulative shift value is

$$a - b + e - f + c - a + d - e + b - c + f - d = 0, \tag{2.42}$$

which evaluates to 0 regardless of the actual values of $(a, b, \ldots, f)$. This shows that, for a CPA structure with $N_c = j \geq 2$ and $N_b = k \geq 3$, the maximum girth is 12, which arises from the commutativity of the shift values under modulo-P additions.

2.4.4 Extension to the GPA Structure

One natural generalization of the CPA structure is to allow a more general set of permutations as the submatrices of the array-structured parity check matrix. We begin by considering the case in which the submatrices $\mathbf{H}_{ij}$ in eq. (2.5) can be any permutation matrices, and describe how the girth-finding problem can be simplified by choosing a particular kind of permutation.

Arbitrary permutations: Using the example in Figure 2.8a, consider a cycle of length 6 in the compact Tanner graph G'. By the first edge connecting B_0 and C_0, the bit nodes in B_0 are connected to the check nodes in C_0 by the permutation $\mathbf{H}_{00}$. Denote the P nodes in B_0 as vector $\mathbf{b}$ and the P nodes in C_0 as vector $\mathbf{c}$, i.e.,

$$\mathbf{c} = [c_0, c_1, \ldots, c_{P-1}]^T; \mathbf{b} = [b_0, b_1, \ldots, b_{P-1}]^T, \tag{2.43}$$

where c_m(b_n) indicates the mth(nth) node in partition C_0(B_0). If we use the notation $c_m = b_n$ to denote the existence of a path from b_n to c_m along the chosen cycle in G', the connection from the

bit nodes in B_0 to the check nodes in C_0 can be collectively represented as

$$\mathbf{c} = \mathbf{H}_{0,0} \cdot \mathbf{b}. \tag{2.44}$$

Similarly, to represent the connection from C_0 to B_3, we can use the following equation.

$$\mathbf{b} = \mathbf{H}_{0,3}^{-1} \cdot \mathbf{c}. \tag{2.45}$$

After traversing all of the edges along the dotted line in Figure 2.8a, we obtain the cumulative permutation Δ_H as the following:

$$\Delta H = \mathbf{H}_{1,0}^{-1} \cdot \mathbf{H}_{1,2} \cdot \mathbf{H}_{2,2}^{-1} \cdot \mathbf{H}_{2,3} \cdot \mathbf{H}_{0,3}^{-1} \cdot \mathbf{H}_{0,0} \tag{2.46}$$

If any of the P diagonal elements in $\mathbf{H}$ is 1, there is a closed path in G that corresponds to the cycle in G'. Therefore, in general, cycle detection involves multiplications of $P \times P$ matrices.

Permutations from regular representations:

Consider a group G_P of size P defined over multiplication. If we choose an ordering of the elements of G_P, i.e., $R = (g_1, \ldots, g_P)$, then for any $g \in G$, $gR = (gg_1, \ldots, gg_p)$ is a permutation of R. This can be shown by the group property as follows. If gR is not a permutation of R, there are at least two distinct elements g_i and g_j satisfying $gg_i = gg_j$. By premultiplying both sides with g^{-1} we get $g_i = g_j$, which violates the assumption.

Now, we can define a $P \times P$ permutation matrix $\phi(g)$ which is determined by the permutation incurred by gR, i.e.,

$$(gg_1, gg_2, \ldots, gg_P)^{\mathrm{T}} = \phi(g)(g_1, g_2, \ldots, g_P)^{\mathrm{T}}. \tag{2.47}$$

The set of all possible $\phi(g)$'s gives a group of permutations

$$\phi(G_P) = \{\phi(g) : g \in G_P\}. \tag{2.48}$$

The mapping $\phi(g)$ from G to $\phi(G)$ preserves the group structure by group isomorphism. Hence, the operation on the elements of $\phi(G_P)$ can be replaced by the operation on the elements of G_P.

The representation $\phi(g)$ is called *regular representation* in group theory, and it has a well-

known property that $\phi(g)$ has a fixed point if and only if $g = 1$. This implies that the product of the permutation matrices chosen from $\phi(G_p)$ cannot have a nonzero diagonal entry unless it is an identity matrix. This can be shown by the following. Suppose there is a nonzero diagonal entry h_{ii} in $\phi(g)$, where g is not an identity. From eq. (2.47), this means that $gg_i = g_i$. This contradicts the assumption that g is not an identity.

Using this property, if we use the elements of the G_p as the entries in the $\mathbf{S}$-matrix, the existence of a cycle in the Tanner graph of the corresponding $\mathbf{H}$-matrix can be detected by checking if the product of the group elements in the $\mathbf{S}$-matrix along a cycle is an identity. The idea for this simplified checking was initially given in [9].

By using the regular representations of a group to form the parity check matrix, we can construct a new class of array-structured LDPC codes, which we will call group permutation array (GPA) structured codes. It can be noticed that the CPA-structured code is merely a special case of GPA where we choose a cyclic group of size P as the underlying group. However, we will use GPA to denote the permutation matrices based on non-cyclic group for the rest of the thesis.

One direct benefit of GPA over CPA is that, by choosing a non-commutative group,the limitation of girth 12 does not apply. Thus, it is expected to achieve a higher girth. The GPA structure is not quasi-cyclic, thus it is more "random" than the CPA-structured codes. Whether it would be possible to devise an efficient encoding algorithm or a efficient hardware decoder architecture remains to be seen and entirely dependent on the underlying group structure.

2.5 A NONPARAMETRIC TEST

In Chapter 5, we discuss the effect of diameter on the performance. To remove the artifact of explicit efforts to generate codes with large diameters, we use a large number of codes uniformly sampled from the code space of CPA-structured codes. From the observations made on the error performance of codes with different diameters, statistical significance will be calculated based on nonparametric hypothesis testing. In this section, we introduce the statistical tools that will be used

in Chapter 5.

2.5.1 Ranking Statistics

For hypothesis testing or estimation problems using samples obtained from an unknown distribution or multiple distributions, it is often necessary to rely on distribution-free statistics whose distribution remains unchanged over the underlying distribution of the random variables being assessed. One of common techniques to construct distribution-free statistics is the ranking of sample observations. We will briefly discuss the basic properties of ranks.

Let $X = (X_1, \ldots, X_N)$ denote the vector of random samples, each independently drawn from the same but unknown continuous distribution with cumulative distribution function $F(x)$. Let $R = (R_1, \ldots, R_N)$ be the vector of the ranks of X, i.e., R_i denotes the rank of X_i in X. Then R takes any one of all possible permutations of $(1, \ldots, N)$ with equal probability, i.e.,

$$P(R = \mathbf{r}) = P\left(R_1 = r_1, \ldots, R_n = r_N\right), \tag{2.49}$$

where $\mathbf{r} = (r_1, \ldots, r_N)$ is any permutation of $(1, \ldots, N)$. With the continuity assumption on X_i, we can disregard the possibility of obtaining a tie in the ranks. From this property in eq. (2.49), it follow that

$$P(R_i = r) = \begin{cases} \frac{1}{N}, & r \in \{1, \ldots, N\} \\ 0, & \text{otherwise,} \end{cases} \tag{2.50}$$

and for $i \neq j$,

$$P(R_i = r, R_j = s) = \begin{cases} \frac{1}{N(N-1)}, & r, s \in \{1, \ldots, N\},\ r \neq s \\ 0, & \text{otherwise.} \end{cases} \tag{2.51}$$

From eq. (2.50) and (2.51), the mean, variance and covariance also follow as

$$\begin{aligned} E[R_i] &= \sum_{r=1}^{N} r P(R_i = r) \\ &= \frac{N+1}{2}, \end{aligned}$$

$$\begin{aligned} Var[R_i] &= E[R_i^2] - E[R_i]^2 \\ &= \sum_{r=1}^{N} r^2 P(R_i = r) - \frac{(N+1)^2}{4} \\ &= \frac{(N+1)(N-1)}{12}, \end{aligned}$$

and for $i \neq j$,

$$\begin{aligned} Cov[R_i, R_j] &= E[R_i R_j] - E[R_i]^2 \\ &= \sum_{r} \sum_{s \neq r} \frac{rs}{N(N-1)} - \frac{(N+1)^2}{4} \\ &= -\frac{N+1}{12}. \end{aligned}$$

A test statistic $T\left(R(X)\right)$, which is dependent upon X only through R, has a distribution independent of the distribution of X, and is called a rank statistic.

2.5.2 Mann-Whitney-Wilcoxon Test

We show the use of the rank statistic in a distribution-free hypothesis test for the two-sample location problem. Suppose you have two vectors of independent random samples, $X = (X_1, \ldots, X_m)$ and $Y = (Y_1, \ldots, Y_n)$, drawn from continuous distributions with distribution functions $F(x)$ and $F(x - \theta)$, respectively, that is, two identical distribution functions with an unknown shift of θ. We want to test the hypothesis that the two sample vectors come from the same distribution, i.e.,

$$\begin{aligned} H_0 : &\quad \theta = 0 \\ H_1 : &\quad \theta > 0. \end{aligned}$$

We combine X and Y to form the vector $Z = (X_1, \ldots, X_m, Y_1, \ldots, Y_n)$ of length $N = m+n$. Let $Q = (Q_1, \ldots, Q_m)$ and $R = (R_1, \ldots, R_n)$ denote the ranks of $X_i's$ and $Y_j's$ in the combined vector Z, respectively. Under the null hypothesis H_0, the combined rank vector $(Q_1, \ldots, Q_m, R_1, \ldots, R_n)$ follows the properties in the previous section.

Now, we construct a test statistic based upon the ranks. We consider the sum of the ranks of Y_i's

in R, called *ranksum*, i.e.,

$$W = \sum_{i=1}^{n} R_i,$$

which was proposed by Wilcoxon [33]. We can also use an equivalent statistic suggested by Mann and Whitney [34], which is given by

$$\begin{aligned} U &= \sum_{i=1}^{m}\sum_{j=1}^{n} I(Y_j - X_i) \\ &= W - \frac{n(n+1)}{2}, \end{aligned}$$

where $I(t)$ is the step function defined as

$$I(t) = \begin{cases} 1, & t > 0 \\ 0, & t \leq 0. \end{cases}$$

Since U differs from W only by a constant, we choose to use W for the discussion.

Under the null hypothesis ($H_0 : \theta = 0$), the discrete random variable W has the following distribution.

$$P(W = w|H_0) = \begin{cases} \frac{t_{N,n}(w)}{\binom{N}{n}}, & w = \frac{n(n+1)}{2}, \frac{n(n+1)}{2}+1, \ldots, \frac{n(N+n+1)}{2} \\ 0, & \text{otherwise}, \end{cases}$$

where $t_{N,n}(w)$ is the number of unordered subsets of n numbers taken from $\{1, \ldots, N\}$ whose sum is w. This indicates that the ranksum W is distribution-free under the null hypothesis for the unknown distribution $F(x)$. The value of $t_{N,n}(w)$ can be found by enumerating all $\binom{N}{n}$ combinations and counting those that sum to w, and the values are tabulated for $N \leq 20$ in the literature (see [35]).

It is also known that, under H_0, the mean of W is

$$E[W|H_0] = \frac{n(n+m+1)}{2} \tag{2.52}$$

and

$$Var[W|H_0] = \frac{nm(n+m+1)}{12}. \tag{2.53}$$

Using eq. (2.52), the null mean and null variance of U can be calculated as the following:

$$E[U|H_0] = E[W|H_0] - \frac{n(n+1)}{2} = \frac{mn}{2}, \tag{2.54}$$

$$Var[U|H_0] = Var[W|H_0] = \frac{mn(m+n+1)}{12}. \tag{2.55}$$

For larger values of N, a Gaussian approximation is known to provide accurate results, i.e., U follows the normal distribution with the mean and variance in eq. (2.54) and eq. (2.55), respectively. Given a data point u calculated from the observation X and Y, we can compute the probability of observing a value of U that is at least as extreme as u, i.e.,

$$p = P(U \geq u|H_0) = Q\left(\frac{u - \frac{mn}{2}}{\sqrt{\frac{mn(m+n+1)}{12}}}\right), \tag{2.56}$$

where $Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,\mathrm{d}x$.

The probability p in eq. (2.56) is called *p-value*, and used as an indicator of the statistical significance of the observed data under the null hypothesis. A small value of p makes the null hypothesis unlikely. Usually it is compared with a predefined significance level α, and the null hypothesis is rejected if p is smaller or equal to α. In other words, the null hypothesis is rejected if the observed value of U is greater than equal to the threshold $u_0(\alpha)$, which is defined as the upper 100α-th percentile point in the distribution of U, i.e., $P\left(U \geq u_0(\alpha)|H_0\right) = \alpha$. Thus, the significance level α is the probability of falsely rejecting the null hypothesis when it is true.

This test is called Mann-Whitney U test, Mann-Whitney-Wilcoxon test or Wilcoxon ranksum test. The form of p in eq. (2.56) is for one-sided test. The modification for two-sided test is straightforward. To test the hypothesis

$$H_0 : \qquad \theta = 0$$

$$H_1 : \quad \theta > 0 \quad \text{or} \quad \theta < 0,$$

the p-value is calculated by considering both tails of the Gaussian distribution, i.e., it is the value of

p that satisfies

$$P(U \geq u|H_0) = \frac{p}{2} \tag{2.57}$$

for a given observed data point u.

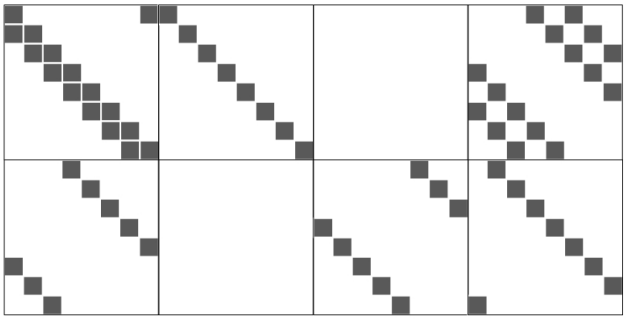
CHAPTER **3**

ANALYSIS OF QC-LDPC CODES: GIRTH AND DIAMETER

Girth and diameter are important graph parameters that can potentially affect the performance of LDPC codes [12, 18] In Chapter 5, we will explore the effect of the two parameters on LDPC code performance. In general, the problem of finding cycles up to length g has complexity of $O\left(N(jk)^{g/2}\right)$, where j and k are the column and row weights, respectively, and N is the codeword length. The problem of finding the diameter has complexity of $O(N^3)$. Thus, it can be too time-consuming to calculate girth and diameter even for moderately long codes ($1000 \leq N \leq 10000$). However, for structured codes, we can expect to find much more efficient algorithms to calculate girth and diameter, if the underlying structure is properly exploited.

In Section 2.4.3, we have shown that, for CPA-structured codes, it is possible to efficiently find the girth by examining cycles in the **S**-matrix, which is a very compact representation of the corresponding **H**-matrix. This algorithm has a runtime of $O\left(N_b(jk)^{g/2}\right)$, thus reducing the number of operations by a factor of P. This is helpful for CPA-structured codes where the girth is limited by 12 and j and k are typically small constants. However, this method can be too costly for CPA*-structured codes which may have larger girth.

In this section, we will introduce two new algorithms for general QC-LDPC codes: a girth-finding algorithm with runtime of $O(jN_bNg)$ and a diameter-finding algorithm with a runtime of $O\left((N_b + N_c)^3P^2\right)$. These algorithms can also be used for any subclass of QC-LDPC codes,



(a) The **H**-matrix

0,1	0	-1	3,5
5	-1	3	7

(b) The corresponding extended **S**-matrix

Figure 3.1: An example of QC(2,4,8) LDPC code

including CPA and CPA*structured codes. We will begin with a discussion of QC-LDPC code representation and then provide the details of the algorithms.

3.1 QC-LDPC CODE REPRESENTATION

In Chapter 1, we introduced the **S**-matrix representation for the CPA-structured LDPC codes. It cannot be used for QC-LDPC codes since there can be more than one 1's in each column (or row) in the $P \times P$ submatrices of the parity check matrix. For a given QC-LDPC code, the parity check matrix $\mathbf{H}$ can be put into an array form, i.e., an array of $P \times P$ circulant submatrices $H_{i,j}$, $i = 0, \ldots, N_c - 1$, $j = 0, \ldots, N_b - 1$. Due to the property of circulant matrices, each submatrix is either the sum of one or more circulant permutation matrices or a $P \times P$ all-zero matrix. With a slight abuse of the **S**-matrix representation, the parity check matrix can be represented in the extended **S**-matrix form, in which multiple shift values can be specified in each entry. For example, a QC-LDPC code with $N_c = 2$, $N_b = 4$ and $P = 8$ and the corresponding extended **S**-matrix are shown in Figure 3.1a and 3.1b, respectively.

3.2 FINDING GIRTH

The array structure of a QC-LDPC code provides natural partitioning of the bit nodes into N_b partitions and the check nodes into N_c partitions. Consider a compact Tanner graph consisting of

N_b bit nodes and N_c check nodes where the i-th (j-th) bit (check) supernode in this compact graph represents P bit (check) nodes in the original Tanner graph. Then, for each of the nonnegative element in the extended $\mathbf{S}$-matrix, there is an edge in the compact Tanner graph. First, we denote the nonnegative shift values in the extended $\mathbf{S}$-matrix as $s_k, k = 0, \ldots N_s - 1$, where N_s is the total number of nonnegative shift values in the extended $\mathbf{S}$-matrix, and the edge corresponding to s_k as e_k. If there is a cycle of length n in the Tanner graph, there is a sequence of shift values $\{s_{(1)}, \ldots, s_{(n)}\}$ in the corresponding cycle on the compact Tanner graph that satisfies the following condition:

$$s_{(1)} - s_{(2)} + \cdots - s_{(n)} \mod P = 0, . \tag{3.1}$$

Now, consider the set R of all possible remainder polynomials of order less than P over integer $\mathcal{Z}$, i.e.,

$$R = \{c_0 + c_1 X + \cdots + c_{P-1} X^{P-1} | c_i \in \mathcal{Z}\}. \tag{3.2}$$

Now, we define the addition and multiplication using the mod-($X^P - 1$) arithmetic as follows:

$$g(X) \oplus h(X) = g(X) + h(X) \mod (X^P - 1);$$
$$g(X) * h(X) = g(X)h(X) \mod (X^P - 1),$$

where $a(X) \bmod b(X)$ indicates the remainder of the polynomial division of $a(X)$ by $b(X)$.

The set and the operators $(R, \oplus, *)$ form a ring. The subset of R, $\{1, X, X^2, \ldots, X^{P-1}\}$ forms a cyclic group of size P under multiplication '$*$', and thus isomorphic to the additive group S of all possible shift values. If we map a shift value s to the element X^s in R, the condition in eq. (3.1) is equivalent to

$$X^{s_{(1)}} \cdot X^{-s_{(2)}} \cdot \cdots \cdot X^{-s_{(n)}} = 1, . \tag{3.3}$$

The problem of finding a cycle of length n in the original Tanner graph is equivalent to finding a cycle in the compact Tanner graph that satisfies the condition in eq. (3.3). This can be efficiently done by simulating the message passing in the compact Tanner graph that works similar to the sum-product algorithm. In this method, we regard the compact Tanner graph as a circuit, where each

edge e_k has a storage element called message $\lambda_k(X)$ and a gain $G_k(X) = X^{s_k}$. An edge is in fact bidirectional, so we will use the notation $\lambda_k(X)$ and $G_k(X)$ when messages flow from bit nodes to check nodes, and use the alternative notation $\bar{\lambda}_k(X)$ and $\bar{G}_k(X) = X^{-s_k}$ when messages flow from check nodes to bit nodes. The messages are represented as the elements of R, and they can be added and multiplied according to the mod-($X^P - 1$) rule.

The algorithm runs by propagating messages along the edges in the compact Tanner graph, starting with an outgoing message from a chosen bit node $c*$. When multiple edges $\{e_1, e_2, .., e_n\}$ are incident on a node, the message λ_m through e_m is computed as the sum of the incoming message in all of the incident edges except e_m, multiplied by the gain $G_m(X)$. In this way, multiple paths with the same cumulative shift values are merged.

When the message passing algorithm is running, a message $\lambda_k(X)$ at time t takes the following general form

$$\lambda_k(X) = \sum_{i=0}^{P-1} a_i X^i, \tag{3.4}$$

where X^i denotes the product of gains that a message has gone through from the beginning of simulation up to time t, and a_i denotes the number of messages with a cumulative gain of X_i. In this way, not only the girth but also the cycle distribution in the compact Tanner graph can be obtained.

At time t, when t is even, the messages coming back to the starting node $b*$ are summed up, and the coefficient of the X^0 term in the sum is examined. This coefficient indicates the number of cycles of length t involving the node $b*$ that satisfies the condition in eq. (3.3). If the first occurrence of a nonzero coefficient of X^0 occurs at time t, it is the shortest of the all cycles that contains $b*$, i.e., the node girth of $b*$. If the algorithm runs up to $t = g$ for the starting node $b*$, the coefficient a_0 for $t = 0, 1, \ldots, g$ is the distribution of cycles for the node $b*$. If the algorithm runs for the girth detection only, it can be stopped at the fist occurrence of the nonzero a_0. For the girth of the whole graph, this algorithm should run for all starting nodes b_j, $i = 0, .., N_b$. The minimum of the node girths is the girth of the graph.

To make a valid cycle, the message coming out of the edge e is not allowed to go into the same edge e. As a result, the message passing algorithm is very similar to the sum-product algorithm. We will use the notation e to denote an edge index, and $\mathcal{E}(b)$ ($\mathcal{E}(c)$) to denote the indices of the edges incident to bit (check) node b (c). We describe the details of the algorithm below.

- *Initialization:* At time $t = 0$, for the starting node $b*$,

$$\forall e \in \mathcal{E}(b*), \lambda_e(X) = G_e(X). \tag{3.5}$$

- *Bit node operation:* At time $t = 2, 4, \ldots,$ for each bit node b,

  First, compute the sum $\sum_{e' \in \mathcal{E}(b)} \bar{\lambda}_{e'}(X)$ and check the coefficient of X^0. If nonzero, stop here for girth only. Continue for cycle distribution.

$$\forall e \in \mathcal{E}(b), \lambda_e(X) = G_e(X) \sum_{e' \in \mathcal{E}(b) \setminus e} \bar{\lambda}_{e'}(X) \tag{3.6}$$

- *Check node operation:* At time $t = 1, 3, 5, \ldots,$ for each check node c,

$$\forall e \in \mathcal{E}(c),\ \bar{\lambda}_e(X) = \bar{G}_e(X) \sum_{e' \in \mathcal{E}(c) \setminus e} \lambda_{e'}(X). \tag{3.7}$$

Now we compute the runtime of this algorithm, assuming a regular code. During the bit node iteration, since there are N_b nodes with degree up to j, jN_b polynomial additions are necessary, resulting in jN_bP operations to be performed. Since there are jN_b edges, jN_b multiplications with a single-term gain X^{s_k} are performed, again resulting in jN_bP operations. Thus, the number of operations in a single time step is $2jN_bP$. With $jN_b = kN_c$, the same number of operations are required in the check node iteration. Since the algorithm runs for g iterations for each of the N_b starting bit nodes, the runtime for the detection of girths up to g is $O(jN_b^2Pg)$ or $O(jN_bNg)$, which is linear in both girth and column weight.

3.3 FINDING DIAMETER

Another important combinatorial property of an LDPC code is *diameter*, which is defined as the maximum, over all pairs of nodes, of the length of the shortest path between them. If the diameter is large compared to the girth, it is conjectured to adversely affect the performance since potentially useful statistically independent messages will be blurred before they propagate to the neighbor node by the dependent messages passing around the short cycles [12].

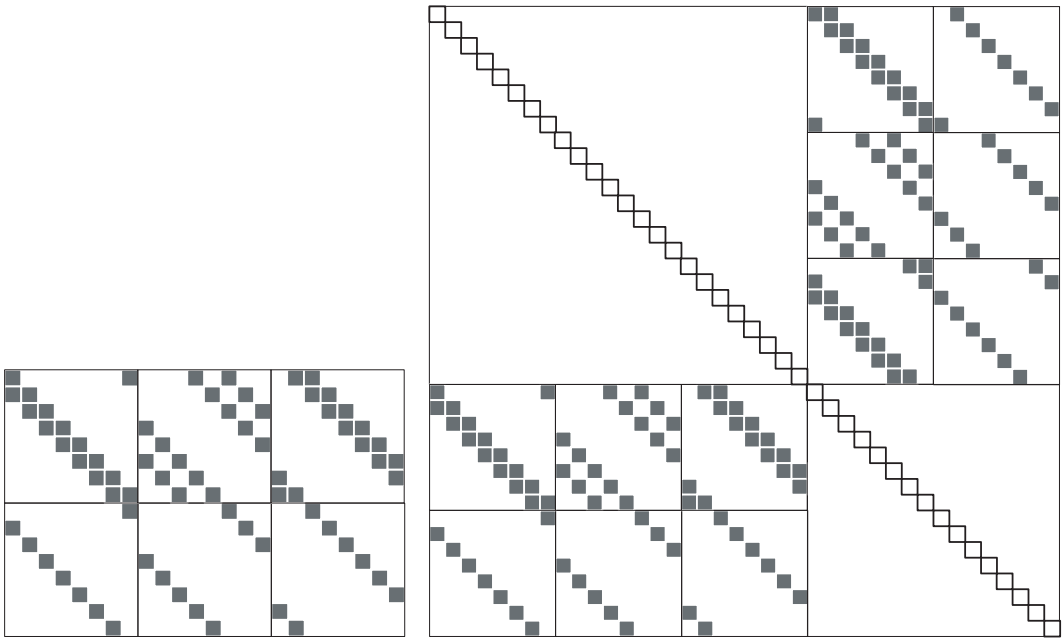
To determine the diameter of a given graph, we need to find the minimum distance between all pairs of nodes. A straight-forward way to do this is to use the Floyd-Warshall (FW) algorithm with a runtime of $Theta(N^3)$, where N is the number of the nodes. To apply the FW algorithm to the Tanner graph, we first relabel the nodes in the Tanner graph such that the indices $\{0, 1, \ldots, PN_b-1\}$ denote the bit nodes and the indices $\{PN_b, PN_b+1, \ldots, P(N_b+N_c)-1\}$ denote the check nodes. Then the $N \times N$ cost matrix C can be initialized as follows:

$$C[i][j] = \begin{cases} 0, & \text{if } i = j \\ 1, & \text{if } PN_b \leq i < P(N_b+N_c),\ 0 \leq j < PN_b, \text{ and } H[i-PN_b][j] = 1 \\ 1, & \text{if } 0 \leq i < PN_b,\ PN_b \leq j < P(N_b+N_c), \text{ and } H[j-PN_b][i] = 1 \\ \infty, & \text{otherwise} \end{cases},$$

where $N = P(N_b + N_c)$ for CPA structure.

An example of a parity check matrix and the corresponding cost matrix is shown in Figure 3.2. In Figure 3.2a, a gray square indicates '1' and white area indicate '0's. In Figure 3.2b, a gray square indicates '1', a white square indicates '0', and the remaining white area are filled with $-\infty$. Note that the initial C matrix is symmetric since the Tanner graph is undirected. It is also an array of circulant matrices, but it is not circulant as a whole.

Now, the FW algorithm can be implemented by the simple triple loop structure written in Matlab in Figure 3.3 [36]. Although there exists a faster implementation of the algorithm that gives a speed-



(a) Parity check matrix $\mathbf{H}$ (b) Cost matrix C

Figure 3.2: A parity check matrix and the corresponding cost matrix.

```
// standard FW algorithm
function FW(C, N)
  for k=1:N
    for i=1:N
      for j=1:N
        C[i][j] = min(C[i][j], C[i][k]+C[k][j]);
```

Figure 3.3: Standard FW algorithm.

up of up to 10 times with adaptive software techniques utilizing cache blocking, loop unrolling and vectorization [37], it is still too slow to be used for finding codes with lengths of practical concern (1000 to 10000). However, by exploiting the structure in the $\mathbf{H}$-matrix, we developed a fast all-pairs shortest path algorithm that can be used when the weight matrix is an array of circulant matrices. With this algorithm, we can reduce the number of computations to $O\left((N_b+N_c)^3P^2\right)$, which is faster than the original algorithm by a factor of P. The initial step of the new algorithm is the application of the tiled FW algorithm developed in [38, 39]. We will give a description of the algorithm with some details omitted.

We begin by defining a generalized version of FW as in Figure 3.4, which was introduced in [39]. It is clear that FWG(C, C, C, N) is the same as FW(C, N). However, the FWG can take

```
// generalized FW algorithm
function FWG(A, B, C, N)
  for k=1:N
    for i=1:N
      for j=1:N
        C[i][j] = min(C[i][j], A[i][k]+B[k][j]);
```

Figure 3.4: Generalized FW algorithm.

different input matrices, and is used as a subroutine in the tiled version of FW shown in Figure 3.5.

```
// tiled FW algorithm (FWT)
// tile size: P x P
function FWT(C, N, P)
  // C_ij: P x P submatrix (i,j) of C, i.e.,
  //      C[(i-1)*P:i*P-1][(j-1)*P:j*P-1];
  M = N/P;
  for k=0:1:M-1
    // Phase 1
    FWG(C_kk, C_kk, C_kk, P);
    // Phase 2
    for i=0:1:M-1, i!=k
      FWG(C_ik, C_kk, C_ik, P);
    // Phase 3
    for j=0:1:M-1, j!=k
      FWG(C_kk, C_kj, C_kj, P);
    // Phase 4
    for i=0:1:M-1, i!=k
    for j=0:1:M-1, j!=k
      FWG(C_ik, C_kj, C_ij, P);
```

Figure 3.5: Tiled FW algorithm. (P divides N.)

In [38, 39], it was proved that the FWT(C, N, P) generates the same result as FW(C, N). The FWT was originally developed for blocking a large weight matrix for better cache performance, but we use it for a different purpose. First, we apply the FWT to the weight matrix. This blocking decomposes the original problem into a sequence of subproblems where the input matrices are circulant. We can show that, for FWG, if all inputs are circulant, the output is also circulant. Since each submatrix C_ij in Figure 3.5 is circulant before and after the execution of FWG, we need to store only the first row of each submatrix, which reduces the storage requirement by a factor of P, from N^2 to PN_cN_b. For the phase 1, since the FWG takes three identical matrices C_kk, it can be replaced with the modified Dijkstra's algorithm shown in Figure 3.6, which computes the shortest distance from the first source node to all destination nodes, i.e., the first row of the FWG result.

For phase 2, when the second input matrix is different from the other two, we can apply DIJK_CBC, a slightly different version of Dijkstra's algorithm, shown in Figure 3.7. A variation of

```
function DIJK_CCC(c, P)
  // c: P x 1 vector where
  //    c[i] denotes the distance from node c_0 to node c_i
  // W: set of permanently labeled nodes
  W = {0};
  for i=1:1:P-1
    x = argmin c[j] for all j not in W
    Add x to W
    for all j not in W
      c[j] = MIN(c[j], c[x] + c[(j-x+P)%P]);
```

Figure 3.6: Modified Dijkstra's algorithm for phase 1.

Dijkstra's algorithm for phase 3 can be similarly derived (not shown).

```
function DIJK_CBC(b, c,  P)
  // b: P x 1 vector where
  //    b[i] denotes the distance from node b_0 to node b_i
  // c: P x 1 vector where
  //    c[i] denotes the distance from node c_0 to node b_i
  // W: set of permanently labeled nodes
  W = {};
  for i=0:1:P-1
    x = argmin c[j] for all j not in W
    Add x to W
    for all j not in W
      c[j] = MIN(c[j], c[x] + b[(j-x+P)%P]);
```

Figure 3.7: Modified Dijkstra's algorithm for phase 2.

Finally, for phase 4 where all input matrices are different, we can apply DIJK_ABC shown in Figure 3.6.

```
function DIJK_ABC(a, b, c, P)
  // a: P x 1 vector where
  //    a[i] denotes the distance from node a_0 to node c_i
  // b: P x 1 vector where
  //    b[i] denotes the distance from node c_0 to node b_i
  // c: P x 1 vector where
  //    c[i] denotes the distance from node a_0 to node b_i
  // W: set of permanently labeled nodes
  for i=0:1:P-1
    for j=0:1:P-1
      c[j] = MIN(c[j], a[i] + b[(j-i+P)%P]);
```

Figure 3.8: Modified Dijkstra's algorithm for phase 4.

By replacing the FWG subroutines in Figure 3.5 by the corresponding Dijkstra routines, we have derived a fast all-pairs shortest path algorithm for a circulant-array weight matrix.

3.4 CPA CONSTRUCTION WITH GIRTH AND DIAMETER

We showed that, for CPA-structured codes, the girth can be found by examining cycles in the shift matrix $\mathbf{S}$, and the diameter can be found by running a modified Dijkstra's algorithm. We incorporated a diameter constraint into the pseudorandom generation algorithm based on girth proposed in [15]. We briefly describe the algorithm for a CPA structure without all-zero matrices as follows:

1. Initialize the first row and the first column of the $\mathbf{S}$-matrix as 0 without loss of generality (since every CPA-structured matrix can be converted to this form with row and column permutations).

2. Set one of the empty $\mathbf{S}$ entries to a randomly generated number s from 0 to $P-1$. Record this number.

3. Check the girth and diameter. If there is no violation, repeat step 2.

4. If the girth or diameter constraint has been violated, try a different s value. If a predefined limit has been reached, empty the current $\mathbf{S}$-matrix entry and backtrack to the previously determined $\mathbf{S}$-matrix entry by repeating step 2.

With this simple algorithm, we could generate several codes with the same code parameters but with different girths and diameters. The performance of the generated codes will be discussed in Chapter 5.

CHAPTER 4

LDPC DECODER IMPLEMENTATION

4.1 INTRODUCTION

While the powerful error correction capability of the LDPC codes has drawn a lot of research interests in the aspect of code performance, the availability of the highly parallelizable decoding algorithm has brought as much interest to the design of efficient hardware implementation. Besides the use of the decoding hardware for the deployment in practical systems, another important usage is to evaluate a given code, usually as part of the design process.

Although there exists an approximate analytical method called density evolution to predict the performance of LDPC code with iterative decoding algorithm, it has to rely on the assumption that the codeword length tends to infinity to make the graph essentially cycle-free [24, 26]. Also, the analytical methods for predicting the waterfall-region performance of finite-length codes in [27] do not consider the error floor. With no analytical methods known to exist that can predict the error performance of a given finite-length LDPC code with finite-precision decoding algorithm, the performance evaluation is often carried out by resorting to Monte-Carlo simulation. Accordingly, the primary goal of a hardware-based implementation for evaluation purposes is the ability to evaluate a given code at much higher speed than is reachable by software-based simulation, which would be critical to exploring the performance in a very low BER regime or for evaluating a large number of codes for design purposes. In addition, the decoder architecture must be flexible with respect to code parameters so as to evaluate a large class of codes.

The execution of the sum-product algorithm involves the generation of bit messages and check messages in multiple iterations. Each iteration consists of two steps: bit node operation and check node operation. For the bit node operation, the outgoing bit messages from each bit node are computed from the incoming check messages. Similarly, the check node operation is to compute outgoing check messages from each check node. The number of bit or check messages to be produced and consumed in each iteration is the same as the number of 1's in the parity check matrix. The messages in a given iteration are computed from the messages in the previous iteration, which makes it possible to choose any order of processing within the same iteration without affecting the result.

The parallelism in the sum-product algorithm is inherently favorable to hardware implementation. Using multiple computation units working in parallel, the hardware implementations with medium-capacity FPGAs are often faster than software simulation using general-purpose CPUs by at least one or two orders of magnitude. While a higher degree of parallelism increases the throughput of the decoder, it also incurs a larger overhead for the interconnection between memory elements and computation units. For example, the highest degree of parallelism can be achieved by a fully parallel architecture in which all of the messages are processed at the same time; however, it results in a very costly implementation in terms of the routing and storage resource. The other extreme, a fully serial architecture in which one message is processed at a time, is just too slow for practical purposes. Accordingly, most practical implementations employ a partially parallel architecture, where the degree of parallelism and the scheduling of the computation are chosen for a proper performance-resource tradeoff.

In general, for structured LDPC codes, the interconnection problem in the decoders can be greatly simplified if the bus connections are designed to match the structure of the code. Accordingly, the fastest decoders tend to put more restrictions on the structure of the supported codes. However, for certain applications such as wireless communication, the code parameters should be flexible to adapt to varying channel conditions.

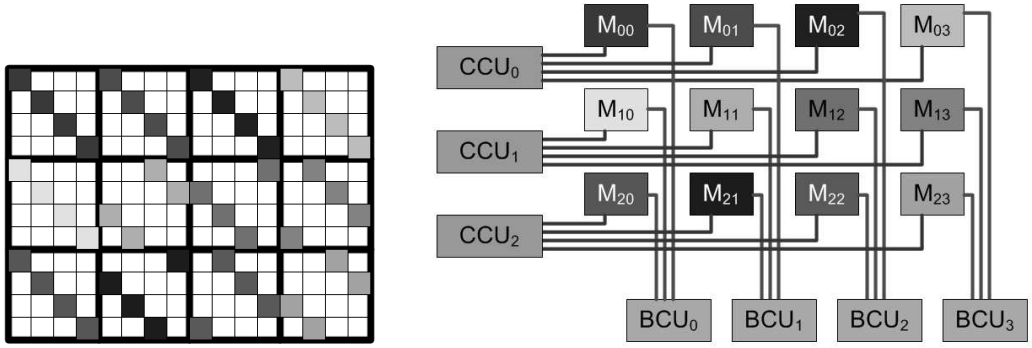
In this section, we propose a hardware decoder for array-structured LDPC codes. The decoder has been designed for scalability in terms of the computational power and the resource usage, that is,

major architectural parameters such as the number of computational units, the number and the depth of the memory blocks can be set by the user before the synthesis. These architectural parameters can be chosen independently from the code parameters unlike other throughput-oriented architectures in which the two sets of parameters are closely coupled. In fact, in our proposed architecture, the actual code parameters can be changed on the fly while the decoder is running.

4.2 DECODER ARCHITECTURES IN LITERATURE

A fully parallel architecture can achieve the highest throughput by mapping each bit and check node in the Tanner graph to a separate computation unit and processing all of the data that are involved in an iteration in parallel. This type of architecture can be seen in [40], where the decoder has been implemented in 0.16 μm CMOS process. Besides the high implementation cost, this architecture has bus connections that are directly associated with a specific code structure, making it very difficult to support a set of different codes. On the other hand, any software-based decoder can be regarded as a fully serial architecture. Since the parallelism is poorly utilized in this architecture, the performance is limited by the clock frequency of the processor in the system, which is often too slow even with the fastest processors available. However, since only a very small number of data need to be fetched from and written back to memory, there is no memory conflict problem and the scheduling of operations is trivial.

Between these two extremes lies the category of partially parallel architectures, where a reasonably small number of data are processed at a time and the computation units are reused over time. In order to fully utilize the potential throughput provided by a large number of computation units, the data to be processed should be accessible at the same time, which requires the data involved to be stored in separate memory blocks. While small memory elements can be implemented with flipflops, they are far less efficient than memory blocks in storing a large amount of data. For VLSI design, it is desirable to have a smaller number of larger memory blocks since a larger memory is more efficient in terms of the area per bit. For FPGAs, there is a limited number of dedicated



(a) **H**-matrix for a CPA(3,4,4) code (b) Memory-CU bus connections

Figure 4.1: CPA(3,4,4) code and the corresponding memory architecture

memory blocks called block RAMs that can be simultaneously accessed. Accordingly, a main goal in FPGA architectural design is to maximize the utilization of the computation units while keeping the number of memory blocks reasonably low.

For array-structured codes, a majority of the decoders employ bus connections and operation scheduling that match the array structure of the parity check matrix. In this architecture, ($N_c \times N_b$) memory blocks are used for (N_c, N_b, P)-structured CPA codes. The data buses are connected horizontally and vertically to provide memory access to N_c check computation units (CCUs) and N_b bit computation units (BCUs). For example, Figure 4.1 shows the memory architecture for CPA(3,4,4), where the messages corresponding to a submatrix (i, j) in Figure 4.1a is stored in a single memory block M_{ij} in Figure 4.1b. Sun used a similar memory architecture in [41] where multiple memory blocks were used for each submatrix to support a more general class of array-structured LDPC codes, where each submatrix can be any regular (i.e., uniform column weight) matrix .

The degree of parallelism can be multiplied by a factor of s by using $s \cdot N_c$ CCUs and $s \cdot N_b$ BCUs. While this normally requires s memory blocks for each submatrix, the number of memory blocks need not increase if multiple messages are stored in a single memory address. In [32], $s = 16$ is implemented by putting 8 messages in each memory address and utilizing the two ports of the Xilinx dual-port block RAMs. However, this technique also puts a serious limit in the code

structure. To guarantee that any $s/2$ messages in a single access are in the same memory address, it is required that the shift values in a $\mathbf{S}$-matrix are multiples of $s/2$.

With an array-style memory architecture as described above, the operands for each check or bit computation can be accessed in a single cycle since the operands belong to different memory blocks. To utilize this memory bandwidth, a BCU (or CCU) is usually designed as a multi-operand adder that can process all of the messages in the same column (or row) in a single cycle. However, as the number of operands increases, the long combinational path created by the adders may deteriorate the maximum operating clock frequency. Although there was no such effect in [32, 41], such designs are not scalable for $\mathbf{H}$ matrices with large column or row weights. Another potential problem of such designs is that, as the BCU and CCU are individually designed for a different number of operands, they cannot be time-shared.

Some of the implementations with the array-style memory architecture can be parameterized for a different CPA structure, that is, they can be synthesized for different N_c, N_b, or P. However, due to the fact that the architecture is closely related with the code structure, the decoding throughput and the amount of required resource is also determined by the code parameters. These architectures cannot be flexibly parameterized for a various performance-resource tradeoff.

There are another category of implementations with a focus on flexibility. A flexible architecture based on communication network in [42] can be parameterized for any number of processing elements. The decoder can be also reprogrammed on the fly for any arbitrary parity check matrix by changing the routing among the processing elements and the memory blocks. However, since it does not utilize the structure of the array structure, there is a large overhead for the bus connections. The throughput to area ratio (TAR) of [42] is reported to be larger than that of code-specific solutions by an order of magnitude.

4.3 RECONFIGURABLE AND REPROGRAMMABLE DECODER

In this section, we propose a novel hardware decoder architecture for array-structured LDPC codes. While the architecture has been designed to utilize the array structure, it is not very tightly correlated with the specific code parameters such as N_c, N_b, p, j, or k. Before synthesis, the number of computation units can be specified for a target throughput, and the depth of the memory blocks can be specified for the maximum values of the code parameters that can be supported. This decoder supports any QC-LDPC code, and the code parameters can be changed by software when the hardware is running.

4.3.1 Decoder Overview

The decoder consists of the input memory to store the input LLR data, computation units to perform bit/check node operations, the message memory to store the messages, the output memory to store the decoder output, and the error counter. In addition, there is a register block for communication with an external device. In our FPGA implementation, we use an embedded processor to give commands and fetch results through the register block. The overall block diagram is shown in Figure 4.2.

The pre-synthesis parameters that determine the amount of logic and memory used for the decoder are as follows.

- Algorithm: the SP or MS/MMS algorithm
- Data precision for message representation (integer part and fractional part)
- The parallelization factor V (even)
- The maximum possible values for N_c, N_b, P, j, and k
- The maximum possible value for $\lceil P/V \rceil + 1$
- The maximum possible value for codeword length N

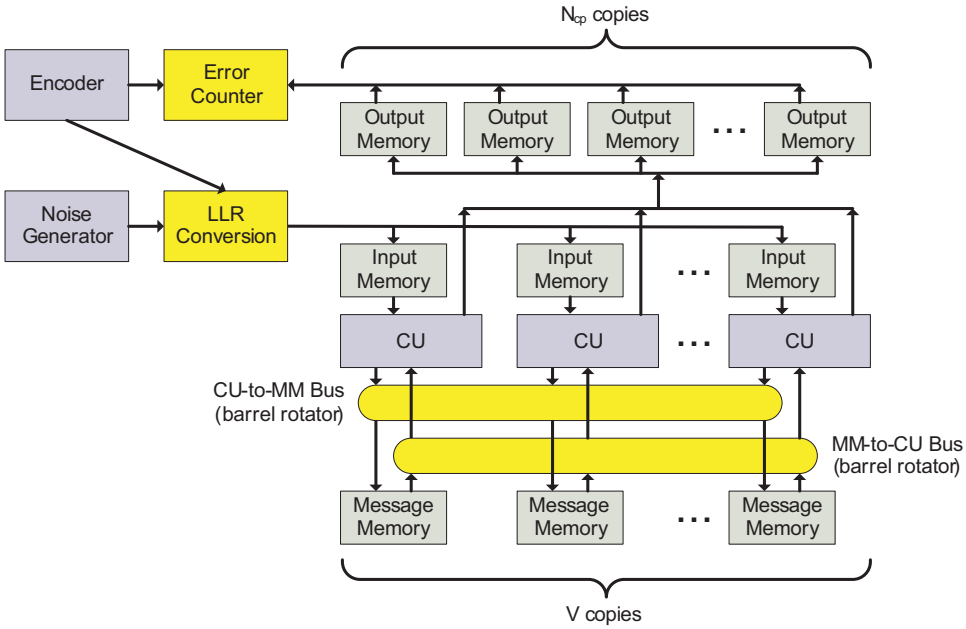


Figure 4.2: Overall block diagram of the decoder

- The maximum possible value for the number of check equations M
- The maximum possible value for the number of N_{ph}, where

$$N_{ph} = \max\left((\text{largest column weight}) * N_b, (\text{largest row weight}) * N_c\right)$$

- The maximum possible value for the number of nonnegative elements in the extended $\mathbf{S}$-matrix N_s
- The depth of each input and output memory blocks IOM_d
- The depth of each message memory blocks MM_d
- The maximum number of message bit errors per codeword that can be counted N_{err}
- The maximum number of iterations N_{iter}
- Total number of intermediate checkpoints N_{cp} (to see the decoding results at different number of iterations)

While the pre-synthesis parameters define a range of code parameters that can be supported by the synthesized hardware, the actual code parameters can be reprogrammed by writing to the register block. The post-synthesis parameters that can be modified at run time are as follows.

- The number of iterations
- Whether to enable early stopping when a valid codeword is found
- Code parameters N_c, N_b, p, j and k
- The shift values for the extended $\mathbf{S}$-matrix
- The coefficients of the look-up table for the $F()$ transform function block (for SP only)
- Scaling factor for the check node output (for MMS algorithm only)
- A list of intermediate checkpoints

4.3.2 Shared Bit/Check Computation Units

In our proposed decoder, either the SP algorithm or the MS/MMS algorithm can be chosen before synthesis.

The computation unit (CU) for the SP algorithm performs additions and calculates the $F(\cdot)$ function, which has been implemented by a piecewise linear interpolator that uses a look-up table. For the MS/MMS algorithm, the CU performs additions, minimum operations, and scaling. In this section, we describe the design of the computation units for both algorithms.

As mentioned in the previous section, most speed-oriented decoders perform a check or bit node operation in a single clock by using multi-operand adders. Since the number of operands is determined by the column and row weight of the code, such an architecture cannot be reprogrammed to support codes with different row/column weights. Also, despite the similarity between the bit and check node operations, the bit computation unit (BCU) and the check computation unit (CCU) are separately designed due to the large difference in the number of operands involved in bit and check node operations.

In the proposed decoder, however, each check or bit node operation is performed in a sequential manner, i.e., one operand is processed at a time. This scheme makes the design of the computation units independent from the actual code parameters since any larger weight can be supported by increasing the number of clock cycles for each bit or check node operation. Also, since the computation unit is not designed for a specific number of operands, it is possible to design a shared bit/check computation unit that works as a BCU during the first half iteration and as a CCU during the second half iteration. In the rest of this section, we provide the details of the CU operation for each mode.

Sum-product algorithm: A direct implementation of the equations (2.20) and (2.21) would require two $F(x)$ units in each shared CU. These units would be idle for bit node operations and also cause longer pipeline depth for the check node operation. This problem can be solved by pre-applying the $F(\cdot)$ to the bit node output, i.e.,

$$\bar{\lambda}_{b\to c} \;=\; F\big(\mu_b + \sum_{c'\in\mathcal{N}(b)\setminus c} \lambda_{c'\to b}\big) \tag{4.1}$$

and

$$\lambda_{c\to b} \;=\; F\Big(\bigoplus_{b'\in\mathcal{N}(c)\setminus b} \bar{\lambda}_{b'\to c}\Big). \tag{4.2}$$

With this modification, the bit and check node operations are identical except for the difference in the additions.

For the sequential processing, the sum is computed first, and the outgoing message is computed by subtracting each of the incoming message from the sum. The SP algorithm can be described as follows.

- *Bit node operation:* At time $t = 0, 2, 4, \ldots$, for each bit node b

$$\text{step 1:}\quad \lambda_b(t) = \begin{cases} \mu_b, & t = 0 \\ \mu_b + \sum_{c\in\mathcal{N}(b)} \lambda_{c\to b}(t-1), & \text{otherwise} \end{cases} \tag{4.3}$$

$$\text{step 2:}\qquad \forall c \in \mathcal{N}(b),\;\; \bar{\lambda}_{b\to c}(t) = F\big(\lambda_b(t) - \lambda_{c\to b}(t-1)\big), \tag{4.4}$$

where $\lambda_b(t)$ is the decoder output for bit b if decoding stops at the iteration t.

- *Check node operation:* At time $t = 1, 3, 5, \ldots$, for each check node c

$$\text{step 1:} \qquad \bar{\lambda}_c(t) = \bigoplus_{b \in \mathcal{N}(c)} \bar{\lambda}_{b \to c}(t-1) \tag{4.5}$$

$$\text{step 2:} \quad \forall b \in \mathcal{N}(c), \;\; \lambda_{c \to b}(t) = F\big(\bar{\lambda}_c(t) \ominus \bar{\lambda}_{b \to c}(t-1)\big), \tag{4.6}$$

where the operation $\ominus$ is defined as

$$a \ominus b \;\; = \;\; \mathrm{sgn}(a)\mathrm{sgn}(b)(|a| - |b|) \tag{4.7}$$

At step 1 of both bit and check node operations, the incoming messages are accumulated in an accumulator. At step 2, the outgoing message is created by subtracting each incoming message from the sum. Because the step 2 operations can be performed only after the step 1 operations are complete, the incoming messages should not be discarded until they are reused in step 2. In the proposed architecture, the step 1 and 2 are pipelined, and the incoming messages are stored in a FIFO with depth $\max(j, k)$. The area overhead incurred by the FIFO's makes the proposed design less efficient than the array-style memory architecture, but a part of the inefficiency can be amortized by the use of the shared bit/check CU's and the elimination of multi-operand adders. The CU for the SP algorithm is shown in Figure 4.3. The adder and the subtracter perform ordinary signed addition and subtraction, respectively, in BCU mode. When in CCU mode, they perform the special $\oplus$ and $\ominus$ operations defined in equations 2.22 and 4.7, respectively.

MS/MMS algorithm: The bit and check node computations take place in a sequential manner as was the case with the SP algorithm. While the bit node operation is identical to that of the SP algorithm, the check node operation requires a fairly different processing. As can be seen in eq. (2.24), the magnitude of a check-to-bit message is always one of the two possible values: the smallest and the second smallest, among all of the incoming bit-to-check messages. The former is taken except when the bit-to-check message with the smallest magnitude is excluded. The sequential processing for the MS or MMS algorithm can be described as follows.

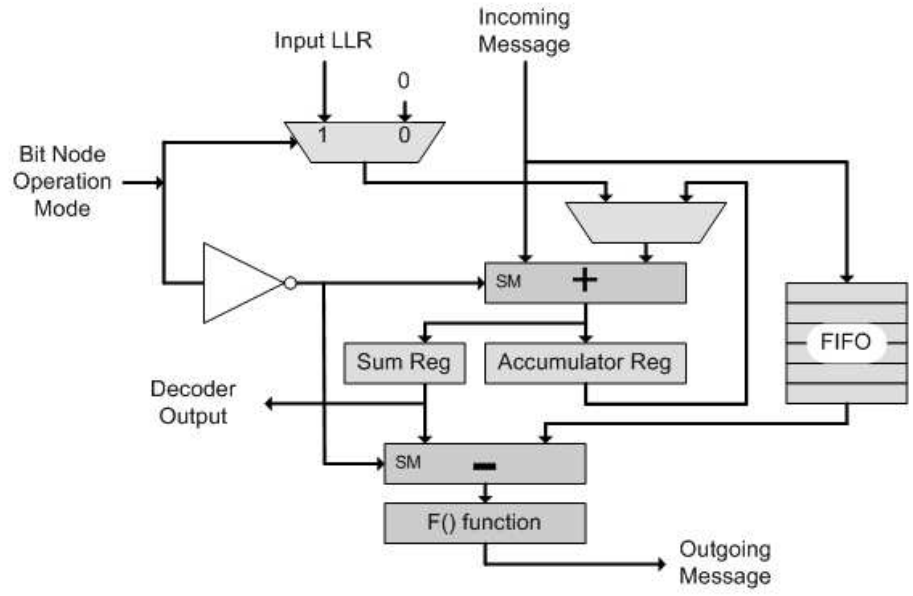


Figure 4.3: Computation unit for the SP algorithm

- *Bit node operation:* At time $t = 0, 2, 4, \ldots$, for each bit node b

$$\text{step 1:} \quad \lambda_b(t) = \begin{cases} \mu_b, & t = 0 \\ \mu_b + \sum_{c \in \mathcal{N}(b)} \lambda_{c \to b}(t-1), & \text{otherwise} \end{cases} \tag{4.8}$$

$$\text{step 2:} \qquad \forall c \in \mathcal{N}(b), \;\; \lambda_{b \to c}(t) = \lambda_b(t) - \lambda_{c \to b}(t-1), \tag{4.9}$$

where $\lambda_b(t)$ is the decoder output for bit b if decoding stops at the iteration t.

- *Check node operation:* At time $t = 1, 3, 5, \ldots$, for each check node c

$$\text{step 1:} \qquad \lambda_c(t) = \bigodot_{b \in \mathcal{N}(c)} \lambda_{b \to c}(t-1) \tag{4.10}$$

$$b_* = \arg \min_{b \in \mathcal{N}(c)} |\lambda_{b \to c}(t-1)| \tag{4.11}$$

$$\lambda'_c(t) = \bigodot_{b \in \mathcal{N}(c) \setminus b_*} \lambda_{b \to c}(t-1) \tag{4.12}$$

$$\text{step 2:} \quad \forall b \in \mathcal{N}(c), \tag{4.13}$$

$$\lambda_{c \to b}(t) = \begin{cases} \alpha \operatorname{sgn}\big(\lambda_{b \to c}(t-1)\big) \operatorname{sgn}\big(\lambda_c(t)\big) |\lambda_c(t)|, & b \neq b_* \\ \alpha \operatorname{sgn}\big(\lambda_{b \to c}(t-1)\big) \operatorname{sgn}\big(\lambda_c(t)\big) |\lambda'_c(t)|, & b = b_*, \end{cases} \tag{4.14}$$

where α is the scaling factor for the MMS algorithm.

Although the bit node and check node operations seem quite different, the shared CU for the MS algorithm can still be implemented using two adders. The two adders, used in the step 1 and 2 for bit mode operation, work as comparators in the step 1 of the check mode operation. The first comparator is used to find the magnitude and the index of the incoming message with the smallest magnitude in eq. (4.10) and (4.11). This can be done sequentially by comparing the magnitude of the incoming message with the previously found minimum and keeping the smaller of the two. The second smallest magnitude in eq. (4.12) is found by the following operations:

- If a new minimum magnitude is found, the previously stored second minimum is replaced by the new minimum.
- Otherwise, the larger of the two operands of the first comparator is compared with the previously found second minimum and the smaller of the two is stored as a new second minimum.

Accordingly, the two comparators work at the same time, as opposed to the pipelined manner. For the magnitude scaling in the MMS algorithm, a multiplier or a third adder should be used. The CU for MS/MMS algorithm is shown if Figure 4.4.

4.3.3 Memory Assignment and Bus Connection

For a hardware-based LDPC decoder, a high throughput can be achieved by deploying a large number of computation units that work simultaneously. For a higher utilization of the available computation units, the memory assignment should ensure that the messages to be processed together are stored in different memory banks so that they can be accessed at the same clock cycle. In the array-style memory/bus architecture (see Section 4.2), this is accomplished by mapping each submatrix of the parity check matrix to a separate memory block and processing no more than one message in each submatrix. Thus, the number of message that can be accessed simultaneously is $j \cdot N_b$. Accordingly, the throughput is determined by the code parameters, and the architecture cannot be reconfigured at run time to support a different set of code parameters.

In this thesis, we take a different approach for the memory allocation and scheduling to provide

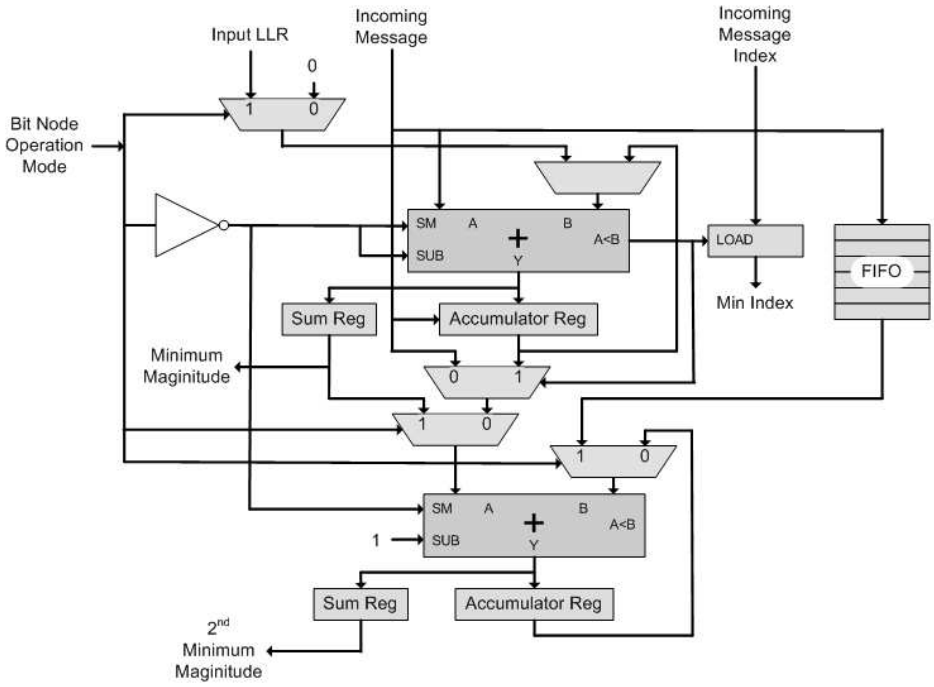


Figure 4.4: Computation unit for the MS/MMS algorithm

a flexible architecture that can support any (N_c, N_b, P, j, k) parameters within the maximum limit imposed by the allocated resource. To design an architecture that is weakly related to the code parameters, we seek to achieve a high degree of parallelism by processing multiple messages within each submatrix of the parity check matrix in parallel, rather than processing multiple messages in the same column or row of the parity check matrix. In this architecture, the degree of parallelism is determined by the pre-synthesis parameter V, which can be decided solely by the desired decoding throughput and the amount of resource available but independently from the code parameters. The number of CUs is V, and there are also as many message memory (MM) banks. Thus, the objective of the memory assignment is to assign each message to one of the V memory banks.

In an iteration of the decoding process for a QC-LDPC code, $P \cdot N_s$ bit-to-check messages and as many check-to-bit messages are produced and consumed, where N_s is the number of nonnegative shift values in the extended $\mathbf{S}$-matrix. The bit and check node computations can be performed in-place, i.e., the messages $\lambda_{b_j \to c_i}$ and $\lambda_{c_i \to b_j}$, both of which correspond to the matrix element h_{ij}

of $\mathbf{H}$, can be stored in the same memory location. Therefore, only $P \cdot N_s$ memory locations are needed.

We can use linear indices to denote the N_s $\mathbf{S}$-matrix elements, i.e., s_m denotes the m-th element. Since the P messages for corresponding to $s_m, m = 0, \ldots N_s - 1$, are to be stored in V memory banks, it would be sufficient to use $N_s\lceil P/V \rceil$ memory locations in each bank. To simplify the address generation logic, each bank has been designed to take up N_s contiguous equal-sized partitions, the size of which is the smallest 2-power number not less than $\lceil P/V \rceil$.

Now, we consider the mapping of the P messages for s_m to the m-th partitions in V memory banks. Reassigning bit and check node indices within the corresponding submatrix, the shift value specifies the connections between bit nodes $b_0, b_1, \ldots, b_{P-1}$ and check nodes $c_0, c_1, \ldots, c_{P-1}$. Denoting the message in column j as $\lambda_j, j = 0, \ldots, P-1$, the message λ_j is assigned to a memory location as follows:

$$\phi(j) = (\lfloor j/V \rfloor, j \mod V), \tag{4.15}$$

where (x, y) denotes the address x of the bank y. With this mapping, each message is assigned to one of the V memory banks in a round-robin manner An example of the bank assignment for a 10×10 submatrix with $V = 4$ is shown in Figure 4.5.

During the bit node computation, normally V contiguous bit nodes are processed at a time, starting from index 0; however, there can be less than V bit nodes involved when V does not divide P. The vector $\mathbf{b}_k$, the set of bit nodes processed at time k, takes the following form:

$$\mathbf{b}_k = \begin{cases} (b_{Vk}, b_{Vk+1}, \ldots, b_{Vk+V-1}), & 0 \le k \le \lceil P/V \rceil - 2 \\ (b_{Vk}, b_{Vk+1}, \ldots, b_{Vk+(P \bmod V)-1}), & k = \lceil P/V \rceil - 1 \end{cases}$$

Since the bit node index is the same as the message index, the memory location assignment of the message associated with b_j can be written as

$$\phi_b(j) = \phi(j).$$

With this mapping, any V consecutive data in bit indices are stored in V different memory banks so

that they can be accessed by the V computation units simultaneously.

Similarly, during the check node computation, the vector $\mathbf{c}_k$ containing the check nodes to be processed at time k takes the following form:

$$\mathbf{c}_k = \begin{cases} (c_{Vk}, c_{Vk+1}, \ldots, c_{Vk+V-1}), & 0 \leq k \leq \lceil P/V \rceil - 2 \\ (c_{Vk}, c_{Vk+1}, \ldots, c_{Vk+(P \bmod V)-1}), & k = \lceil P/V \rceil - 1 \end{cases}$$

The check node index i can be converted to the message index j by the following:

$$j = i - s \mod P,$$

where s is the shift value of the m-th $\mathbf{S}$-matrix element. Thus, the memory location assignment for the message associated with c_i is

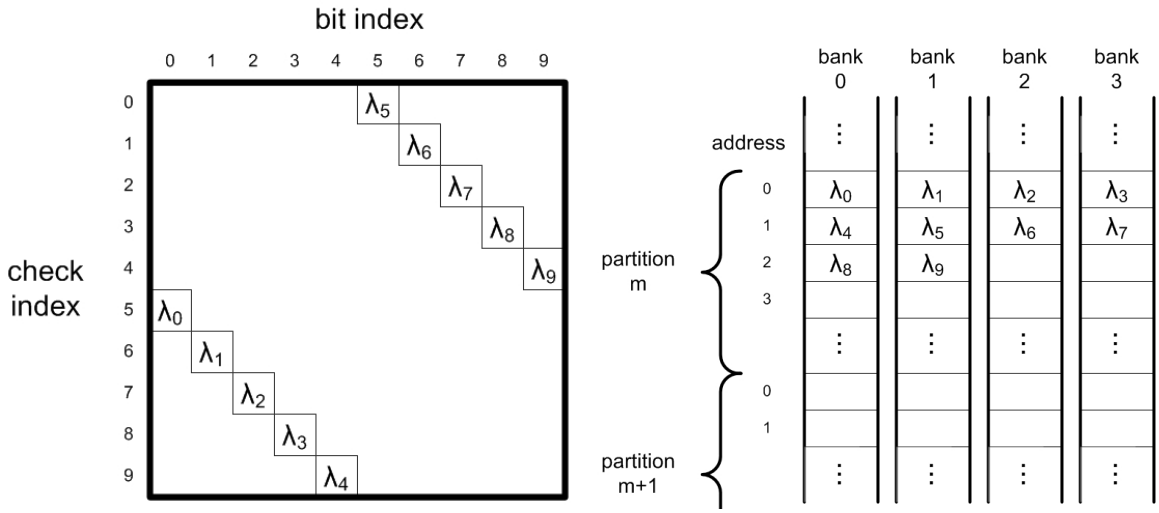
$$\phi_c(i) = \phi(i - s \mod P). \tag{4.16}$$

However, with this assignment, V consecutive data in check indices are not always in in V different banks, which results from the discontinuity in the index caused by the ' mod ' operation in eq. (4.16). For example, in Figure 4.5, the four messages $(\lambda_5, \lambda_6, \lambda_7, \lambda_8)$ corresponding to check indices (0,1,2,3) are in different memory banks. However, the messages $(\lambda_9, \lambda_0, \lambda_1, \lambda_2)$ corresponding to the check indices (4,5,6,7) are not, i.e., the messages λ_1 and λ_9 are in the same memory bank 1.

To overcome this memory conflict problem, which arises when P is not a multiple of V, we perform a special processing called "copy phase" at the end of each bit or check node operation, i.e., we keep a redundant copy of the messages beyond the boundary set by P to provide contiguous bank assignment. For this, we use extra cycles to copy the first portion of the memory to the last portion after bit node (half) iteration, and copy the last to the first after check node (half) iteration. The details of the copy phase operations are as follows:

- *Determine the number of messages to copy:*

$$r = (V\lfloor P/V \rfloor - s \mod P) \mod V,$$



(a) Message indexing for m-th $\mathbf{S}$-matrix element (b) Memory allocation in the m-th partition

Figure 4.5: Memory block assignment for one submatrix with $P = 10$ and $V = 4$.

- *Copy r messages after bit mode iteration:*

$$(\mathcal{M}[\phi(P)], \mathcal{M}[\phi(P+1)], \ldots, \mathcal{M}[\phi(P+r-1)])$$
$$\leftarrow (\mathcal{M}[0,0], \mathcal{M}[0,1], \ldots, \mathcal{M}[0,r-1]),$$

  where $\mathcal{M}[i,j]$ denotes the memory location at the address j of the bank i, '$\leftarrow$' denotes a copy operation from right to left.

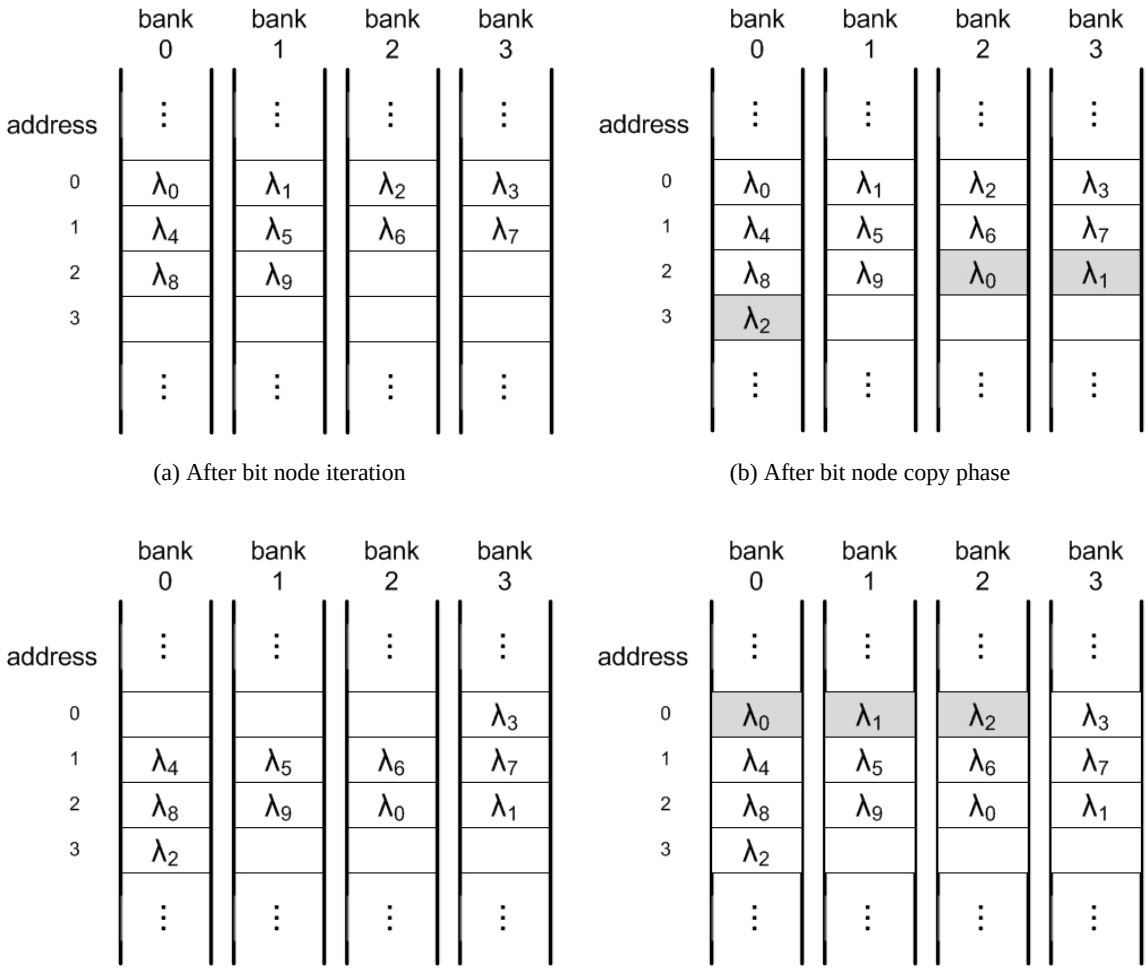
- *Copy r messages after check mode iteration:*

$$(\mathcal{M}[0,0], \mathcal{M}[0,1], \ldots, \mathcal{M}[0,r-1])$$
$$\leftarrow (\mathcal{M}[\phi(P)], \mathcal{M}[\phi(P+1)], \ldots, \mathcal{M}[\phi(P+r-1)])$$

With the copy phase, it is guaranteed that any V consecutive check messages in eq. (4.16) are in V different memory banks by determining the bank and address of the first message λ_i by eq. (4.16) and the rest by taking the "next" positions in the message memory, i.e.,

$$\phi_c(i+n) = \phi\big((i-s \mod P) + n\big),$$

where $0 \leq n \leq V-1$. This operation is best illustrated by an example. In Figure 4.6, the contents of the m-th partition of the message memory is shown for each step of the bit and check iteration.



(a) After bit node iteration

(b) After bit node copy phase

(c) After check node iteration

(d) After check node copy phase

Figure 4.6: Message memory contents for an iteration. Duplicated messages are shown in gray.

Parameter	Value
Algorithm	MS/MMS
Data Precision	Integer = 4 bit, Fractional = 2 bit
V	$2, 12, 24, 36, 48$
Max value for (N_c, N_b, P, j, k)	$(32, 32, 2048, 32, 32)$
Max value for $\lceil P/V \rceil + 1$	512
Max N	16384
Max M	8192
Max N_{ph}	256
Max N_s	128
Max IOM_d	512
Max MM_d	2048
Max N_{err}	256
Max N_{iter}	1024
Max N_{cp}	8

Table 4.1: Example: Pre-synthesis parameter set.

Figure 4.6a shows the memory contents after the bit mode iteration is completed. In the bit mode copy phase, three messages are copied from the beginning to the end, as shown in Figure 4.6b. Now, we can see the messages $(\lambda_9, \lambda_0, \lambda_1, \lambda_2)$ corresponding to check indices (4,5,6,7) are in different memory banks, using the duplicated messages. During the check mode iteration in Figure 4.6c, the messages are modified using the addressing in eq. (4.17). Finally, in Figure 4.6d, the three messages at the end are copied back to the beginning to ensure that V messages are in different banks when the next bit node iteration begins.

The copy phase requires $2N_s$ clock cycles per iteration. Since the total number of clock cycles for an iteration without the copy phase is $2N_s\lceil P/V \rceil$, this overhead becomes negligibly small when P is large compared to V. When V divides P, the copy phase is not necessary.

4.3.4 Synthesis Results

The hardware-based evaluation system including the proposed decoder has been implemented on the Xilinx Virtex-II Pro XC2VP30 FPGA. Each FPGA contains two embedded PowerPC processors, 13,696 slices, 136 18k-bit BRAMs, and 136 18×18-bit multipliers. One of the processors is used for downloading code parameters and recording the decoded results, but is not used by the decoder.

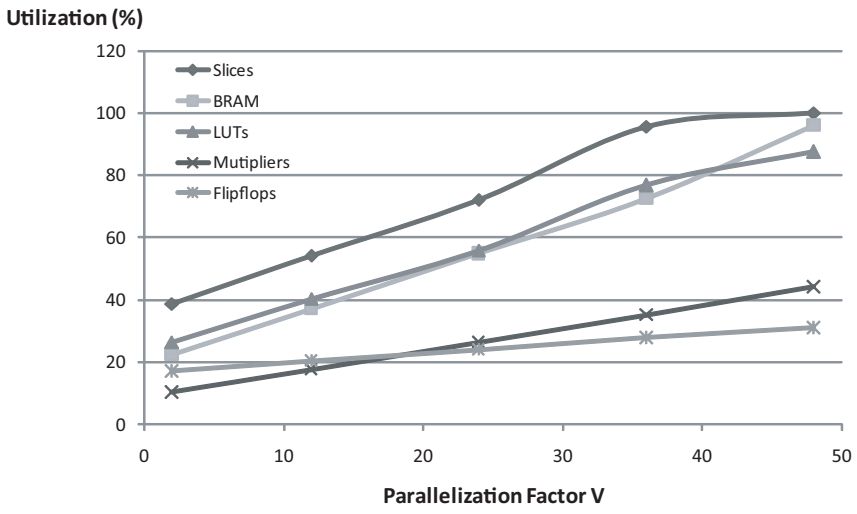


Figure 4.7: Chip utilization for MMS decoder

In order to demonstrate the flexibility of the proposed architecture in the resource usage and throughput, the evaluation system has been synthesized with a wide range of parallelization factors for the decoder. The pre-synthesis parameters used for the decoder are shown in Table 4.1. The resource utilization with respect to the parallelization factor is given in Figure 4.7, where the resource usage is shown for the entire FPGA chip including the decoder, the random number generator, and the peripherals for the embedded processor. By increasing V until the FPGA is fully utilized, we could utilize 100% of slices and 94% of BRAMs.

The decoding complexity of a given LDPC code with the SP algorithm is directly proportional to the number of edges in the Tanner graph. In [43], the required processing power $P_{c,req}$ was

	Parallelization V					
	2	12	24	36	48	total
Area [slices]	886	2967	5233	8130	10466	13696
Block RAM	14	34	58	82	114	136
Multiplier	6	16	28	40	52	136
Max frequency [MHz]	71.56	69.19	67.03	67.18	66.78	
Peak P_c [edges/cycle]	1	6	12	18	24	
Peak P_s [M edges/sec]	72	415	804	1209	1603	
PAR [edges/sec/ 1000 slices]	81	140	154	149	153	

Table 4.2: Decoder-only area and throughput results with parameters in Table 4.1.

defined as the number of edges to be processed per cycle, which can be derived as

$$P_{c,req} = \frac{\varepsilon i_{\text{max}} D}{K f_{\text{CLK}}} \text{ [edges/cycle]}, \tag{4.17}$$

where ε is the number of edges in the Tanner graph, i_{max} is the maximum number of iterations, D is the desired information throughput, K is the number of information bits per codeword, and f_{CLK} is the clock frequency of the decoder. For a given implementation, *processing power* P_c, the actual number of edges that are processed per cycle, can be used as a measure of parallelism. Since each edge corresponds to two messages (bit-to-check and check-to-bit) and each CU is capable of processing one message per clock, the peak processing power of the proposed architecture is simply $P_c = V/2$. However, when V does not divide P, the actual processing power depends on the code parameters due to the copy phase, and can be derived as follows.

$$P_c = \frac{N_s P \text{ [edges/codeword]}}{2N_s(\lceil P/V \rceil + 1) \text{ [cycles/codeword]}} \tag{4.18}$$

$$= \frac{P}{2(\lceil P/V \rceil + 1)} \text{ [edges/cycle]}. \tag{4.19}$$

In order to make a comparison across different LDPC decoder implementations, we can define *processing rate* P_s as the number of edges processed per second, which is given by

$$P_s = P_c \cdot f_{\text{CLK}} \text{ [edges/sec]}. \tag{4.20}$$

As a metric for of area efficiency, we will use *processing rate to area ratio* (PAR), which gives the processing rate per unit area.

The resource usage and peak processing power of the proposed architecture synthesized with the parameters in Table 4.1 are shown in Table 4.2, where the resource usage is for the decoder only. The table shows that the maximum clock frequency deteriorates only slightly with increasing V due to the sequential processing structure. In addition, the table shows that the PAR is almost constant for sufficiently large parallelization factors.

4.3.5 Architecture Comparison

Due to the lack of a standard framework to compare different LDPC decoder implementations, it is a difficult task to carry out fair comparisons, especially with differences in the code parameters and the implementation technology. For example, the TAR in [42] cannot be used for comparing decoders designed for different code rates since it is computed from throughput in bits per second, which is often reported in literature as a part of the synthesis results. In this thesis, we will use processing rate P_s defined in Section 4.3.4 to quantify the computational capability, because the decoders to be compared are based on the SP or MS algorithm and for these algorithms the number of edges correctly represents the decoding complexity of a given LDPC code. Accordingly, we will use PAR to compare the area efficiency of different decoder implementations.

In Table 4.3, three FPGA decoder implementations in the literature with array-style memory architectures are compared with the proposed architecture when the code being decoded has a regular CPA structure. While the proposed decoder has run-time reconfigurability, the other decoders are synthesized for a specific array structure of LDPC codes. As a result, the resource usage including the number of CUs and memory blocks and the processing power are closely linked to the structural parameter such as N_b or j. On the other hand, the memory usage of the proposed decoder is only related with the parameter V, which is independent of the code structure. The processing power P_c is weakly related with P, and becomes irrelevant as P/V increases.

The throughput, processing power, and area efficiency are compared in Table 4.4. A code has been chosen from each of the architectures in Table 4.2 and the corresponding throughput and area are taken for comparison as shown in Table 4.4a. Since it is a common practice in the literature

	Author			
	Sun [41]	Karkooti [32]	Zhang [44]	Proposed Architecture
Flexibility				
Supported codes	array of regular submatrices	CPA(3,6,P) $P = 2^{\theta}$	CPA	QC-LDPC
Parallelization scalability	No	Yes	No	Yes
Precision flexibility	Yes	Yes	Yes	Yes
Run-time reconfigurability	No	No	No	Yes
Algorithm	MS	MMS	SP	SP/MS/MMS
Resource and Computational Power				
Number of input memory blocks	N_b	sN_b	N_b	V
Number of message memory blocks	jN_b	sjN_b	$2N_b$	V
Number of CUs (BCU,CCU)	N_b, N_c	sN_b, sN_c	N_b, N_c	V (SCU)
CU processing power (BCU,CCU) [edges/cyc]	$j/2, k/2$	$j/2, k/2$	$1/2, 1/2$	$1/2$

Table 4.3: Architecture comparison of LDPC decoders.
(BCU: bit computation unit, CCU: check computation unit, SCU: shared computation unit)

to report the throughput in bits per second, the P_c and P_s have been calculated from the reported throughput using eq. (4.17) and (4.20), respectively. If the proposed decoder is synthesized by the parameters in Table 4.1, the three codes considered in Table 4.4a can be decoded by the same implementation. The throughput of the proposed decoder for four codes including the aforementioned three are shown in Table 4.4b.

It can be seen from the PAR (P_s/Area) that the architectures designed for a specific array structure process between 2.9 and 8.1 times more edges per second with the same amount of hardware resource than the proposed architecture, from which we can see the overhead incurred by the run-time reconfigurability. To understand how much overhead is reasonable for the flexibility, the results in [42] may be helpful, where a network-on-chip decoder for any kind of LDPC codes with run-time reconfigurability is compared with more code-specific solutions implemented in application-specific integrated circuit (ASIC). In this work, it has been shown that a fully parallel solution and a partially parallel solution have 22 and 17 times higher area efficiency than the flexible solution, respectively.

It is also notable that, despite the large difference in PAR between the proposed decoder and Sun's decoder in [41], the difference in the throughput in bits per second (D) is only a factor of two. This is because of the pre-synthesis flexibility of the proposed architecture that allows the highest resource utilization independently from the code parameters.

	Author		
Parameter	Sun [41]	Karkooti [32]	Zhang [44]
Code	DDS($j = 3, N = 4923$)	CPA(3,6,256)	CPA(6,32,64)
Algorithm	MS	MMS	SP
Precision	6 bit	5 bit	6 bit
ε	14769	4608	12288
$i_{\max}$	1	20	1
D[Mbps]	832	63.5	195
K[bits]	4376	768	1664
f_{CLK}[MHz]	208	121	100
Area [slices]	2571	11352	7320
P_c [edges/cycle]	13.5	62.98	14.4
P_s [edges/sec]	2.81×10^9	7.62×10^9	1.44×10^9
PAR [edges/sec/slice]	1090×10^3	671×10^3	197×10^3

(a) Code-specific architectures in the literature

	Code			
Parameter	DDS($j = 3, N = 4923$)	CPA(3,6,256)	CPA(6,32,64)	CPA(3,9,500)
Algorithm	MMS			
Precision	6 bit			
ε	14769	4608	12288	13500
$i_{\max}$	1	20	1	10
D[Mbps]	412	11.6	188	30.9
K[bits]	4376	768	1664	3000
f_{CLK}[MHz]	66.67			
Area [slices]	10466			
P_c [edges/cycle]	21.0	18.3	10.7	20.83
P_s [edges/sec]	1.40×10^9	1.22×10^9	0.71×10^9	1.39×10^9
PAR [edges/sec/slice]	134×10^3	116×10^3	67.9×10^3	133×10^3

(b) Proposed architecture with $V = 48$ for different codes

Table 4.4: Throughput and area efficiency comparison.

CHAPTER 5

PERFORMANCE STUDY

5.1 INTRODUCTION

The pseudorandom construction of the CPA-structured LDPC codes has a number of advantages over the other types of constructions. First, the size of the $\mathbf{S}$-matrix and the submatrix size P can be arbitrarily chosen, which makes it possible to design a code for a desired code rate and codeword length in a very flexible manner. Also, the girth of the code can be directly incorporated in the design process to allow better performance with the iterative decoding algorithm. The inherent graph partitioning in the CPA-structure also facilitates a highly parallel decoder architecture with simple bus connections, resulting in a much more area-efficient decoder than the more general class of QC-LDPC codes. However, the performance of the pseudorandom CPA-structured codes (PR-CPA) depends on the actual $\mathbf{S}$-matrix elements chosen by a computer search. Due to the random nature, two codes with the same $\mathbf{S}$-matrix dimension and the same girth can have very different performance.

Accordingly, a natural question would be whether it would be possible to find a good code by examining the structural properties of the code. In this chapter, we study by simulation the effect of the structural parameters of CPA-structured codes on the error performance. For this purpose, we construct CPA-structured LDPC codes by using the pseudorandom method described in Chapter 3. Another point of interest is to find out how the PR-CPA LDPC codes compare with other construction methods. For this, we compare the performance of pseudorandomly constructed CPA

codes with well-known deterministically generated codes. For the simulation results in this chapter, we used the FPGA decoder described in Chapter 4, synthesized for the MMS algorithm with 6-bit LLR quantization. The performance was measured after 50 iterations without early stopping.

5.2 HARDWARE-BASED NOISE GENERATION

A correct implementation of a random number generator (RNG) plays an integral role in obtaining accurate results from the simulation. We designed a Gaussian noise generator based on the well-known Box-Muller method [45] that converts two uniformly distributed random variables over the interval $[0, 1)$ to two samples of Gaussian distribution $\mathcal{N}(0, 1)$., i.e.,

$$f(u_1) = \sqrt{-\ln(u_1)} \tag{5.1}$$

$$g_1(u_2) = \sqrt{2}\sin(2\pi u_2) \tag{5.2}$$

$$g_2(u_2) = \sqrt{2}\cos(2\pi u_2) \tag{5.3}$$

$$y_1 = f(u_1)g_1(u_2) \tag{5.4}$$

$$y_2 = f(u_1)g_2(u_2). \tag{5.5}$$

The hardware design is based on [46] as shown in Fig. 5.1, but we replaced the uniform random number generator with the Mersenne Twister (MT) 19937 that provides an extremely long period of $2^{19937} - 1$ of 32-bit numbers using relatively small hardware resource [47].

A high-dimensional uniformity is also regarded as one of the desired properties of good RNGs. An RNG with period P is said to be k-distributive to v-bit if, when a set of kv-bit vectors is formed by collecting v most significant bits from k consecutive numbers starting from each of P numbers, each possible bit vector occurs the same number of times in the set, except for the all-zero vector that occurs once less often. In such a test, with the largest possible k for a given v, the RNG is said to be $k(v)$-dimensionally equidistributed with v-bit accuracy. The MT 19937 has a very large $k(v)$ for $v = 1, 2, \ldots, 32$, having 623-dimensional equidistribution with 32-bit accuracy.

The hardware MT 19937 has been designed based upon the parallelization idea in [48], support-

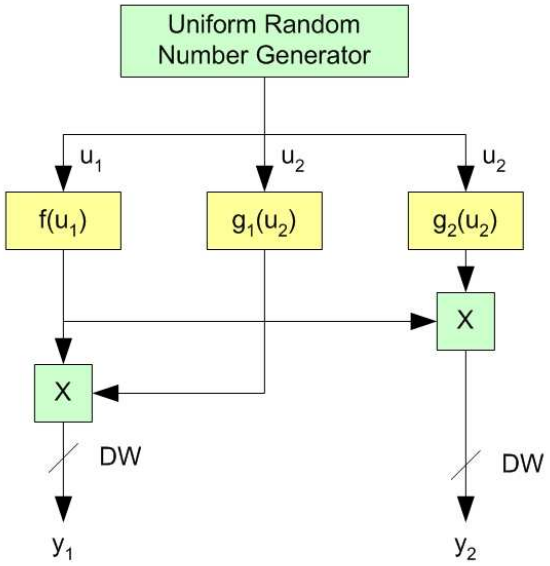


Figure 5.1: Gaussian noise generator

ing any even parallelization factor that divides 624. With this flexibility, the RNG can be configured to keep up with the speed of the decoder.

The currently designed RNG is good enough for BER=2.5×10^{-9} when we collect 100 bit error samples in the sense that the probability of any sample from a population of 4×10^{10} Gaussian samples exceeding the maximum representable value is less than 0.5. More recently, Lee proposed a hardware Gaussian RNG with much higher accuracy that can be used to explore for BER as low as 10^{-12} or 10^{-13} [49]. Although our Gaussian conversion unit has not been designed with such high accuracy, our RNG has been shown to have a sufficient accuracy to obtain correct results with our fixed-point hardware decoder. We have measured the performance of a selected set of codes by replacing our RNG with that in [49] that the authors kindly provided, and found no noticeable differences down to the BER of 10^{-11}.

5.3 PSEUDO-RANDOMLY GENERATED CPA-STRUCTURED CODE

5.3.1 Effect of Girth

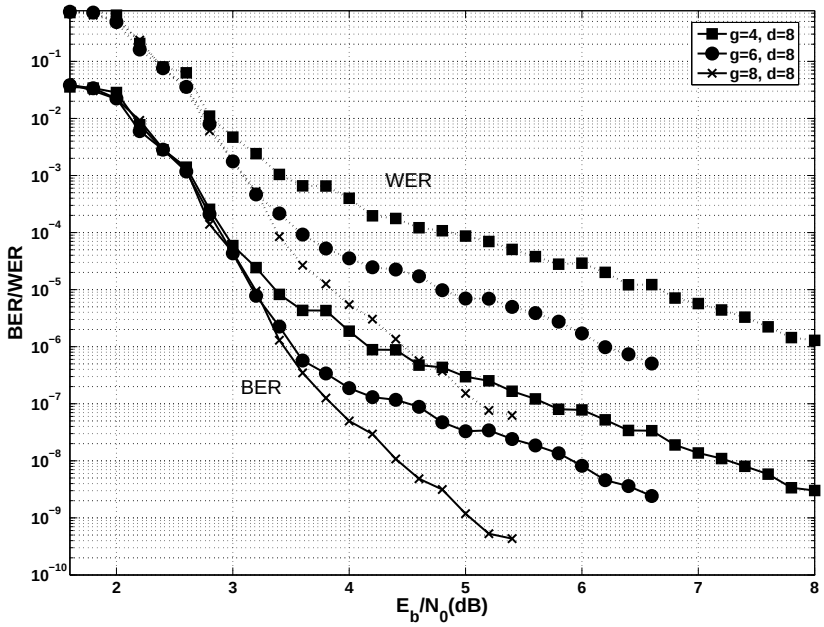
To see how the girth affects the performance, we have constructed two sets of codes of different code rate, with various girths but similar diameters.

- Code rate=$2/3$: CPA(3,9,100) and CPA(3,9,500)
- Code rate=$1/2$: CPA(3,6,150), CPA(3,6,750) and CPA(4,8,1023)

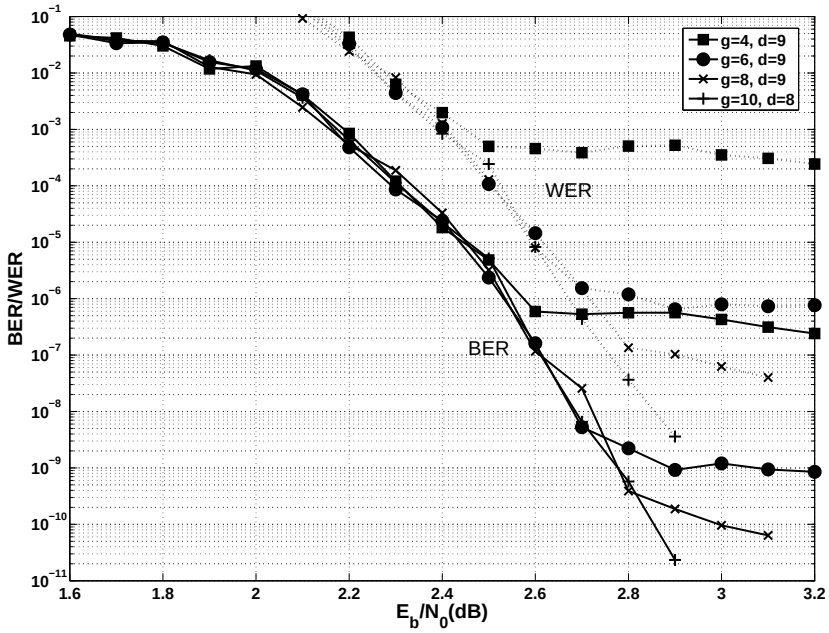
The performance of the codes with code rate $2/3$ is shown in Figure 5.2. For both $N = 900$ (Figure 5.2a) and $N = 4500$ (Figure 5.2b), it can be seen that the girth has a major effect on the performance in the error floor region, which justifies the code search to enlarge the girth. The performance of the codes with code rate $\frac{1}{2}$ in Figure 5.3 shows a similar trend with an exception of the girth-6 and girth-8 codes in Figure 5.3a. The cycle distribution of the two codes was checked but the difference was not large enough to explain the higher error floor of the code with the higher girth. On the other hand, for the CPA(4,8,1023) in Figure 5.3c, the girths make little difference in the entire SNR region. This is probably because the error floor did not occur within the simulated SNR region. In both plots, the girths do not make difference in the performance of the waterfall region but it is evident that higher girths lower error floors.

5.3.2 Effect of Diameter

To demonstrate the effect of diameter on the performance, a number of CPA($4, 8, 1023$) codes with different diameters have been generated using the method described in Chapter 3. While the largest possible diameter is infinity, it would simply mean that the corresponding Tanner graph is not connected, effectively shortening the code length. Excluding this trivial case, the largest finite diameter found with a reasonable amount of search efforts was 62. From the performance shown in Figure 5.4, the diameter has a direct influence on the performance in the whole SNR range under consideration; the code with a smaller diameter has a steeper slope in the waterfall region.

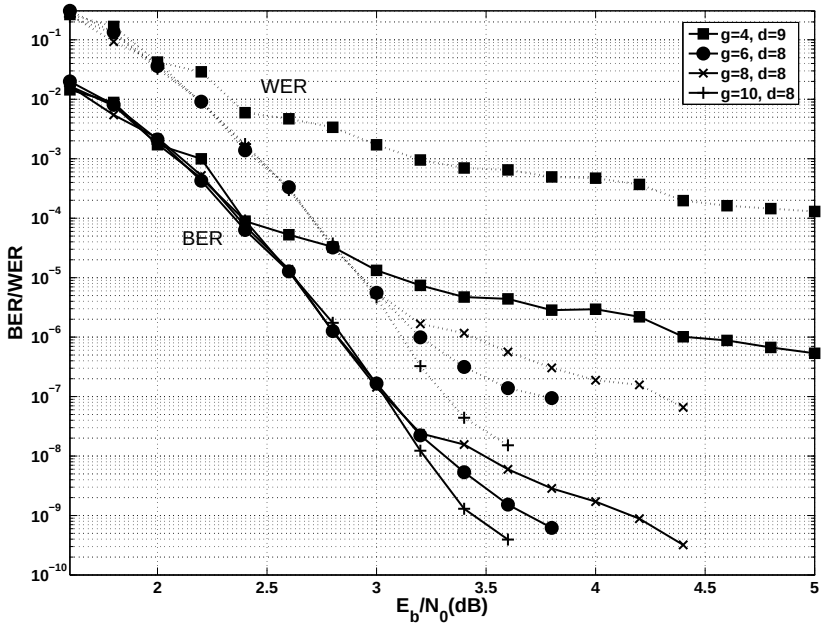


(a) CPA(3,9,100)

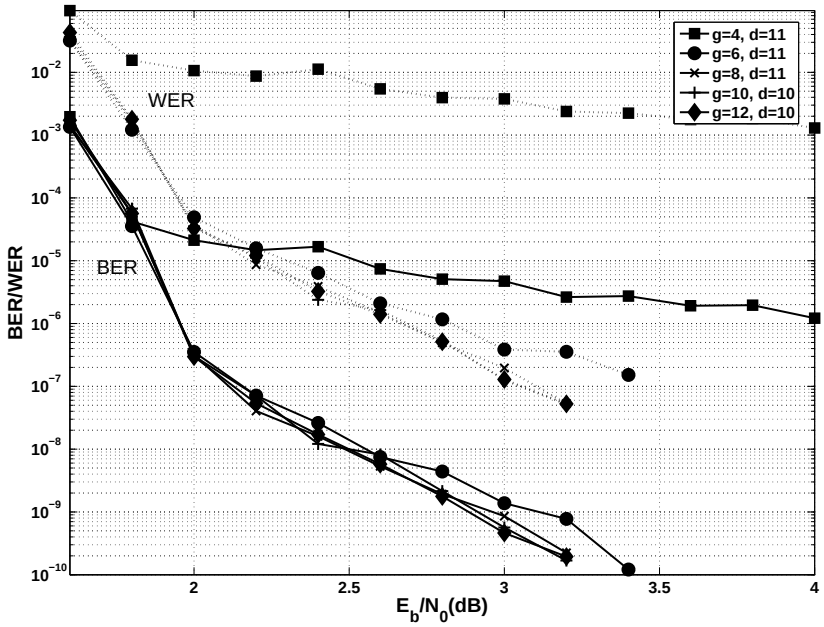


(b) CPA(3,9,500)

Figure 5.2: The effect of girth on the performance (code rate = $\frac{2}{3}$) (g:girth, d:diameter)

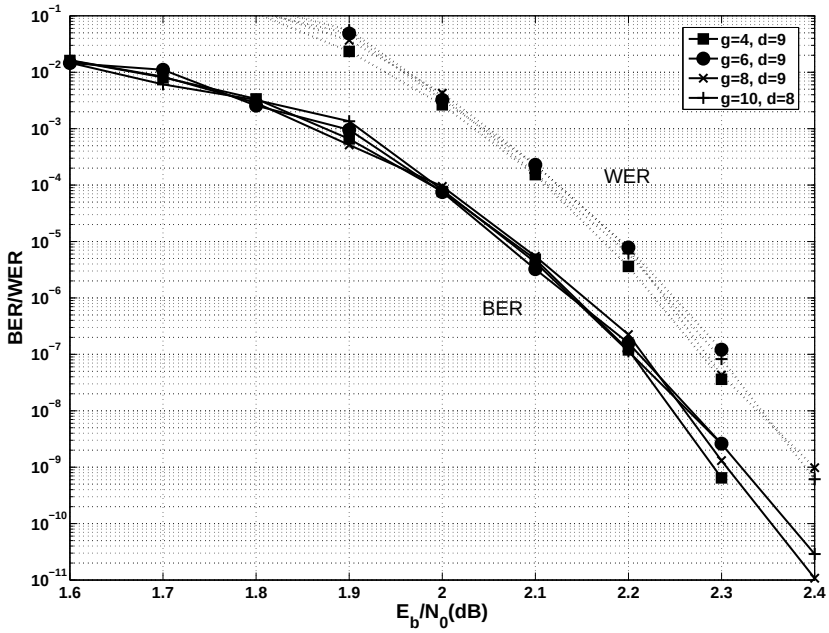


(a) CPA(3,6,150)



(b) CPA(3,6,750)

Figure 5.3: The effect of girth on the performance (code rate = $\frac{1}{2}$) (g:girth, d:diameter)



(c) CPA(4,8,1023)

Figure 5.3: The effect of girth on the performance (code rate = 1/2, cont'd) (g:girth, d:diameter)

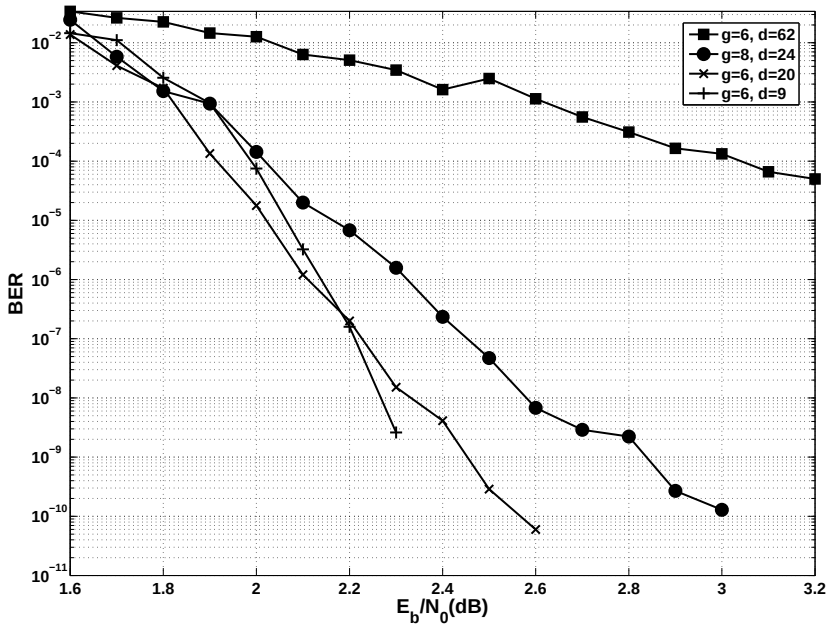


Figure 5.4: The effect of diameter on the performance of CPA(4,8,1023) code

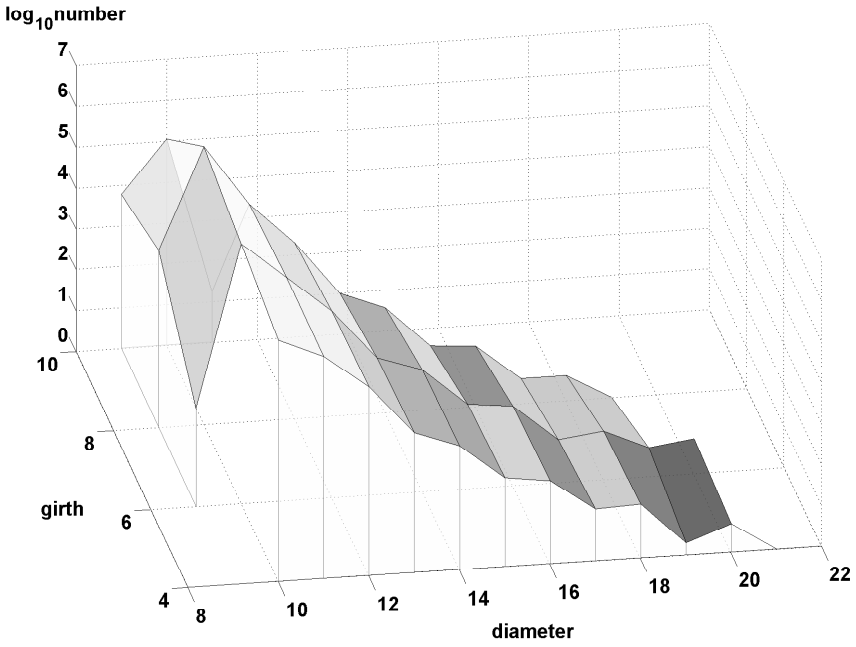


Figure 5.5: The distribution of uniformly sampled CPA(3,6,384) codes

Although increasing the diameter results in deteriorated performance, a pseudorandom generation based only on the girth results in very low diameters, which are only slightly larger than the lowest diameter that can be obtained by explicit efforts to reduce it in the search process. In addition, in the search space of PR-CPA codes, the codes with a very high (but finite) diameter seem to be very rare, and can be found only after a time-consuming search. In the search process for a high-diameter code, a new $\mathbf{S}$-matrix entry is chosen to maximize the diameter while keeping already determined $\mathbf{S}$-matrix entries . With the rarity of the high-diameter codes, it might be suspected that the bad performance is the artifact of the search process, rather than the effect of the high diameter.

To demonstrate that high diameters are a reliable indicator of bad performance in the space of CPA-structured codes, a test with uniform sampling was conducted, in which 10 million CPA(3,6,384) codes were generated by randomly choosing the $\mathbf{S}$-matrix elements without any efforts to control the girth or diameter. The number of sampled codes for each (girth, diameter) pair is shown in log scale in Figure 5.5. To measure the effect of diameter on the performance, we formed 8 sets

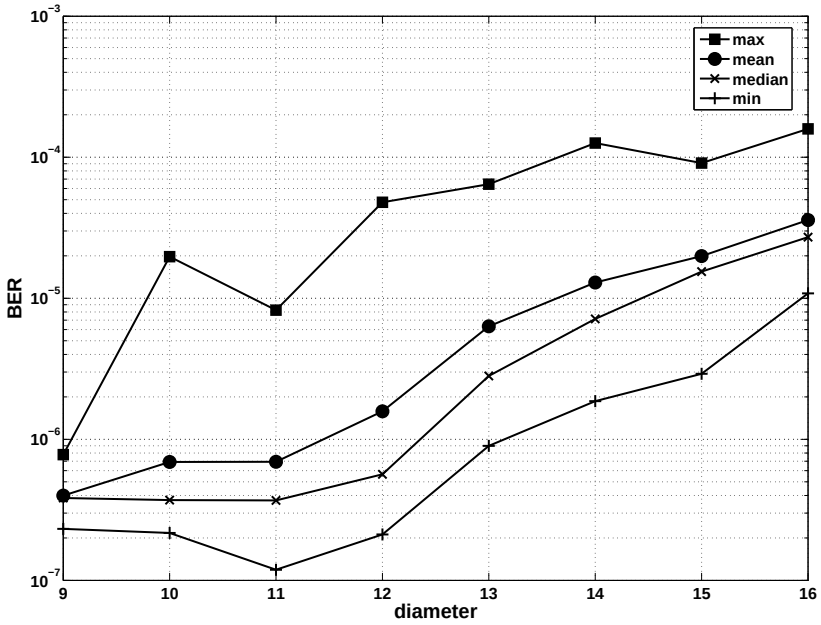


Figure 5.6: The performance of uniformly sampled CPA(3,6,384) codes with girth 6

of codes, each of which contains up to 100 codes of girth 6 and diameters ranging from 9 to 16, respectively. We will denote as $S^{(d)}$ the set of the codes with diameter d. Each set contains 100 codes except for $S^{(16)}$ that contains only 85 codes. The BER of the codes are measured at the SNR of 2.3 dB, and the minimum, mean, median, and maximum of the BER are shown in Figure 5.6.

To investigate statistical significance of the observed difference in BER performance, we can regard the BER of each code set $S^{(d)}$ as a random variable. Due to our lack of knowledge of the actual distribution of the BER random variable, we used the nonparametric hypothesis testing introduced in Section 2.5.

First, we define random variables $X^{(d)}$, $d = 9, \ldots, 16$ as the BER of a code in $S^{(d)}$, respectively. We perform the one-sided Mann-Whitney-Wilcoxon test for each pair $(X^{(d_1)}, X^{(d_2)})$ to test H_0: both variables have the same distribution against H_1: both have different distributions. The test measures the p-value, i.e., the probability that we will obtain the observed result when in reality the two variables $X^{(i)}$ and $X^{(j)}$ have the same distribution.

$d_1 \backslash d_2$	9	10	11	12	13	14	15	16
9	0.50	0.78	0.46	2.6×10^{-10}	1.3×10^{-34}	1.3×10^{-34}	1.3×10^{-34}	5.7×10^{-32}
10	0.22	0.50	0.34	1.6×10^{-9}	4.7×10^{-32}	1.3×10^{-32}	1.5×10^{-33}	1.1×10^{-31}
11	0.54	0.65	0.50	4.8×10^{-7}	1.5×10^{-29}	2.4×10^{-32}	2.8×10^{-34}	5.7×10^{-32}
12	1.00	1.00	1.00	0.50	4.3×10^{-24}	8.1×10^{-30}	1.7×10^{-32}	5.7×10^{-31}
13	1.00	1.00	1.00	1.00	0.50	4.3×10^{-12}	1.0×10^{-22}	1.5×10^{-26}
14	1.00	1.00	1.00	1.00	1.00	0.50	1.5×10^{-13}	9.3×10^{-22}
15	1.00	1.00	1.00	1.00	1.00	1.00	0.50	2.6×10^{-10}
16	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.50

Table 5.1: p-value from the Wilcoxon test for diameter d_1 and diameter d_2.

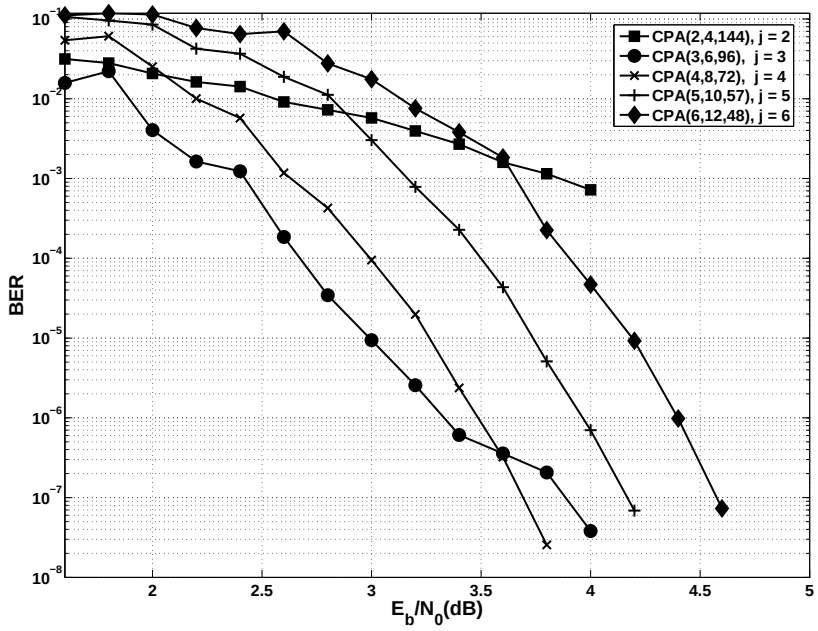
The test results are shown in Table 5.1. For diameters up to 11, these is no statistical relation between diameter and BER. However, for diameters 11 and above, any difference in diameters makes a statistically meaningful difference in the BER, as indicated by very small p-values in the upper diagonal portion of the table. For any d_1 and d_2 that satisfy $11 < d_1 < d_2$, we draw the conclusion that the codes with diameter d_2 have higher BER than the codes with diameter d_1. Due to the extremely small p-values in the corresponding results, the conclusion will not change for any significance level down to $\alpha = 4.8 \times 10^{-7}$. For two-sided tests, the new p-value p' can be simply computed by the following formula:

$$p' = \begin{cases} 2p, & p < 0.5 \\ 2(1-p), & p \geq 0.5. \end{cases} \tag{5.6}$$

In the distribution of uniformly sampled codes shown in Figure 5.5, the codes with diameter greater than 11 make up 1.72% of the all codes with girth 6. This indicates that, even though the random construction generates good CPA codes in terms of diameter with high probability, there is still a nonnegligible probability that bad codes will be chosen. Thus, this result justifies the incorporation of diameters in the CPA code construction.

5.3.3 Effect of Column Weight

The column weight of LDPC codes determine the number edges per coded bit and directly affects the decoding complexity of a given code when the iterative decoding algorithm is used. In

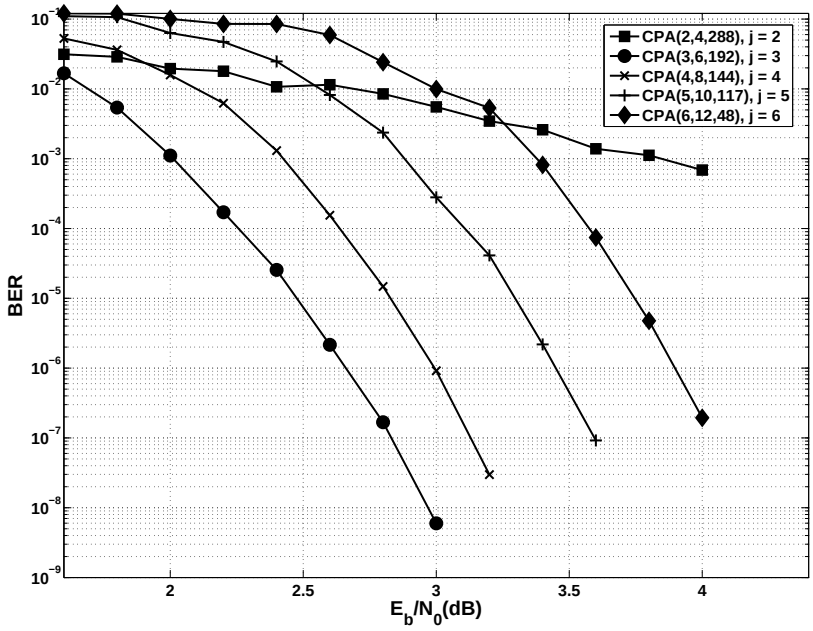


(a) N=576

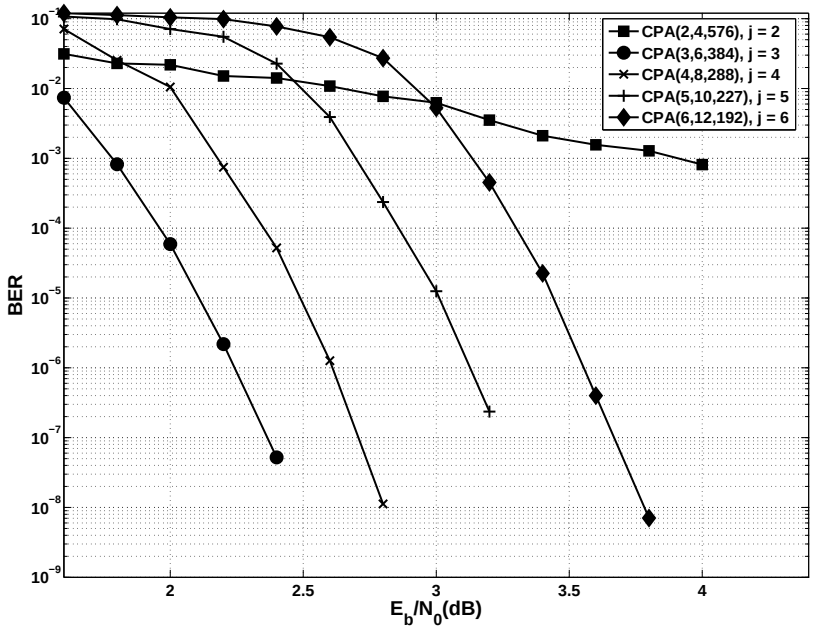
Figure 5.7: The effect of column weight on the performance

general, a higher column weight can provide a strong error correction capability since there are more check equations that involve each bit. However, a high column weight also generates a large number of short cycles in the Tanner graph, having a detrimental effect on the performance with the SP algorithm. The performance of half-rate PR-CPA codes with the column weight $j = 2, 3, 4, 5, 6$ are shown Figure 5.7. The plots are shown for the code length $N = 576, 1152, 2304$, and the size of the submatrix P has been appropriately adjusted to maintain the code length. For all three cases, the codes with $j = 3$ have the best result within the simulated range of SNR with the exception for $N = 576$ where the code with $j = 3$ shows a higher error floor. It can be seen that a higher column weight does not always lead to better BER performance although the lower bound on the minimum is an increasing function of the column weight as shown in eq. (1.3).

Also, the codes with $j = 2$ exhibit a very flat BER curve, showing a substantially high error rate in the entire SNR region.



(b) N=1152



(c) N=2304

Figure 5.7: The effect of column weight on the performance (cont'd)

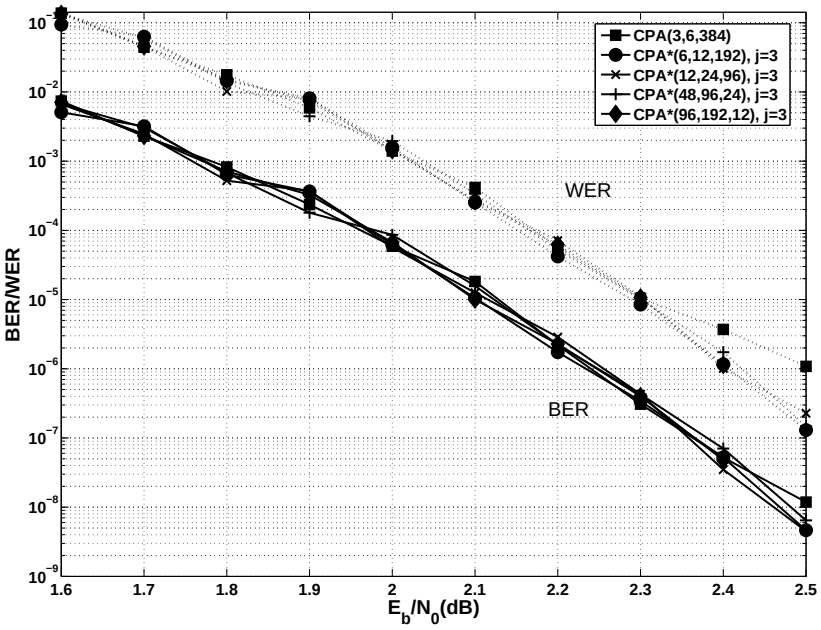


Figure 5.8: The effect of submatrix size P with with constant column weight

5.3.4 Effect of Submatrix Size P

With the CPA-structured codes, reducing the submatrix size P with the number of checks fixed increases the column weight by the relation $P = M/N_c$ and $j = N_c$. On the other hand, with the CPA* structure, the column weight can be maintained by replacing some of the submatrices by all-zero matrices. To see if the submatrix size P has any influence on the performance, several half-rate CPA* codes with different P are compared with a CPA(3,6,384) code, where all tested codes have the same rate, length, and column weight. The codes do not show any noticeable difference as shown in Figure 5.8. However, when designing a code with a very long code length, it would be more advantageous to use CPA* structure with a smaller P than to use CPA structure with a larger P since the CPA* structure allows the girth to be increased beyond 12 with a sufficiently large P while the CPA structure has the limitation of $g \leq 12$ regardless of P.

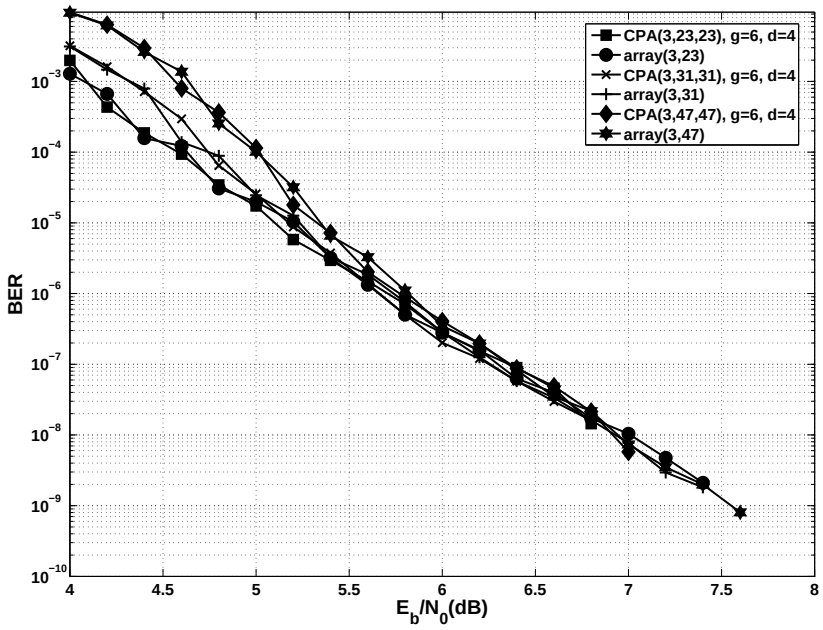
5.4 ARRAY CODE

The array code array(j, P), using the notation introduced in Section 1.1, has a CPA(j, P, P) structure where P is a prime and $j < P$. They are guaranteed to have a girth of 6 by the algebraic construction. It has been also found that they have diameter of 4, which is the lowest possible diameter for CPA-structured codes with girth 6. To compare the performance of array codes with pseudorandomly generated CPA codes, nine array codes array(j, P) with $j = 3, 4, 5$ and $P = 23, 31, 47$ are constructed and compared with PR-CPA codes with the same N_c, N_b and P. As seen in Figure 5.9a, the array codes with $j = 3$ show similar performance with their PR-CPA counterparts, where all codes have the same girth. On the other hand, for $P = 31$ and 47, the pseudorandom generation could not find CPA codes with girths more than 4. Accordingly, in Figure 5.9b, the array codes with $j = 4$ outperform the PR-CPA codes due to the higher error floor in the PR-CPA codes. However, the array codes are on par with their respective CPA counterparts before the error floor occurs. Finally, for the case of $j = 5$ shown in Figure 5.9c, the array codes show similar performance with PR-CPA, probably because the error floor of the CPA codes did not occur within the simulated SNR range.

5.5 FINITE GEOMETRY CODE

In this section, we evaluate the performance of the two-dimensional type-I (0,s)th-order EG-LDPC codes for $s = 2, 3, 4, 5$. For this class of codes, the parity check matrix is a $(2^{2s}-1)\times(2^{2s}-1)$ square matrix with column and row weight $j = k = 2^s$, forming a QC($1, 1, 2^{2s} - 1$) structure. Since $N_c = N_b = 1$, the codes are cyclic. The performance of EG-LDPC codes are shown in Figure 5.10.

To compare EG-LDPC codes with the pseudorandomly generated CPA codes, PR-CPA codes with $j = 3, 4, 5$ have been constructed where the $\mathbf{S}$-matrix parameters (N_c, N_b, P) have been chosen to make the code rate and codeword length close to those of the EG counterparts. Figure 5.11a shows that the (0,4)-th order EG code outperforms the CPA counterparts. The steep slope of the EG code in the waterfall region seems to be caused by the column weight $j = 16$, which is much higher

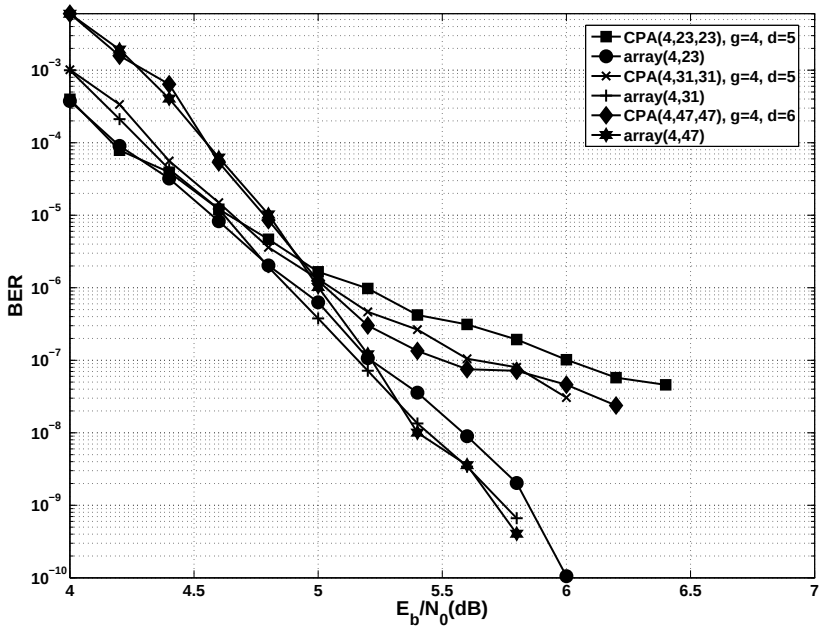


(a) j=3

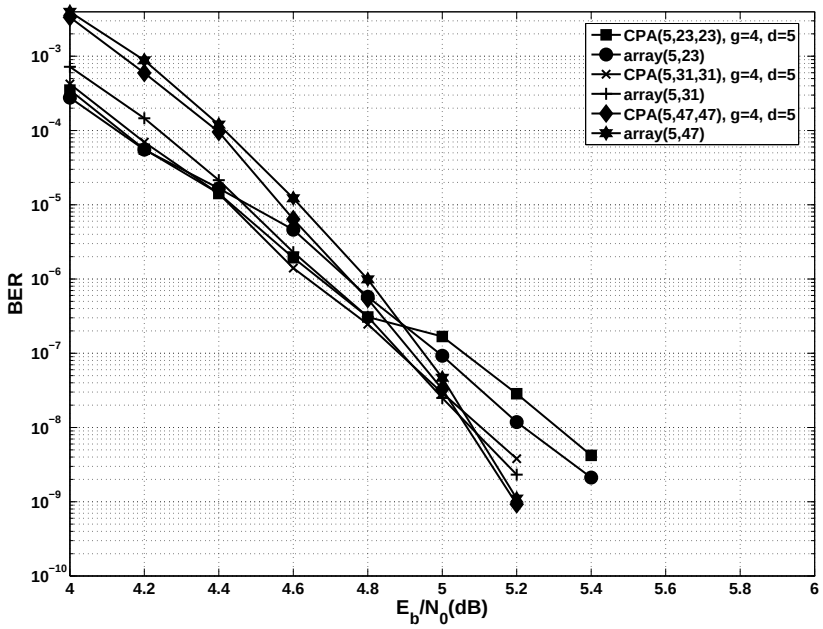
Figure 5.9: Comparison of array code and PR-CPA

than the CPA codes. However, it should be also noted that the CPA code with $j = 4$ has only 0.2 dB difference at the BER of 10^{-6} from the EG code while it has much lower decoding complexity. Another observation is that, unlike the case with half rate in Section 5.3.3, the column weight affects the slope of the BER curve.

The same comparison is made for the (0,5)-th order EG, as shown in Figure 5.11b. In this case, all CPA codes except for the one with $j = 3$ outperforms the EG-LDPC code within the simulated range. Although the EG-LDPC is expected to eventually outperform the CPA codes at very low BER due to the steeper curve, the short cycles generated by the high column weight of $j = 32$ seems to adversely affect the performance at the relatively high BER range. This implies that, for applications where the BER requirement is not very stringent such as wireless communication, the CPA-structured codes provide larger coding gain than the EG-LDPC code while they require less processing power at the decoder.



(b) j=4



(c) j=5

Figure 5.9: Comparison of array code and PR-CPA (cont'd)

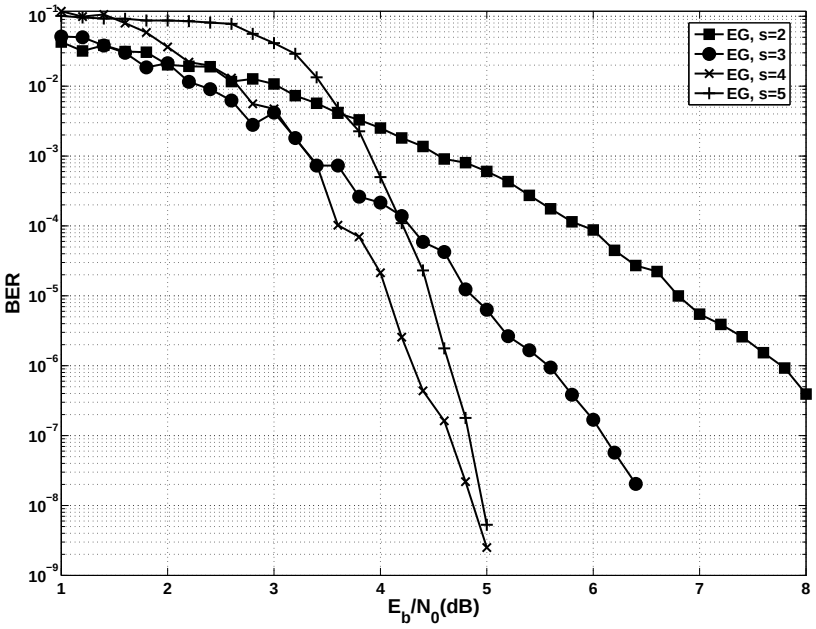
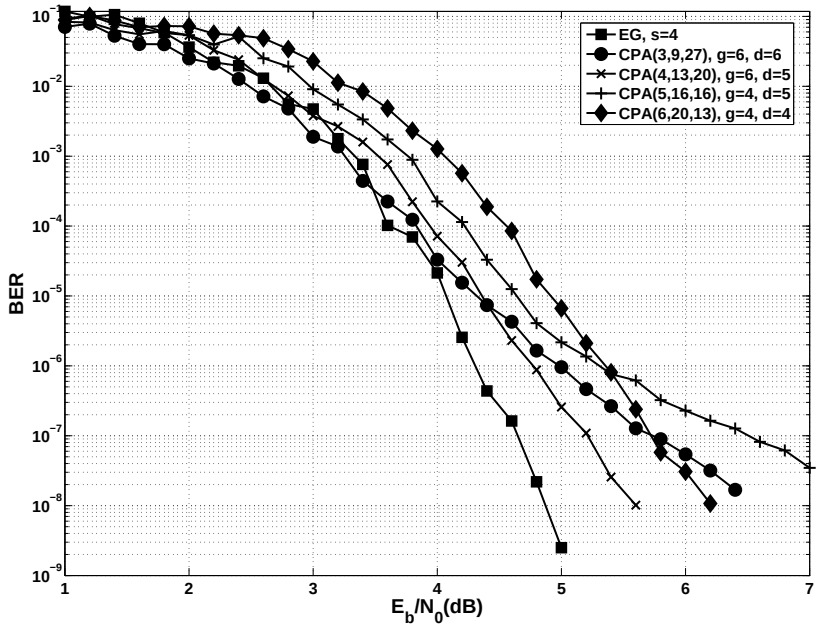


Figure 5.10: Comparison of EG and CPA codes

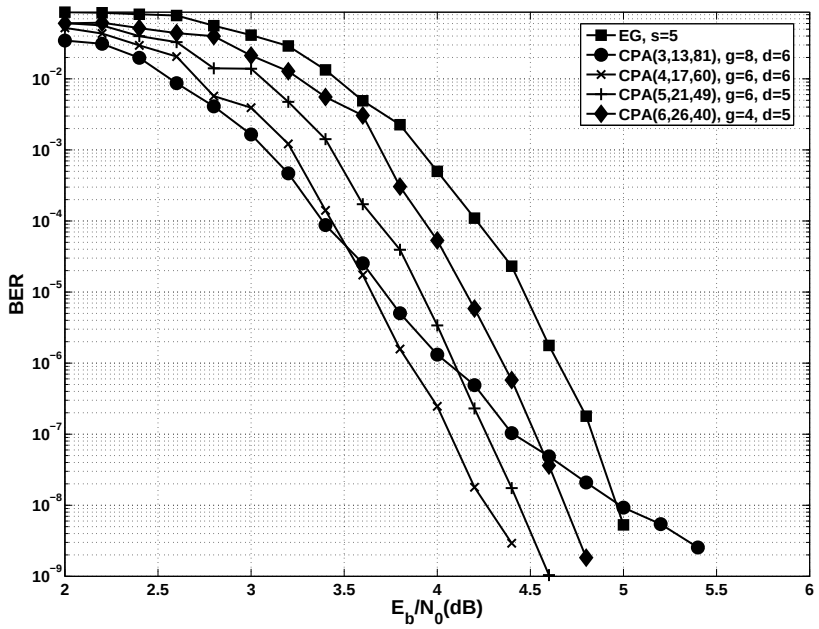
5.6 802.16E LDPC CODE

In the IEEE 802.16e standards for wireless metropolitan area network, the CPA*-structured LDPC codes have been adopted as an option for channel coding [16]. In the standards, the codes take the form of CPA*(N_c, N_b, P), where $N_b = 12$ and $N_c \in \{12, 8, 6, 4\}$ to support code rates $1/2, 2/3, 3/4$, and $5/6$. The submatrix size P is a multiple of 4 in the range from 24 and 96, which corresponds to the codeword lengths between $N = 576$ and $N = 2304$ at the multiples of 96 bits. The $\mathbf{S}$-matrix elements are specified for submatrix size $P = 96$, one representation for each of the rates $1/2$ and $5/6$, and two representations for each of the rates $2/3$ and $3/4$ (called type A and B). The $\mathbf{S}$-matrix elements for smaller P are derived by proportional scaling for code rates $1/2$, $3/4$ (type A and B), $2/3$ (type B) and $5/6$, i.e.,

$$S_{i,j}(P) = \begin{cases} S_{i,j}(96), & S_{i,j}(96) < 0 \\ \lceil S_{i,j}(96)\frac{P}{96} \rceil, & \text{otherwise} \end{cases}$$



(a) (0,4)th-order EG(N=255,K=175)



(b) (0,5)th-order EG(N=1023,K=781)

Figure 5.11: Comparison of type-I EG and CPA

```
-1 94 73 -1 -1 -1 -1 -1 55 83 -1 -1  7  0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 27 -1 -1 -1 22 79  9 -1 -1 -1 12 -1  0  0 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 24 22 81 -1 33 -1 -1 -1  0 -1 -1  0  0 -1 -1 -1 -1 -1 -1 -1 -1
61 -1 47 -1 -1 -1 -1 -1 65 25 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1 -1
-1 -1 39 -1 -1 -1 84 -1 -1 41 72 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 46 40 -1 82 -1 -1 -1 79  0 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1
-1 -1 95 53 -1 -1 -1 -1 -1 14 18 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1
-1 11 73 -1 -1 -1  2 -1 -1 47 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1 -1
12 -1 -1 -1 83 24 -1 43 -1 -1 -1 51 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1
-1 -1 -1 -1 -1 94 -1 59 -1 -1 70 72 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1
-1 -1  7 65 -1 -1 -1 -1 39 49 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0  0
43 -1 -1 -1 -1 66 -1 41 -1 -1 -1 26  7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0
```

Figure 5.12: **S**-matrix for 802.16e half-rate code for $P = 96$

or by modulo operation for code rate $2/3$ (type A), i.e.,

$$S_{i,j}(P) = \begin{cases} S_{i,j}(96), & S_{i,j}(96) < 0 \\ S_{i,j}(96) \mod P, & \text{otherwise} \end{cases}$$

The **S**-matrix representation for a half-rate code with $P = 96$ is shown in Figure 5.12. The S-matrices in the standards are partitioned into two parts, $S = [S_i S_p]$, where S_i is composed of the left $N_b - N_c$ columns of the original **S**-matrix and S_p contains the rest. In the encoding process, the S_i corresponds to the information bits and the S_p corresponds to parity bits, and S_i has higher column weights than S_p. Due to this irregularity, the information bits are involved in more check equations, and are thus better protected than the parity bits.

The 802.16e half-rate code is compared with a regular PR-CPA code with $j = 3$ in Figure 5.13, shown for the shortest ($P = 24, N = 576$) and longest ($P = 96, N = 2304$) codeword length. For $N = 576$, the regular PR-CPA code outperforms the 802.16e counterpart for in the entire SNR region. For $N = 2304$, the 802.16e code shows larger coding gain for BER less than 10^{-6} although it shows a higher error floor at around BER=10^{-6}. Considering that the target operating BER specified in the standards is 10^{-6}, it seems reasonable to use irregular codes to get better performance at the specific BER.

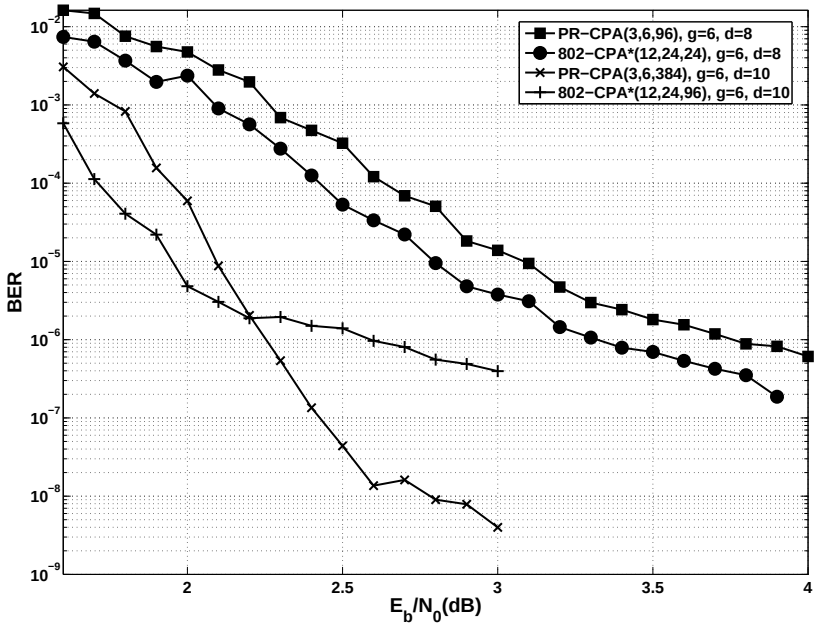
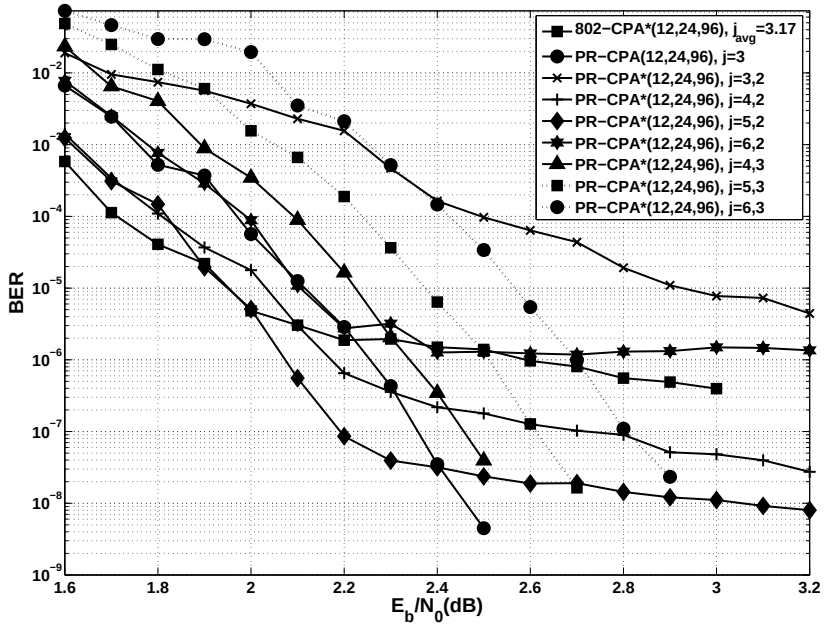


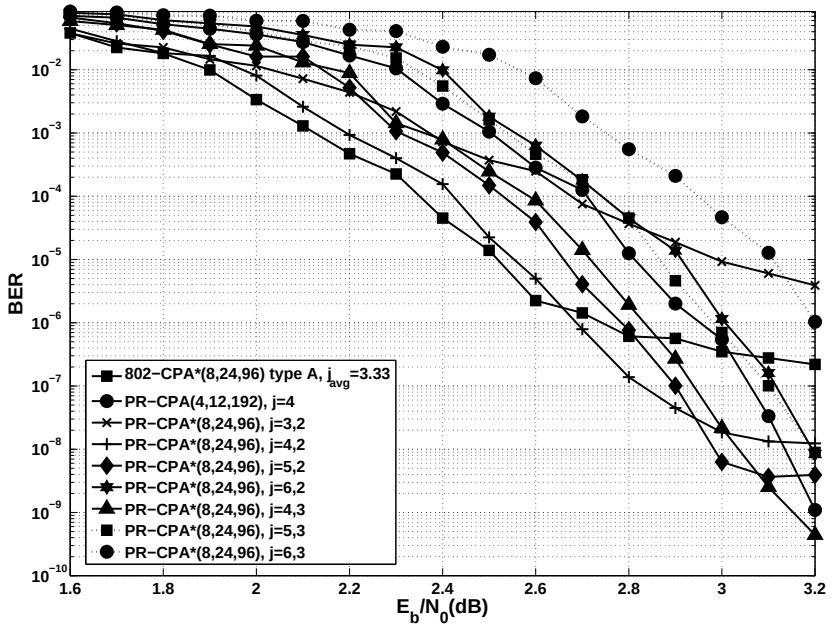
Figure 5.13: Comparison of half-rate 802.16e and PR-CPA codes

To better see the effect of nonuniform column weights, CPA*-structured codes have been constructed from the regular PR-CPA codes CPA(12,24,96) and CPA(8,24,96) by eliminating some of the nonnegative $\mathbf{S}$-matrix entries, i.e., replacing them by -1. The elimination is arranged to give two different column weights, larger for S_i and smaller for S_p. The performance of the CPA*-structured codes including the 802.16e code and the regular CPA-structured code are shown in Figure 5.14. For the half-rate codes in Figure 5.14a, the 802.16e codes are outperformed at BER=10^{-6} by some of the PR-CPA codes including the regular $j = 3$ codes. Irregular codes outperform the regular code when $j = 3$. On the other hand, for rate $2/3$ codes, the 802.16e code outperformed the regular code with $j = 4$. It is also observed that the irregular code with $j = (4, 2)$ gives a slightly larger coding gain than the 802.16e code.

As a further comparison between the 802.16e codes and the PR-CPA codes, the performance of the codes at the BER of 10^{-5}, 10^{-6} and 10^{-7} have been measured for all codeword lengths defined in the standards. The result is shown in Figure 5.15. For code rate = 1/2 (Figure 5.15a),



(a) rate = 1/2



(b) rate = 2/3

Figure 5.14: The performance comparison of regular/irregular column weights

as previously noticed, the regular CPA codes provides the SNR gain of between 0.1 and 0.5 dB at the target BER of 10^{-6} over the 802.16e codes. The gain becomes substantially larger for the BER of 10^{-7} although it would not be helpful for the wireless application. It is also noteworthy that the regular codes show a smooth increase in the coding gain as the codeword length increases while the 802.16e codes have fluctuations. For code rate = 2/3 (Figure 5.15b), the regular codes still tend to perform better than the 802.16e codes for shorter codes ($N < 1000$), but the opposite is true for longer codes ($N > 1000$). However, the PR-CPA*codes with $j = (4, 2)$, chosen based on the result in Figure 5.14b, slightly outperformed the 802.16e codes for $N \geq 1152$ with an exception at $N = 1344$.

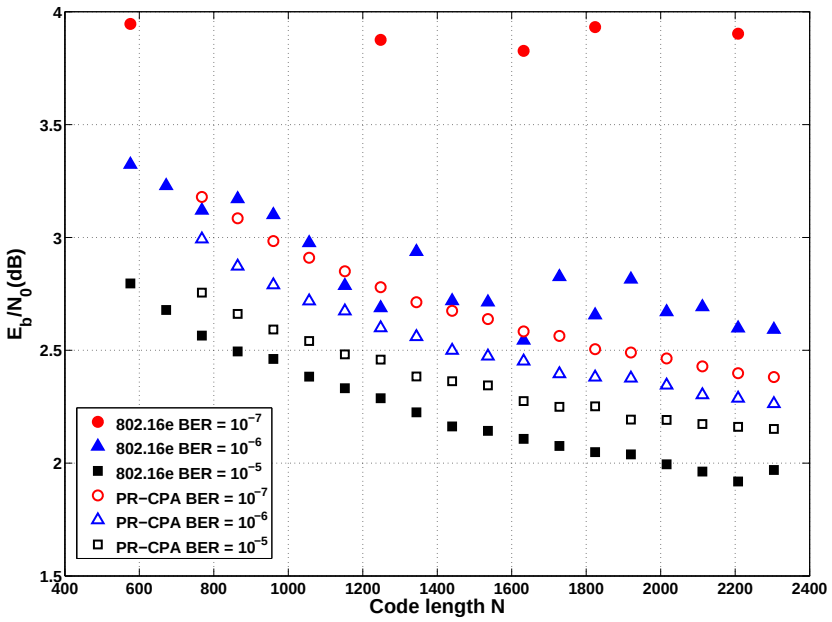
5.7 GPA-STRUCTURED LDPC CODE

As described in Section 2.4.4, GPA-structured codes have the potential to achieve larger girths than CPA structured codes, since they do not have the limitation of girth 12 that the CPA-structured codes have. Also, they are expected to be able to achieve large girths without a substantial increase in the codeword length, which is the case with the CPA*-structured codes. The search for good GPA-structured codes heavily relies on the existence of good non-abelian groups suitable for constructing large-girth codes. Once such group is found, a computer-based search should be conducted to check if a high girth code can be constructed with the elements in the chosen group. For this purpose, a search has been conducted using the GAP computer algebra system [50], which provides a library of all groups of order up to 1000. Due to the enormous number of groups and the considerable search time, the search space has been reduced by considering only the following two classes of groups:

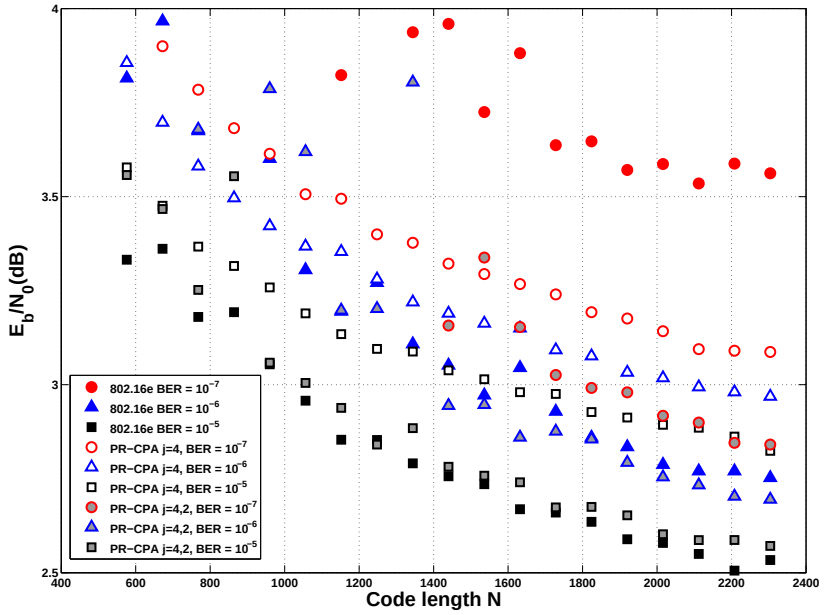
$$\text{group } G \text{ with order } n = pq,\ \ p : \text{prime}, p \mid (q-1) \tag{5.7}$$

where '|' denotes 'divides' and

$$\text{group } G \text{ with order } n = pqr,\ \ p : \text{prime}, p \mid (q-1), r \nmid \frac{q-1}{p}, G = C_{pr} : C_q, \tag{5.8}$$



(a) rate = 1/2



(b) rate = 2/3

Figure 5.15: Comparison between 802.16e and regular/irregular PR-CPA codes

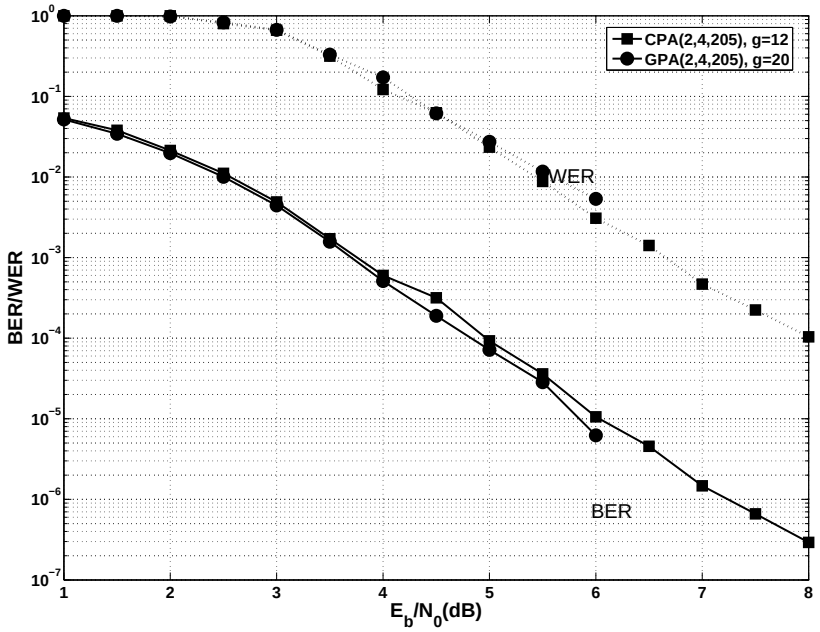
(N_c, N_b)	$(2,4)$ CPA,GPA1	$(3,4)$ CPA,GPA1,GPA2	$(3,6)$ CPA,GPA1,GPA2	$(3,9)$ CPA,GPA1,GPA2	$(4,8)$ CPA,GPA1,GPA2
$g=6$	$5,10$	$7,10,28$	$8,10,28$	$10,10,28$	$11,14,28$
$g=8$	$5,10$	$10,10,28$	$18,21,28$	$35,38,44$	$59,74,76$
$g=10$	$13,20$	$39,39,44$	$111,111,117$	$367,417,412$	$754,831,873$
$g=12$	$13,20$	$73,93,92$	$366,543,412$	$-,-,-$	$-,-,-$
$g=14$	$-,55$	$-,305,549$	$-,-,-$	$-,-,-$	$-,-,-$
$g=16$	$-,55$	$-,-,-$	$-,-,-$	$-,-,-$	$-,-,-$
$g=18$	$-,205$	$-,-,-$	$-,-,-$	$-,-,-$	$-,-,-$
$g=20$	$-,205$	$-,-,-$	$-,-,-$	$-,-,-$	$-,-,-$

Table 5.2: Smallest group order for each target girth g ('-' indicates that no codes have been found.)

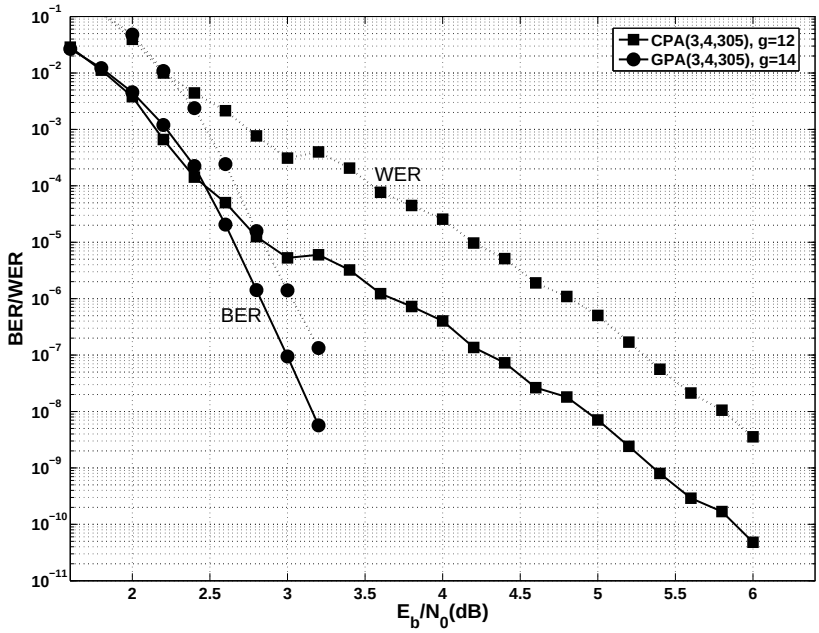
where C_n is a cyclic group of order n and ":" denotes semidirect product. The GPA codes based on the groups in eq. (5.7) and eq. (5.8) will henceforth be referred to as GPA1 and GPA2, respectively.

In the search process, for each of the groups belonging to GPA1 or GPA2, the GPA codes are constructed using the same sequential filling method as used by the CPA construction to find a code that has a target girth ranging from 6 to 20. From the search results for several $\mathbf{S}$-matrix sizes, the smallest group order n (which is also the submatrix size P in the GPA-structured codes) for each target girth is shown in Table 5.2. The search results for CPA-structured codes are also included here for comparison purposes. With the groups considered, the GPA-structured codes tend to require higher group order than the CPA-structured codes. While a dramatic increase in the girth from 12 to 20 could be achieved with a GPA(2,4,205) code, it was difficult to find GPA-structured codes having girths more than 12 for $j \geq 3$. The only such case found is GPA(3,4,305) with girth 14 compared to CPA(3,4,305) code with girth 12.

Since the GPA-structured codes are not quasi-cyclic, they cannot be simulated by the implemented hardware decoder. The result obtained from software simulation is shown in Figure 5.16. The GPA(2,4,205) code does not show a noticeable performance improvement over the CPA(2,4,205) code in spite of the large difference in the girth, which seems to have been caused by the pathological behavior of codes with column weight 2. On the other hand, the GPA(3,4,305) code shows a considerable performance improvement with a much steeper slope in the waterfall region than the CPA(3,4,305). Such large improvement with only an increase of 2 in the girth has not been seen in



(a) $(N_c, N_b, P) = (2, 4, 205)$



(b) $(N_c, N_b, P) = (3, 4, 305)$

Figure 5.16: The performance comparison of CPA and GPA structured codes

the experiments with CPA codes and makes further research on this class of codes very promising.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

The primary goals of this thesis are the design of a flexible decoder for QC-LDPC codes and the investigation of the performance of QC-LDPC codes with a focus on pseudorandom construction of circulant permutation arrays. The main results of this research can be summarized as the following:

1. **The design of a highly flexible decoder:** This thesis presents a generalized QC-LDPC decoder architecture designed with a priority placed on flexibility. The proposed decoder is capable of supporting a wide range of array-structured codes in the selected set of codes under investigation.

    - **Flexibly parameterizable architecture:** Unlike throughput-oriented architectures, this architecture decouples the degree of parallelism and resource utilization from the code parameters. This is accomplished by a new approach in which parallelism is achieved within submatrices rather than across submatrices. As a result, the decoder can be synthesized for a user-defined processing power and resource usage.
    - **Run-time reconfigurability:** As a benefit of the flexible architecture, the entire set of code parameters of QC-LDPC codes can be changed while the hardware is running. These parameters include the dimension of the array structure (N_c, N_b, P), the

column/row weights (j, k), and the actual shift values of circulants in the parity check matrix.

- **Architecture comparison:** The designed decoder has been implemented in FPGA, and the synthesis results have been compared with a selection of decoder architectures in the literature in terms of processing rate to area ratio (PAR). The area efficiency indicated by the PAR shows that the proposed architecture uses between 2.9 and 8.1 more area resource to generate the same processing power than more code-specific architectures. We believe that the order of the difference is reasonable considering the degree of flexibility provided by this architecture.

2. **Analysis tools for QC-LDPC codes:** This thesis presents new methods of finding the girth and the diameter of QC-LDPC codes.

   - **Girth-finding algorithm:** Finding the girths of a QC-LDPC decoder is more complicated than the CPA-structured codes. The proposed algorithm finds the cycle distribution of given codes by propagating messages in a compact form of Tanner graph where a node represents a group of actual nodes. The messages are represented as polynomials in X to keep track of the shift value differences. The presence of true cycles in the original Tanner graph can be detected by examining the coefficient of the term X^0. This algorithm is linear in the girth and the column/row weights.
   - **Diameter-finding algorithm:** Based on the observation that the input cost matrix for all-pairs shortest-path (APSP) algorithm for QC-LDPC codes is an array of circulants but not a circulant as a whole, a blocked version of APSP is applied to convert the original problem to subproblems where the input cost matrices are circulants. Within each subproblem, the APSP algorithm is replaced by modified versions of Dijkstra's algorithm to best exploit the circulant property. The runtime of the resulting algorithm is $O(N^2)$, where N is the codeword length while the direct application of the Floyd-Warshall algorithm has a runtime of $Theta(N^3)$.

3. **Performance study of CPA-structured codes:** The performance of a large selection of CPA-structured codes have been evaluated for the investigation of the following items.

   - **Effect of girth:** The effect of the girth on the BER performance was shown by constructing CPA-structured codes with the different girths. It has been observed that the girth does not have a major effect on the performance in the waterfall region but lowers the error floor. Thus, it justifies the pseudorandom construction of CPA-structured codes directed toward higher girth which is adopted in many construction methodologies based on computer search. However, it is not clear from the observed results whether this is a unique trait of CPA-structured codes or a general property of array-structured codes since no other construction methods provide a wide range of choices for the girth while keeping the other code parameters constant.

   - **Effect of diameter:** The effect on the diameter on the performance has been quantitatively measured by constructing CPA-structured codes with a wide range of diameters. It has been shown that the high diameter seriously affects the slope of the BER curve. To construct codes with a desired diameter, both the girth and diameter were incorporated into the pseudorandom code construction process. To separate the artifact of the explicit efforts to control the diameter, a large set of codes have been generated by uniform random sampling. Using the random samples, the Mann-Whitney-Wilcoxon nonparametric test has been conducted. From the test results with uniformly sampled girth-6 CPA$(3, 6, 384)$ codes, it has been shown that every increase in diameter beyond 11 has an adverse effect on the BER performance with very high statistical significance.

   - **Effect of column weight:** The column weight linearly increases the decoding complexity of the sum-product algorithm and provides a lower bound on the minimum distance. From the observed results from the experiment designed for the column weight effect, it is seen that the higher column weight increases the BER without an apparent change in the slope of the BER curve. However, in some other experiments with higher rate,

it has been noticed that the column weight increased the slope. For the simulated BER region down to 10^{-9}, the column weight of 3 or 4 gave the best results.

4. **Performance comparison between pseudorandomly constructed CPA-structured codes and deterministically constructed codes:** The pseudorandom construction of CPA-structured codes is very flexible in choosing a desired rate or codeword length. Using this design flexibility, deterministically constructed codes based on finite-geometry and arithmetic progression in powers have been compared with CPA-structured codes with matching code rate and length.

   - **Finite geometry codes:** A selection of (0,s)-th cyclic Euclidean geometry codes of length $2^{2s} - 1$ have been compared with the CPA-structured codes. For $s = 4$, EG codes gave the best performance, but for $s = 5$, CPA-structured code with column weight $4, 5$ or 6 outperformed the EG with column weight 32 at the BER range down to 10^{-9}. Considering the low column weight of such CPA codes, it shows that the CPA structure can be a cost-effective solution for certain applications.
   - **Array codes:** Array codes have the same CPA structure with a deterministic arithmetic progression in the powers. They have a fixed girth of 6 and fixed diameter 4. When the pseudorandomly constructed codes have the same girth, no difference in the performance is observed. However, for $\mathbf{S}$-matrices with large dimensions, it is not always possible for CPA codes to achieve girth 6. When there is a girth difference, the array codes have better performance due to the higher error floor of the pseudorandomly constructed counterparts with girth 4. It can be seen that, for a high rate CPA-structure, the array codes are a better choice than the pseudorandom construction in that the girth of 6 is guaranteed. Conversely, for low-rate codes with smaller $\mathbf{S}$-matrix dimensions, the pseudorandom construction can yield better codes by enlarging the girth.
   - **802.16e codes:** The LDPC codes in the 802.16e are derived from a fixed base $\mathbf{S}$-matrix. They have unbalanced column weights for the information bits and the parity bits. In the

experiments with half-rate codes, it has been shown that a regular CPA codes outperform the 802.16e codes by between 0.1dB and 0.5dB at the target BER of 10^{-6}. For code rate $2/3$, the 802.16e codes outperform the regular CPA codes, but give comparable performance to the pseudorandomly constructed irregular codes.

- **GPA-structured codes:** The underlying group structure of the GPA-structured codes is not cyclic and thus the limitation of girth 12 does not apply to these codes. In the computer search for GPA-structured codes with a selection of group structures, it has been observed that GPA codes tend to require large submatrix size P to attain the same girth and GPA codes with higher girths than 12 are very difficult to find. However, the GPA(3,4,305) code gave a performance improvement of 0.8 dB at the BER of 10^{-6} with a much steeper BER curve.

6.2 FUTURE WORK

The possible directions of the future research are as follows:

- **Pseudorandom construction methods for QC-LDPC codes:** The QC-LDPC code structure encompasses a much larger code space than the CPA structure while having the same encoding complexity as the CPA-structured codes. Since many good algebraic codes have a QC structure, a pseudorandom construction method of QC-LDPC codes will provide a very flexible design tool for more powerful LDPC codes that can compete with any other deterministically generated codes.

- **Search for other graph parameters that affect the performance:** Even though the girth and diameter have been shown to have major effects on the performance of a code, they are by no means sufficient to predict the superiority of a given code to others. For example, in the uniform random sampling test in Chapter 5.3.2, a large difference between the best code and the worst code has been observed within the set of codes with the same girth and diameter. If other graph parameters that affect the performance can be found, they can be incorpo-

rated into the pseudorandom code construction process. One possibility is the use of cycle distribution that can already be obtained using our girth-finding algorithm. Another possibility is the incorporation of trapping set (or pseudo-codewords) analysis into the construction process, which has been developed to explain and analyze the error floor [51, 52]. With the cyclic-symmetry in the QC-LDPC codes, an efficient method for trapping set analysis can be expected to exist.

- **Computer search for more GPA-structured codes:** The GPA-structured codes with high girth are hard to find, but it has been shown that they bring a considerable improvement to performance. If a sufficiently large number of GPA codes can be found, it will form a new class of very good structured codes.

- **Hardware support for GPA-structured codes:** While GPA-structured codes may bring a major improvement in the performance, they do not belong to the class of the QC-LDPC codes. It will be crucial to find ways to exploit the structure of the GPA-structured codes in order to design efficient encoders and decoders that can be used in practical systems.

REFERENCES

[1] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding 5th IMA Conf., in Lecture Notes in Computer Science*, no. 1025. C. Boyd., Editor, 1995, pp. 100–111.

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

[3] Y. Kou, S. Lin, and M. Fossorier, "Low density parity check codes: Construction based on finite geometries," in *Proc. IEEE Global Telecommunications Conference*, vol. 2, Nov. 2000, pp. 825–829.

[4] ——, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inform. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[5] S. J. Johnson and S. R. Weller, "Construction of low density parity check codes from kirkman triple systems," in *Proc. IEEE Global Telecommunications Conference*, vol. 2, Nov. 2001, pp. 970–974.

[6] H. Song, J. Liu, and B. V. K. V. Kumar, "Low complexity LDPC codes for partial response channels," in *Proc. IEEE Global Telecommunications Conference*, vol. 2, Nov. 2002, pp. 1294–1299.

[7] B. Ammar, B. Honary, Y. Kou, J. Xu, and S. Lin, "Construction of low density parity check codes based on balanced incomplete block designes," *IEEE Trans. Inform. Theory*, vol. 50, no. 6, pp. 1257–1269, Jun. 2004.

[8] B. Vasic and O. Milenkovic, "Combinatorial constructions of LDPC codes," *IEEE Trans. Inform. Theory*, vol. 50, no. 6, pp. 1156–1176, Jun. 2004.

[9] J. L. Fan, "Array codes as low-density parity-check codes," in *Proc. 2nd Int. Symp. Turbo Codes and Related Topics*, Brest, France, Sep. 2000, pp. 553–556.

[10] M. Blaum and R. Roth, "New array codes for multiple phased burst corrections," *IEEE Trans. Inform. Theory*, vol. 39, no. 1, pp. 66–67, January 1993.

[11] M. Blaum, P. Farrell, and H. van Tilborg, *Handbook of COding Theory*. V.S. Pless and W.C. Huffman (Eds.), Elsevier Science B.V., 1998.

[12] R. M. Tanner, D. Srkdhara, and T. Fuja, "A class of group-structured LDPC codes," in *ICSTA'01*, Ambleside, England, 2001.

[13] B. Vasic and O. Milenkovic, "Combinatorial constructions of LDPC codes," *IEEE Trans. Inform. Theory*, vol. 50, no. 6, pp. 1156–1176, Jun. 2004.

[14] J. Lu, J. M. F. Moura, and U. Niesen, "Grouping-and-shifting designs for structured LDPC codes with large girth," in *ISIT'04: Int. Symp. on Information Theory*, Chicago, IL, Jun. 2004, p. 236.

[15] J. Lu, "Designing structured low density parity check codes with large girth," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, Dec. 2004.

[16] *IEEE 802.16e: Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, IEEE Std. 802.16e, 2006.

[17] G. A. Margulis, "Explicit constructions of graphs without short cycles and low density codes," *Combinatorica*, vol. 2, no. 1, pp. 71–78, 1982.

[18] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: MIT Press, 1963.

[19] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533–547, Nov. 1981.

[20] J. Campello, D. S. Modha, and S. Rajagopalan, "Designing LDPC codes using bit filling," in *Proc. IEEE International Conference on Communications*, vol. 1, Jun. 2001, pp. 55–59.

[21] Y. Mao and A. H. Banihashemi, "A heuristic search for good LDPC codes at short block lengths," in *Proc. IEEE International Conference on Communications*, vol. 1, Jun. 2001, pp. 41–44.

[22] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding," in *Proc. IEEE International Conference on Communications*, May. 1993, pp. 1064–70.

[23] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 585–598, Feb 2001.

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

[25] T. J. Richardson, A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 618–637, Feb. 2001.

[26] S.-Y. Chung and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 657–618, Feb. 2001.

[27] R. Yazdani and M. Ardakani, "An efficient analysis of inite-length LDPC codes," in *Proc. IEEE International Conference on Communications*, June 2007, pp. 677–682.

[28] Z. Li, L. Chen, L. Zeng, S. Lin, and W. H. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Trans. Commun.*, vol. 54, no. 1, pp. 71–81, Jan. 2006.

[29] Z. He, S. Roy, and P. Fortier, "Encoder architecture with throughput over 10 gbit/sec for quasi-cyclic LDPC codes," in *IEEE International Symposium on Circuit and Systems (ISCAS'06)*, May 2006, pp. 3269–3272.

[30] F. Kschischang, J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, pp. 498–519, Feb. 2001.

[31] J. Heo, "Analysis of scaling soft information on low density parity check codes," in *Elect. Letters*, vol. 39, no. 2, Jan. 2003, pp. 219–221.

[32] M. Karkooti and J. Cavallaro, "Semi-parallel reconfigurable architectures for real-time LDPC decoding," in *Proc. IEEE International Conference on Information Technology: Coding and Computing (ITCC'04)*, vol. 1, 2004, pp. 579–585.

[33] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, pp. 80–83, 1945.

[34] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, pp. 50–60, 1947.

[35] D. A. Wolf, "Some general results about uncorrelated statistics," *J. Amer. Statist. Assoc.*, vol. 68, pp. 1013–1018, 1973.

[36] D. Bertsekas and R. Gallager, *Data Networks*. Upper Saddle River: Prentice Hall, 1992.

[37] S.-C. Han, F. Franchetti, and M. Püschel, "Program generation for the all-pairs shortest path problem," in *PACT '06: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*. New York, NY, USA: ACM Press, 2006, pp. 222–232.

[38] G. Venkataraman, S. Sahni, and S. Mukhopadhyaya, "A blocked all-pairs shortest-paths algorithm," *J. Experimental Algorithms*, vol. 8, p. 2.2, 2003.

[39] J.-S. Park, M. Penner, and V. K. Prasanna, "Optimizing graph algorithms for improved cache performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 769–782, 2004.

[40] A. Blanksby and C. Howland, "A 690-mw 1-gb/s 1024-b, rate 1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, March 2002.

[41] L. Sun and B. V. Kumar, "Field programmable gate array implementation of a generalized decoder for structured low-density parity check codes," in *Proc. IEEE International Conference on Field-Programmable Technology*, 2004, pp. 17–24.

[42] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Trans. Circuits Syst.*, vol. 54, no. 6, pp. 542–546, June 2007.

[43] F. Guilloud, E. Boutillon, J. Tousch, and J. Danger, "Generic description and synthesis of LDPC decoders," *IEEE Trans. Commun.*, to be published.

[44] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Investigation of error floors of structured low-density parity-check codes by hardware emulation," in *Proc. IEEE Global Telecommunications Conference*, Nov. 2006, pp. 1–6.

[45] G. Box and M. Muller, "A note on the generation of random normal deviates," *Ann. Math. Statist*, vol. 29, pp. 610–611, 1958.

[46] D. Lee, W. Luk, and J. Villasenor, "A hardware gaussian noise generator for channel code evaluation," in *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'03)*, Apr. 2003, pp. 69–78.

[47] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.

[48] S. Konuma and S. Ichikawa, "Design and evaluation of hardware pseudo-random number generator mt19937," *IEICE = Trans. Inf. Syst.*, vol. E88-D, no. 12, pp. 2876–2879, 2005.

[49] D. Lee, W. J. Villasenor, W. Luk, and P. Leong, "A hardware gaussian noise generator using the box-muller method and its error analysis," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 659–671, June 2006.

[50] T. G. Team. (1997) GAP - *Groups, Algorithms, and Programming*. [Online]. Available: http://www-gap.dcs.st-and.ac.uk/ gap/

[51] D. MacKay and M. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, pp. 80–83, 2003.

[52] S. Landner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes," in *Proc. Wireless Networks, Communications and Mobile Computing Conference*, vol. 1, 2005, pp. 630–635.