PROTOGRAPH-BASED LOW-DENSITY PARITY-CHECK HADAMARD CODES

PENGWEI ZHANG

PhD

The Hong Kong Polytechnic University

2021

The Hong Kong Polytechnic University
Department of Electronic and Information Engineering

Protograph-Based Low-Density Parity-Check Hadamard Codes

Pengwei Zhang

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

July 2021

Certificate of Originality

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

________________ (Signed)

Pengwei ZHANG (Name of student)

Abstract

This thesis proposes and analyzes a new class of ultimate-Shannon-limit approaching codes, namely protograph-based low-density parity-check (PLDPC) Hadamard codes. This class of code has a low code rate and can achieve excellent error performance even at a very low bit-energy-to-noise-power-spectral-density ratio (i.e., $E_b/N_0 < 0$ dB). Application scenarios include multiple access wireless systems with a huge number of non-orthogonal users and deep space communications.

Firstly, we describe the protograph structure and protomatrix of a protograph-based low-density parity-check Hadamard block code (PLDPCH-BC). To optimize the structure of the PLDPCH-BC, we propose a low-complexity Protograph Extrinsic Information Transfer (PEXIT) method based on Monte Carlo simulations. Given multiple *a priori* information and channel information, the proposed method can obtain multiple extrinsic mutual information (MI) from the symbol-by-symbol maximum *a posteriori* probability (symbol-MAP) Hadamard decoder. Moreover, this method is applicable to low/high and/or even/odd order of Hadamard codes, and can compute the theoretical thresholds of PLDPCH-BCs with degree-1 or/and punctured variable nodes. Optimized designs for PLDPCH-BCs with Hadamard codes of different orders are derived. Simulations are performed on the constructed codes and the simulated error rates are compared with those of traditional LDPC-Hadamard codes. In addition, PLDPCH-BCs are punctured and their simulation results are compared with unpunctured PLDPCH-BCs.

Secondly, we propose an efficient and effective layered decoding algorithm for PLDPCH-BCs, and compare its convergence speed with that of the standard decoding algorithm. We further implement the proposed layered decoding algorithm onto hardware, namely an FPGA board, and evaluate its error performance under different throughputs. The error degradation due to fixed-point computation is also evaluated.

Thirdly, we make use of the optimized PLDPCH-BC designs to construct spatially-coupled PLDPC-Hadamard convolutional codes (SC-PLDPCH-CCs), the error performance of which is also close to the ultimate Shannon limit. We introduce the encoding of SC-PLDPCH-CCs using their convolutional parity-check matrices. We propose a pipelined decoding strategy with a layered decoding algorithm so as to perform efficient and effective decoding for the SC-PLDPCH-CCs. We simulate the error performance of SC-PLDPCH-CCs with different rates and different number of processors

contained in pipeline decoding. The error performance of the SC-PLDPCH-CCs is compared with that of PLDPCH-BCs.

Publications

Journal papers:

- **Peng-Wei Zhang**, F. C. M. Lau, and C.-W. Sham, "Layered decoding for protograph-based low-density parity-check Hadamard codes," ***IEEE Communications Letters***, vol. 25, no. 6, pp. 1776-1780, 2021, doi: 10.1109/LCOMM.2021.3057717.
- **Peng-Wei Zhang**, F. C. M. Lau, and C.-W. Sham, "Protograph-based low-density parity-check Hadamard codes," ***IEEE Transactions on Communications***, vol. 69, no. 8, pp. 4998-5013, 2021, doi: 10.1109/TCOMM.2021.3077939.
- **Peng-Wei Zhang**, F. C. M. Lau, and C.-W. Sham, "Spatially coupled PLDPC-Hadamard convolutional codes," in preparation.
- **Peng-Wei Zhang**, F. C. M. Lau, and C.-W. Sham, "Hardware Architecture of Layered Decoders for PLDPC-Hadamard Codes," in preparation.

Conference papers:

- **Peng-Wei Zhang**, F. C. M. Lau and C.-W. Sham, "Protograph-based LDPC-Hadamard codes," **in *Proceedings of 2020 IEEE Wireless Communications and Networking Conference (WCNC)***, pp. 1-6, 2020, doi: 10.1109/WCNC45663.2020.9120683.
- **Peng-Wei Zhang**, F. C. M. Lau and C.-W. Sham, "Design of a high-throughput low-latency extended Golay decoder," **in *Proceedings of 2017 23rd Asia-Pacific Conference on Communications (APCC)***, pp. 1-4, 2017, doi: 10.23919/APCC.2017.8304002.

Others:

- **Peng-Wei Zhang**, F. C. M. Lau, and C.-W. Sham, "Protograph-based low-density parity-check Hadamard codes," *arXiv:2010.08285*, 2021.

Acknowledgments

The doctoral studies in the past five years have benefited me a lot. On the occasion of graduation, I would like to thank my supervisor Prof. Francis C. M. Lau for giving me the opportunity to pursue my PhD degree, and thank him for the key guidance on my research direction. His rigorous attitude towards academic has impressed me deeply, which is also worthy of my learning.

I would like to thank Dr. Bruce C. W. Sham for answering the questions I encountered in the process of learning hardware. I would like to thank Dr. Wai-Man Tam and Dr. Sheng Jiang for discussing and solving the difficulties encountered in the projects. While solving these problems and difficulties, it helps a lot with my research topics. I would also like to thank the visiting student Zhaojie Yang for talking about some interesting things in study and life.

Finally, I would like to thank my parents and my uncle. With their support and encouragements, I can complete my study more motivated.

Contents

List of Figures

List of Tables

Part I

INTRODUCTION

Chapter 1

INTRODUCTION

1.1 DEVELOPMENT OF LDPC CODES

In 1943, Claude Shannon derived the channel capacity theorem [2], based on which the maximum rate that information can be sent through a channel without errors can be evaluated. In 1993, Berrou *et al*. invented the turbo codes and demonstrated that with a code rate of 0.5, the proposed turbo code and decoder could work within 0.7 dB from the capacity limit at a bit error rate (BER) of 10^{-5} [3], [4]. Besides turbo codes, other well-known capacity-approaching codes are low-density parity-check codes (proposed by Gallager in 1960s [5] and rediscovered by MacKay and Neal in 1990s [6]) and polar codes (proposed by Arikan in 2009 [7]). These capacity-approaching codes have since been used in many wireless communication systems (e.g., 3G/4G/5G, Wifi, satellite communications) [8–10], optical communication systems [11] and magnetic recording systems [12, 13]. The progresses of the aforementioned three types of capacity-approaching codes over the past decades can be found in the survey papers [14–18] and the references therein.

In particular, an LDPC code can be represented by a matrix containing a low density of "1"s and also by its corresponding Tanner graph [19]. In the Tanner graph, there are two sets of nodes, namely variables nodes (VNs) and check nodes (CNs), sparsely connected by links. Messages are updated and passed iteratively along the links during the decoding process [20, 21]. Density evolution (DE) [22] is a kind of analytical method that tracks the probability density function (PDF) of the messages after each iteration. It not only can predict the convergence of the decoder, but also can be used for optimizing LDPC code designs [21]. The extrinsic information transfer (EXIT) chart is another common technique employed to analyze and optimize LDPC codes [23–25]. An optimal LDPC code design is found when the EXIT curves of the VNs and CNs are "matched" with the smallest bit-energy-to-noise-power-spectral-density ratio (E_b/N_0).

For an LDPC code with given degree distributions and code length, the progressive-edge-growth (PEG) method [26–28] is commonly used to connect the VNs and CNs with an aim to maximizing the girth (shortest cycle) of the code. The method is simple and the code can achieve good error performance. However, the code has a quadratic encoding complexity with its length because it is unstructured. The hardware implementation of the encoder/decoder also consumes a lot of resources and has high routing complexity.

Subsequently, structured quasi-cyclic (QC) LDPC codes are proposed [29]. QC-LDPC codes have a linear encoding complexity and allow parallel processing in the hardware implementation. Other structured codes, such as the repeat-accumulate (RA) codes and their variants, can be formed by the repeat codes and the accumulators [30–33]. They belong to a subclass of LDPC codes that have a fast encoder structure and good error performance [34, 35]. Structured LDPC codes can also be constructed by protographs [36]. By expanding a protomatrix (corresponding to a protograph) with a small size, a QC matrix (corresponding to a lifted graph) that possesses the same properties as the protomatrix can be obtained. The codes corresponding to the lifted graphs are called protograph-based LDPC (PLDPC) codes. The traditional EXIT chart cannot be used to analyze protographs where degree-1 and/or punctured variable nodes exist. Subsequently, the protograph EXIT (PEXIT) chart method is developed [37] for analyzing and designing PLDPC codes, and well-designed PLDPC codes are found to achieve performance close to the Shannon limit [17], [38]. Moreover, in the case of block-fading channels, root-protograph LDPC codes are analyzed [39] and found to achieve near-outage-limit performance [40].

Based on the LDPC block codes (LDPC-BCs) mentioned above, memory is introduced into the code designs to construct LDPC convolutional codes (LDPC-CCs). LDPC-CCs were first proposed in [41] and characterized by the degree distributions of the underlying LDPC-BCs. By applying a sliding window decoding [42, 43], LDPC-CCs can achieve convolutional gains over their block-code counterparts. In addition, spatially coupled LDPC (SC-LDPC) codes are constructed by coupling L LDPC-BCs, which can enhance their theoretical thresholds and decoding performance [44, 45]. As L tends to infinity, spatially coupled LDPC convolutional codes (SC-LDPC-CCs) are obtained. In [46] and [47], SC-LDPC-CCs have been shown to achieve capacity over binary memoryless symmetric channels under belief propagation (BP) decoding. Moreover, the spatially coupled codes have been applied in multiuser detection [48] and multiple access channels [49–52]. In [53], SC-LDPC-CCs have been constructed from the perspective of protographs, forming SC-PLDPC-CCs. Through the edge-spreading procedure on a protomatrix, the threshold, convergence behavior and error performance of SC-PLDPC ensembles have also been systematically investigated.

In the Tanner graph of an LDPC code, the VNs are equivalent to repeat codes while CNs correspond to single-parity-check (SPC) codes. If other block codes, such as Hamming codes and BCH codes, are used to replace the repeat codes and/or SPC codes, generalized LDPC (GLDPC) codes are obtained [54–56]. In [57–59], doped-Tanner codes are formed by replacing the SPC component codes in the structured LDPC codes with Hamming codes and recursive systematic convolutional codes. Ensemble codeword weight enumerators are used to find good GLDPC codes while Hamming codes have been used to design medium-length GLDPC codes with performances approaching the channel capacity (> 0 dB). In [60] and [61], EXIT functions of block codes over binary symmetric channels have been derived and used for analyzing LDPC codes. The use of linear programming algorithm to optimize a rate-8/9 GLDPC code from the perspective of degree distribution is further demonstrated [62]. To achieve good error performance (BER = 10^{-5}) at very low E_b/N_0, say < -1.15 dB, Hadamard codes have been proposed to replace the SPC codes, forming the low-rate ($\leqslant 0.05$) LDPC-Hadamard codes [63], [1]. By adjusting the degree distribution of the VNs and using the EXIT chart technique, the EXIT curves of the Hadamard "super CNs" and VNs are matched and excellent error performance at low E_b/N_0 is obtained.

In practice, different channels possess different capacities, depending on factors such as modulation scheme, signal-to-noise ratio and code rate. However, the "ultimate Shannon limit" over an additive-white-Gaussian-noise (AWGN) channel remains at -1.59 dB, i.e., $E_b/N_0 = -1.59$ dB [64]. Scenarios where digital communications may need to work close to the ultimate Shannon limit include deep space communications, multiple access (e.g. code-division multiple-access [65] and interleave-division multiple-access [66–68]) with severe inter-user interferences, or embedding low-rate information in a communication link. The most notable channel codes with performance close to this limit are turbo-Hadamard codes [69–72], concatenated zigzag Hadamard codes [73], [74], and LDPC-Hadamard codes [63], [1]. When applying these codes in the scenarios above, we can not only ensure reliable data transmission, but also increase the transmission distance under the same transmission energy, or reduce the transmission energy under the same transmission distance. However, both turbo-Hadamard codes and concatenated zigzag Hadamard codes require the use of forward/backward decoding algorithms and hence will have long decoding latencies [69], [73]. The LDPC-Hadamard codes allow parallel processing and hence the decoding latency can be made much shorter [1]. However, in optimizing the threshold of LDPC-Hadamard codes, only the degree distribution of the variable nodes has been found for a given order of the Hadamard code used. Therefore, the method used in optimizing LDPC-Hadamard codes has the following drawbacks.

- For the same variable-node degree distribution, many different code realizations with very diverse bit-error-rate performances can be obtained.
- The code is unstructured, making both encoding and decoding very complex to realize in practice. We take the LDPC-Hadamard code with code rate $R = 0.05$ and Hadamard code order $r = 4$ as an example. For an information length of $65,536$, the degree distributions optimized by [1] indicate that there are $113,426$ Hadamard check nodes and $n = 178,962$ variable nodes. When these large number of nodes are connected by the PEG algorithm, the resultant graph has little structure and is therefore not conducive to parallel encoding/decoding and reduces encoding/decoding efficiency. In the hardware implementation, the unstructured conventional LDPC-Hadamard code further results in high routing complexity and low throughput.
- The degree distribution analysis requires a minimum variable-node degree of 2 because an EXIT curve cannot be produced for degree-1 variable nodes. Moreover, LDPC-Hadamard codes with punctured variable nodes cannot be analyzed.

The concept in [1] has been applied to designing other low-rate generalized LDPC codes [75]. However, the main criterion of those codes is to provide low latency communications and hence their performance is relatively far from the ultimate Shannon limit [64].

1.2 Thesis innovations

To solve the issues of traditional LDPC-Hadamard codes, we design LDPC-Hadamard codes from the perspective of protographs. Hence, this thesis consists of three main innovations: PLDPC-Hadamard block codes (PLDPCH-BCs), layered decoding algorithm, and spatially coupled PLDPC-Hadamard convolutional codes (SC-PLDPCH-CCs).

Firstly, we propose a method to design LDPC-Hadamard codes which possess degree-1 and/or punctured VNs. The technique is based on applying Hadamard constraints to the CNs in a generalized PLDPC code, followed by lifting the generalized protograph. We name the codes formed *protograph-based LDPC Hadamard* (PLDPC-Hadamard) codes [76]. We also propose a modified PEXIT algorithm for analyzing and optimizing PLDPC-Hadamard code designs. Codes with decoding thresholds ranging from -1.53 dB to -1.42 dB have been found, and simulation results show a bit error rate of 10^{-5} can be achieved at $E_b/N_0 = -1.43$ dB. Moreover, the BER performances of these codes after puncturing are simulated and compared. We summarize the contributions as follows

1) It is the first attempt to use protographs to design codes with performance close to the ultimate Shannon limit [64]. By appending additional degree-1 Hadamard VNs to the CNs of a protograph, the SPC check nodes are converted into more powerful Hadamard constraints, forming the generalized protograph of PLDPC-Hadamard codes. After using the copy-and-permute operations to lift the protograph, the matrix corresponding to the lifted graph is a structured QC matrix which is greatly beneficial to linear encoding, parallel decoding and hardware implementation.

2) To analyze the decoding threshold of a PLDPC-Hadamard code, we propose a modified PEXIT method. We replace the SPC mutual information (MI) updating with our proposed Hadamard MI updating based on Monte Carlo simulations. Different from the EXIT method used in optimizing the degree distribution of VNs in an LDPC-Hadamard code [1], our proposed PEXIT method searches and analyzes protomatrices corresponding to the generalized protograph of the PLDPC-Hadamard codes. The proposed method, moreover, is applicable to analyzing PLDPC-Hadamard codes with degree-1 VNs and/or punctured VNs. Using the analytical technique, we have found PLDPC-Hadamard codes with very low decoding thresholds (< -1.40 dB) under different code rates.

3) Extensive simulations are performed under an AWGN channel. For each case, 100 frame errors are collected before the simulation is terminated. Results show that the PLDPC-Hadamard codes can obtain comparable BER performance to the traditional LDPC-Hadamard codes [1]. At a BER of 10^{-5}, the gaps to the ultimate Shannon limit [64] are 0.40 dB for the rate-0.0494 code, 0.35 dB for the rate-0.021 code, 0.24 dB for the rate-0.008 code and 0.16 dB for the rate-0.003 code, respectively.

4) Punctured PLDPC-Hadamard codes are studied. Puncturing different VNs in the protograph of a PLDPC-Hadamard code sometimes can produce different BER/FER performance improvement/degradation compared with the unpunctured code. Moreover, when the order of the Hadamard code $r = 5$, puncturing the extra degree-1 Hadamard VNs provided by the non-systematic Hadamard encoding is found to degrade the error performance.

Secondly, we propose a layered decoding algorithm for PLDPCH-BCs with an aim of improving the convergence rate. Based on the layered algorithm, we propose a hardware architecture for PLDPC-Hadamard layered decoders. We summarize the contributions of this part as follows.

1) Compared with the standard decoding algorithm, the layered decoding algorithm improves the convergence rate by about two times. At a bit error rate of 2.0×10^{-5}, the layered decoder using 20 decoding iterations shows a very small degradation of 0.03 dB compared with the standard decoder using 40 decoding iterations. Moreover, the layered decoder using 21 decoding iterations shows the same error performance as the standard decoder using 41 decoding iterations.

2) For the implementation of PLDPC-Hadamard layered decoders, it consists mainly of control logics, random address memories, and Hadamard sub-decoders. Two slightly different pipelined structures are designed to cater for different numbers of Hadamard sub-decoders running in parallel. The latency and throughput of these two different structures are derived. Implementation of the decoder design on an FPGA board shows that a throughput of 1.48 Gbps is achieved with a bit error rate (BER) of 10^{-5} at around $E_b/N_0 = -0.40$ dB. The decoder can also achieve the same BER at $E_b/N_0 = -1.11$ dB with a reduced throughput of 0.20 Gbps.

Thirdly, we make use of PLDPC-Hadamard block codes to design spatially coupled PLDPC-Hadamard convolutional codes (SC-PLDPCH-CCs). We summarize the contributions of this part as follows.

1) We propose spatially coupled PLDPC-Hadamard codes, which are constructed by spatially coupling PLDPC-Hadamard block codes.

2) We describe the encoding method of SC-PLDPCH-CCs and propose a pipeline decoding structure to decode SC-PLDPCH-CCs.

3) We make use of optimized PLDPCH-BCs to design good SC-PLDPCH-CCs with different rates. Simulation results show that SC-PLDPCH-CCs outperform their PLDPC-Hadamard block code counterparts in terms of bit error performance. Moreover, we find that error floors appear in PLDPC-Hadamard block codes but not in SC-PLDPCH-CCs. For the rate-0.00295 SC-PLDPCH-CC, a BER of 2×10^{-7} is achieved at $E_b/N_0 = -1.45$ dB.

1.3 Thesis Organization

The organization of the thesis is as follows.

Chapter 2 briefly describes some important channel codes such as LDPC block codes, protograph-based LDPC block codes, spatially coupled LDPC codes, Hadamard codes, and LDPC-Hadamard codes. Moreover, the basic analysis, encoding and decoding methods are presented.

Chapter 3 introduces our proposed PLDPC-Hadamard block code, including its structure, encoding and decoding methods, and code rate. In particular, the cases in which the order of the Hadamard code used is even or odd are described and analyzed. A low-complexity PEXIT method for analyzing PLDPC-Hadamard codes is proposed and an optimization algorithm is provided. Moreover, this chapter presents the protomatrices of the PLDPC-Hadamard codes found by the proposed algorithms, their decoding thresholds and simulated error results. The error performance of these codes after puncturing are further evaluated.

Chapter 4 presents the standard decoding algorithm and our proposed layered decoding for PLDPCH-BCs. This chapter then presents the BER results for the standard and layered decoders. Based on the layered algorithm, this chapter proposes a hardware architecture of PLDPCH-BC layered decoders, derives the latency and throughput, and report the implementation results.

Chapter 5 introduces the structure and encoding process of SC-PLDPCH-CCs. It proposes a pipelined strategy combined with layered scheduling for decoding SC-PLDPCH-CCs. Using the proposed pipeline decoding, error performance of SC-PLDPCH-CCs with different rates and different number of processors is evaluated. This chapter also compares the simulated BER results of the SC-PLDPCH-CCs with those of the underlying PLDPCH-BCs.

Chapter 6 concludes this thesis and suggests some possible future works.

Part II

LITERATURE REVIEW

Chapter 2

BACKGROUND

In this chapter, we review some channel codes that are related to this thesis.

2.1 Low-density parity-check codes

This section briefly introduces traditional LDPC block codes, protograph-based LDPC (PLDPC) block codes and spatially coupled LDPC codes. We also review their analysis method, encoding and decoding.

2.1.1 *Traditional LDPC Block Codes*

An LDPC code with code length N, information length $k = N - M$ and code rate $R = k/N$ can be represented by a $M \times N$ parity-check matrix $\boldsymbol{H}_{M\times N}$ whose entries only include 0 or 1. Moreover, the matrix $\boldsymbol{H}_{M\times N}$ needs to satisfy the following conditions:

1. The number of "1"s in the matrix should be much less than the number of elements MN, i.e., a low density of "1"s.

2. The codeword bits corresponding to the "1"s in each row of the matrix must take part in the same parity-check equation, i.e., each LDPC codeword $\boldsymbol{c}$ satisfies $\boldsymbol{c}\boldsymbol{H}_{M\times N}^{T} = \mathbf{0}$, where $\mathbf{0}$ represents a zero vector of appropriate length.

The matrix $\boldsymbol{H}_{M\times N}$ can also be represented by a Tanner graph, as shown in Fig. 1. The circles denote the variable nodes (VNs) corresponding to the columns of the matrix; the squares denote the check nodes (CNs) corresponding to the rows of the matrix; and the edges connecting the VNs and CNs correspond to the "1"s in the matrix. Moreover, the number of edges connecting each VN/CN is called the degree of the VN/CN and corresponds to the column/row weight. Denote $\boldsymbol{\lambda} = \{\lambda_j\}$ and $\boldsymbol{\rho} = \{\rho_i\}$ as the fraction of degree-d_j VNs and the fraction of degree-d_i CNs, respectively. If $\boldsymbol{\lambda} = \{1\}$ and $\boldsymbol{\rho} = \{1\}$, we call such code as a regular LDPC code; otherwise, it is called an irregular LDPC code. The degree distribution $(\boldsymbol{\lambda}, \boldsymbol{\rho})$ not only determines the "1"s distribution in $\boldsymbol{H}_{M\times N}$, but also can be used by the extrinsic information transfer (EXIT) chart technique to estimate the theoretical threshold of the LDPC code [23], [24].

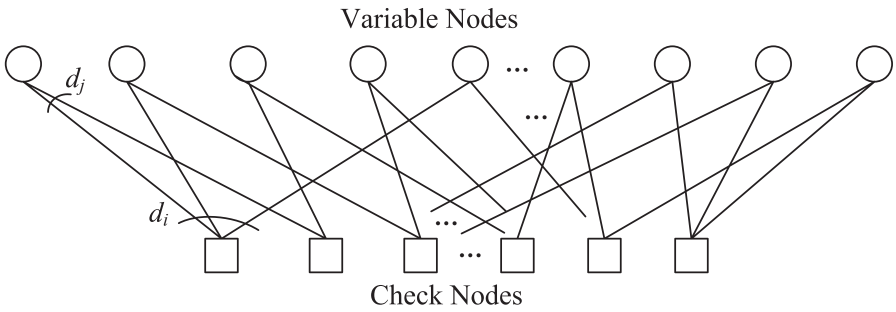

Figure 1: Representation of an LDPC code by a Tanner graph.

To illustrate the EXIT method, different types of mutual information (MI) are defined as follows:

- I_{av}: *a priori* MI value of VNs;
- I_{ac}: *a priori* MI value of CNs;
- I_{ev}: extrinsic MI value of VNs;
- I_{ec}: extrinsic MI value of CNs;
- I_{ch}: MI value from the channel.

We summarize the method as follows

1. Select a relatively large E_b/N_0.
2. Set all MI values to 0.
3. Initialize I_{ch} based on E_b/N_0 and the code rate.
4. Compute I_{ev} based on I_{av}, I_{ch} and $\boldsymbol{\lambda}$.
5. Set $I_{ac} = I_{ev}$.
6. Compute I_{ec} based on I_{ac} and $\boldsymbol{\rho}$.
7. Set $I_{av} = I_{ec}$.
8. Repeat Steps 4) to 7) I_{iter} times.
9. Plot the two EXIT curves (I_{av}, I_{ev}) and (I_{ec}, I_{ac}). If the two curves in the EXIT chart only intersects at the point $(1, 1)$, reduce E_b/N_0 and go to Step 2); otherwise set the previous E_b/N_0 when the two curves intersect only at the point $(1, 1)$ as the threshold $(E_b/N_0)_{th}$ and stop.

In the EXIT chart method and based on degree distribution $(\boldsymbol{\lambda}, \boldsymbol{\rho})$, we compute

$$I_{ac} = I_{ev} = \sum_{j=2}^{d_v} \lambda_j \cdot J\left(\sqrt{(d_j - 1)\left(J^{-1}\left(I_{av}\right)\right)^2 + \left(J^{-1}\left(I_{ch}\right)\right)^2}\right) \quad (1)$$

and

$$I_{av} = I_{ec} = 1 - \sum_{i=2}^{d_c} \rho_i \cdot J\left(\sqrt{(d_i - 1)\left(J^{-1}\left(1 - I_{ac}\right)\right)^2}\right), \tag{2}$$

where d_v and d_c are denoted as the maximum degrees of VNs and CNs, respectively. Moreover, the functions $I = J(\sigma)$ and $\sigma = J^{-1}(I)$ are given by [17, 23]

$$J(\sigma) = \begin{cases} a_1\sigma^3 + b_1\sigma^2 + c_1\sigma, & 0 \leqslant \sigma \leqslant 1.6363 \\ 1 - e^{(a_2\sigma^3 + b_2\sigma^2 + c_2\sigma + d_2)}, & 1.6363 < \sigma < 10 \\ 1, & 10 \leqslant \sigma \end{cases} \tag{3}$$

and

$$J^{-1}(I) = \begin{cases} a_1' I^2 + b_1' I + c_1' \sqrt{I}, & 0 \leqslant I \leqslant 0.3646 \\ -a_2' \ln[b_2'(1 - I)] - c_2' I, & 0.3646 < I < 1 \end{cases} \tag{4}$$

where

- $a_1 = -0.0421061$, $a_2 = 0.00181491$, $b_1 = 0.209252$, $b_2 = -0.142675$, $c_1 = -0.00640081$, $c_2 = -0.0822054$, $d_2 = 0.0549608$; and
- $a_1' = 1.09542$, $a_2' = 0.706692$, $b_1' = 0.214217$, $b_2' = 0.386013$, $c_1' = 2.33727$ and $c_2' = -1.75017$.

We use the EXIT method to analyze the (3,6) regular LDPC code and obtain a threshold of 1.127 dB. The matched (not crossed) EXIT curves from VNs and CNs are plotted in Fig. 2 when $E_b/N_0 = 1.127$ dB.

2.1.2 *Protograph-based LDPC Block Codes*

When an LDPC code contains degree-1 VNs or punctured VNs, the traditional EXIT chart cannot evaluate its decoding performance. However, for LDPC codes constructed based on protographs, their theoretical performance can be estimated by the protograph EXIT (PEXIT) algorithm even if they contain degree-1 VNs or punctured VNs [37].

A protograph can be denoted by $G = (V, C, E)$ where V is a set of VNs, C is a set of CNs and E is a set of edges [36]. Fig. 3 illustrates a protograph, and the corresponding protomatrix (also called base matrix) is given by

$$\boldsymbol{B}_{m\times n} = \begin{bmatrix} 1 & 3 & \cdots & 0 & 1 \\ 2 & 1 & \cdots & 2 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 2 \end{bmatrix}. \tag{5}$$

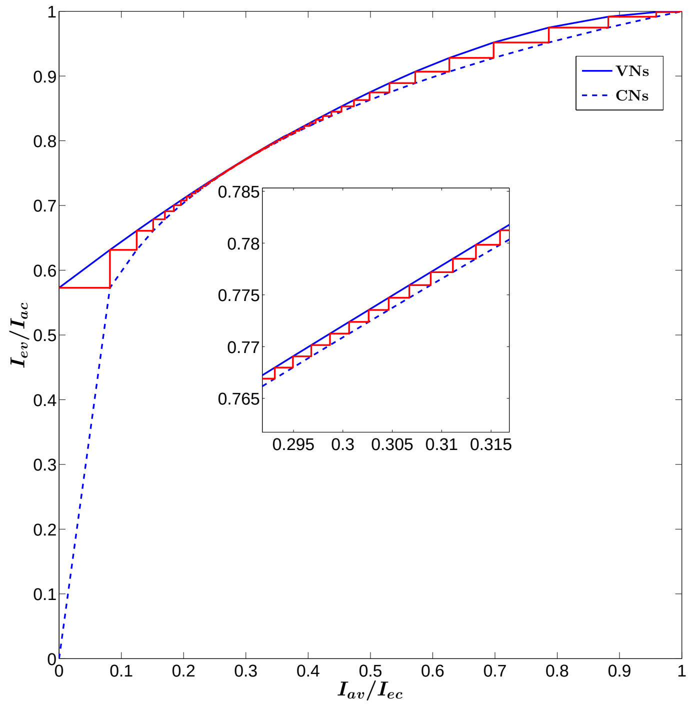

Figure 2: The EXIT curves for (3,6) regular LDPC code at the threshold of 1.127 dB.

The entries in $\boldsymbol{B}_{m\times n} = \{b_{i,j} : i = 0, 1, 2 \ldots, m-1; j = 0, 1, 2 \ldots, n-1\}$ are allowed to be larger than 1 and they correspond to the multiple edges connecting the same pair of VN and CN in the protograph. The parity-check matrix $\boldsymbol{H}_{M\times N}$ of a protograph-based LDPC (PLDPC) code can be constructed by expanding the protomatrix $\boldsymbol{B}_{m\times n}$ where $m \ll M$ and $n \ll N$.

To obtain a larger $\boldsymbol{H}_{M\times N}$, the following copy-and-permute operations can be used to expand $\boldsymbol{B}_{m\times n}$.

1. Duplicate the protograph z times.
2. Permute the edges which connect the same type of VNs and CNs among these duplicated protographs.

This expansion process is also called lifting and the parameter z is called the lifting factor. The equivalent process in the "matrix domain" is to replace each $b_{i,j}$ by

- a $z \times z$ zero matrix if $b_{i,j} = 0$; or
- a summation of $b_{i,j}$ non-overlapping $z \times z$ permutation matrices if $b_{i,j} \neq 0$.

As mentioned, permutations occur only among the edges connecting to the same type of nodes and the lifted matrix $\boldsymbol{H}_{M\times N}$ (where $M =$

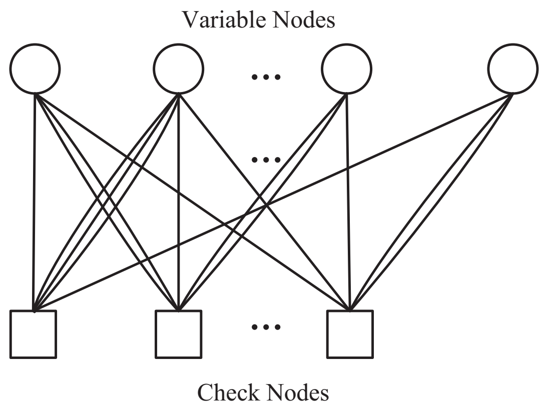

Figure 3: A protograph corresponding to the protomatrix in (5).

zm and $N = zn$) keeps the same degree distribution and code rate as $\boldsymbol{B}_{m\times n}$. The code represented by $\boldsymbol{H}_{M\times N}$ is called a PLDPC code.

To make the lifted matrix having quasi-cyclic (QC) structure, this thesis uses a two-step lifting method [77]. In the first step, we "lift" a base matrix $\{b(i,j)\}$ by replacing each non-zero entry $b(i,j)$ with a summation of $b(i,j)$ different $z_1 \times z_1$ permutation matrices and replacing each zero entry with the $z_1 \times z_1$ zero matrix. After the first lifting process, all entries in the lifted matrix are either "0" or "1". In the second step, we lift the resultant matrix again by replacing each entry "1" with a $z_2 \times z_2$ circulant permutation matrix (CPM), and replacing each entry "0" with the $z_2 \times z_2$ zero matrix. As can be seen, the final connection matrix can be easily represented by a series of CPMs. Note that in each lifting step, the permutation matrices and CPMs are selected using the progressive-edge-growth (PEG) algorithm [28] such that the girth (shortest cycle) in the resultant matrix can be maximized.

To analyze the decoding performance of a PLDPC code, the PEXIT algorithm is applied to $\boldsymbol{B}_{m\times n}$. In the PEXIT method, the MI values on all types of edges are updated separately and iteratively [37].

To illustrate the method, different types of MI are first defined as follows:

- $I_{ac}(i,j)$: *a priori* MI from *j*-th VN to *i*-th CN in $\mathbf{B}_{m\times n}$;
- $I_{av}(i,j)$: *a priori* MI from *i*-th CN to *j*-th VN in $\mathbf{B}_{m\times n}$;
- $I_{ev}(i,j)$: extrinsic MI from *j*-th VN to *i*-th CN in $\mathbf{B}_{m\times n}$;
- $I_{ec}(i,j)$: extrinsic MI from *i*-th CN to *j*-th VN in $\mathbf{B}_{m\times n}$;
- $I_{app}(j)$: *a posteriori* MI value of the *j*-th VN;
- I_{ch}: MI from the channel.

Without going into the details, the steps below show how to determine the threshold $(E_b/N_0)_{th}$.

1. Select a relatively large E_b/N_0.
2. Set all MI values to 0.
3. Initialize I_{ch} based on E_b/N_0 and the code rate.
4. Compute $I_{ev}(i,j)$ and set $I_{ac}(i,j) = I_{ev}(i,j)\ \forall i,j$.
5. Compute $I_{ec}(i,j)$ and set $I_{av}(i,j) = I_{ec}(i,j)\ \forall i,j$.
6. Repeat Steps 4) to 5) I_{iter} times.
7. Compute $I_{app}(j)$.
8. If $I_{app}(j) = 1\ \forall j$, reduce E_b/N_0 and go to Step 2); otherwise set the previous E_b/N_0 that achieves $I_{app}(j) = 1\ \forall j$ as the threshold $(E_b/N_0)_{th}$ and stop.

In the PEXIT method, for $b_{i,j} > 0$

$$I_{ac}(i,j) = I_{ev}(i,j)$$
$$= J\left(\sqrt{\sum_{s\neq i} b_{s,j}\left(J^{-1}\left(I_{av}(s,j)\right)\right)^2 + \left(b_{i,j}-1\right)\cdot\left(J^{-1}\left(I_{av}(i,j)\right)\right)^2 + \left(J^{-1}\left(I_{ch}\right)\right)^2}\right)$$
$$\forall\ i,j; \tag{6}$$

$$I_{av}(i,j) = I_{ec}(i,j)$$
$$= 1 - J\left(\sqrt{\sum_{s\neq j} b_{i,s}\left(J^{-1}\left(1 - I_{ac}(i,s)\right)\right)^2 + \left(b_{i,j}-1\right)\cdot\left(J^{-1}\left(1 - I_{ac}(i,j)\right)\right)^2}\right)$$
$$\forall\ i,j; \tag{7}$$

and

$$I_{app}(j) = J\left(\sqrt{\sum_{i} b_{i,j}\left(J^{-1}\left(I_{av}(i,j)\right)\right)^2 + \left(J^{-1}\left(I_{ch}\right)\right)^2}\right) \quad \forall\ j. \tag{8}$$

The above analytical process can be regarded as the repeated computation and exchange between the *a priori* MI matrices $\{I_{av}(i,j)\}/\{I_{ac}(i,j)\}$ and extrinsic MI matrices $\{I_{ev}(i,j)\}/\{I_{ec}(i,j)\}$. Moreover, these matrices have the same size as $\boldsymbol{B}_{m\times n}$. Note that the PEXIT algorithm can be used to analyze protographs with degree-1 VNs, i.e., columns in the protomatrix with weight 1. Protographs with punctured VNs will also be analyzed in a similar way, except that the code rate will be changed accordingly and the corresponding I_{ch} will be initialized as 0.

2.1.3 *Spatially Coupled LDPC Codes*

2.1.3.1 *LDPC convolutional codes*

Given an LDPC block code, an LDPC convolutional code can be constructed by introducing memory in the code design and allowing multiple consecutive block codes to become related [41]. The parity-check matrix $\boldsymbol{H}_{\text{CC}}$ of an LDPC convolutional code is semi-infinite and structurally repeated, and can be written as

$$\boldsymbol{H}_{\text{CC}} = \begin{bmatrix} \boldsymbol{H}_0(1) & & & & \\ \boldsymbol{H}_1(1) & \boldsymbol{H}_0(2) & & & \\ \vdots & \boldsymbol{H}_1(2) & \ddots & & \\ \boldsymbol{H}_{m_s}(1) & \vdots & \ddots & \boldsymbol{H}_0(t) & \\ & \boldsymbol{H}_{m_s}(2) & \ddots & \boldsymbol{H}_1(t) & \ddots \\ & & \ddots & \vdots & \ddots \\ & & & \boldsymbol{H}_{m_s}(t) & \ddots \\ & & & & \ddots \end{bmatrix},$$

where each $\boldsymbol{H}_i(t)$ $(i = 0, 1, \ldots, m_s)$ is a $M \times N$ component matrix, t denotes the time index, and m_s is the syndrome former memory. Each codeword $\boldsymbol{c}$ should satisfy $\boldsymbol{c}\boldsymbol{H}_{\text{CC}}{}^{\mathsf{T}} = \mathbf{0}$, where $\mathbf{0}$ is the semi-infinite zero vector.

2.1.3.2 *Spatially coupled PLDPC codes*

Spatially coupled PLDPC codes are constructed based on underlying PLDPC block codes. We denote W as the coupling width (equivalent to the aforementioned syndrome former memory m_s) and L as the coupling length. Based on the $m \times n$ protomatrix $\boldsymbol{B}$ of an underlying PLDPC code, an edge spreading procedure can be first used to obtain $W + 1$ split protomatrices $\boldsymbol{B}_i$ $(i = 0, 1, \ldots, W)$ under the constraint $\boldsymbol{B} = \sum_{i=0}^{W} \boldsymbol{B}_i$. Then L sets of such protomatrices are coupled to construct a spatially coupled PLDPC (SC-PLDPC) code [53, 78]. Depending on how the coupling ends, three types of SC-PLDPC codes, namely SC-PLDPC terminated code (SC-PLDPC-TDC), SC-PLDPC tail-biting code (SC-PLDPC-TBC) and SC-PLDPC convolutional codes (SC-PLDPC-CC), are formed.

When the L sets of protomatrices are coupled and then directly terminated, the resultant protomatrix is given by

$$\boldsymbol{B}_{\mathrm{SC-PLDPC-TDC}} = \left.\overbrace{\begin{bmatrix} \boldsymbol{B}_0 & & & \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & & \\ \vdots & \boldsymbol{B}_1 & \ddots & \\ \boldsymbol{B}_W & \vdots & \ddots & \boldsymbol{B}_0 \\ & \boldsymbol{B}_W & \ddots & \boldsymbol{B}_1 \\ & & \ddots & \vdots \\ & & & \boldsymbol{B}_W \end{bmatrix}}^{nL}\right\} m(L+W). \tag{9}$$

Such code is called a SC-PLDPC-TDC. The code rate equals

$$\begin{aligned} R_{\mathrm{SC-PLDPC-TDC}} &= \frac{nL - m(L+W)}{nL} \\ &= 1 - \frac{L+W}{L}\left(1 - R_{\mathrm{PLDPC-BC}}\right), \end{aligned} \tag{10}$$

where $R_{\mathrm{PLDPC-BC}} = 1 - \frac{m}{n}$ is the code rate of its underlying block code.

Example: We make use of the protomatrix (11) to construct the protomatrix of a SC-PLDPC-TDC.

$$\boldsymbol{B} = \begin{bmatrix} 2 & 0 & 2 & 2 \\ 0 & 2 & 2 & 2 \\ 3 & 2 & 0 & 1 \end{bmatrix}. \tag{11}$$

We assume a coupling width $W = 1$. Hence we split $\boldsymbol{B}$ into $\boldsymbol{B}_0$ and $\boldsymbol{B}_1$ under the constraint $\boldsymbol{B} = \boldsymbol{B}_0 + \boldsymbol{B}_1$, and obtain

$$\boldsymbol{B}_0 = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix} \tag{12}$$

and

$$\boldsymbol{B}_1 = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 1 & 1 \\ 2 & 0 & 0 & 0 \end{bmatrix}. \tag{13}$$

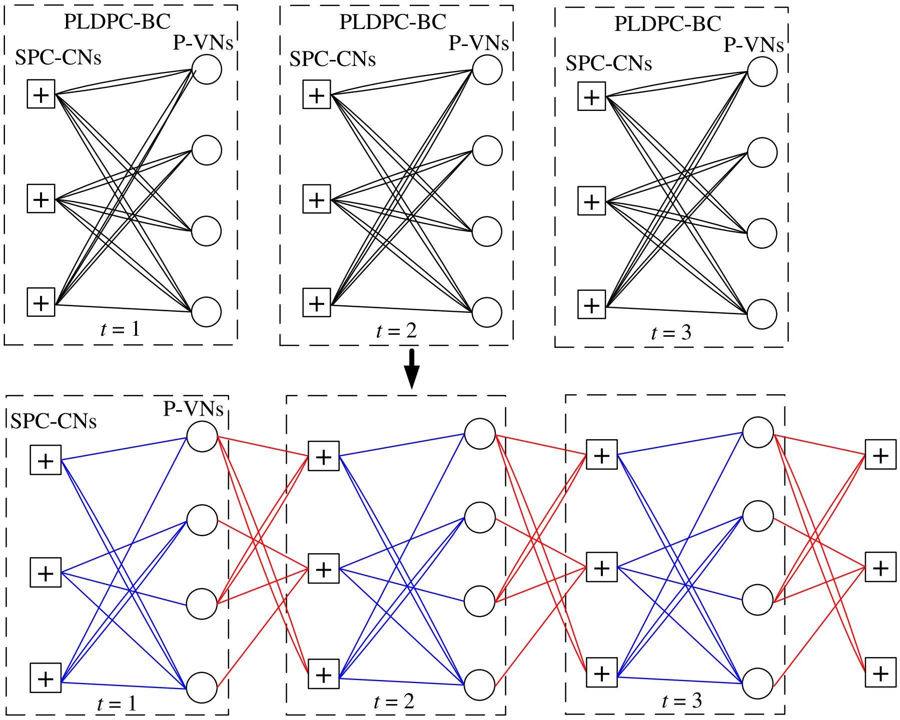

Figure 4: Constructing the protograph of a SC-PLDPC-TDC from the protographs of a PLDPC block code. $W = 1$ and $L = 3$.

Assuming a coupling length $L = 3$, we can construct the protomatrix of a SC-PLDPC-TDC as

$$\boldsymbol{B}_{\text{TDC},W=1,L=3} = \begin{bmatrix} \boldsymbol{B}_0 & & \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & \\ & \boldsymbol{B}_1 & \boldsymbol{B}_0 \\ & & \boldsymbol{B}_1 \end{bmatrix}. \tag{14}$$

The protograph of the above SC-PLDPC-TDC is shown in Fig. 4, which is formed by coupling $L = 3$ PLDPC-BC protographs. The blue edges (connecting P-VNs and SPC-CNs) correspond to $\boldsymbol{B}_0$ (12) while the red ones correspond to $\boldsymbol{B}_1$ (13). According to edge spreading operations, the edges from the P-VNs at time t will be spread to connect the SPC-CNs at time $t+1, t+2, \ldots, t+W$ in addition to the SPC-CNs at time t. As $W = 1$ in Fig. 4, the P-VNs at time $t = 1$ connect SPC-CNs at time $t = 1$ and $t = 2$; and P-VNs at time $t = 2$ connect SPC-CNs at time $t = 2$ and $t = 3$.

In Fig. 5, we illustrate another example where $W = 2$ and $L = 4$. The $\boldsymbol{B}$ in (12) is split into

$$\boldsymbol{B}_0 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \tag{15}$$

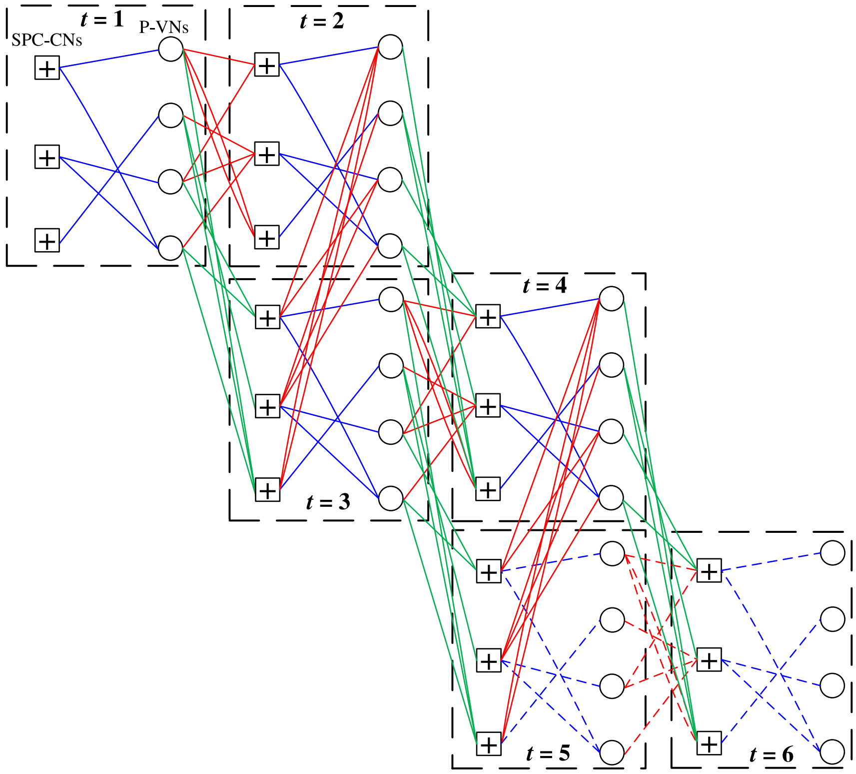

Figure 5: The protograph of a SC-PLDPC-TDC with $W = 2$ and $L = 4$. The P-VNs corresponding to $t = 5$ and $t = 6$ and their associated connections do not exist.

$$\boldsymbol{B}_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 2 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

$$\boldsymbol{B}_2 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}. \tag{17}$$

under the constraint $\boldsymbol{B} = \boldsymbol{B}_0 + \boldsymbol{B}_1 + \boldsymbol{B}_2$. With $L = 4$, the protomatrix of the SC-PLDPC-TDC is given by

$$\boldsymbol{B}_{\text{TDC},W=2,L=4} = \begin{bmatrix} \boldsymbol{B}_0 & & & \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & & \\ \boldsymbol{B}_2 & \boldsymbol{B}_1 & \boldsymbol{B}_0 & \\ & \boldsymbol{B}_2 & \boldsymbol{B}_1 & \boldsymbol{B}_0 \\ & & \boldsymbol{B}_2 & \boldsymbol{B}_1 \\ & & & \boldsymbol{B}_2 \end{bmatrix}. \tag{18}$$

In Fig. 5, the blue edges correspond to $\boldsymbol{B}_0$ (15), the red ones correspond to $\boldsymbol{B}_1$ (16), and the green ones correspond to $\boldsymbol{B}_2$ (17).

Note that the "connections" represented by the dashed (red and blue) lines do not exist.

When the protograph of a spatially coupled code is terminated with "end-to-end" connections, the corresponding code is called SC-PLDPC-TBC, whose protomatrix can be written as

$$\boldsymbol{B}_{\mathrm{SC-PLDPC-TBC}} = \left.\overbrace{\begin{bmatrix} \boldsymbol{B}_0 & & & & & \boldsymbol{B}_W & \cdots & \boldsymbol{B}_1 \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & & & & & \ddots & \vdots \\ \vdots & \boldsymbol{B}_1 & \boldsymbol{B}_0 & & & & & \boldsymbol{B}_W \\ \boldsymbol{B}_W & \vdots & \boldsymbol{B}_1 & \ddots & & & & \\ & \boldsymbol{B}_W & \vdots & \ddots & \boldsymbol{B}_0 & & & \\ & & \boldsymbol{B}_W & \ddots & \boldsymbol{B}_1 & \boldsymbol{B}_0 & & \\ & & & \ddots & \vdots & \ddots & \boldsymbol{B}_0 & \\ & & & & \boldsymbol{B}_W & \cdots & \boldsymbol{B}_1 & \boldsymbol{B}_0 \end{bmatrix}}^{nL}\right\} mL \tag{19}$$

The code rate $R_{\mathrm{SC-PLDPC-TBC}}$ of a SC-PLDPC-TBC is the same as that of its underlying block code, i.e.,

$$R_{\mathrm{SC-PLDPC-TBC}} = \frac{nL - mL}{nL} = R_{\mathrm{PLDPC-BC}}. \tag{20}$$

Using the example shown in Fig. 4 and allowing the spatially coupled protographs connected end-to-end, we obtain the spatially coupled tail-biting protograph shown in Fig. 6. The protomatrix of the SC-PLDPC-TBC is given by

$$\boldsymbol{B}_{\mathrm{TBC},W=1,L=3} = \begin{bmatrix} \boldsymbol{B}_0 & & \boldsymbol{B}_1 \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & \\ & \boldsymbol{B}_1 & \boldsymbol{B}_0 \end{bmatrix}. \tag{21}$$

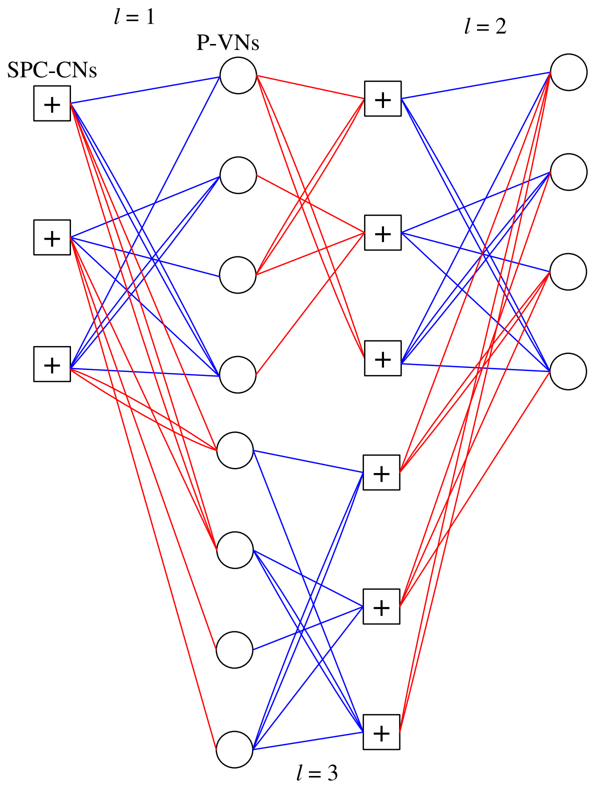

Figure 6: The protograph of a SC-PLDPC-TBC derived from Fig. 4. $W = 1$ and $L = 3$.

By extending the coupling length L of a SC-PLDPC-TDC to infinite, a SC-PLDPC-CC is formed. The semi-infinite protomatrix of a SC-PLDPC-CC is given by

$$\boldsymbol{B}_{\mathrm{SC-PLDPC-CC}} = \begin{bmatrix} \boldsymbol{B}_0 & & & & \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & & & \\ \vdots & \boldsymbol{B}_1 & \ddots & & \\ \boldsymbol{B}_W & \vdots & \ddots & \boldsymbol{B}_0 & \\ & \boldsymbol{B}_W & \ddots & \boldsymbol{B}_1 & \ddots \\ & & \ddots & \vdots & \ddots \\ & & & \boldsymbol{B}_W & \ddots \\ & & & & \ddots \end{bmatrix}. \quad (22)$$

The code rate of SC-PLDPC-CC equals that of the underlying LDPC block code [53], i.e.,

$$\begin{aligned} R_{\mathrm{SC-PLDPC-CC}} &= \lim_{L\to\infty} R_{\mathrm{SC-PLDPC-TDC}} \\ &= \lim_{L\to\infty} 1 - \frac{L+W}{L}\left(1 - R_{\mathrm{PLDPC-BC}}\right) \\ &= R_{\mathrm{PLDPC-BC}} \end{aligned} \quad (23)$$

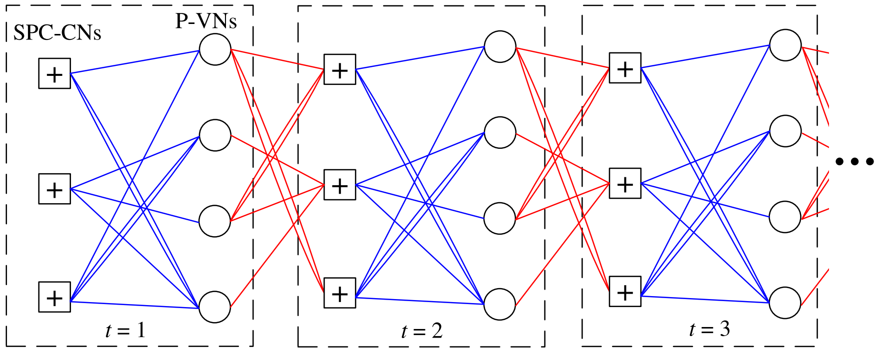

Figure 7: The protograph of a SC-PLDPC-CC derived from Fig. 4. $W = 1$ and $L = \infty$.

Fig. 7 depicts part of the spatially coupled convolutional protograph when the coupling length L of the SC-PLDPC-TDC shown in Fig. 4 is extended to infinity.

Once the protomatrix of a spatially coupled code is derived, the two-step lifting can be used to construct the SC-PLDPC code (SC-PLDPC-TDC, SC-PLDPC-TBC or SC-PLDPC-CC).

2.1.4 *Encoding*

Given a $M \times N$ parity-check matrix $\boldsymbol{H}_{M \times N}$, any length-N LDPC codeword $\boldsymbol{c}$ needs to satisfy $\boldsymbol{c}\boldsymbol{H}^T_{M \times N} = \mathbf{0}$. One direct encoding method for LDPC codes is to use Gaussian elimination to convert $\boldsymbol{H}_{M \times N}$ into the lower triangular matrix $\boldsymbol{H}'_{M \times N}$ shown in Fig. 8. The entries on the red diagonal are all 1's, the entries in the white part are all 0's, and the entries in the gray part can be ether 0 or 1. We can use $\boldsymbol{H}'_{M \times N}$ to systematically encode the length-$(N - M)$ information sequence $\boldsymbol{u} = [u_0\ u_1\ \ldots\ u_{N-M-1}]$ into length-N codeword $\boldsymbol{c} = [\boldsymbol{u}\ \boldsymbol{p}]$, where $\boldsymbol{p} = [p_0\ p_1\ \ldots\ p_{M-1}]$ denotes the M parity-check bits. Assuming that $\boldsymbol{H}'_{M \times N} = \{h_{i,j}\}$, $\boldsymbol{p}$ can be obtained by backward recursion [79], i.e.,

$$p_i = \sum_{j=0}^{N-M-1} u_j h_{i,j} + \sum_{j=0}^{i-1} p_j h_{i,j+N-M};\ i = 0, 1, \ldots, M - 1. \tag{24}$$

We also can encode LDPC codes using parity-check matrices with an approximate lower triangular form [80]. For encoding of PLDPC or SC-LDPC codes, we can use Gaussian elimination to adjust their lifted matrices or directly design their protomatrices with lower triangular structures.

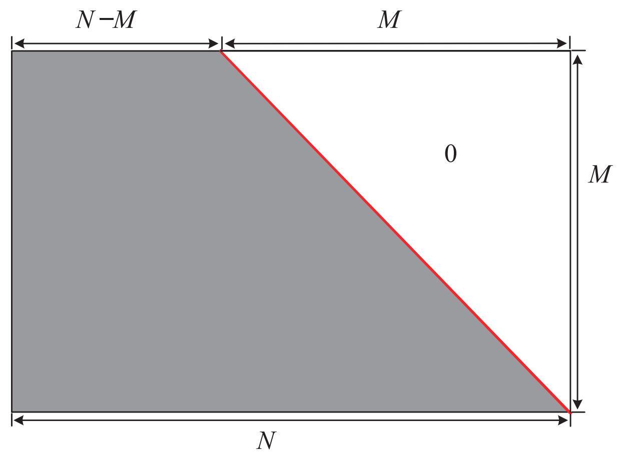

Figure 8: Use Gaussian elimination way to generate $\boldsymbol{H}'_{M\times N}$ for encoding.

2.1.5 *Decoding*

Once receiving the channel observations, which is denoted as a length-N vector $\boldsymbol{y} = [y_0 \ y_1 \ \dots \ y_{N-1}]$, we can use the classic belief propagation (BP) algorithm to decode LDPC codes. When the BP algorithm is calculated in the logarithmic domain, its operations only involve addition and multiplication, thus it is also called the sum-product algorithm (SPA) [21]. We denote

- R_α as the set of VNs connected to the α-th CN ($\alpha = 1, 2, \dots, M$);
- $R_{\alpha\setminus\beta}$ as the set of VNs connected to the α-th CN excluding the β-th VN ($\alpha = 1, 2, \dots, M$);
- C_β as the set of CNs connected to the β-th VN ($\beta = 1, 2, \dots, N$);
- $C_{\beta\setminus\alpha}$ as the set of CNs connected to the β-th VN excluding the α-th CN ($\beta = 1, 2, \dots, N$);
- $L^{\mathrm{VN}}_{\mathrm{ch}}(\beta)$ as the channel LLR value of the β-th VN ($\beta = 1, 2, \dots, N$);
- $L^{\mathrm{VN}}_{\mathrm{app}}(\beta)$ as the *a posteriori* (APP) LLR value of the β-th VN ($\beta = 1, 2, \dots, N$);
- $L^{\mathrm{VN}}_{\mathrm{ex}}(\alpha, \beta)$ as the extrinsic LLR sent from the β-th VN to the α-th CN ($\alpha = 1, 2, \dots, M$; $\beta = 1, 2, \dots, N$);
- $L^{\mathrm{CN}}_{\mathrm{ex}}(\alpha, \beta)$ as the extrinsic LLR sent from the α-th CN to the β-th VN ($\alpha = 1, 2, \dots, M$; $\beta = 1, 2, \dots, N$).

Assuming that the variance σ^2 of the zero-mean AWGN channel is known at the receiving end, the SPA is described as follows

1. Initialization: Set $L_{ex}^{VN}(\alpha, \beta) = L_{ch}^{VN}(\beta) = 2y_\beta/\sigma^2$, $\forall \alpha = 1, 2, \ldots,$ M and $\beta = 1, 2, \ldots, N$.

2. CN processor: For $\alpha = 1, 2, \ldots, M$, compute

$$L_{ex}^{CN}(\alpha, \beta) = 2\tanh^{-1}\left(\prod_{\beta' \in R_{\alpha \backslash \beta}} \tanh\left(L_{ex}^{VN}(\alpha, \beta')/2\right)\right) \ \forall\, \beta \in R_\alpha.$$

3. VN processor: For β-VN ($\beta = 1, 2, \ldots, N$), compute

$$L_{ex}^{VN}(\alpha, \beta) = L_{ch}^{VN}(\beta) + \sum_{\alpha' \in C_{\beta \backslash \alpha}} L_{ex}^{CN}(\alpha', \beta) \ \forall\, \alpha \in C_\beta.$$

4. Repeat Step 2 to 3 I times and make decisions based on the sign of $L_{app}^{PVN}(\beta)$ ($\beta = 1, 2, \ldots, N$), where

$$L_{app}^{VN}(\beta) = L_{ch}^{VN}(\beta) + \sum_{\alpha' \in C_\beta} L_{ex}^{CN}(\alpha', \beta).$$

Based on the SPA decoding algorithm, the shuffled decoding algorithm [81, 82] and layered decoding algorithm [83, 84] have been proposed to speed up the convergence rate while maintaining the same computational complexity. The simplified decoding algorithms such as normalized or offset BP-based decoding [85–87] and bit-flipping decoding methods [88–91] have also been proposed. For decoding of (spatially coupled) LDPC convolutional codes, we can use a windowed decoding strategy or pipeline decoding arrangement [42, 43, 92] to perform the BP decoding.

2.2 Traditional LDPC-Hadamard Block Codes

2.2.1 *Hadamard Codes*

We first review the Hadamard codes and their decoding. A Hadamard code with an order r is a class of linear block codes. We consider a $q \times q$ positive Hadamard matrix $+\boldsymbol{H}_q = \{+\boldsymbol{h}_j, j = 0, 1, \ldots, q-1\}$, which can be constructed recursively using

$$+\boldsymbol{H}_q = \begin{bmatrix} +\boldsymbol{H}_{q/2} & +\boldsymbol{H}_{q/2} \\ +\boldsymbol{H}_{q/2} & -\boldsymbol{H}_{q/2} \end{bmatrix} \tag{25}$$

with $q = 2^r$ and $\pm\boldsymbol{H}_1 = [\pm 1]$. Each column $+\boldsymbol{h}_j$ is a Hadamard codeword and thus $\pm\boldsymbol{H}_q$ contains $2q = 2^{r+1}$ codewords $\pm\boldsymbol{h}_j$. Note that Hadamard codewords can also be represented by mapping $+1$ in $\pm\boldsymbol{h}_j$ to bit "0" and -1 to bit "1".

Considering an information sequence $\boldsymbol{u} \in \{0,1\}^{r+1}$ of length $r+1$ and denoted by $\boldsymbol{u} = [u_0 \ u_1 \ \ldots \ u_r]^T$, the Hadamard encoder encodes $\boldsymbol{u}$ into a codeword $\boldsymbol{c}^H$ of length q, i.e., $\boldsymbol{c}^H \in \{0,1\}^{2^r} = [c_0^H \ c_1^H \ \ldots \ c_{2^r-1}^H]^T$, where $(\cdot)^T$ represents the transpose operation. Assuming that the $+\boldsymbol{h}_j$ or $-\boldsymbol{h}_j$ corresponding to $\boldsymbol{c}$, i.e., by mapping bit "0" in $\boldsymbol{c}$ to $+1$ and bit "1" to -1, is uniformly transmitted through an AWGN channel with mean 0 and variance σ_{ch}^2, we denote the received signal by $\boldsymbol{y} = [y_0 \, y_1 \ \ldots \ y_{2^r-1}]^T$. Given y_i, the log-likelihood ratio (LLR) value $L(c_i^H \mid y_i)$ is computed by the ratio between the conditional probabilities $\Pr(c_i^H = \text{"0"} \mid y_i)$ and $\Pr(c_i^H = \text{"1"} \mid y_i)$, i.e.,

$$L(c_i^H \mid y_i) = \ln \frac{\Pr(c_i^H = \text{"0"} \mid y_i)}{\Pr(c_i^H = \text{"1"} \mid y_i)} \quad i = 0, 1, ..., 2^r - 1. \tag{26}$$

Applying the following Bayes' rule to (26)

$$\Pr(c_i^H \mid y_i) = \frac{p(y_i \mid c_i^H) \cdot \Pr(c_i^H)}{p(y_i)}, \tag{27}$$

we obtain

$$L(c_i^H \mid y_i) = \ln \frac{p(y_i \mid c_i^H = \text{"0"}) \cdot \Pr(c_i^H = \text{"0"})}{p(y_i \mid c_i^H = \text{"1"}) \cdot \Pr(c_i^H = \text{"1"})} \tag{28}$$

where $p(y_i \mid c_i^H = \text{"0"})$ and $p(y_i \mid c_i^H = \text{"1"})$ denote the channel output probability density function (PDF) conditioned on the code bit $c_i^H = \text{"0"}$ and $c_i^H = \text{"1"}$, respectively, being transmitted; $\Pr(c_i^H = \text{"0"})$ and $\Pr(c_i^H = \text{"1"})$ denote the *a priori* probabilities that $c_i^H = \text{"0"}$ and $c_i^H = \text{"1"}$ are transmitted, respectively; and $p(y_i)$ denotes the PDF of received signal y_i. We denote $L_{ch}^H(i)$ as the channel LLR value of the i-th bit, i.e.,

$$L_{ch}^H(i) = \ln \frac{p(y_i \mid c_i^H = \text{"0"})}{p(y_i \mid c_i^H = \text{"1"})} = \frac{2y_i}{\sigma_{ch}^2}, \quad i = 0, 1, ..., 2^r - 1; \tag{29}$$

and denote $L_{apr}^H(i)$ as the *a priori* LLR of the i-th bit, i.e.,

$$L_{apr}^H(i) = \ln \frac{\Pr(c_i^H = \text{"0"})}{\Pr(c_i^H = \text{"1"})}, \quad i = 0, 1, ..., 2^r - 1. \tag{30}$$

Thus, (28) is rewritten as

$$L(c_i^H \mid y_i) = L_{ch}^H(i) + L_{apr}^H(i), \quad i = 0, 1, ..., 2^r - 1. \tag{31}$$

We also define

$$\boldsymbol{L}_{ch}^H = [L_{ch}^H(0) \ L_{ch}^H(1) \ \cdots \ L_{ch}^H(2^r - 1)]^T; \tag{32}$$

$$\boldsymbol{L}_{apr}^H = [L_{apr}^H(0) \ L_{apr}^H(1) \ \cdots \ L_{apr}^H(2^r - 1)]^T. \tag{33}$$

In [69], a symbol-by-symbol maximum *a posteriori* probability (symbol-MAP) Hadamard decoder has been developed, in which the *a posteriori* LLR values of the code bits are computed based on the received vector $\boldsymbol{y}$. In the following, we show the steps to derive the *a posteriori* LLR values.

1. We re-write (26) for $i = 0, 1, \dots, 2^r - 1$ into

$$\Pr(c_i^H = \text{“}0\text{"} \mid y_i) = \frac{\exp(L(c_i^H \mid y_i)/2)}{\exp(L(c_i^H \mid y_i)/2) + \exp(-L(c_i^H \mid y_i)/2)} \tag{34}$$

and

$$\Pr(c_i^H = \text{“}1\text{"} \mid y_i) = \frac{\exp(-L(c_i^H \mid y_i)/2)}{\exp(L(c_i^H \mid y_i)/2) + \exp(-L(c_i^H \mid y_i)/2)}. \tag{35}$$

We also denote $\pm H[i, j]$ as the i-th bit in $\pm \boldsymbol{h}_j$. Given $\boldsymbol{y}$ and applying (34) and (35), the *a posteriori* probabilities of the transmitted Hadamard codeword $\boldsymbol{c}^H$ being $+\boldsymbol{h}_j$ or $-\boldsymbol{h}_j$ ($j = 0, 1, \cdots, 2^r - 1$) are given by

$$\begin{aligned}\Pr\left(\boldsymbol{c}^H = \pm\boldsymbol{h}_j \mid \boldsymbol{y}\right) &= \prod_i \Pr\left(c_i^H = \pm H[i, j] \mid y_i\right) \\ &= \prod_i \frac{\exp\left(\pm H[i, j] \cdot L(c_i^H \mid y_i)/2\right)}{\exp(L(c_i^H \mid y_i)/2) + \exp(-L(c_i^H \mid y_i)/2)} \\ &= \kappa \cdot \gamma\left(\pm \boldsymbol{h}_j\right) \end{aligned} \tag{36}$$

where

$$\kappa = \left[\prod_i \left[\exp\left(L(c_i^H \mid y_i)/2\right) + \exp\left(-L(c_i^H \mid y_i)/2\right)\right]\right]^{-1}$$

is independent of $\pm \boldsymbol{h}_j$;

$$\gamma\left(\pm \boldsymbol{h}_j\right) = \exp\left(\frac{1}{2}\left\langle \pm\boldsymbol{h}_j, \boldsymbol{L}_{\mathrm{ch}}^H + \boldsymbol{L}_{\mathrm{apr}}^H \right\rangle\right) \tag{37}$$

represents the *a posteriori* “information” of the codeword $\pm \boldsymbol{h}_j$; and $\langle \cdot \rangle$ denotes the inner-product operator.

2. Based on $\Pr\left(\boldsymbol{c}^H = \pm\boldsymbol{h}_j \mid \boldsymbol{y}\right)$, the *a posteriori* LLR of the i-th ($i = 0, 1, ..., 2^r - 1$) code bit, which is denoted by $L^H_{app}(i)$, is computed using

$$\begin{aligned} L^H_{app}(i) &= \ln \frac{\Pr(c^H_i = \text{"0"} \mid \boldsymbol{y})}{\Pr(c^H_i = \text{"1"} \mid \boldsymbol{y})} \\ &= \ln \frac{\sum\limits_{\pm H[i,j]=+1} \Pr(\boldsymbol{c}^H = \pm\boldsymbol{h}_j \mid \boldsymbol{y})}{\sum\limits_{\pm H[i,j]=-1} \Pr(\boldsymbol{c}^H = \pm\boldsymbol{h}_j \mid \boldsymbol{y})} \\ &= \ln \frac{\sum\limits_{\pm H[i,j]=+1} \gamma\left(\pm\boldsymbol{h}_j\right)}{\sum\limits_{\pm H[i,j]=-1} \gamma\left(\pm\boldsymbol{h}_j\right)}. \end{aligned} \tag{38}$$

We define

$$\boldsymbol{L}^H_{app} = [L^H_{app}(0)\ L^H_{app}(1)\ \cdots\ L^H_{app}(2^r - 1)]^T. \tag{39}$$

Based on the butterfly-like structure of the Hadamard matrix, $\boldsymbol{L}^H_{app}$ can be computed using the fast Hadamard transform (FHT) and the dual FHT (DFHT) [69, 71, 72]. Hard decisions can then be made on $L^H_{app}(i)$ to estimate code bits. In the case of iterative decoding, the Hadamard decoder subtracts the $L^H_{apr}(i)$ from $L^H_{app}(i)$ and feeds back "new" extrinsic information to other component decoders.

2.2.2 *LDPC-Hadamard Codes*

In the Tanner graph of an LDPC code, a VN with degree-d_j emits $d_j (d_j > 1)$ edges connecting to d_j different CNs and forms a $(d_j, 1)$ repeat code; whereas a CN with degree-d_i emits $d_i (d_i > 1)$ edges connecting d_i different VNs and forms a $(d_i, d_i - 1)$ single-parity-check (SPC) code. A generalized LDPC code is obtained when the repeat code and/or SPC code is/are replaced by other block codes.

In [1], the SPC codes of an LDPC code are replaced with Hadamard codes, forming an LDPC-Hadamard code. Fig. 9 depicts the Tanner graph of an LDPC-Hadamard code. In particular, the code possesses the following characteristics.

- *Structure:* Hadamard parity-check bits are added to the CNs in the Tanner graph such that the SPC constraints become the Hadamard constraints (see the Hadamard check node shown in Fig. 9).
- *Encoding:* LDPC coded bits are first generated based on LDPC parity-check matrix and then the LDPC coded bits are used to generate Hadamard parity-check bits (corresponding to Hadamard degree-1 variable nodes).

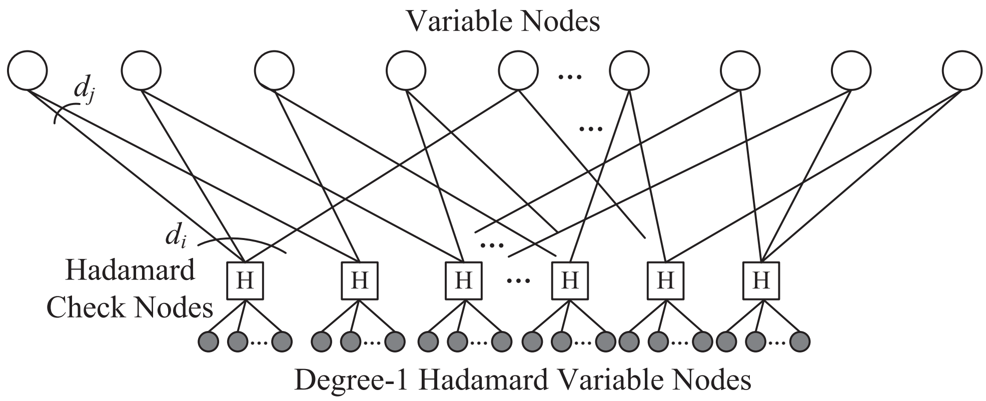

Figure 9: Tanner graph of LDPC-Hadamard codes. The unfilled circles denote variable nodes which are the same as variable nodes in Tanner graph of LDPC codes, while SPC-CNs in Tanner graph of LDPC codes are replaced with Hadamard check nodes (denoted as squares with symbol "H") with some attached Hadamard degree-1 variable nodes (denoted as filled circles).

- *Decoding:* LDPC-Hadamard decoding retains the same variable-node updating method while replacing the check-node updating method with the symbol-MAP Hadamard decoding.
- *Optimization:* With a fixed Hadamard order and a given code rate, the EXIT method is used to adjust the degree distribution of VNs. The aim is to find an optimal degree distribution of the VNs such that the EXIT curves of the repeat codes (i.e., VNs) and Hadamard codes are matched under a low E_b/N_0.

2.3 summary

In this chapter, we have reviewed some important component codes, i.e., LDPC code ensembles, Hadamard codes and LDPC-Hadamard codes, their code structures, analysis techniques, and encoding and decoding methods. In the next chapter, we begin introducing our own works, i.e., PLDPC-Hadamard codes.

Part III

PROPOSED PROTOGRAPH-BASED LOW-DENSITY PARITY-CHECK HADAMARD CODES

CHAPTER 3

PROTOGRAPH-BASED LDPC-HADAMARD BLOCK CODES

In this chapter, we propose a variation of LDPC-Hadamard code called *protograph-based LDPC Hadamard (PLDPC-Hadamard) code* [93, 94]. We provide the detailed decoding for even/odd-order Hadamard decoder when evaluating the error performance of PLDPC-Hadamard codes. We also propose a PEXIT chart method to compute the threshold of PLDPC-Hadamard codes while providing the optimization criterion. By the proposed methods, we search for protomatrices with low thresholds for PLDPC-Hadamard codes. Extensive simulations and the corresponding analysis for unpunctured/punctured PLDPC-Hadamard codes are conducted.

3.1 CODE STRUCTURE

The base structure of a PLDPC-Hadamard code is shown in Fig. 10, where

- each blank circle denotes a protograph variable-node (P-VN);
- each square with an "H" inside denotes a Hadamard check-node (H-CN); and
- each filled circle denotes a degree-1 Hadamard variable-node (D1H-VN).

We assume that there are n P-VNs and m H-CNs. The protomatrix of the proposed PLDPC-Hadamard codes is then denoted by $\boldsymbol{B}_{m\times n} = \{b_{i,j}\}$, where $b_{i,j}$ represents the number of edges connecting the i-th H-CN ($i = 0, 1, \ldots, m-1$) and the j-th P-VN ($j = 0, 1, \ldots, n-1$). Moreover, we denote the weight of the i-th row by $d_{c_i} = \sum_{j=0}^{n-1} b_{i,j}$, which represents the total number of edges connecting the i-th H-CN to all P-VNs. For example in Fig. 10, the number of edges connecting each of the three displayed H-CNs to all P-VNs is equal to $d_{c_i} = 6$. These d_{c_i} edges are considered as (input) information bits to the i-th Hadamard code while the connected D1H-VNs represent the corresponding (output) parity bits in the Hadamard code. Recall that an order-r Hadamard code contains 2^{r+1} codewords with each codeword containing $r+1$ information bits. Suppose a Hadamard code of order-$(d_{c_i}-1)$ is used to encode these d_{c_i} inputs and generate

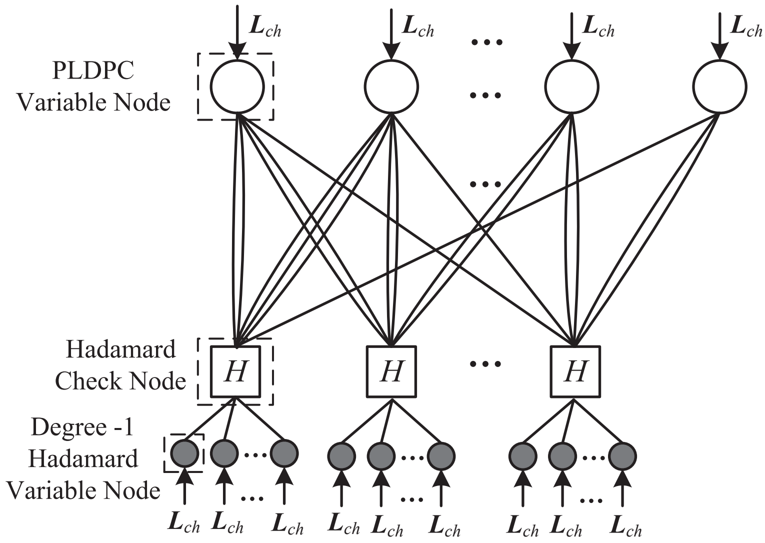

Figure 10: The protograph of a PLDPC-Hadamard code.

$2^{(d_{c_i}-1)} - d_{c_i}$ Hadamard parity-check bits. As these $d_{c_i} = r + 1$ bits take part in the same parity-check equation of an LDPC code and need to fulfill the SPC constraint [1], the number of possible combinations of these d_{c_i} bits is only $2^{(d_{c_i}-1)}$ and thus $2^{(d_{c_i}-1)} = 2^r$ Hadamard codewords will be generated. In other words, only half of the 2^{r+1} available Hadamard codewords are used, making the encoding process very inefficient.

Same as in LDPC-Hadamard codes [1], we utilize Hadamard codes with order $r = d_{c_i} - 2$ $(r > 2)$ in the proposed PLDPC-Hadamard codes. With such an arrangement,

- all possible Hadamard codewords, i.e., $2^{(d_{c_i}-1)} = 2^{r+1}$ can be utilized;
- fewer Hadamard parity bits compared with the case of $r = d_{c_i} - 1$ need to be added (only $(2^{(d_{c_i}-2)} - d_{c_i})$ and $(2^{(d_{c_i}-2)} - 2)$ Hadamard parity-check bits are generated for r is even and odd, respectively);
- the encoding process becomes most efficient;
- the overall code rate is increased; and
- the decoding performance is improved.

Note that a Hadamard code with order $r = 2$ is equivalent to the $(4, 3)$ SPC code. No extra parity-check bits (i.e., D1H-VNs) will

1 If these $d_{c_i} = r + 1$ input bits are not required to satisfy a SPC constraint, two other types of LDPC-Hadamard codes can be formed and they are briefly described in Appendix A.

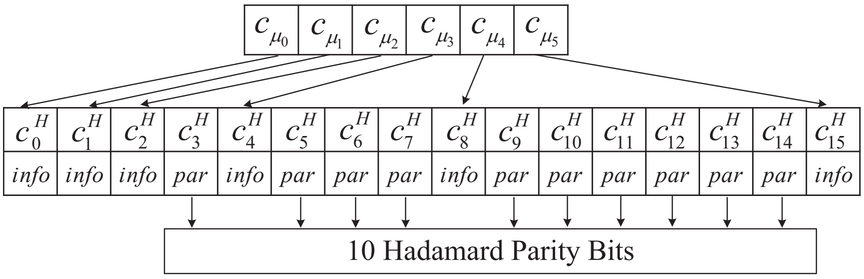

Figure 11: Example of encoding a length-6 SPC codeword into a length-16 ($r = 4$) Hadamard codeword.

be generated if such an Hadamard code is used in the PLDPC-Hadamard code. Thus Hadamard codes with order $r = 2$ are not considered.

In the following, we consider the cases when r is even and odd separately. It is because systematic Hadamard encoding is possible when r is even and non-systematic Hadamard encoding needs to be used when r is odd.

3.1.1 $r = d_{c_i} - 2$ *Is An Even Number*

We denote a Hadamard codeword by $\boldsymbol{c}^H = [c_0^H \; c_1^H \; \ldots c_{2^r-1}^H]$. For r being an even number, it has been shown that [1]

$$[c_0^H \oplus c_1^H \oplus c_2^H \oplus \cdots \oplus \; c_{2^{k-1}}^H \oplus \cdots \oplus \; c_{2^{r-1}}^H] \oplus c_{2^r-1}^H = 0. \qquad (40)$$

Viewing from another perspective, if there is a length-$(r+2)$ SPC codeword denoted by $\boldsymbol{c}_\mu = [c_{\mu_0} \; c_{\mu_1} \; \ldots \; c_{\mu_r} \; c_{\mu_{r+1}}]$, these bits can be used as inputs to a systematic Hadamard encoder and form a Hadamard codeword where

$$c_0^H = c_{\mu_0}, \; c_1^H = c_{\mu_1}, \cdots, c_{2^{k-1}}^H = c_{\mu_k}, \cdots, c_{2^{r-1}}^H = c_{\mu_r}, c_{2^r-1}^H = \; c_{\mu_{r+1}} \qquad (41)$$

correspond to $r+2$ P-VNs and the remaining Hadamard parity bits in $\boldsymbol{c}^H$ correspond to $2^r - (r+2)$ D1H-VNs. Fig. 11 shows an example in which a $(6,5)$ SPC codeword is encoded into a length-16 ($r = 4$) Hadamard codeword. For the length-6 SPC code, we use the order $r = 6-2 = 4$ Hadamard matrix, i.e., $\pm\boldsymbol{H}_{16}$, to generate 10 Hadamard parity bits. Suppose $+1$ is mapped to bit "0" and -1 to bit "1" in Hadamard matrix. We show $\pm\boldsymbol{H}_{16}$ in Fig. 12, where $\pm h_{0,j} \oplus \pm h_{1,j} \oplus \pm h_{2,j} \oplus \pm h_{4,j} \oplus \pm h_{8,j} \oplus \pm h_{15,j} = 0 \; \forall \; j$ and $c_{\mu_0} = \pm h_{0,j}$, $c_{\mu_1} = \pm h_{1,j}$, $c_{\mu_2} = \pm h_{2,j}$, $c_{\mu_3} = \pm h_{4,j}$, $c_{\mu_4} = \pm h_{8,j}$, $c_{\mu_5} = \pm h_{15,j}$.

Referring to Fig. 10, the links connecting the P-VNs to the i-th H-CN always form a SPC. These links can make use of the above mechanism to derive the parity bits of the Hadamard code (denoted

$$
\begin{array}{c}
\begin{array}{cccccccccccccccc}
\pm\boldsymbol{h}_0 & \pm\boldsymbol{h}_1 & \pm\boldsymbol{h}_2 & \pm\boldsymbol{h}_3 & \pm\boldsymbol{h}_4 & \pm\boldsymbol{h}_5 & \pm\boldsymbol{h}_6 & \pm\boldsymbol{h}_7 & \pm\boldsymbol{h}_8 & \pm\boldsymbol{h}_9 & \pm\boldsymbol{h}_{10} & \pm\boldsymbol{h}_{11} & \pm\boldsymbol{h}_{12} & \pm\boldsymbol{h}_{13} & \pm\boldsymbol{h}_{14} & \pm\boldsymbol{h}_{15}
\end{array} \\
\pm\boldsymbol{H}_{16} = \begin{bmatrix}
\pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 \\
\pm1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & +1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 \\
\pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 \\
\pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 \\
\pm1 & \pm1 & \pm1 & \pm1 & \mp1 & \mp1 & \mp1 & \mp1 & \pm1 & \pm1 & \pm1 & \pm1 & \mp1 & \mp1 & \mp1 & \mp1 \\
\pm1 & \mp1 & \pm1 & \mp1 & \mp1 & \pm1 & \mp1 & \pm1 & \pm1 & \mp1 & \pm1 & \mp1 & \mp1 & \pm1 & \mp1 & \pm1 \\
\pm1 & \pm1 & \mp1 & \mp1 & \mp1 & \mp1 & \pm1 & \pm1 & \pm1 & \pm1 & \mp1 & \mp1 & \mp1 & \mp1 & \pm1 & \pm1 \\
\pm1 & \mp1 & \mp1 & \pm1 & \mp1 & \pm1 & \pm1 & \mp1 & \pm1 & \mp1 & \mp1 & \pm1 & \mp1 & \pm1 & \pm1 & \mp1 \\
\pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \pm1 & \mp1 & \mp1 & \mp1 & \mp1 & \mp1 & \mp1 & \mp1 & \mp1 \\
\pm1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & \pm1 \\
\pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 \\
\pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 \\
\pm1 & \pm1 & \pm1 & \pm1 & \mp1 & \mp1 & \mp1 & \mp1 & \mp1 & \mp1 & \mp1 & \mp1 & \pm1 & \pm1 & \pm1 & \pm1 \\
\pm1 & \mp1 & \pm1 & \mp1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & \pm1 & \mp1 & \pm1 & \pm1 & \mp1 & \pm1 & \mp1 \\
\pm1 & \pm1 & \mp1 & \mp1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \pm1 & \pm1 & \mp1 & \mp1 \\
\pm1 & \mp1 & \mp1 & \pm1 & \mp1 & \pm1 & \pm1 & \mp1 & \mp1 & \pm1 & \pm1 & \mp1 & \pm1 & \mp1 & \mp1 & \pm1
\end{bmatrix}
\end{array}
\quad
\boxed{\begin{array}{c}
\pm h_{0,j} \\ \oplus \\ \pm h_{1,j} \\ \oplus \\ \pm h_{2,j} \\ \oplus \\ \pm h_{4,j} \\ \oplus \\ \pm h_{8,j} \\ \oplus \\ \pm h_{15,j} \\ \parallel \\ 0
\end{array}}
$$

Figure 12: Hadamard matrix $\pm\boldsymbol{H}_{16}$ and the Hadamard codewords $\{\pm\boldsymbol{h}_j : j = 0, 1, \ldots, 15\}$. When $+1$ is mapped to bit "0" and -1 is mapped to bit "1", $\pm h_{0,j} \oplus \pm h_{1,j} \oplus \pm h_{2,j} \oplus \pm h_{4,j} \oplus \pm h_{8,j} \oplus \pm h_{15,j} = 0 \; \forall \, j$.

as D1H-VNs of the Hadamard check node in Fig. 10) if d_{c_i} is even. In this case, the Hadamard code length equals $2^{d_{c_i}-2}$, and the number of D1H-VNs equals $2^{d_{c_i}-2} - d_{c_i}$. Assuming d_{c_i} is even for all $i = 0, 1, \ldots, m-1$, the total number of D1H-VNs is given by $\sum_{i=0}^{m-1} \left(2^{d_{c_i}-2} - d_{c_i}\right)$. When all VNs are sent to the channel, the code rate of the protograph given in Fig. 10 equals

$$R^{\text{even}} = \frac{n - m}{\sum\limits_{i=0}^{m-1} \left(2^{d_{c_i}-2} - d_{c_i}\right) + n}. \tag{42}$$

If we further assume that all rows in $\boldsymbol{B}_{m \times n}$ have the same weight which is equal to d, i.e., $d_{c_i} = d$ for all i, the code rate is simplified to

$$R^{\text{even}}_{d_{c_i}=d} = \frac{n - m}{m\left(2^{d-2} - d\right) + n}. \tag{43}$$

When $n_p \, (< n)$ P-VNs are punctured, the code rate becomes

$$R^{\text{even}}_{\text{punctured}} = \frac{n - m}{m\left(2^{d-2} - d\right) + n - n_p}. \tag{44}$$

3.1.2 $r = d_{c_i} - 2$ *Is An Odd Number*

For r being an odd number, the 2^r Hadamard codewords in $+\boldsymbol{H}_q$ can satisfy (40) but all the 2^r Hadamard codewords in $-\boldsymbol{H}_q$ cannot. We

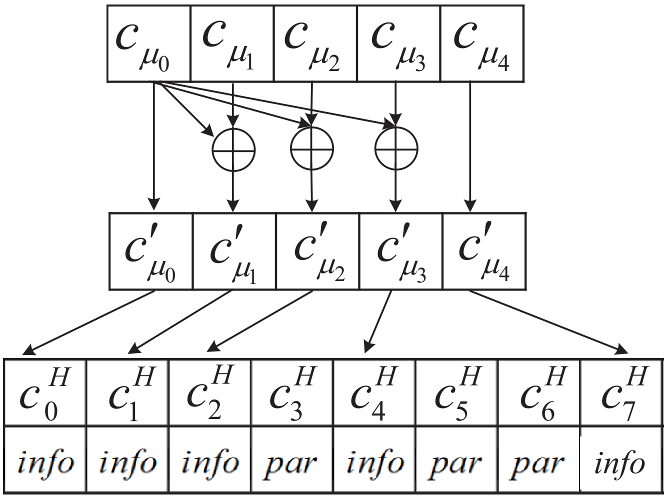

Figure 13: Example of encoding a length-5 SPC codeword into a length-8 ($r = 3$) Hadamard codeword.

apply the same non-systematic encoding method in [1] to encode the SPC codeword [2]. Supposing $\boldsymbol{c}_\mu$ is a SPC codeword, we preprocess $\boldsymbol{c}_\mu = [c_{\mu_0}\ c_{\mu_1} \dots c_{\mu_{r+1}}]$ to obtain $\boldsymbol{c}'_\mu = [c'_{\mu_0}\ c'_{\mu_1} \dots c'_{\mu_{r+1}}]$, and then we perform Hadamard encoding for $\boldsymbol{c}'_\mu$ to obtain $\boldsymbol{c}^H = [c_0^H\ c_1^H \dots c_{2^r-1}^H]$, where

$$\begin{aligned}
c_0^H &= c'_{\mu_0} &&= c_{\mu_0} \\
c_1^H &= c'_{\mu_1} &&= c_{\mu_1} \oplus c_{\mu_0} \\
&\vdots \\
c_{2^{k-1}}^H &= c'_{\mu_k} &&= c_{\mu_k} \oplus c_{\mu_0} \\
&\vdots \\
c_{2^{r-1}}^H &= c'_{\mu_r} &&= c_{\mu_r} \oplus c_{\mu_0} \\
c_{2^r-1}^H &= c'_{\mu_{r+1}} &&= c_{\mu_{r+1}}.
\end{aligned} \tag{45}$$

Fig. 13 shows an example in which a (5,4) SPC codeword is encoded into a length-8 ($r = 3$) Hadamard codeword. It can be seen that after the non-systematic encoding, only the first and last code bits are the same as the original information bits, i.e., $c_0^H = c'_{\mu_0} = c_{\mu_0}$ and $c_{2^r-1}^H = c'_{\mu_{r+1}} = c_{\mu_{r+1}}$. Thus we send the remaining code bits, i.e., c_1^H to $c_{2^r-2}^H$, to provide more channel observations for the decoder and the number of D1H-VNs equals $2^{d_{c_i}-2} - 2$. For example, the code bits $[c_1^H\ c_2^H\ c_3^H\ c_4^H\ c_5^H\ c_6^H]$ shown in Fig. 13 will be sent.

2 Note that there are other non-systematic encoding methods, e.g., preprocess $\boldsymbol{c}_\mu = [c_{\mu_0}\ c_{\mu_1} \dots c_{\mu_{r+1}}]$ to obtain $\boldsymbol{c}'_\mu = [c'_{\mu_0}\ c'_{\mu_1} \dots c'_{\mu_{r+1}}]$, where $c'_{\mu_i} = c_{\mu_i}$ for $i = 0, 1, 2, \dots, r$; and $c'_{\mu_{r+1}} = c_{\mu_{r+1}} \oplus c_{\mu_0}$.

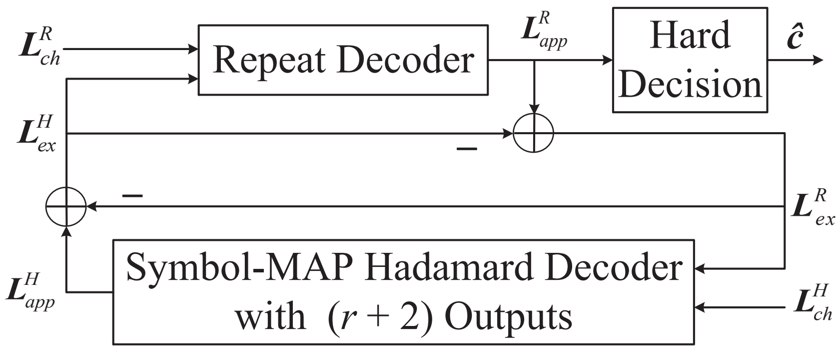

Figure 14: Block diagram of a PLDPC-Hadamard decoder. The repeat decoder is the same as the variable-node processor used in LDPC decoder. For the symbol-MAP Hadamard decoder, the number of outputs is always $r+2$; the number of inputs is 2^r when r is even; the number of inputs is $2^r + r$ when r is odd.

Assuming all the rows in $\boldsymbol{B}_{m \times n}$ have the same weight d, the code rate is given by

$$R^{\text{odd}}_{d_{c_i}=d} = \frac{n-m}{m\,(2^{d-2}-2)+n}. \tag{46}$$

If n_p $(< n)$ P-VNs are punctured, the code rate becomes

$$R^{\text{odd}}_{\text{punctured}} = \frac{n-m}{m\,(2^{d-2}-2)+n-n_p}. \tag{47}$$

Note that for $k = 1, 2, \ldots, r$,

- $c^H_{2^{k-1}} = c'_{\mu_k} = c_{\mu_k} \oplus c_0$ and hence $c_{\mu_k} = c^H_{2^{k-1}} \oplus c_0$;
- c_{μ_k} is transmitted as P-VN; and
- $c^H_{2^{k-1}}$ is transmitted as D1H-VN.

Thus the r information bits c_{μ_k} can have both the *a priori* information provided by the extrinsic information from P-VNs and the channel information of $c^H_{2^{k-1}} = c'_{\mu_k} = c_{\mu_k} \oplus c_0$ from D1H-VNs. However, the two information bits c_{μ_0} and $c_{\mu_{r+1}}$ only have the *a priori* information from P-VNs and the $2^r - (r+2)$ Hadamard parity bits only have the channel information from D1H-VNs. Supposing for every H-CN, n_h $(\leqslant r)$ D1H-VNs corresponding to code bits $c^H_{2^{k-1}}$ $(k = 1, 2, \ldots, r)$ are also punctured. The code rate further becomes

$$R^{\text{odd}}_{\text{punctured D1H}-\text{VN}} = \frac{n-m}{m\,(2^{d-2}-2-n_h)+n-n_p}. \tag{48}$$

3.2 Decoder of PLDPC-Hadamard codes

To evaluate the performance of PLDPC-Hadamard codes, the iterative decoder shown in Fig. 14 is used. It consists of a repeat decoder and a symbol-MAP Hadamard decoder. The repeat decoder is the same as the variable-node processor used in an LDPC decoder. The operations of a repeat decoder can be found in (65) and (66) in Section 4.1.

As described in the previous section, each H-CN with an order-r Hadamard constraint is connected to $r+2$ P-VNs in the protograph of a PLDPC-Hadamard code. The symbol-MAP Hadamard decoder of order-r has a total of 2^r or 2^r+r inputs, among which $r+2$ come from the repeat decoder and are updated in each iteration; and the remaining inputs come from the channel LLR information which do not change during the iterative process. Moreover, the symbol-MAP Hadamard decoder will produce $r+2$ extrinsic LLR outputs which are fed back to the repeat decoder. The iterative process between the repeat decoder and symbol-MAP Hadamard decoder continues until the information bits corresponding to all Hadamard codes (after hard decision) become valid SPCs or the maximum number of iterations has been reached. In the following, we show the details of the operations of the symbol-MAP Hadamard decoder.

3.2.1 *Even-Order Hadamard Decoder*

A H-CN has $r+2$ links to P-VNs and is connected to $2^r-(r+2)$ D1H-VNs. Specifically, we denote

- $\boldsymbol{L}^{\mathrm{R}}_{\mathrm{ex}} = [\mathrm{L}^{\mathrm{R}}_{\mathrm{ex}}(0)\ \mathrm{L}^{\mathrm{R}}_{\mathrm{ex}}(1)\ \cdots\ \mathrm{L}^{\mathrm{R}}_{\mathrm{ex}}(r+1)]^{\mathrm{T}}$ as the $r+2$ extrinsic LLR information values coming from the repeat decoder (P-VNs),
- $\boldsymbol{L}^{\mathrm{H}}_{\mathrm{apr}} = [\mathrm{L}^{\mathrm{H}}_{\mathrm{apr}}(0)\ \mathrm{L}^{\mathrm{H}}_{\mathrm{apr}}(1)\ \cdots\ \mathrm{L}^{\mathrm{H}}_{\mathrm{apr}}(2^r-1)]^{\mathrm{T}}$ as the 2^r *a priori* LLR values of $\boldsymbol{c}^{\mathrm{H}}$,
- $\boldsymbol{y}^{\mathrm{H}}_{\mathrm{ch}} = [y^{\mathrm{H}}_{\mathrm{ch}}(0)\ y^{\mathrm{H}}_{\mathrm{ch}}(1)\ \cdots\ y^{\mathrm{H}}_{\mathrm{ch}}(2^r-1)]^{\mathrm{T}}$ as the length-2^r channel observation vector corresponding to $\boldsymbol{c}^{\mathrm{H}}$ and is derived from the D1H-VNs (note that $r+2$ channel observations are zero),
- $\boldsymbol{L}^{\mathrm{H}}_{\mathrm{ch}} = [\mathrm{L}^{\mathrm{H}}_{\mathrm{ch}}(0)\ \mathrm{L}^{\mathrm{H}}_{\mathrm{ch}}(1)\ \cdots\ \mathrm{L}^{\mathrm{H}}_{\mathrm{ch}}(2^r-1)]^{\mathrm{T}}$ as the length-2^r channel LLR observations corresponding to $\boldsymbol{c}^{\mathrm{H}}$.

Based on (41) and the transmission mechanism, *a priori* LLR values exist only for the $r+2$ information bits in $\boldsymbol{c}^{\mathrm{H}}$ and they are equal to the extrinsic LLR values $\boldsymbol{L}^{\mathrm{R}}_{\mathrm{ex}}$ from the repeat decoder. Correspondingly, channel LLR values only exist for the 2^r-r-2 Hadamard parity bits in $\boldsymbol{c}^{\mathrm{H}}$ and they are obtained from the received channel observations

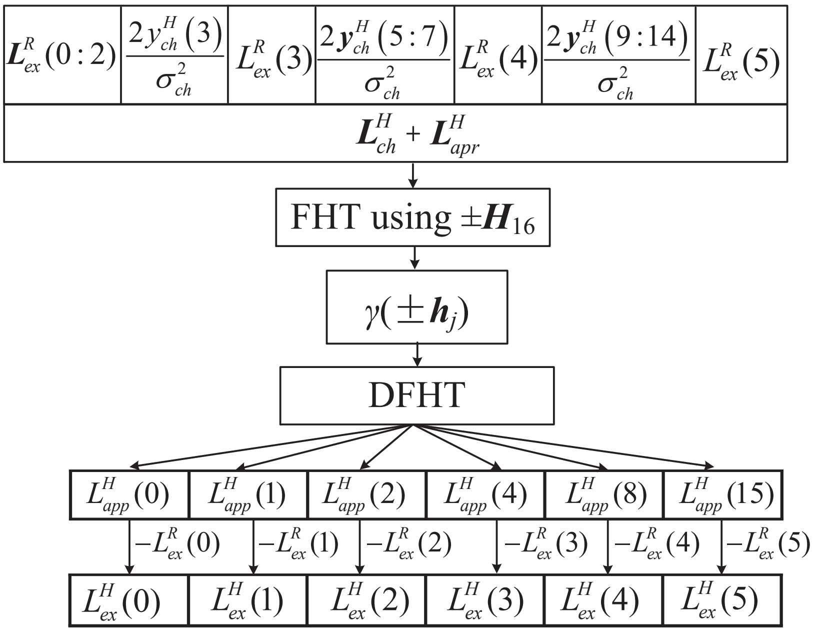

Figure 15: Operations in the symbol-MAP Hadamard decoder for $r = 4$, i.e., 16 LLR inputs and 6 output LLR values for the information bits.

$\boldsymbol{y}_{ch}^{H}$. In other words, only $2^r - r - 2$ entries in $\boldsymbol{y}_{ch}^{H}$ and also $\boldsymbol{L}_{ch}^{H}$ are non-zero. Thus the entries of $\boldsymbol{L}_{ch}^{H}$ and $\boldsymbol{L}_{apr}^{H}$ are assigned as

$$\begin{cases} L_{apr}^{H}(k) = L_{ex}^{R}(0) \\ L_{ch}^{H}(k) = \frac{2y_{ch}^{H}(0)}{\sigma_{ch}^{2}} = 0 \end{cases} \text{for } k = 0;$$
$$\begin{cases} L_{apr}^{H}(k) = L_{ex}^{R}(i) \\ L_{ch}^{H}(k) = \frac{2y_{ch}^{H}(k)}{\sigma_{ch}^{2}} = 0 \end{cases} \text{for } k = 1, 2, \cdots, 2^{i-1}, \cdots, 2^{r-1};$$
$$\begin{cases} L_{apr}^{H}(k) = L_{ex}^{R}(r+1) \\ L_{ch}^{H}(k) = \frac{2y_{ch}^{H}(k)}{\sigma_{ch}^{2}} = 0 \end{cases} \text{for } k = 2^r - 1;$$
$$\begin{cases} L_{apr}^{H}(k) = 0 \\ L_{ch}^{H}(k) = \frac{2y_{ch}^{H}(k)}{\sigma_{ch}^{2}} \end{cases} \text{for the } 2^r - r - 2 \text{ remaining } k. \tag{49}$$

The symbol-MAP Hadamard decoder then computes the *a posteriori* LLR ($\boldsymbol{L}_{app}^{H}$) of the code bits using (38) and (37). By subtracting the *a priori* LLR values from the *a posteriori* LLR values, the extrinsic LLR values ($\boldsymbol{L}_{ex}^{H}$) can be obtained. Fig. 15 illustrates the flow of the computation of $\boldsymbol{L}_{app}^{H}$ and hence $\boldsymbol{L}_{ex}^{H}$ for $r = 4$, which corresponds to $r + 2 = 6$ information bits (and $2^r - (r+4) = 10$ Hadamard parity bits).

3.2.2 *Odd-Order Hadamard Decoder*

A H-CN is connected to $r+2$ P-VNs and 2^r-2 D1H-VNs, and the bits corresponding to the $r+2$ P-VNs form a SPC codeword $\boldsymbol{c}_\mu$. Similar to the "r is an even number" case, we denote

- $\boldsymbol{L}^R_{ex} = [L^R_{ex}(0)\ L^R_{ex}(1)\ \cdots\ L^R_{ex}(r+1)]^T$ as the $r+2$ extrinsic LLR information values coming from the repeat decoder (P-VNs),
- $\boldsymbol{L}^H_{apr} = [L^H_{apr}(0)\ L^H_{apr}(1)\ \cdots\ L^H_{apr}(2^r-1)]^T$ as the 2^r *a priori* LLR values of $\boldsymbol{c}^H$,
- $\boldsymbol{y}^H_{ch} = [y^H_{ch}(0)\ y^H_{ch}(1)\ \cdots\ y^H_{ch}(2^r-1)]^T$ as the length-2^r channel observation vector corresponding to $\boldsymbol{c}^H$ and is derived from the D1H-VNs (note that the first and the last channel observations are zero),
- $\boldsymbol{L}^H_{ch} = [L^H_{ch}(0)\ L^H_{ch}(1)\ \cdots\ L^H_{ch}(2^r-1)]^T$ as the length-2^r channel LLR observations corresponding to $\boldsymbol{c}^H$.

Since non-systematic Hadamard code is used, $\boldsymbol{c}_\mu$ does not represent the information bits in $\boldsymbol{c}^H$ for c_{μ_0} = "1". Thus, we cannot directly apply (38) to obtain the *a posteriori* LLR of $\boldsymbol{c}_\mu$. Here, we present the decoding steps when r is odd. Detailed derivations are shown in Appendix B.

Referring to (45), the assignment of $\boldsymbol{L}^H_{apr}$ depends on c_{μ_0}. For convenience of explanation, we denote $\boldsymbol{L}^{+H}_{apr}/\boldsymbol{L}^{-H}_{apr}$ as the assignment of $\boldsymbol{L}^H_{apr}$ for c_{μ_0} = "0"/"1", respectively. We use (B4) to assign $\boldsymbol{L}^{\pm H}_{apr}$ and (B5) to assign $\boldsymbol{L}^H_{ch}$. Since the first bit in all $+\boldsymbol{h}_j/-\boldsymbol{h}_j$ is "0"/"1" (+1 mapped to "0" and −1 to "1"), we apply $\boldsymbol{L}^{\pm H}_{apr}$ and $\boldsymbol{L}^H_{ch}$ to compute $\gamma(\pm\boldsymbol{h}_j)$, i.e.,

$$\gamma\left(\pm\boldsymbol{h}_j\right) = \exp\left(\frac{1}{2}\left\langle \pm\boldsymbol{h}_j, \boldsymbol{L}^H_{ch} + \boldsymbol{L}^{\pm H}_{apr}\right\rangle\right). \tag{50}$$

We define the $r+2$ *a posteriori* LLR values ($\boldsymbol{L}^H_{app}$) of the original bits $\boldsymbol{c}_\mu$ by

$$\boldsymbol{L}^H_{app} = [L^H_{app}(0)\ L^H_{app}(1)\ \cdots\ L^H_{app}(2^{i-1})\ \cdots \\ L^H_{app}(2^{r-1})\ L^H_{app}(2^r-1)]^T. \tag{51}$$

We use (37) and (38) to compute $L^H_{app}(0)$ and $L^H_{app}(2^r-1)$; (B9) to obtain $\gamma'(-\boldsymbol{h}_j)$ and then DFHT to compute (B10) to obtain $L^H_{app}(2^{k-1})$ $k = 1, 2, \cdots, r$. Fig. 16 illustrates for the case $r = 3$, the transformation from $\gamma(-\boldsymbol{h}_j)$ to $\gamma'(-\boldsymbol{h}_j)$, i.e., $\gamma'(-\boldsymbol{h}_j) = \gamma(-\boldsymbol{h}_{2^r-1-j})$. Then $\boldsymbol{L}^H_{ex} = \boldsymbol{L}^H_{app} - \boldsymbol{L}^R_{ex}$ of length $r+2$ is computed and fed back to the repeat decoder. The steps to compute $\boldsymbol{L}^H_{ex}$ for the case $r = 3$ is shown in Fig. 17.

$\gamma'(-\boldsymbol{h}_0)$	$\gamma'(-\boldsymbol{h}_1)$	$\gamma'(-\boldsymbol{h}_2)$	$\gamma'(-\boldsymbol{h}_3)$	$\gamma'(-\boldsymbol{h}_4)$	$\gamma'(-\boldsymbol{h}_5)$	$\gamma'(-\boldsymbol{h}_6)$	$\gamma'(-\boldsymbol{h}_7)$
$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
$\gamma(-\boldsymbol{h}_7)$	$\gamma(-\boldsymbol{h}_6)$	$\gamma(-\boldsymbol{h}_5)$	$\gamma(-\boldsymbol{h}_4)$	$\gamma(-\boldsymbol{h}_3)$	$\gamma(-\boldsymbol{h}_2)$	$\gamma(-\boldsymbol{h}_1)$	$\gamma(-\boldsymbol{h}_0)$

Figure 16: Illustration of $\gamma'(-\boldsymbol{h}_j) = \gamma(-\boldsymbol{h}_{2^r-1-j})$ for $r = 3$.

Remark: If some more bits in $c^H_{2^{k-1}}$ for $k = 1, 2, \ldots, r$ are punctured, the corresponding channel observation $y^H_{ch}(2^{k-1})$ and LLR values of $L^H_{ch}(2^{k-1})$ are set to 0 and the overall code rate will slightly increase.

3.3 CODE DESIGN OPTIMIZATION

We propose a low-complexity PEXIT algorithm for analyzing PLDPC-Hadamard codes. Our low-complexity PEXIT algorithm uses the same MI updating method as the original PEXIT algorithm [37] for the P-VNs. However, our algorithm computes extrinsic MI for the symbol-MAP Hadamard decoder whereas the original PEXIT algorithm computes extrinsic MI for the SPC decoder. We use Monte Carlo method in obtaining the extrinsic MI values of the symbol-MAP Hadamard decoder. The algorithm not only has a low complexity, but also is generic and applicable to analyzing both systematic and non-systematic Hadamard codes.

We define the following symbols.

- $I_{av}(i, j)$: the *a priori* mutual information (MI) from the i-th H-CN to the j-th P-VN;
- $I_{ev}(i, j)$: extrinsic MI from the j-th P-VN to the i-th H-CN;
- $I_{ah}(i, k)$: the *a priori* MI of the k-th information bit in the i-th H-CN;
- $I_{eh}(i, k)$: extrinsic MI of the k-th information bit in the i-th H-CN;
- $I_{app}(j)$ the *a posteriori* MI of the j-th P-VN.

Referring to Fig. 10, the channel LLR value L_{ch} follows a normal distribution $\mathcal{N}(\sigma^2_{L_{ch}}/2, \sigma^2_{L_{ch}})$ where $\sigma^2_{L_{ch}} = 8R \cdot E_b/N_0$ and R is the code rate. When the output MI of a decoder is I, the corresponding LLR values of the extrinsic information obeys a Gaussian distribution of $(\pm\sigma^2/2, \sigma^2)$. The relationship between I and σ can be approximately computed by functions $I = J(\sigma)$ (3) and $\sigma = J^{-1}(I)$ (4) [17, 23].

3.3.1 *Modified PEXIT Algorithm*

To generate the PEXIT curves for the repeat decoder and symbol-MAP Hadamard decoder, we apply the following steps for a given

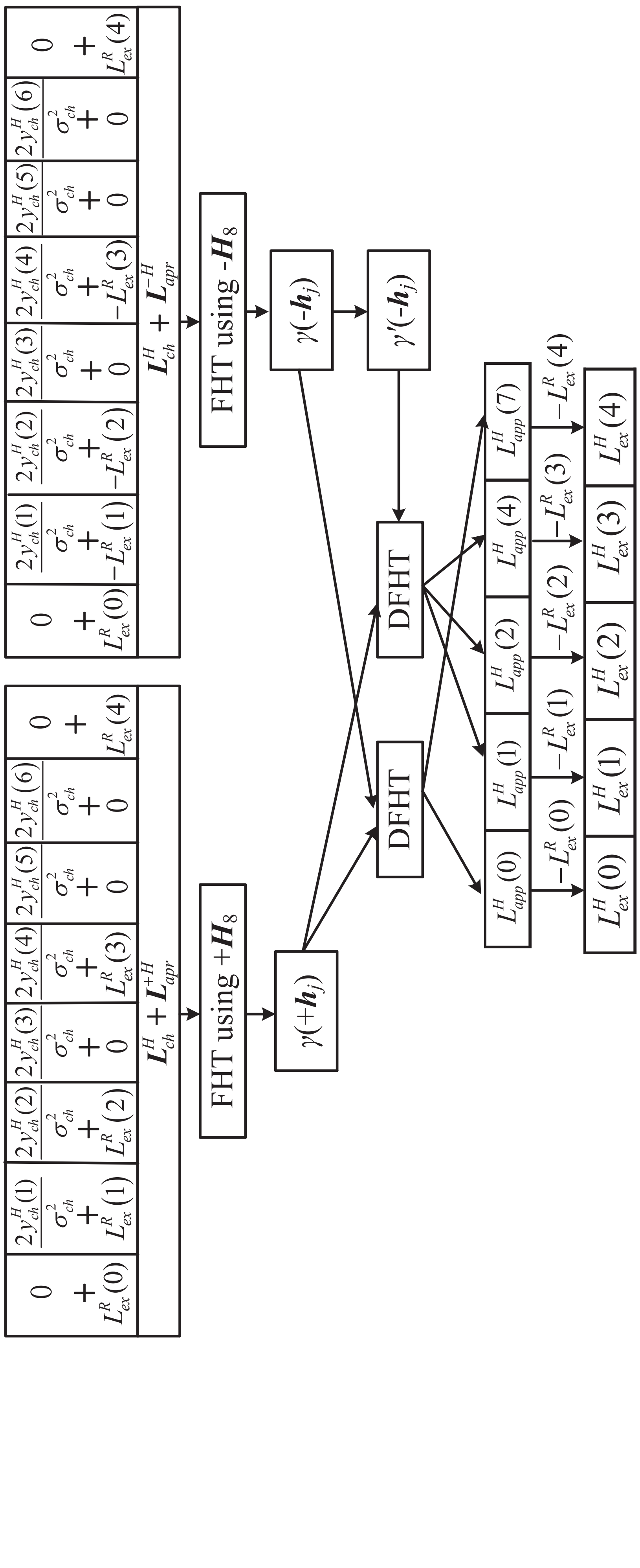

Figure 17: Operations in the symbol-MAP Hadamard decoder for r = 3, i.e., 11 LLR inputs and 5 output LLR values for the information bits.

$$\boldsymbol{B}_{3\times 4} = \begin{bmatrix} 2 & 0 & 2 & 2 \\ 0 & 2 & 2 & 2 \\ 3 & 2 & 0 & 1 \end{bmatrix} \tag{52}$$

$$I_{ev}(i,j) = J\left(\sqrt{\sum_{s\neq i} b_{s,j}\left(J^{-1}\left(I_{av}\left(s,j\right)\right)\right)^2 + \left(b_{i,j}-1\right)\cdot\left(J^{-1}\left(I_{av}\left(i,j\right)\right)\right)^2 + \sigma^2_{L_{ch}}}\right) \tag{53}$$

$$I_{ev} = \begin{bmatrix} I_{ev}(0,0) & 0 & I_{ev}(0,2) & I_{ev}(0,3) \\ 0 & I_{ev}(1,1) & I_{ev}(1,2) & I_{ev}(1,3) \\ I_{ev}(2,0) & I_{ev}(2,1) & 0 & I_{ev}(2,3) \end{bmatrix} \tag{54}$$

$$\begin{aligned} I_{ah} &= \begin{bmatrix} I_{ah}(0,0) & I_{ah}(0,1) & I_{ah}(0,2) & I_{ah}(0,3) & I_{ah}(0,4) & I_{ah}(0,5) \\ I_{ah}(1,0) & I_{ah}(1,1) & I_{ah}(1,2) & I_{ah}(1,3) & I_{ah}(1,4) & I_{ah}(1,5) \\ I_{ah}(2,0) & I_{ah}(2,1) & I_{ah}(2,2) & I_{ah}(2,3) & I_{ah}(2,4) & I_{ah}(2,5) \end{bmatrix} \\ &= \begin{bmatrix} I_{ev}(0,0) & I_{ev}(0,0) & I_{ev}(0,2) & I_{ev}(0,2) & I_{ev}(0,3) & I_{ev}(0,3) \\ I_{ev}(1,1) & I_{ev}(1,1) & I_{ev}(1,2) & I_{ev}(1,2) & I_{ev}(1,3) & I_{ev}(1,3) \\ I_{ev}(2,0) & I_{ev}(2,0) & I_{ev}(2,0) & I_{ev}(2,1) & I_{ev}(2,1) & I_{ev}(2,3) \end{bmatrix} \end{aligned} \tag{55}$$

$$I_E = \frac{1}{2}\sum_{x\in\{0,1\}}\int_{-\infty}^{\infty} p_e\left(\xi|X=x\right)\log_2\frac{2\cdot p_e\left(\xi|X=x\right)}{p_e\left(\xi|X=\text{“}0\text{"}\right)+p_e\left(\xi|X=\text{“}1\text{"}\right)}d\xi \tag{56}$$

$$I_{eh} = \begin{bmatrix} I_{eh}(0,0) & I_{eh}(0,1) & I_{eh}(0,2) & I_{eh}(0,3) & I_{eh}(0,4) & I_{eh}(0,5) \\ I_{eh}(1,0) & I_{eh}(1,1) & I_{eh}(1,2) & I_{eh}(1,3) & I_{eh}(1,4) & I_{eh}(1,5) \\ I_{eh}(2,0) & I_{eh}(2,1) & I_{eh}(2,2) & I_{eh}(2,3) & I_{eh}(2,4) & I_{eh}(2,5) \end{bmatrix} \tag{57}$$

$$\begin{aligned} I_{av} &= \begin{bmatrix} I_{av}(0,0) & 0 & I_{av}(0,2) & I_{av}(0,3) \\ 0 & I_{av}(1,1) & I_{av}(1,2) & I_{av}(1,3) \\ I_{av}(2,0) & I_{av}(2,1) & 0 & I_{av}(2,3) \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2}\sum\limits_{k=0}^{1} I_{eh}(0,k) & 0 & \frac{1}{2}\sum\limits_{k=2}^{3} I_{eh}(0,k) & \frac{1}{2}\sum\limits_{k=4}^{5} I_{eh}(0,k) \\ 0 & \frac{1}{2}\sum\limits_{k=0}^{1} I_{eh}(1,k) & \frac{1}{2}\sum\limits_{k=2}^{3} I_{eh}(1,k) & \frac{1}{2}\sum\limits_{k=4}^{5} I_{eh}(1,k) \\ \frac{1}{3}\sum\limits_{k=0}^{2} I_{eh}(2,k) & \frac{1}{2}\sum\limits_{k=3}^{4} I_{eh}(2,k) & 0 & I_{eh}(2,5) \end{bmatrix} \end{aligned} \tag{58}$$

set of protomatrix $\boldsymbol{B}_{m \times n}$ (e.g., (52)), code rate R and E_b/N_0 in dB (denoted as $E_b/N_0(\text{dB})$).

i) Compute $\sigma_{L_{ch}} = \left(8 \cdot R \cdot 10^{(E_b/N_0(\text{dB}))/10}\right)^{1/2}$ for L_{ch}.

ii) For $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, n-1$, set $I_{av}(i, j) = 0$.

iii) For $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, n-1$, compute (53) if $b_{i,j} > 0$; else set $I_{ev}(i,j) = 0$. Taking the 3×4 protomatrix in (52) as an example, the weight of each row is $d = 6$ and hence $r + 2 = 6 \Rightarrow r = 4$. After analyzing the MI of the P-VNs, the corresponding 3×4 $\{I_{ev}(i,j)\}$ MI matrix can be represented by (54).

iv) Convert the $m \times n$ $\{I_{ev}(i,j)\}$ MI matrix into an $m \times d$ $\{I_{ah}(i,k)\}$ MI matrix by eliminating the 0 entries and repeating $\{I_{ev}(i,j)\}$ $b_{i,j}(\geqslant 1)$ times in the same row. Using the previous example, the 3×4 $\{I_{ev}(i,j)\}$ MI matrix is converted into the 3×6 $\{I_{ah}(i,k)\}$ MI matrix shown in (55).

v) For $i = 0, 1, \ldots, m-1$, using the d entries in the i-th row of I_{ah} and $\sigma^2_{L_{ch}}$, generate a large number of sets of LLR values as inputs to the symbol-MAP Hadamard decoder and record the output extrinsic LLR values of the k-th information bit ($k = 0, 1, \ldots, d-1$). Compute the extrinsic MI of the information bit using (56), where $p_e(\xi|X = x)$ denotes the PDF of the LLR values given the bit x being "0" or "1". Form the extrinsic MI matrix $\{I_{eh}(i,k)\}$ of size $m \times d$. (Details of the method is shown in Appendix C.) Using the previous example, the matrix is represented by (57).
Remark: Our technique makes use of multiple *a priori* MI values ($\{I_{ah}(i,k)\}$) as well as channel information $\sigma_{L_{ch}}$ and produces multiple extrinsic MI values ($\{I_{eh}(i,k)\}$). In [95], an EXIT function of symbol-MAP Hadamard decoder under the AWGN channel is obtained. However, the function involves very high computational complexity, which increases rapidly with an increase of the Hadamard order r. The function also cannot be used for analyzing non-systematic Hadamard codes. In [1], simulation is used to characterize the symbol-MAP Hadamard decoder but the method is based on a single *a priori* MI value as well as channel information and produces only one output extrinsic MI.

vi) Convert the $m \times d$ $\{I_{eh}(i,k)\}$ MI matrix into an $m \times n$ $\{I_{av}(i,j)\}$ MI matrix. For $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, n-1$; if $b_{i,j} > 0$, set the value of $I_{av}(i,j)$ as the average of the corresponding $b_{i,j}$ MI values in the i-th row of $\{I_{eh}(i,k)\}$; else set $I_{av}(i,j) = 0$. In the above example, $\{I_{av}(i,j)\}$ becomes (58).

vii) Repeat Steps iii) to vi) until the maximum number of iterations is reached; or when $I_{app}(j) = 1$ for all $j = 0, 1, \ldots, n-1$ where $I_{app}(j) = J\left(\sqrt{\sum_{i=0}^{m-1} b_{i,j}\left(J^{-1}\left(I_{av}(i,j)\right)\right)^2 + \sigma^2_{L_{ch}}}\right)$.

Note that our PEXIT algorithm can be used to analyze PLDPC-Hadamard designs with degree-1 and/or punctured VNs. In case of puncturing, the corresponding channel LLR values in the analysis will be set to zero.

3.3.2 *Optimization Criterion*

For a given code rate, our objective is to find a protograph of the PLDPC-Hadamard code such that it achieves $I_{app}(j) = 1 \ \forall\, j$ within a fixed number of iterations and with the lowest threshold E_b/N_0. To reduce the search space, we impose the following constraints:

- the weights of all rows in the protomatrix are fixed at d;
- the maximum column weight, the minimum column weight, and the maximum value of each entry in protomatrix are preset according to the code rate and order of the Hadamard code;
- the maximum number of iterations used in the PEXIT algorithm is set to 300; and
- a target threshold is set to below -1.40 dB.

Algorithm 1 shows the steps to find a protomatrix with a low threshold. A protomatrix is first randomly generated according to the constraints above [3]. Then it is iteratively analyzed by the PEXIT algorithm to see if the corresponding PEXIT curves converge under the current E_b/N_0 (dB). If the protomatrix is found satisfying $I_{app}(j) = 1$ for all j, E_b/N_0 (dB) is reduced by 0.01 dB and the protomatrix is analyzed again. If the number of iterations reaches 300 and the condition $I_{app}(j) = 1$ for all j is not satisfied, the analysis is terminated and the E_b/N_0 threshold is determined. The process is repeated until a protomatrix with a satisfactory E_b/N_0 threshold is found. (On average, the PEXIT algorithm takes 35s (for $r = 4$) to 120s (for $r = 10$) to determine the threshold of a protomatrix. For the case of $r = 4$, we generate 659 protomatrices to find the protomatrix presented in our thesis, which takes about 6.4 hours. Among the 659 protomatrices, 18 protomatrices have a threshold less than or equal to -1.40 dB. Using annealing approaches or genetic algorithms to generate the protomatrices would speed up the search and should be looked into in the future.)

3 We randomly generate each row, satisfying that each entry is less than or equal to maximum entry value and row weight equals d; while satisfying each column weight greater than or equal to minimum column weight, and less than or equal to the maximum column weight.

Algorithm 1: Searching $\boldsymbol{B}_{m \times n}$ with a low threshold

1 Generate a random protomatrix $\boldsymbol{B}_{m \times n}$ according to the corresponding constraints;
2 $E_b/N_0(\text{dB}) = -1.40$ dB;
3 **while** $E_b/N_0(\text{dB}) > -1.59$ dB **do**
4 $\sigma_{L_{ch}} = \sqrt{8 \cdot R \cdot 10^{(E_b/N_0(\text{dB}))/10}}$;
5 $I_{ch}(j) = J(\sigma_{L_{ch}})$ for $\forall$ j;
6 $I_{av}(i, j) = 0$ for $\forall$ i, j;
7 $It = 0$;
8 **while** $It < 300$ **do**
9   Use the proposed PEXIT algorithm to analyze $\boldsymbol{B}_{m \times n}$ and obtain $I_{app}(j)$ for $j = 0, 1, \ldots, n-1$;
10   **if** $I_{app}(j) = 1$ for $\forall$ j **then**
11    $E_b/N_0(\text{dB}) = E_b/N_0(\text{dB}) - 0.01$ dB; Goto line 3;
12   $It = It + 1$;
13 **Break**;
14 Threshold equals $E_b/N_0(\text{dB}) + 0.01$ dB.

3.4 SIMULATION RESULTS

In this section, we report our simulation results. Once a protomatrix with low threshold is found, we use a two-step lifting mechanism together with the PEG method [28] to construct an LDPC code. (See Appendix D for details of the lifting process.) Subsequently, each CN will be replaced by a Hadamard CN connected to an appropriate number of D1H-VNs. Without loss of generality, we transmit all-zero codewords. Moreover, the code bits are modulated using binary phase shift keying and sent through an AWGN channel. The maximum number of iterations performed by the decoder is 300. At a particular E_b/N_0, we run the simulation until 100 frame errors are collected. Then we record the corresponding bit error rate (BER), frame error rate (FER) and average number of iterations per decoded frame.

3.4.1 *Unpunctured PLDPC-Hadamard Codes*

3.4.1.1 $r = 4$ *and* $d = r + 2 = 6$

We attempt to find a PLDPC-Hadamard code with a target code rate of approximately 0.05. We substitute $R \approx 0.05$ and $d = 6$ into (43), and obtain $\frac{m}{n} \approx 0.63$. We therefore select a protomatrix $\boldsymbol{B}_{7 \times 11}$ of size 7×11, i.e., $m = 7$ and $n = 11$, and hence the code rate equals $R = 0.0494$. Moreover, we set the minimum column weight to 1, maximum column weight to 9, and maximum entry value to 3. The overall constraints of the protomatrix are listed as follows:

- size equals 7×11,

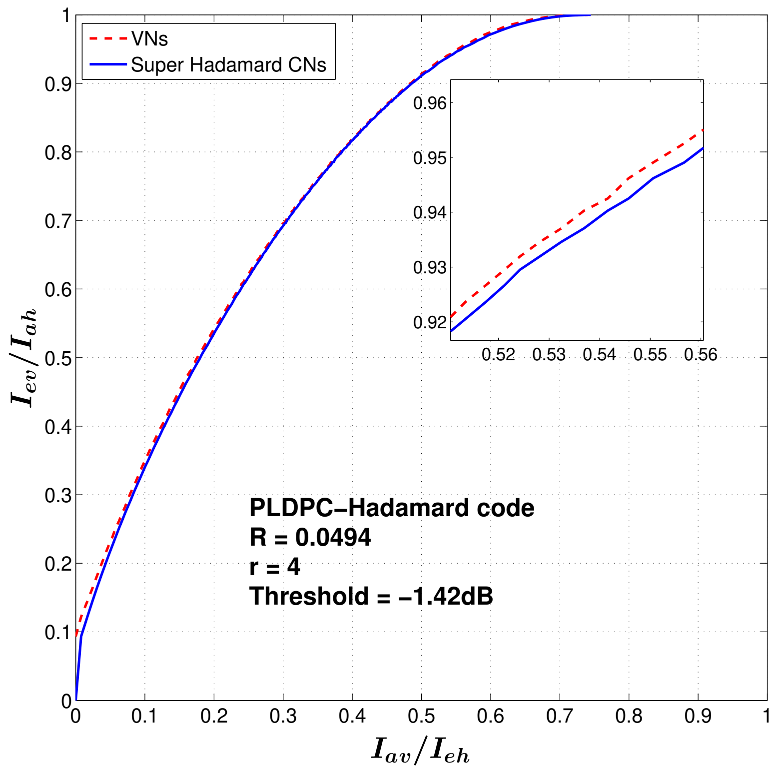

Figure 18: The PEXIT chart of the PLDPC-Hadamard code given in (59) with $R = 0.0494$ and $r = 4$.

- row weight equals $\sum_{j=0}^{10} b_{i,j} = d = 6$,
- minimum column weight equals $\sum_{i=0}^{6} b_{i,j} = 1$,
- maximum column weight equals $\sum_{i=0}^{6} b_{i,j} = 9$, and
- maximum entry value in $\boldsymbol{B}_{7\times 11}$ equals 3.

Using the proposed analytical method under the constraints above, we find the following protomatrix which has a theoretical threshold of -1.42 dB.

$$\boldsymbol{B}_{7\times 11} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \\ 2 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 \\ 3 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 2 & 0 \end{bmatrix} \quad (59)$$

Fig. 18 plots the PEXIT curves of the repeat decoder and the symbol-MAP Hadamard decoder under $E_b/N_0 = -1.42$ dB. It can be

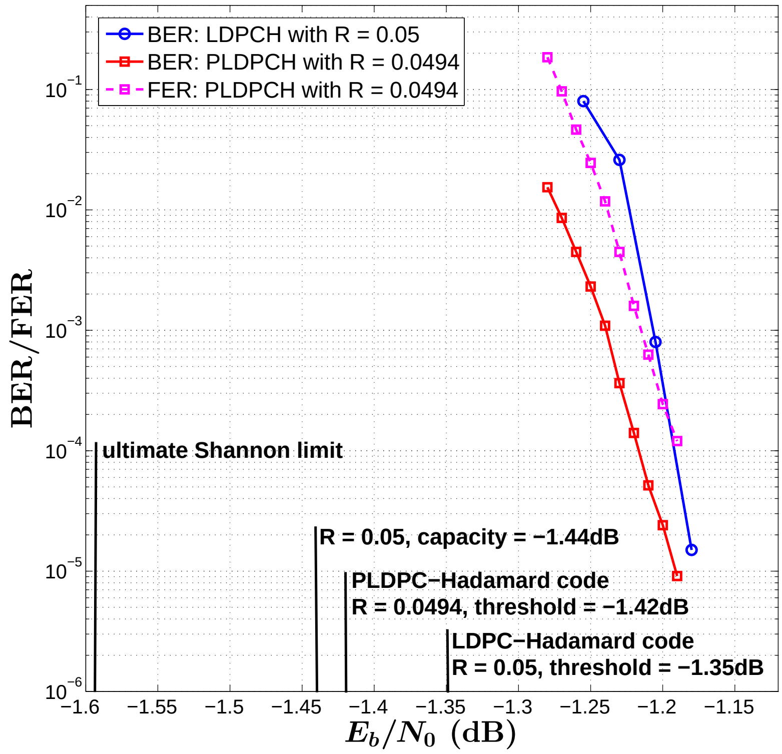

Figure 19: BER (red curve) and FER (pink curve) performance of the proposed PLDPC-Hadamard code compared with the BER of the LDPC-Hadamard code (blue curve) in [1]. $r = 4$ and $k = 65,536$.

observed that the two curves are matched. By lifting the protomatrix with factors of $z_1 = 32$ and $z_2 = 512$, we obtain a PLDPC-Hadamard code with information length $k = z_1 z_2(n - m) = 65,536$ and code length $N_{total} = z_1 z_2[m(2^{d-2} - d) + n] = 1,327,104$. (See Table 13 in Appendix D for details of the code structure after the lifting process.)

The BER and FER results of the PLDPC-Hadamard code found are plotted in Fig. 19. Our code achieves a BER of 10^{-5} at $E_b/N_0 = -1.19$ dB, which is 0.23 dB from the threshold. Table 1 lists the detailed results at -1.19 dB. A total of 832,056 frames need to be sent before 100 frame errors are collected. Hence a FER of 1.2×10^{-4} is achieved. Fig. 20 plots the average number of iterations versus E_b/N_0. Our code requires an average of 127 iterations for decoding at $E_b/N_0 = -1.19$ dB. At a BER of 10^{-5}, the gaps of our rate-0.0494 PLDPC-Hadamard code to the Shannon capacity for $R = 0.05$ and to the ultimate Shannon limit are 0.25 dB and 0.40 dB, respectively. The comparison of gaps is also listed in Table 2.

In Fig. 21, we further compare the BER results of the rate-0.05 LDPC-Hadamard code in [1] at $E_b/N_0 = -1.18$ dB and our rate-0.0494 PLDPC-Hadamard code at $E_b/N_0 = -1.18$ dB and -1.19 dB under different number of iterations. Note that the result of the LDPC-

Table 1: Detail results achieving a BER of 10^{-5} for $r = 4$ PLDPC-Hadamard code with rate-0.494 until 100 frame errors are reached.

E_b/N_0	-1.19 dB
No. of frame sent	$832,056$
FER	1.2×10^{-4}
BER	9.1×10^{-6}
Avg. no. of iterations	127

Hadamard code is the average from 20 simulations [1], whereas our result is the average from $10,000$ simulations. In other words, our simulation results are statistically very accurate due to the large number of simulations involved. For the same number of iterations at $E_b/N_0 = -1.18$ dB, our PLDPC-Hadamard code produces a lower BER compared with the LDPC-Hadamard code in [1]. When our proposed PLDPC-Hadamard code operates at a slightly lower E_b/N_0, i.e., -1.19 dB, the BER of the proposed code still outperforms the conventional code except for iteration numbers beyond 200. Thus, we conclude that the proposed code achieves a faster convergence rate compared with the conventional code. In particular, our results are more precise because $10,000$ simulations are used for our code compared with only 20 simulations used for the conventional code in [1].

Compared with the LDPC-Hadamard code in [1] which uses $R = 0.05$ and $r = 4$, our proposed PLDPC-Hadamard code has a slight performance improvement. The relatively advantage of our proposed PLDPC-Hadamard code over the LDPC-Hadamard code is probably due to degree-1 VNs in the protograph. Such degree-1 VNs are regarded as a kind of precoding structure which can increase the linear minimum distance [17, 33].

3.4.1.2 $r = 5$ *and* $d = 7$

We attempt to search a PLDPC-Hadamard code with a target code rate of approximately $R \approx 0.02$. Using (46), we obtain $m = 6$, $n = 10$ and $\frac{m}{n} \approx 0.61$. Hence the actual code rate is $R = 0.021$. The constraints of the protomatrix are as follows:

- size equals 6×10,
- row weight equals $\sum_{j=0}^{9} b_{i,j} = d = 7$,
- minimum column weight equals $\sum_{i=0}^{5} b_{i,j} = 1$,
- maximum column weight equals $\sum_{i=0}^{5} b_{i,j} = 9$,

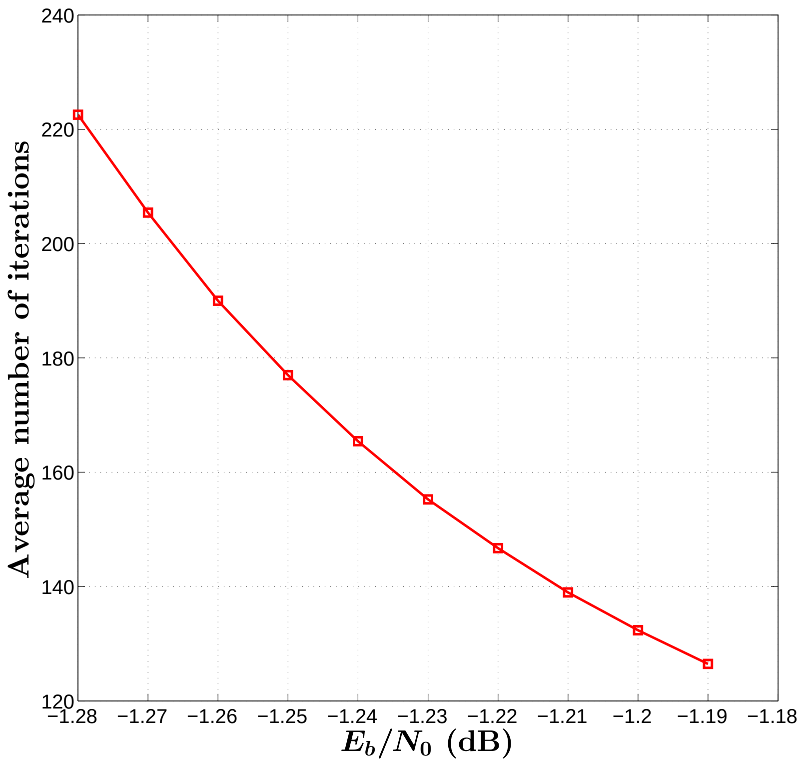

Figure 20: Average number of iterations required to decode the PLDPC-Hadamard code versus E_b/N_0 with $r = 4$ and $k = 65,536$.

- maximum entry value in $\boldsymbol{B}_{6\times10}$ equals 3.

The following protomatrix with a threshold of -1.51 dB is found.

$$\boldsymbol{B}_{6\times10}=\begin{bmatrix} 3 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 2 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 1 & 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 3 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & 2 & 0 & 2 \\ 2 & 0 & 1 & 1 & 0 & 0 & 0 & 2 & 0 & 1 \end{bmatrix}. \tag{60}$$

The same lifting factors $z_1 = 32$ and $z_2 = 512$ are used to expand $\boldsymbol{B}_{6\times10}$. The rate-0.021 PLDPC-Hadamard code has an information length of $k = z_1z_2(n-m) = 65,536$ and a code length of $N_{total} = z_1z_2[m(2^{d-2}-2)+n] = 3,112,960$. Fig. 22 shows the PEXIT chart of the code at $E_b/N_0 = -1.51$ dB. We can observe that the two curves do not crossed and are matched.

Fig. 23 plots the BER and FER performance of the PLDPC-Hadamard code. The code achieves a BER of 1.4×10^{-5} and a FER of 1.3×10^{-4} at $E_b/N_0 = -1.24$ dB (red curve), which is 0.27 dB away

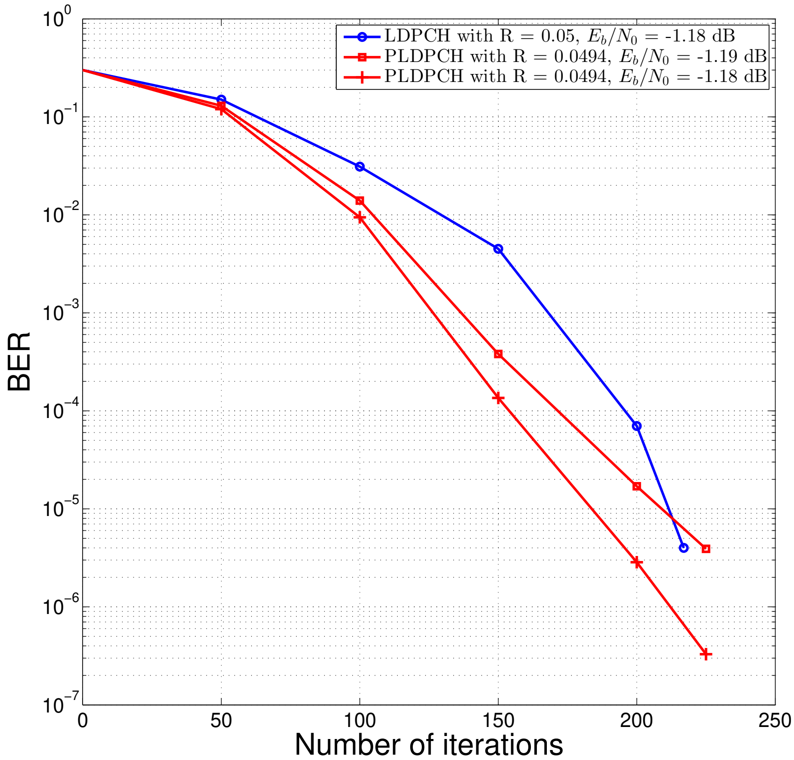

Figure 21: BER performance versus number of iterations for the LDPC-Hadamard code in [1] ($E_b/N_0 = -1.18$ dB) and PLDPC-Hadamard code ($E_b/N_0 = -1.18$ and -1.19 dB). $r = 4$ and $k = 65,536$.

from the designed threshold. Compared with the BER curve (blue curve) of the rate-0.022 LDPC-Hadamard code in [1], our PLDPC-Hadamard code can achieve comparable results. Table 3 shows that at $E_b/N_0 = -1.24$ dB, $780,660$ frames have been decoded with an average of 119 decoding iterations per frame. Fig. 24 plots the average number of iterations for the code at different E_b/N_0 values. At a BER of 10^{-5}, the gaps to the Shannon capacity of $R = 0.020$ and to the ultimate Shannon limit are 0.29 dB and 0.35 dB, respectively, which are listed in Table 4.

3.4.1.3 $r = 8$ *and* $d = 10$

A rate-0.008 PLDPC-Hadamard code is constructed using $m = 5$ and $n = 15$. The constraints of the protomatrix are as follows:

- size equals 5×15,
- row weight equals $\sum_{j=0}^{14} b_{i,j} = d = 10$,
- minimum column weight equals $\sum_{i=0}^{4} b_{i,j} = 1$,

Table 2: Gaps to theoretical threshold, rate-0.05 Shannon limit and ultimate Shannon limit for $r = 4$ LDPC-Hadamard and PLDPC-Hadamard codes at a BER of 10^{-5}.

Type of Code	[1] Rate-0.05 LDPCH code	Rate-0.0494 PLDPCH code
Theoretical threshold	-1.35 dB by EXIT chart	-1.42 dB by PEXIT chart
E_b/N_0 at a BER of 10^{-5}	-1.18 dB	-1.19 dB
Gap to theoretical threshold	0.17 dB	0.23 dB
Gap to rate-0.05 Shannon limit (-1.44 dB)	0.26 dB	0.25 dB
Gap to ultimate Shannon limit (-1.59 dB)	0.41 dB	0.40 dB

- maximum column weight equals $\sum_{i=0}^{4} b_{i,j} = 11$,
- maximum entry value in $\boldsymbol{B}_{5\times15}$ equals 3.

Compared with the constraints for low-order ($r = 4$ or 5) PLDPC-Hadamard protomatrices, the maximum column weight is increased to 11. Based on these constraints and using our proposed analytical method, the following protomatrix is found with a threshold of -1.53 dB.

$$\boldsymbol{B}_{5\times15} = \begin{bmatrix} 2 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 2 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 3 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 0 & 0 & 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 2 & 3 & 0 & 0 \end{bmatrix} \tag{61}$$

We use lifting factors of $z_1 = 16$ and $z_2 = 1280$. The rate-0.008 PLDPC-Hadamard code thus has an information length of $k = 204,800$ and a code length of $N_{total} = 25,497,600$. This code has the same theoretical threshold as the rate-0.008 LDPC-Hadamard code in [1]. The FER and BER curves of our code and the BER of the code in [1] are plotted in Fig. 25. At $E_b/N_0 = -1.35$ dB, the PLDPC-Hadamard code achieves a FER of 2.1×10^{-4}, and a BER of 3.8×10^{-6}

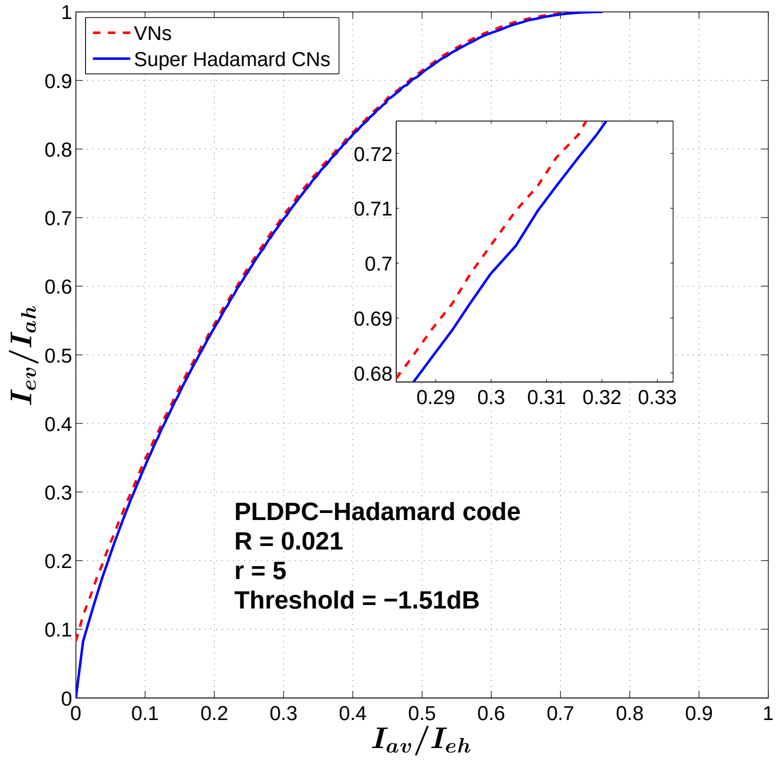

Figure 22: The PEXIT chart of the PLDPC-Hadamard code given in (60) with $R = 0.021$ and $r = 5$.

which is 0.18 dB away from the designed threshold. Compared with the BER curve of [1], there is a performance gap of about 0.03 dB at a BER of 10^{-5}. At the same BER, the gaps of our code to the rate-0.008 Shannon limit and to the ultimate Shannon limit are 0.22 dB and 0.24 dB, respectively, as shown in Table 5. The convergence rate of the code can be found in Fig. 26 and the average number of iterations is 139 at $E_b/N_0 = -1.35$ dB.

3.4.1.4 $r = 10$ *and* $d = 12$

A rate-0.00295 PLDPC-Hadamard code is constructed using $m = 6$ and $n = 24$. (The target rate is approximately 0.003.) The constraints of the protomatrix are as follows:

- size equals 6×24,
- row weight equals $\sum_{j=0}^{23} b_{i,j} = d = 12$,
- minimum column weight equals $\sum_{i=0}^{5} b_{i,j} = 1$,
- maximum column weight equals $\sum_{i=0}^{5} b_{i,j} = 11$,
- maximum entry value in $\boldsymbol{B}_{6\times 24}$ equals 4.

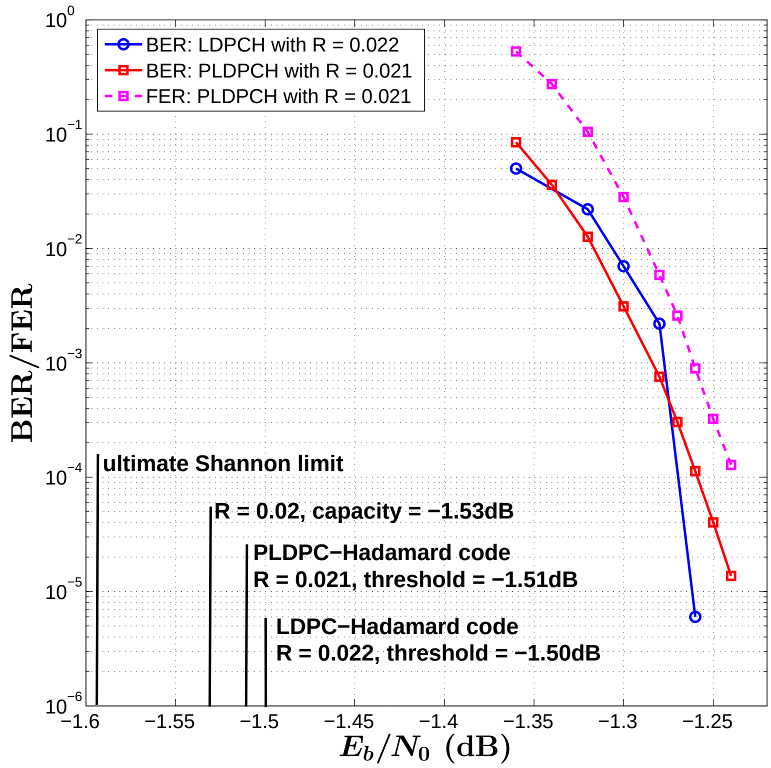

Figure 23: BER (red curve) and FER (pink curve) performance of the proposed PLDPC-Hadamard code compared with the BER of the LDPC-Hadamard code (blue curve) in [1]. $r = 5$ and $k = 65,536$.

$$
\boldsymbol{B}_{6\times 24} =
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 4 & 0 & 1 & 0 \\
0 & 0 & 0 & 3 & 2 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3 & 0 & 4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 1 \\
2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 3 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 3 & 2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\tag{62}
$$

Note that the maximum value of the entries in $\boldsymbol{B}_{6\times 24}$ is increased to 4. For the rate-0.00295 PLDPC-Hadamard code, the protomatrix in (62) with a theoretical threshold of -1.53 dB is found by the proposed analytical method. The threshold is slightly higher (0.02 dB) than that of the LDPC-Hadamard code in [1]. We use lifting factors of $z_1 = 20$ and $z_2 = 1280$. The information length equals $k = 460,800$ and the code length equals $N_{total} = 156,057,600$.

Fig. 27 plots the error performance of our constructed code and that in [1]. Our PLDPC-Hadamard code achieves a BER of 2.8×10^{-6} at

Table 3: Detail results achieving a BER of 10^{-5} for $r = 5$ PLDPC-Hadamard code until 100 frame errors are reached.

E_b/N_0	-1.24 dB
No. of frame sent	$780,660$
FER	1.3×10^{-4}
BER	1.4×10^{-5}
Avg. no. of iterations	119

$E_b/N_0 = -1.43$ dB, which is 0.01 dB higher compared with the LDPC-Hadamard code in [1]. However, our code is 29.11% shorter compared with the code in [1]. This performance has 0.10 dB gap from the designed threshold. The gaps of our code to the rate-0.003 Shannon limit and to the ultimate Shannon limit are 0.15 dB and 0.16 dB, respectively. Table 6 lists the detailed comparison. The convergence rate of the code can be found in Fig. 28 and the average number of iterations is 202 at $E_b/N_0 = -1.43$ dB.

Remark: For cases with $r = 5, 8$ and 10 (Figs. 23, 25 and 27), the BER results may appear that our proposed PLDPC-Hadamard codes are slightly outperformed by the LDPC-Hadamard codes in [1] at the high E_b/N_0 region. For our codes, we keep running the simulations until 100 block errors are recorded. Thus our reported results have a high degree of accuracy. However, the stopping criterion of the LDPC-Hadamard code simulation in [1] is not known. If an inadequate number of simulations are performed, there could be some statistical difference between the actual error performance and the reported results.

3.4.2 *Punctured PLDPC-Hadamard Codes*

When a code is punctured, the code rate increases. The signals corresponding to the punctured variable nodes are not sent to the receiver and hence their channel LLR values are initialized to zero. In this section, we evaluate the performance of the PLDPC-Hadamard codes designed in the previous section when the codes are punctured. (Note that our proposed PEXIT chart method can be used to design good punctured PLDPC-Hadamard codes.)

We use α to denote a column number in a protomatrix and β to denote the weight of a column. For example in the protomatrix shown in (59), $[4, 6]$ refers to the 4-th column $[0\ 0\ 0\ 3\ 0\ 2\ 1]^T$ which has a column weight of 6. Thus we use "punctured $[\alpha, \beta]$" to denote a PLDPC-Hadamard code in which the P-VN corresponding to the α-th column in the protomatrix is punctured. Moreover, the punctured P-VN has a degree of β.

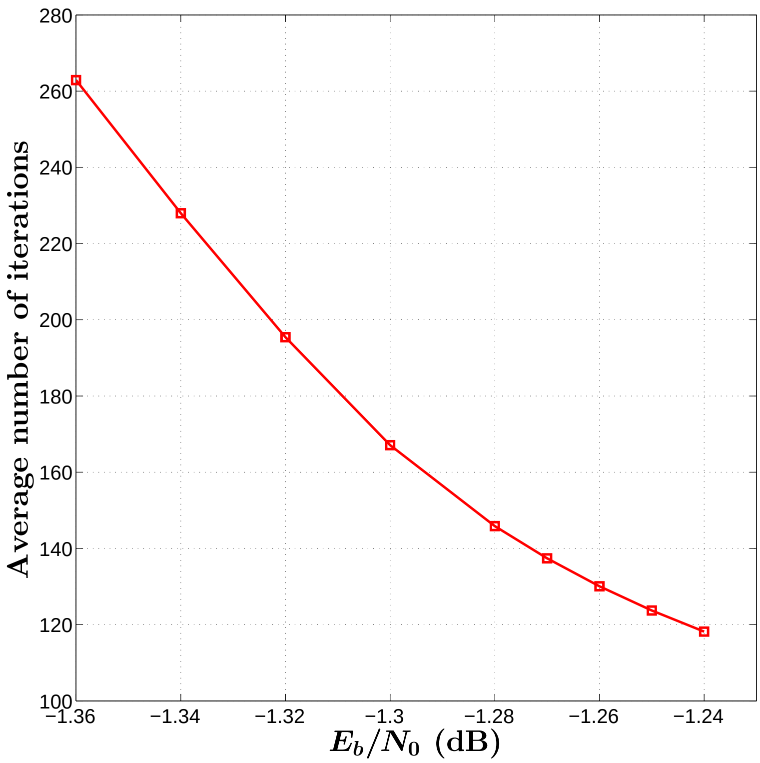

Figure 24: Average number of iterations required to decode the PLDPC-Hadamard code with $r = 5$ and $k = 65,536$.

3.4.2.1 $r = 4$

We first consider the rate-0.0494 PLDPC-Hadamard code shown in (59) and puncture one P-VN with the largest degree (i.e., 9) or lowest degree (i.e., 1). Four cases are therefore considered, i.e., $[1,9]$, $[10,9]$, $[6,1]$ and $[8,1]$. After puncturing, all codes have a rate of 0.0500 (by applying (44)). Fig. 29 shows that at a BER of 10^{-4}, punctured $[10,9]$, $[1,9]$, $[6,1]$ and $[8,1]$ have performance losses of about 0.075 dB, 0.065 dB, 0.012 dB and 0.004 dB, respectively, compared with the unpunctured code. Fig. 30 plots the FER of the unpunctured/punctured codes and it shows a similar relative error performance. We also simulate the code when both $[6,1]$ and $[8,1]$ P-VNs are punctured. The code rate is further increased to 0.0506. The error performance of the code, as shown in Figs. 29 and Fig. 30, is found to be between punctured $[6,1]$ and $[8,1]$.

Fig. 31 plots the average number of iterations required to decode a codeword at different E_b/N_0. Punctured $[8,1]$ has the fastest convergence speed compared with other punctured codes and has almost the same convergence speed as the unpunctured code. Table 7 lists (i) the number of frame sent, (ii) BER, (iii) FER and (iv) average number of iterations until 100 frame errors are reached, at $E_b/N_0 =$

Table 4: Gaps to theoretical threshold, rate-0.02 Shannon limit and ultimate Shannon limit for $r = 5$ LDPC-Hadamard and PLDPC-Hadamard codes at a BER of 10^{-5}.

Type of Code	[1] Rate-0.022 LDPCH code	Rate-0.021 PLDPCH code
Theoretical Threshold	-1.50 dB by EXIT chart	-1.51 dB by PEXIT chart
E_b/N_0 at a BER of 10^{-5}	-1.26 dB	-1.24 dB
Gap to theoretical Threshold	0.24 dB	0.27 dB
Gap to rate-0.020 Shannon limit (-1.53 dB)	0.27 dB	0.29 dB
Gap to ultimate Shannon limit (-1.59 dB)	0.33 dB	0.35 dB

-1.19 dB for the unpunctured/punctured PLDPC-Hadamard codes with $r = 4$. The aforementioned results conclude that punctured $[8, 1]$ outperforms other punctured codes being considered and has a very similar performance as the unpunctured code.

3.4.2.2 $r = 5$

We consider the rate-0.021 PLDPC-Hadamard code shown in (60); and puncture $[8, 9]$ (largest degree) and $[9, 1]$ (lowest degree), respectively. After puncturing, both codes have a rate of 0.02116 (by applying (47)). Fig. 32 shows that punctured $[9, 1]$ achieves the lowest BER while punctured $[8, 9]$ achieves the lowest FER. Fig. 33 plots the average number of decoding iterations. The results indicate that punctured $[8, 9]$ converges faster than the unpunctured code, which in turn converges faster than punctured $[9, 1]$.

We further consider puncturing D1H-VNs corresponding to code bits $c^H_{2^{k-1}}$ $(k = 1, 2, \dots, r)$ for every H-CN. The rate of such punctured codes is computed using (48). We use $[c^H_1\ c^H_2\ \cdots c^H_{2^{k-1}}]$ $(1 \leqslant k \leqslant r)$ to denote the set of bits being punctured. Three sets of punctured bits are being considered. They are $[c^H_8\ c^H_{16}]$, $[c^H_2\ c^H_4\ c^H_8\ c^H_{16}]$ and $[c^H_1\ c^H_2\ c^H_4\ c^H_8\ c^H_{16}]$; and their corresponding rates are 0.022, 0.024 and 0.025, respectively. Fig. 34 shows that in terms of BER and FER, all the punctured codes are degraded compared with the unpunctured rate-0.022 PLDPC Hadamard code. Particularly compared with the

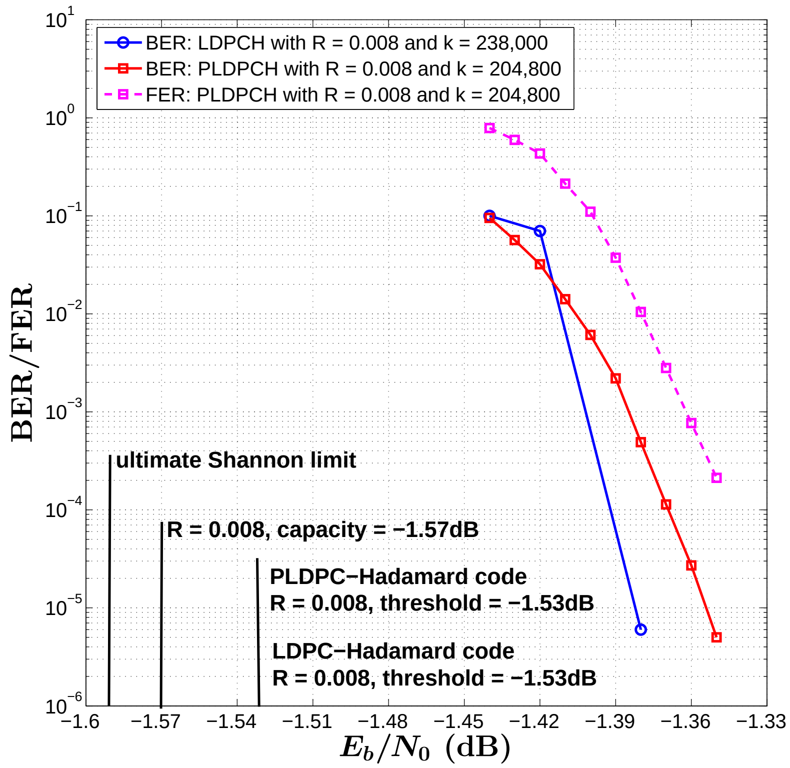

Figure 25: BER (red curve) and FER (pink curve) performance of the proposed PLDPC-Hadamard code compared with the BER of the LDPC-Hadamard code (blue curve) in [1]. $r = 8$.

unpunctured code, punctured $[c_8^H\ c_{16}^H]$ has a 0.02 dB performance loss at a BER of 3.6×10^{-5}; punctured $[c_2^H\ c_4^H\ c_8^H\ c_{16}^H]$ has a 0.03 dB performance loss at a BER of 4.7×10^{-5}; and punctured $[c_1^H\ c_2^H\ c_4^H\ c_8^H\ c_{16}^H]$ has a 0.04 dB performance loss at a BER of 1.4×10^{-5}. The BER/FER results indicate that the channel observations corresponding to these D1H-VNs provide very useful information for the decoder to decode successfully. Fig. 35 plots the average number of decoding iterations. It shows that the unpunctured code requires the lowest number of decoding iterations.

3.4.2.3 $r = 8$

We consider the rate-0.008 PLDPC-Hadamard code shown in (61); and puncture [12, 11] (largest degree) and [2, 2] (lowest degree), respectively. The code rate is increased slightly from 0.008032 to 0.008038. Figs. 36 and 37 show that compared with the unpunctured code, the punctured ones are degraded only very slightly in terms of BER/FER and have almost the same convergence rates.

Table 5: Gaps to theoretical threshold, rate-0.008 Shannon limit and ultimate Shannon limit for $r = 8$ LDPC-Hadamard and PLDPC-Hadamard codes at a BER of 10^{-5}.

Type of Code	[1] Rate-0.008 LDPCH code	Rate-0.008 PLDPCH code
Theoretical Threshold	-1.53 dB by EXIT chart	-1.53 dB by PEXIT chart
E_b/N_0 at a BER of 10^{-5}	-1.38 dB	-1.35 dB
Gap to theoretical Threshold	0.15 dB	0.18 dB
Gap to rate-0.008 Shannon limit (-1.57 dB)	0.19 dB	0.22 dB
Gap to ultimate Shannon limit (-1.59 dB)	0.21 dB	0.24 dB

3.4.2.4 $r = 10$

We consider the rate-0.002950 PLDPC-Hadamard code shown in (62); and puncture $[21, 11]$ (largest degree) and $[3, 2]$ (lowest degree), respectively. The code rate is increased slightly from 0.002950 to 0.002953. Figs. 38 and 39 show that compared with the unpunctured code, the punctured ones have almost the same performance in terms of BER/FER and convergence rate.

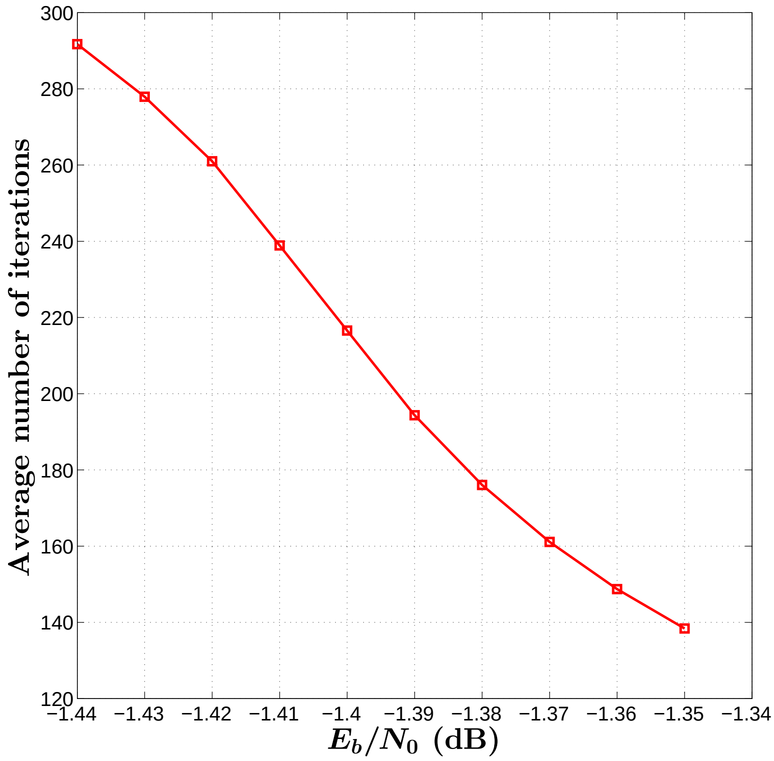

Figure 26: Average number of iterations required to decode the PLDPC-Hadamard code with $r = 8$ and $k = 204,800$.

Table 6: Gaps to theoretical threshold, rate-0.003 Shannon limit and ultimate Shannon limit for $r = 10$ LDPC-Hadamard and PLDPC-Hadamard codes at a BER of 10^{-5}.

Type of Code	[1] Rate-0.003 LDPCH code	Rate-0.00295 PLDPCH code
Theoretical Threshold	−1.55 dB by EXIT chart	−1.53 dB by PEXIT chart
E_b/N_0 at a BER of 10^{-5}	−1.44 dB	−1.43 dB
Gap to theoretical Threshold	0.11 dB	0.10 dB
Gap to rate-0.003 Shannon limit (−1.58 dB)	0.14 dB	0.15 dB
Gap to ultimate Shannon limit (−1.59 dB)	0.15 dB	0.16 dB

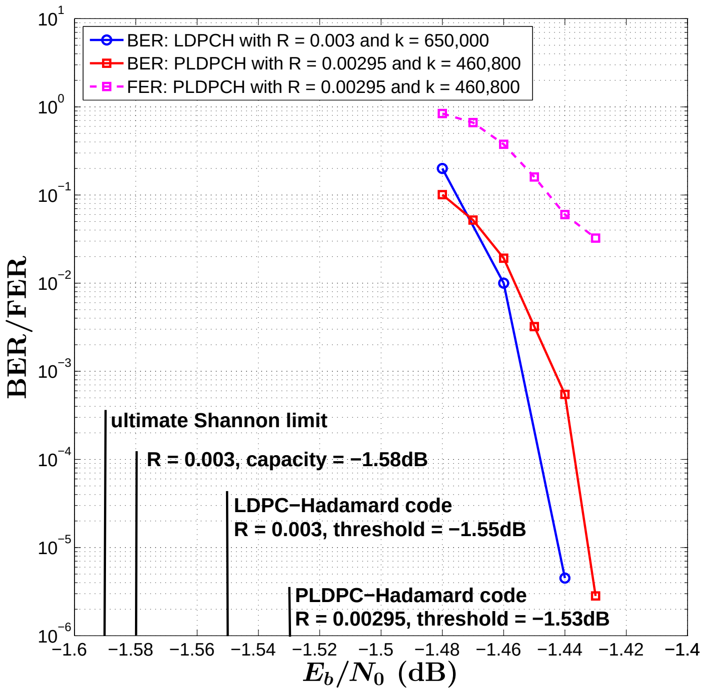

Figure 27: BER (red curve) and FER (pink curve) performance of the proposed PLDPC-Hadamard code compared with the BER of the LDPC-Hadamard code (blue curve) in [1]. $r = 10$.

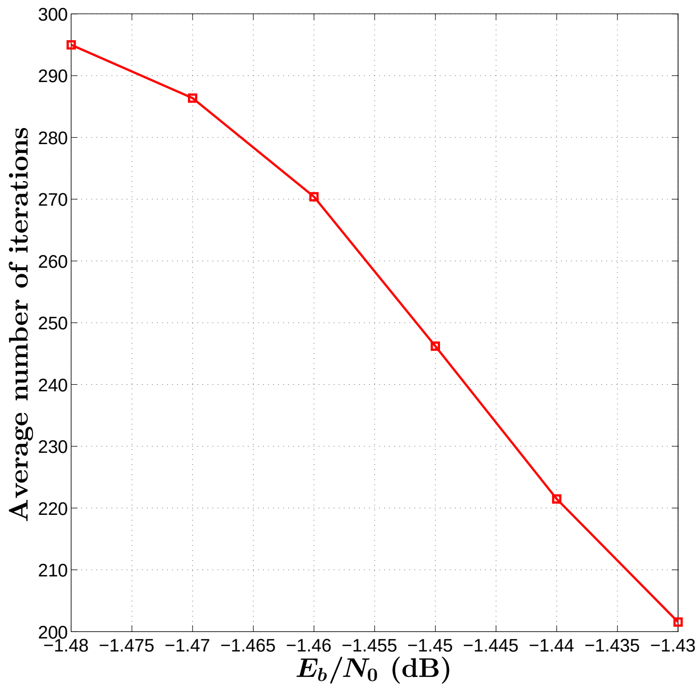

Figure 28: Average number of iterations required to decode the PLDPC-Hadamard code with $r = 10$ and $k = 460,800$.

Table 7: Performance of unpunctured/punctured PLDPC-Hadamard codes with $r = 4$ at $E_b/N_0 = -1.19$ dB.

Punctured P-VN(s)	Unpunctured	$[1, 9]$	$[10, 9]$	$[6, 1]$	$[8, 1]$	$[6, 1]\&[8, 1]$
Code rate	0.0494	0.0500	0.0500	0.0500	0.0500	0.0506
FER	1.2×10^{-4}	1.1×10^{-2}	1.4×10^{-2}	1.7×10^{-3}	1.4×10^{-4}	8.0×10^{-4}
BER	9.1×10^{-6}	2.8×10^{-3}	3.9×10^{-3}	4.1×10^{-5}	1.2×10^{-5}	1.7×10^{-5}
No. of frames sent	832,056	9,314	6,962	59,564	727,997	124,672
Avg. no. of iterations	127	135	137	138	127	135

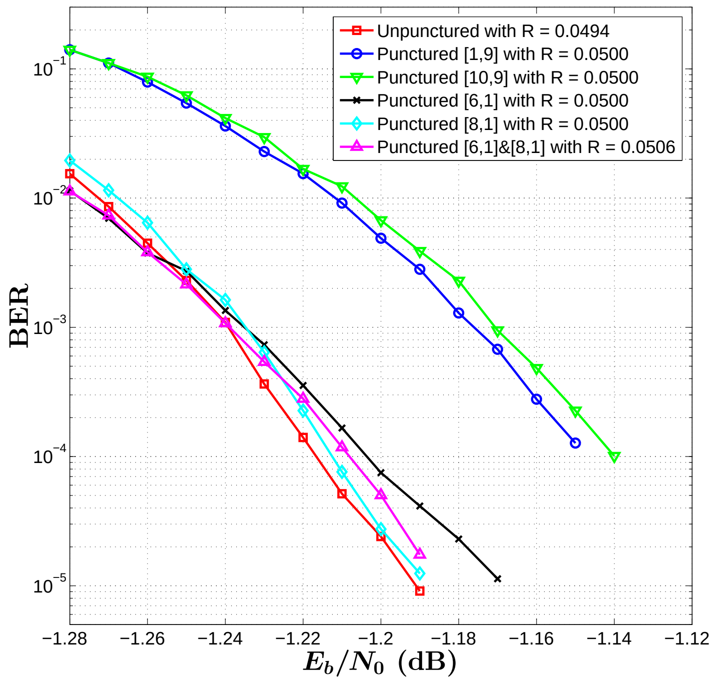

Figure 29: BER performance of unpunctured/punctured PLDPC-Hadamard codes. One or two P-VNs is/are punctured. $r = 4$ and $k = 65,536$.

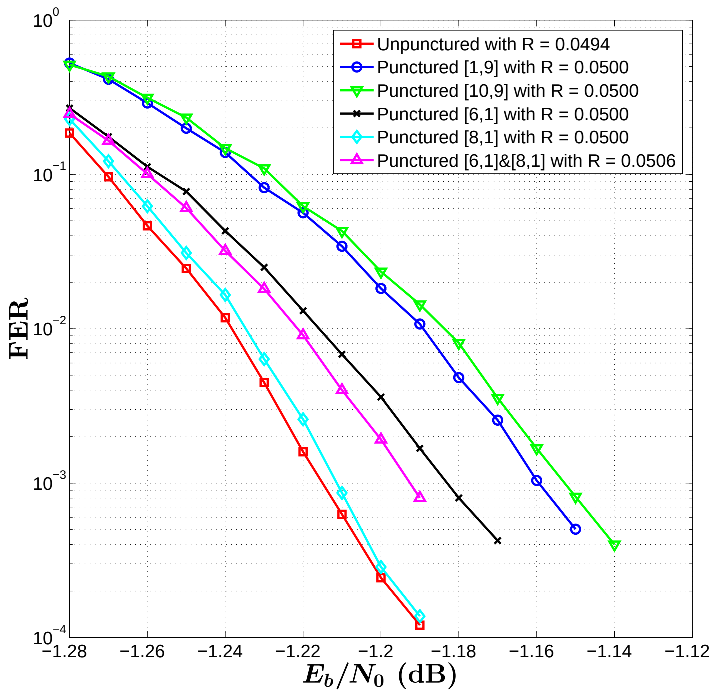

Figure 30: FER performance of unpunctured/punctured PLDPC-Hadamard codes. One or two P-VNs is/are punctured. $r = 4$ and $k = 65,536$.

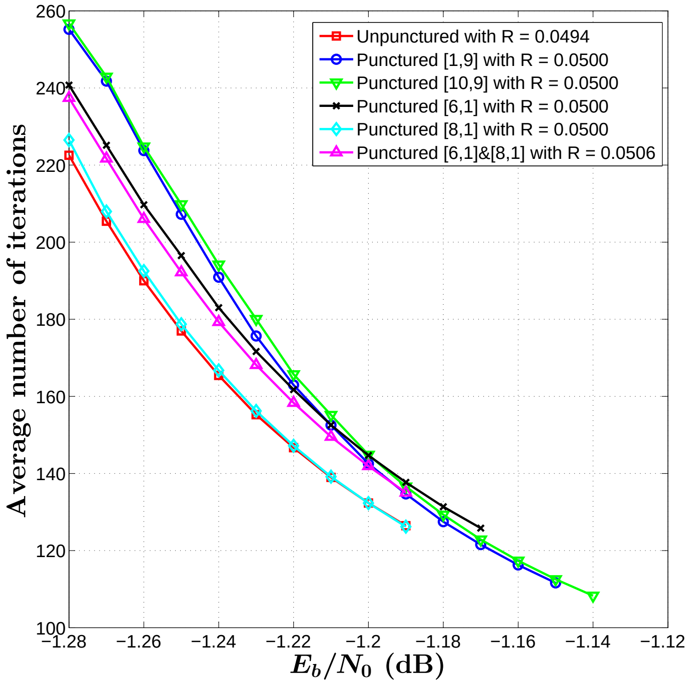

Figure 31: Average number of iterations required to decode unpunctured/punctured PLDPC-Hadamard codes. One or two P-VNs is/are punctured. $r = 4$ and $k = 65,536$.

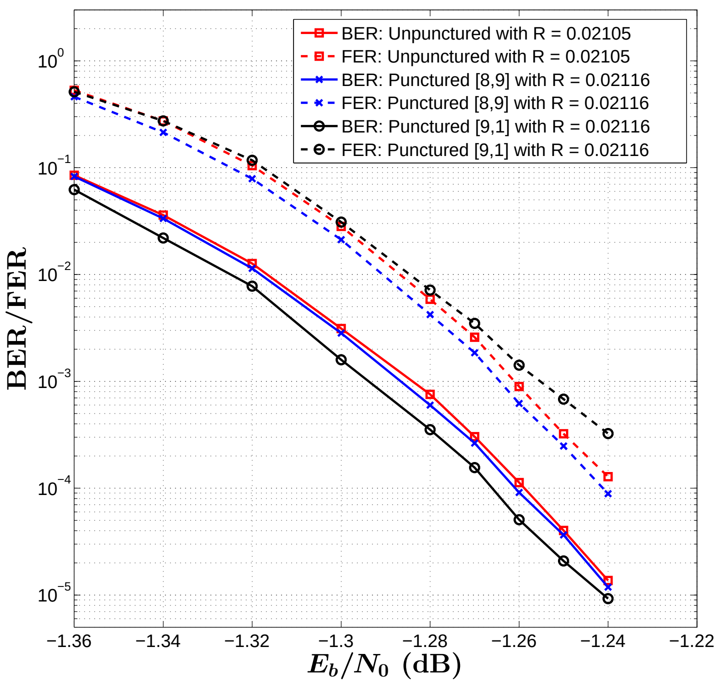

Figure 32: BER/FER performance of unpunctured/punctured PLDPC-Hadamard codes. One P-VN is punctured. $r = 5$ and $k = 65,536$.

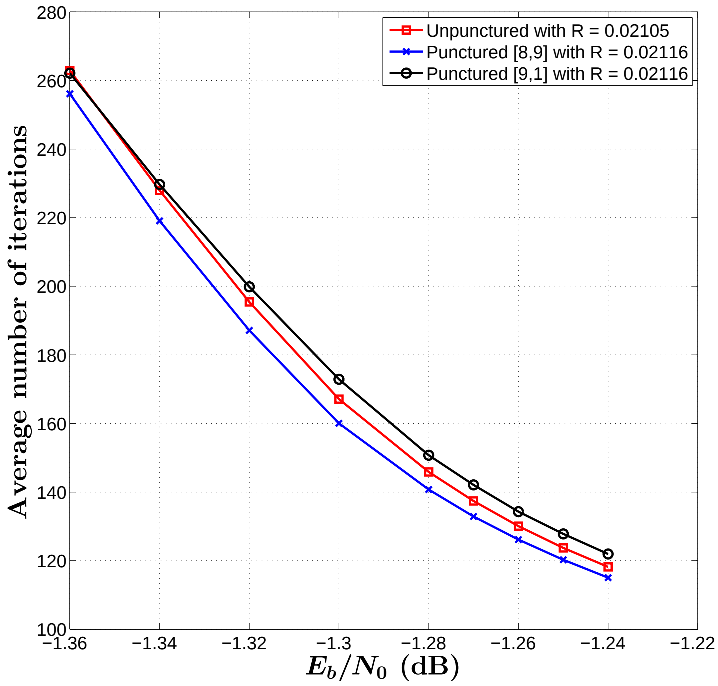

Figure 33: Average number of iterations required to decode unpunctured/punctured PLDPC-Hadamard codes. One P-VN is punctured. $r = 5$ and $k = 65,536$.

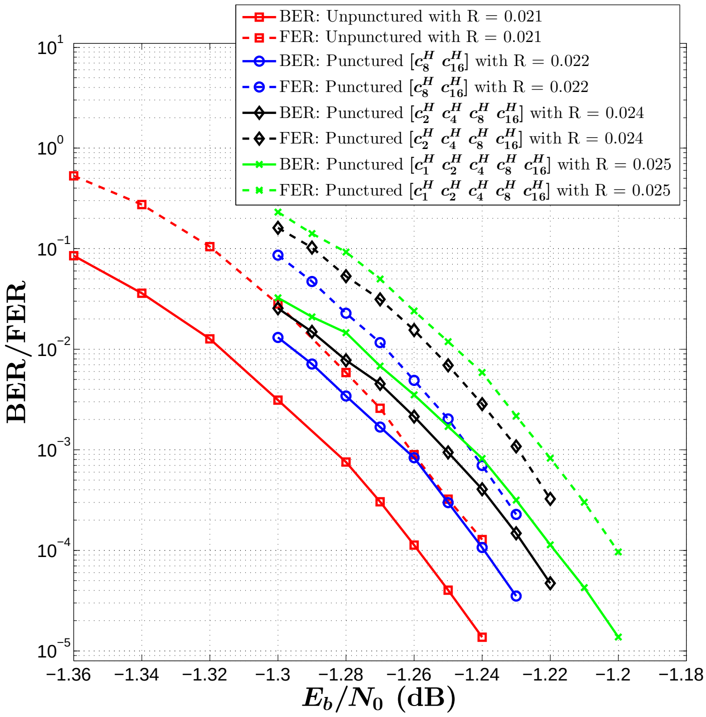

Figure 34: BER/FER performance of unpunctured/punctured PLDPC-Hadamard codes. Two, four and five D1H-VNs are punctured. $r = 5$ and $k = 65,536$.

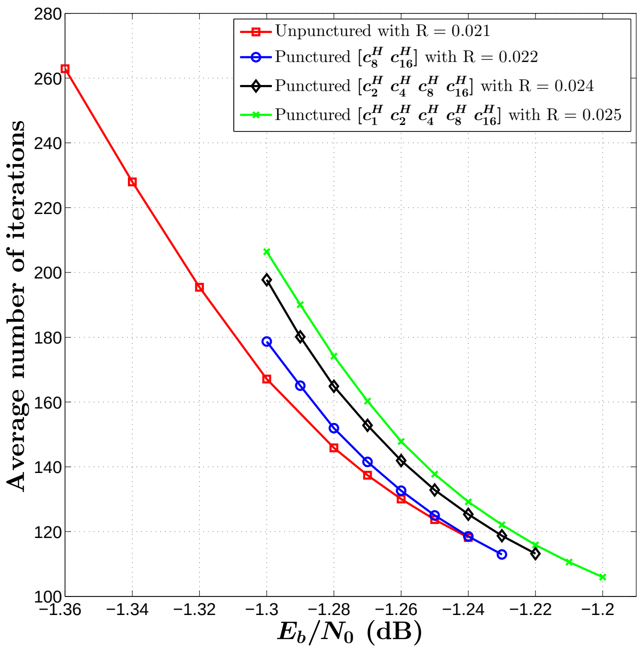

Figure 35: Average number of iterations required to decode unpunctured/punctured PLDPC-Hadamard codes. Two, four and five D1H-VNs are punctured. $r = 5$ and $k = 65,536$.

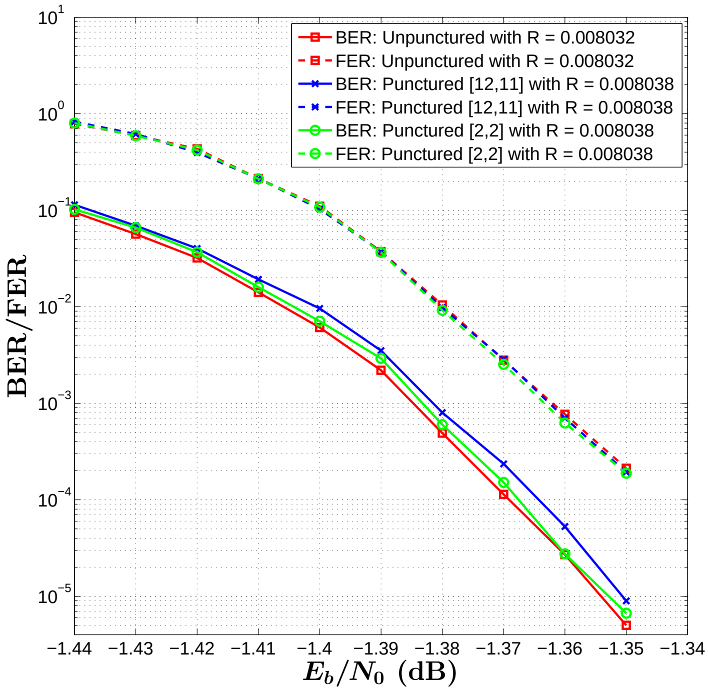

Figure 36: BER/FER performance of unpunctured/punctured PLDPC-Hadamard codes. One P-VN is punctured. $r = 8$ and $k = 204,800$.

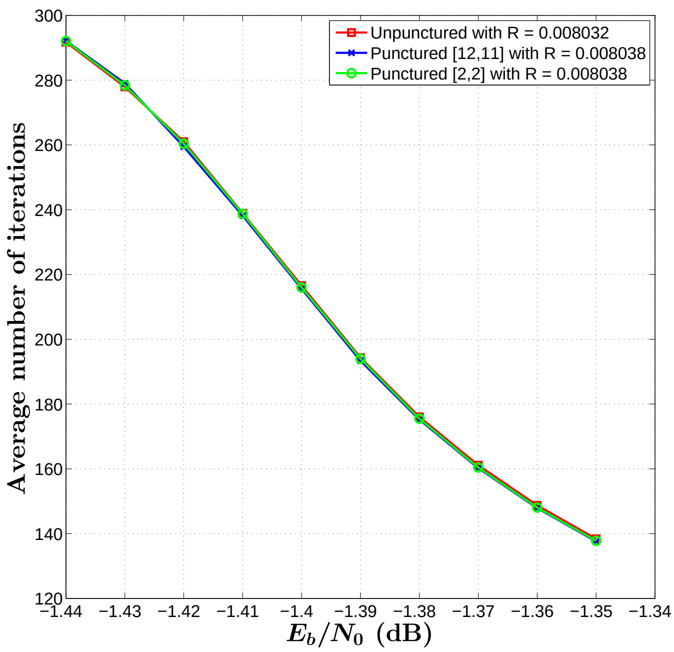

Figure 37: Average number of iterations required to decode unpunctured/punctured PLDPC-Hadamard codes. One P-VN is punctured. $r = 8$ and $k = 204,800$.

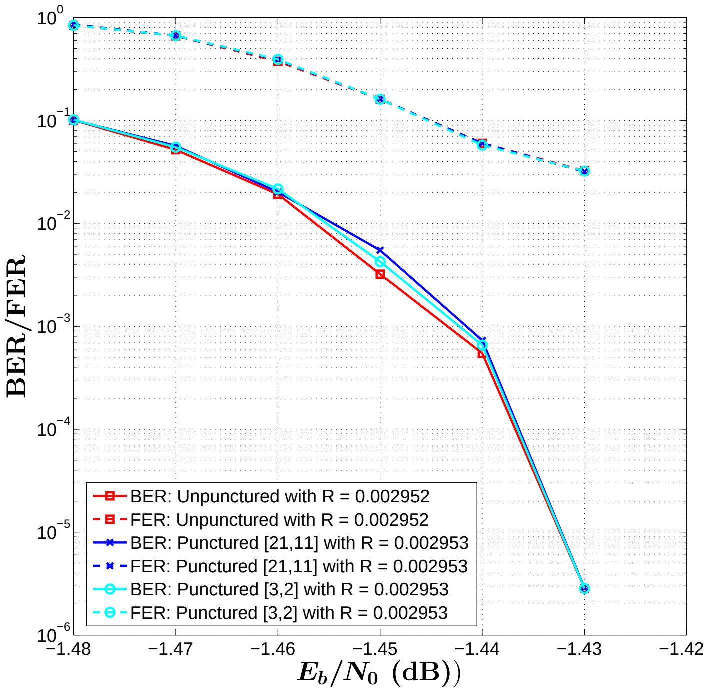

Figure 38: BER/FER performance of unpunctured/punctured PLDPC-Hadamard codes. One P-VN is punctured. $r = 10$ and $k = 460,800$.

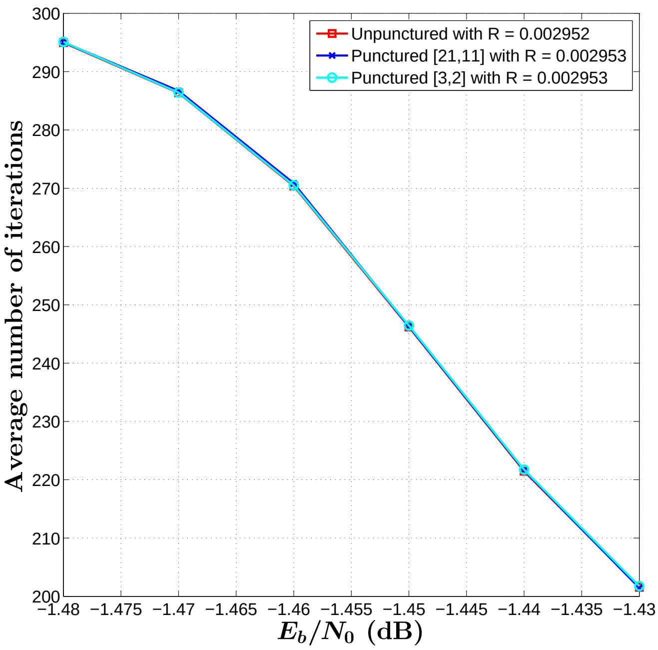

Figure 39: Average number of iterations required to decode unpunctured/punctured PLDPC-Hadamard codes. One P-VN is punctured. $r = 10$ and $k = 460,800$.

3.5 summary

In this chapter, we have proposed an alternate method of designing ultimate-Shannon-limit-approaching LDPC-Hadamard codes —- protograph-based LDPC-Hadamard (PLDPC-Hadamard) codes. By appending degree-1 Hadamard variable nodes (D1H-VNs) to the protograph of LDPC codes, a generalized protograph can be formed to characterize the structure of PLDPC-Hadamard codes. We have also proposed a low-complexity PEXIT algorithm to analyze the threshold of the codes, which is valid for PLDPC-Hadamard protographs with degree-1 variable nodes and/or punctured variable nodes/D1H-VNs. Based on the proposed analysis method, we have found good PLDPC-Hadamard codes with different code rates and have provided the corresponding protomatrices with very low thresholds (< -1.40 dB).

Reliable BER, FER and average number of decoding iterations are derived by running simulations until 100 frame errors are obtained. At a BER of 10^{-5}, the gaps of our codes to the ultimate-Shannon-limit range from 0.40 dB (for rate = 0.0494) to 0.16 dB (for rate = 0.003). Moreover, the error performance of our codes is comparable to that of the traditional LDPC-Hadamard codes. We have also investigated punctured PLDPC-Hadamard codes. When the order of the Hadamard code $r = 4$, puncturing different variable nodes in the protograph produces quite different BER/FER performance degradations compared with the unpunctured code. When $r = 5$, puncturing one VN can actually improve the BER/FER performance slightly. When $r = 8$ or 10, puncturing one VN does not seem to have any effect. Moreover, we conclude that when $r = 5$, puncturing the extra D1H-VNs provided by the non-systematic Hadamard code degrades the error performance quite significantly.

In the next chapter, we look into the decoding algorithms of PLDPC-Hadamard codes. We will propose a layered decoder, simulate its decoding performance, and design its hardware architecture.

Chapter 4

Layered Decoder for PLDPC-Hadamard Block Codes

In this chapter, we first review the standard decoding algorithm for PLDPC-Hadamard block codes (PLDPCH-BCs). To speed up the decoding convergence rate, we propose a layered decoding algorithm for PLDPCH-BCs [96]. We also compare the complexity between the standard and layered decoding algorithms. Finally, we design and implement the layered decoder onto an FPGA board [97].

4.1 Standard Decoding Algorithm

The receiver obtains the channel log-likelihood-ratio (LLR) values of the P-VNs and D1H-VNs, based on which the transmitted PLDPCH-BC is decoded. We denote

- $L_{ch}^{PVN}(\beta)$ as the channel LLR value of the β-th P-VN ($\beta = 1, 2, \ldots, N$);
- $\boldsymbol{L}_{ch}^{D1H(\alpha)}$ as a vector consisting of the channel LLR values of the D1H-VNs connected to the α-th H-CN ($\alpha = 1, 2, \ldots, M$);
- $L_{app}^{PVN}(\beta)$ as the *a posteriori* (APP) LLR value of the β-th P-VN ($\beta = 1, 2, \ldots, N$);
- $L_{ex}^{PVN}(\alpha, \beta)$ as the extrinsic LLR sent from the β-th P-VN to the α-th H-CN ($\alpha = 1, 2, \ldots, M$; $\beta = 1, 2, \ldots, N$);
- $L_{app}^{H}(\alpha, \beta)$ as the APP LLR computed by the α-th H-CN for the β-th P-VN ($\alpha = 1, 2, \ldots, M$; $\beta = 1, 2, \ldots, N$);
- $L_{ex}^{H}(\alpha, \beta)$ as the extrinsic LLR sent from the α-th H-CN to the β-th P-VN ($\alpha = 1, 2, \ldots, M$; $\beta = 1, 2, \ldots, N$).

We also denote

- $\mathcal{P}(\alpha)$ as the set of P-VNs connected to the α-th H-CN;
- $\mathcal{H}(\beta)$ as the set of H-CNs connected to the β-th P-VN.

The standard PLDPC-Hadamard decoder [76, 94] consists of two component decoders, i.e., repeat decoder and Hadamard decoder, which are shown in Fig. 40 and Fig. 41, respectively. The repeat

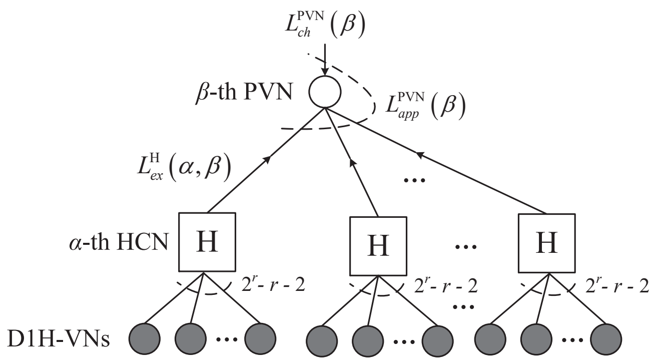

Figure 40: A repeat decoder for P-VN message processing.

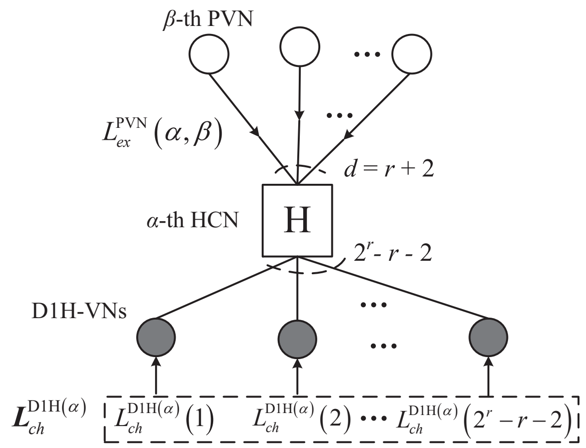

Figure 41: A symbol-MAP Hadamard decoder for H-CN message processing when r is even.

decoder is the same as the variable-node processor used in an LDPC decoder. The check node processor used in an LDPC decoder is replaced by a symbol-by-symbol maximum a posteriori probability (MAP) Hadamard decoder. Referring to Fig. 41, the symbol-MAP Hadamard decoder receives $d = r + 2$ inputs from the repeat decoders and $2^r - r - 2$ inputs from the D1H-VNs, and computes d outputs and feeds them back to the repeat decoders.

The standard decoding method is described as follows.

1. Initialization: Set $L_{ex}^{\mathrm{PVN}}(\alpha, \beta) = L_{ch}^{\mathrm{PVN}}(\beta)$, $\forall \alpha = 1, 2, \ldots, M$ and $\beta = 1, 2, \ldots, N$.

2. Symbol-MAP Hadamard decoder: For the α-th H-CN ($\alpha = 1, 2, \ldots, M$), compute the following.
   a) Compute $L_{app}^{\mathrm{H}}(\alpha, \beta)$ for the β-th P-VN ($\beta \in \mathcal{P}(\alpha)$) using

$$
\begin{aligned}
\boldsymbol{L}_{app}^{\mathrm{H}}(\alpha) &= \{L_{app}^{\mathrm{H}}(\alpha, \beta) : \beta \in \mathcal{P}(\alpha)\} \\
&= \mathcal{T}\left[\{L_{ex}^{\mathrm{PVN}}(\alpha, \beta) : \beta \in \mathcal{P}(\alpha)\}, \boldsymbol{L}_{ch}^{\mathrm{D1H}(\alpha)}\right]
\end{aligned}
\tag{63}
$$

where $\mathcal{T}$ is a transformation involving the fast Hadamard transform (FHT) and the dual FHT (DFHT) operations [1, 94]. Fig. 8 and Fig. 9 illustrate the arrangement of the $2^r = 16$ / $2^r \times 2 = 16$ inputs when they are fed to the FHT block in a symbol-MAP Hadamard decoder for the case $r = 4$ / $r = 3$. In the example of Fig. 8, the $r+2$ extrinsic LLR values from P-VNs are assigned to the 1st, 2nd,...,$(2^{k-1}+1)$-th, ..., $(2^{r-1}+1)$-th and 2^r-th positions; while the channel LLR values of the D1H-VNs are assigned to the remaining $2^r - r - 2$ positions [94].

b) Compute $L_{ex}^{H}(\alpha, \beta)$ by subtracting $L_{ex}^{PVN}(\alpha, \beta)$ from $L_{app}^{H}(\alpha, \beta)$, i.e.,

$$L_{ex}^{H}(\alpha, \beta) = L_{app}^{H}(\alpha, \beta) - L_{ex}^{PVN}(\alpha, \beta); \quad \forall \beta \in \mathcal{P}(\alpha). \tag{64}$$

3. Repeat decoder: For the β-th P-VN ($\beta = 1, 2, \ldots, N$), compute the following.

   a) Compute $L_{app}^{PVN}(\beta)$ for the β-th P-VN using

$$L_{app}^{PVN}(\beta) = \sum_{\alpha \in \mathcal{H}(\beta)} L_{ex}^{H}(\alpha, \beta) + L_{ch}^{PVN}(\beta). \tag{65}$$

   b) Compute $L_{ex}^{PVN}(\alpha, \beta)$ by subtracting $L_{ex}^{H}(\alpha, \beta)$ from $L_{app}^{PVN}(\beta)$, i.e.,

$$L_{ex}^{PVN}(\alpha, \beta) = L_{app}^{PVN}(\beta) - L_{ex}^{H}(\alpha, \beta); \quad \forall \alpha \in \mathcal{H}(\beta). \tag{66}$$

4. Decoding: Repeat Step 2 and Step 3 I times and make decisions based on the sign of $L_{app}^{PVN}(\beta)$ ($\beta = 1, 2, \ldots, N$).

4.2 LAYERED DECODING ALGORITHM

It is well known that using layered BP decoding for LDPC codes can accelerate the convergence and to reduce the hardware requirements compared with using standard BP decoding [98]. In [99], an efficient check-node-update scheduling has been proposed for rate-compatible punctured LDPC codes, and is shown to outperform conventional scheduling and conventional BP decoding in terms of convergence speed. In [100], an efficient dynamic scheduling scheme has been proposed to speed up the convergence rate of LDPC decoders at medium to high signal-to-noise (SNR) region. In [101], a safe early termination strategy has been developed for layered LDPC decoding in order to help saving resources such as power and processing time. To improve the convergence rate of the PLDPC-BC decoder, we propose a layered decoding algorithm. Moreover, we

1	2	3	4	5	6 : 8	9	10 : 15	16
$L_{ex}^{PVN}(\alpha, \beta_1)$	$L_{ex}^{PVN}(\alpha, \beta_2)$	$L_{ex}^{PVN}(\alpha, \beta_3)$	$L_{ch}^{D1H(\alpha)}(1)$	$L_{ex}^{PVN}(\alpha, \beta_4)$	$\boldsymbol{L}_{ch}^{D1H(\alpha)}(2:4)$	$L_{ex}^{PVN}(\alpha, \beta_5)$	$\boldsymbol{L}_{ch}^{D1H(\alpha)}(5:10)$	$L_{ex}^{PVN}(\alpha, \beta_6)$

Table 8: Arrangement of the $2^r = 16$ inputs when they are fed to the FHT block in a symbol-MAP Hadamard decoder for the case $r = 4$. $\{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\} = \mathcal{P}(\alpha)$.

1	2	3	4	5	6	7	8
0 $+$ $L_{ex}^{PVN}(\alpha, \beta_1)$	$\boldsymbol{L}_{ch}^{D1H(\alpha)}(1)$ $\pm$ $L_{ex}^{PVN}(\alpha, \beta_2)$	$\boldsymbol{L}_{ch}^{D1H(\alpha)}(2)$ $\pm$ $L_{ex}^{PVN}(\alpha, \beta_3)$	$\boldsymbol{L}_{ch}^{D1H(\alpha)}(3)$ $+$ 0	$\boldsymbol{L}_{ch}^{D1H(\alpha)}(4)$ $\pm$ $L_{ex}^{PVN}(\alpha, \beta_4)$	$\boldsymbol{L}_{ch}^{D1H(\alpha)}(5)$ $+$ 0	$\boldsymbol{L}_{ch}^{D1H(\alpha)}(6)$ $+$ 0	0 $+$ $L_{ex}^{PVN}(\alpha, \beta_5)$

Table 9: Arrangement of the $2^r \times 2 = 16$ inputs when they are fed to two set of FHT blocks in a symbol-MAP Hadamard decoder for the odd case $r = 3$. $\{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\} = \mathcal{P}(\alpha)$. The odd case also needs two set of DFHT blocks for computing extrinsic information and the *a posteriori* information.

conduct a complexity analysis and compare the simulation results for the standard and layered decoding algorithms.

4.2.1 *Proposed Layered Decoding Algorithm*

Recall in Appendix D that the base matrix $\boldsymbol{B}_{m\times n}$ is lifted twice using factors z_1 and z_2, respectively, to form $\boldsymbol{H}_{M\times N}$ which has a size of $M \times N$ $(= mz_1z_2 \times nz_1z_2)$. Here we divide $\boldsymbol{H}_{M\times N}$ into mz_1 layers, where each layer is composed of $1 \times nz_1$ CPMs each of size $z_2 \times z_2$ (or equivalently a block-row of size $z_2 \times nz_1z_2$). In other words, each layer consists of z_2 H-CNs, each connected to d independent P-VNs. ($d = r + 2$ is the row weight of $\boldsymbol{B}_{m\times n}$ and also that of $\boldsymbol{H}_{M\times N}$.)

We use the same symbols in Section 4.1. Moreover, we define k as the layer number ($k = 1, 2, \ldots, mz_1$) and $\mathcal{L}(k)$ as the set of H-CNs in layer k. Our layered decoding algorithm is described as follows.

1. Initialization: Set $L^{\text{PVN}}_{\text{app}}(\beta) = L^{\text{PVN}}_{\text{ch}}(\beta)$, $\forall\beta = 1, 2, \ldots, N$; and set $L^{\text{H}}_{\text{ex}}(\alpha, \beta) = 0$ $\forall\alpha = 1, 2, \ldots, M$ and $\beta = 1, 2, \ldots, N$.

2. Symbol-MAP Hadamard layered decoder: Set $k = 1$.

   a) For the α-th H-CN in layer k ($\alpha \in \mathcal{L}(k)$), compute the following.

      i. For $\beta \in \mathcal{P}(\alpha)$, compute $L^{\text{PVN}}_{\text{ex}}(\alpha, \beta)$ by subtracting $L^{\text{H}}_{\text{ex}}(\alpha, \beta)$ from $L^{\text{PVN}}_{\text{app}}(\beta)$, i.e.,

      $$L^{\text{PVN}}_{\text{ex}}(\alpha, \beta) = L^{\text{PVN}}_{\text{app}}(\beta) - L^{\text{H}}_{\text{ex}}(\alpha, \beta);\quad \forall\beta \in \mathcal{P}(\alpha). \tag{67}$$

      ii. Compute $L^{\text{H}}_{\text{app}}(\alpha, \beta)$ for the β-th P-VN ($\beta \in \mathcal{P}(\alpha)$) using

      $$\begin{aligned} \boldsymbol{L}^{\text{H}}_{\text{app}}(\alpha) &= \{L^{\text{H}}_{\text{app}}(\alpha, \beta) : \beta \in \mathcal{P}(\alpha)\} \\ &= \mathcal{T}\left[\{L^{\text{PVN}}_{\text{ex}}(\alpha, \beta) : \beta \in \mathcal{P}(\alpha)\}, \boldsymbol{L}^{\text{D1H}(\alpha)}_{\text{ch}}\right]. \end{aligned} \tag{68}$$

      iii. Update $L^{\text{H}}_{\text{ex}}(\alpha, \beta)$ and $L^{\text{PVN}}_{\text{app}}(\beta)$ using

      $$L^{\text{H}}_{\text{ex}}(\alpha, \beta) = L^{\text{H}}_{\text{app}}(\alpha, \beta) - L^{\text{PVN}}_{\text{ex}}(\alpha, \beta);\quad \forall\beta \in \mathcal{P}(\alpha) \tag{69}$$

      $$L^{\text{PVN}}_{\text{app}}(\beta) = L^{\text{H}}_{\text{app}}(\alpha, \beta);\quad \forall\beta \in \mathcal{P}(\alpha). \tag{70}$$

   b) If k is smaller than the number of layers, i.e., $k < mz_1$, increment k by 1 and goto Step 2a).

3. Repeat Step 2 I times and make decisions based on the sign of $L^{\text{PVN}}_{\text{app}}(\beta)$ ($\beta = 1, 2, \ldots, N$).

Note that (70) is derived as follows. We consider the associated P-VNs in layer k. Note that each of the associated P-VNs is connected to one and only one H-CN in layer k. We suppose the β-th P-VN is connected to the α-th H-CN in layer k. After this layer is processed, the updated APP for the β-th P-VN is given by

$$\begin{aligned} L_{app}^{PVN}(\beta) &= \sum_{\alpha \in \mathcal{H}(\beta)} L_{ex}^{H}(\alpha, \beta) + L_{ch}^{PVN}(\beta) \\ &= L_{ex}^{H}(\alpha, \beta) + \sum_{\substack{\alpha' \in \mathcal{H}(\beta) \\ \alpha' \notin \mathcal{L}(k)}} L_{ex}^{H}(\alpha', \beta) + L_{ch}^{PVN}(\beta) \\ &= L_{ex}^{H}(\alpha, \beta) + L_{ex}^{PVN}(\alpha, \beta) \\ &= L_{app}^{H}(\alpha, \beta). \end{aligned} \tag{71}$$

4.2.2 *Complexity Analysis*

We compare the complexity of the proposed layered decoding algorithm and the standard decoding algorithm in terms of memory requirement and computational logic.

4.2.2.1 *Memory requirement*

Considering the layered decoding algorithm in Section 4.2.1, memory storage (i.e., RAM) for the following sets of LLRs is required — $\{L_{ch}^{PVN}(\beta)\}$, $\{L_{app}^{PVN}(\beta)\}$, $\{L_{ex}^{H}(\alpha, \beta)\}$ and $\{\boldsymbol{L}_{ch}^{D1H(\alpha)}\}$. Moreover, $\{L_{app}^{H}(\alpha, \beta)\}$ and $\{L_{ex}^{PVN}(\alpha, \beta)\}$ are only intermediate variables generated during the computation process and thus need no storage. Note also that $\{L_{ch}^{PVN}(\beta)\}$ is only required during the initialization process but not in the iterative process. Thus, it can be immediately released for storing the LLRs for the next codeword.

For the standard decoding algorithm in Section 4.1, besides $\{L_{ch}^{PVN}(\beta)\}$, $\{L_{app}^{PVN}(\beta)\}$, $\{L_{ex}^{H}(\alpha, \beta)\}$ and $\{\boldsymbol{L}_{ch}^{D1H(\alpha)}\}$, $\{L_{ex}^{PVN}(\alpha, \beta)\}$ needs to be stored after the computation in (66). On the other hand, we can observe that $L_{ex}^{H}(\alpha, \beta)$, after being used to update $L_{ex}^{PVN}(\alpha, \beta)$ in (66), is no longer needed. The memory location used to store $L_{ex}^{H}(\alpha, \beta)$ can therefore be used to store $L_{ex}^{PVN}(\alpha, \beta)$. Similarly, $L_{ex}^{PVN}(\alpha, \beta)$ is no longer needed after computing (64), and its memory location can be used to store $L_{ex}^{H}(\alpha, \beta)$ afterwards. In other words, $\{L_{ex}^{H}(\alpha, \beta)\}$ and $\{L_{ex}^{PVN}(\alpha, \beta)\}$ can share the same set of memory locations. But unlike in the layered decoding algorithm, $\{L_{ch}^{PVN}(\beta)\}$ in the standard decoding algorithm is required throughout the iterative process (in (65)). Thus another set of memory is required to store $\{L_{ch}^{PVN}(\beta)\}$ for the next codeword. Note that the number of $L_{ch}^{PVN}(\beta)$ is equal to number of P-VNs, i.e., $N = nz_1z_2$. For the $r = 4$ PLDPCH-BC optimized in [76], N equals $11 \times 32 \times 512 = 180224$, which implies quite a large memory.

4.2.2.2 *Computational logic*

Both the layered decoding algorithm and the standard decoding algorithm involve FHT, DFHT and simple additions/subtractions. Moreover, the summation term in (65) of the standard decoding algorithm requires the addition of all $L_{ex}^{H}(\alpha, \beta)$ terms corresponding to the same P-VN. The number of terms equals the column weight and varies from column to column. In the example given in Fig. 42, the column weight ranges from 1 to 9. When 9 values are to be added together, more combinational logics (especially many P-VNs are processed in parallel) are required and a slightly larger latency is needed. However, for layered decoding algorithm, (70) updates L_{app}^{PVN} without consuming any combinational logics (i.e., use 100% less combinational logics compared with the standard decoding algorithm).

In summary, the layered decoding algorithm requires less memory storage and computational logic compared with the standard decoding algorithm.

4.2.3 *Simulation Results*

We simulate the $r = 4$ and $R = 0.0494$ PLDPCH-BC optimized in Section 3.4.1.1 [76] (whose base matrix is shown in (72) and protograph is shown in Fig. 42).

$$\boldsymbol{B}_{7\times 11} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \\ 2 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 \\ 3 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 2 & 0 \end{bmatrix} \tag{72}$$

We transmit all-zero codewords using binary-phase-shift-keying modulation over an additive white Gaussian noise channel. To compare with the BER performance of the standard decoder used in Section 3.4.1.1 [76], we use the same lifting factors, i.e., $z_1 = 32$ and $z_2 = 512$, and the same code length, i.e., $l = 1,327,104$ (See Appendix D for details of the code structure after the lifting process).

Fig. 43 plots the bit error rate (BER) results of the standard and layered decoders. We denote the maximum number of decoding iterations used by the layered decoder as I. When $I = 30, 40, 50, 60, 75, 150$, the layered decoding algorithm using I iterations has almost the same error rate as the standard decoder using 2I iterations. When $I = 20$, there is a 0.03 dB difference between the layered decoding algorithm using $I = 20$ iterations and the standard decoder using

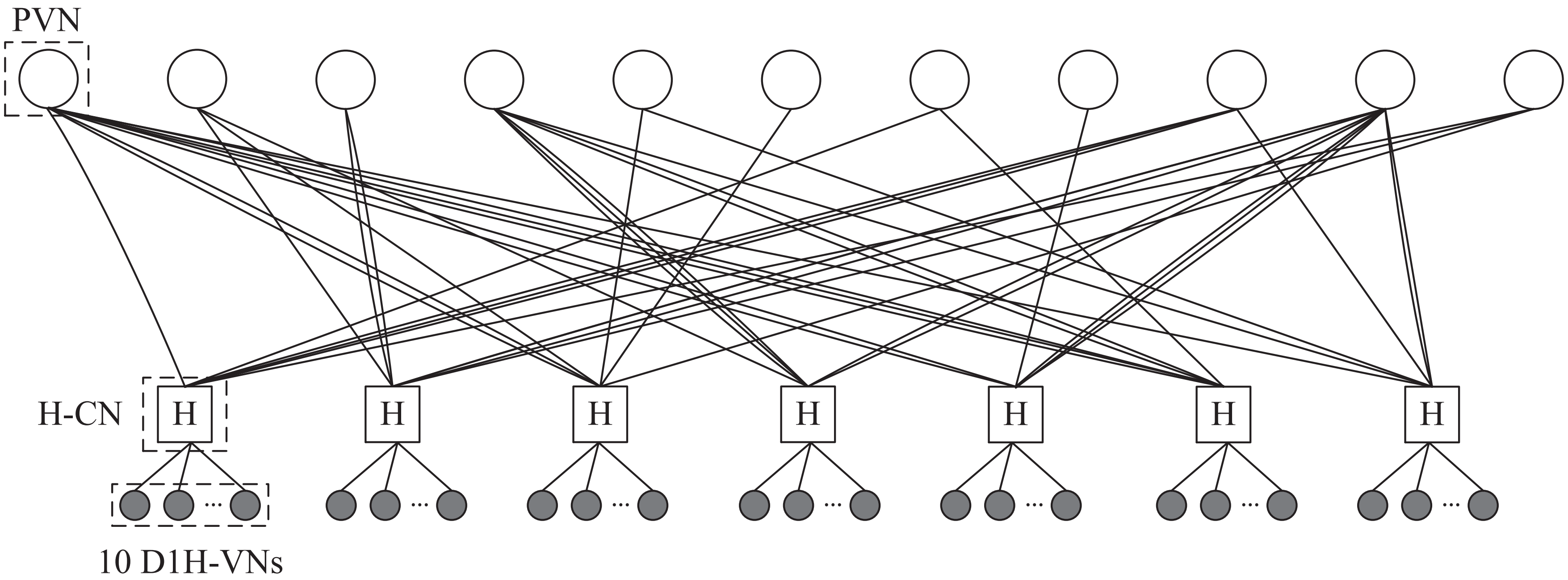

Figure 42: The protograph of the rate-0.0494 PLDPC-Hadamard code. A circle denotes a protograph variable node (P-VN), a square with "H" denotes a Hadamard check node (H-CN), and a filled circle denotes a degree-1 Hadamard variable node (D1H-VN). Row weight $d = 6$, $r = d - 2 = 4$ and $2^r - r - 2 = 10$ D1H-VNs are attached to each H-CN. Code rate $R = 0.0494$.

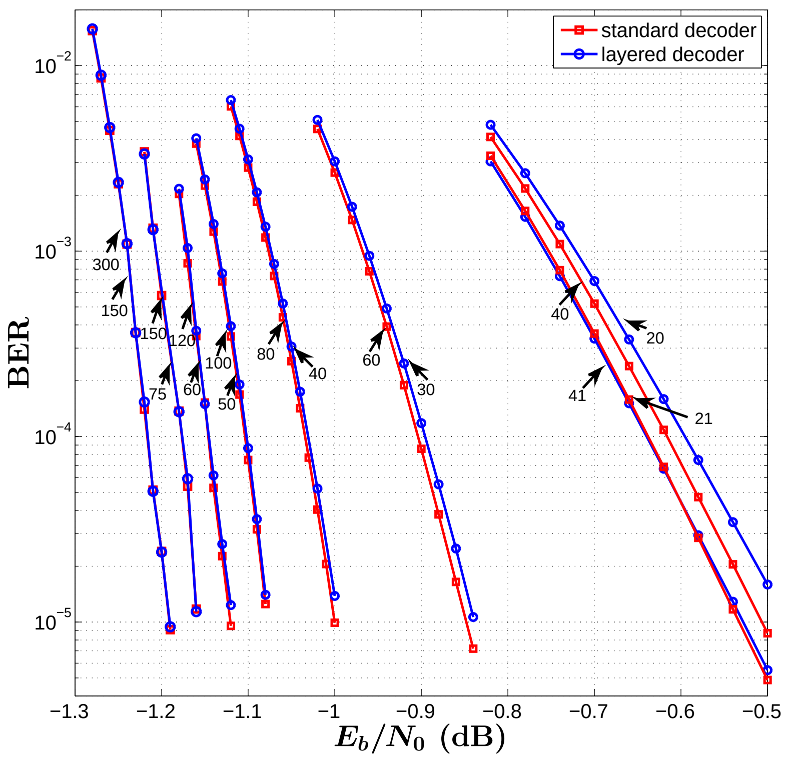

Figure 43: Comparison of BER performance of the standard PLDPC-Hadamard decoder and the layered PLDPC-Hadamard decoder. The maximum number of decoding iterations ranges from 40 to 300 for the standard decoder; and ranges from 20 to 150 for the layered decoder. $r = 4$ and $R = 0.0494$.

$2I = 40$ iterations at a bit error rate of 2.0×10^{-5}. Note that in most scenarios, a 0.03 dB difference is considered as insignificant, but has been shown in our figure to be a relatively large gap due to the scale being used. We further find that when $I = 21$, the layered decoding algorithm outperforms the standard decoder using 40 iterations and has the same performance of the standard decoder using 41 iterations. Thus we can conclude that compared with the standard decoding algorithm, the layered decoding algorithm improves the convergence rate by about two times. Fig. 44 plots the corresponding average number of iterations required to decode a codeword. At a given E_b/N_0, the average number of iterations required by the layered decoder is about half of that required by the standard decoder.

Remark: The simulation results reported in this section are obtained by software simulation. To improve decoding efficiency, we run our programs on GPU platform. Since there is sufficient memory on this platform, we can design our decoders with high-degree parallelism. For the standard decoding, in one iteration, we can update all the P-VNs at the same time, and then update all the

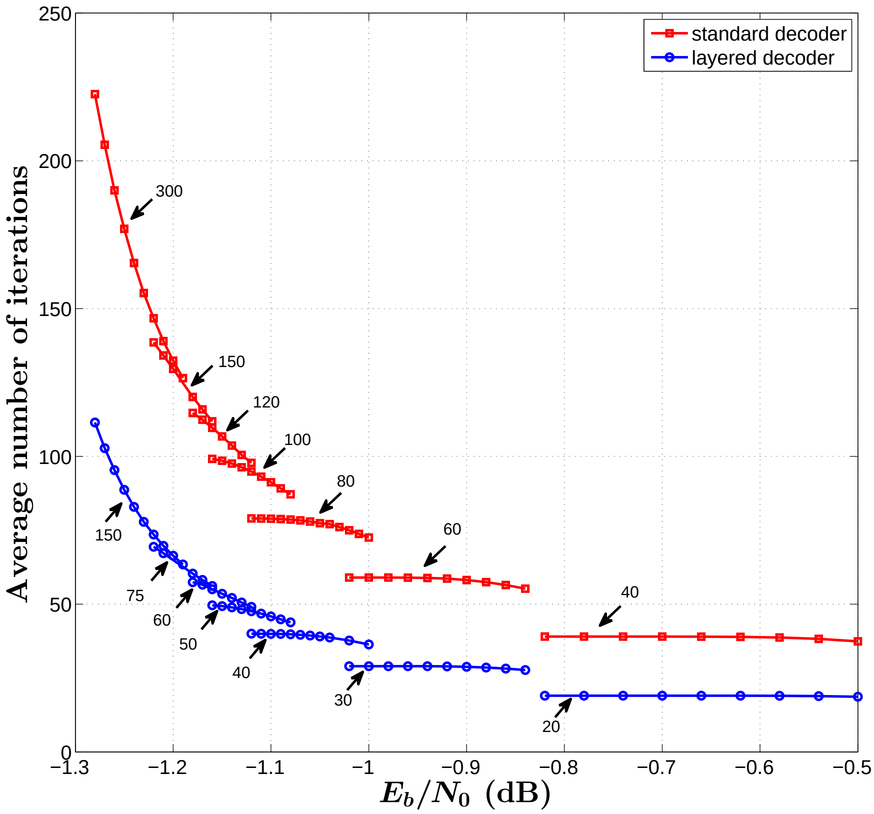

Figure 44: Average number of iterations required for a standard PLDPC-Hadamard decoder and a layered PLDPC-Hadamard decoder to decode a codeword. The maximum numbers of iterations allowed are given next to the curves.

H-CNs at the same time. However, for the layered decoding, in one iteration, we can only update the H-CNs in the same layer at the same time, and then update the remaining layers layer-by-layer. Therefore, computation time for the standard decoding will be less than that for the layered decoding. But our proposed layered decoding algorithm will play an important role when implementing it on hardware. Because the resource on hardware is limited, the decoder with high-degree parallelism cannot be realized. Compared with the standard decoding, the layered decoding consumes less memory and computational logics, has faster convergence rate, which is beneficial to achieving high working frequency, high-throughput and low-latency.

4.3 HARDWARE ARCHITECTURE OF LAYERED DECODERS

Based on the layered decoding algorithm, we propose a hardware architecture of layered decoders for PLDPC-BCs [97]. In our proposed layered decoder for the PLDPCH-BC, there are four types of random access memory (RAM) which are used to store, respectively, $\{L_{ch}^{PVN}(\beta)\}$, $\{L_{app}^{PVN}(\beta)\}$, $\{L_{ex}^{H}(\alpha,\beta)\}$ and $\{\boldsymbol{L}_{ch}^{D1H(\alpha)}\}$. As can be seen in the previous section, $\{L_{ex}^{PVN}(\alpha,\beta)\}$ in (67) and $\{\boldsymbol{L}_{app}^{H}(\alpha,\beta)\}$ in (68) are only temporary values in the computation process and thus need no storage. Moreover, dual-port RAMs are used in our design, meaning that two memory locations can be accessed (read and/or write) at the same time.

4.3.1 *Operation of a symbol-MAP Hadamard sub-decoder*

Referring to Step 2a) of the decoding algorithm, the inputs to each symbol-maximum-a-posterior (symbol-MAP) Hadamard sub-decoder are $\{L_{app}^{PVN}(\beta)\}$ (or $\{L_{ch}^{PVN}(\beta)\}$ in the first iteration), $\{L_{ex}^{H}(\alpha,\beta)\}$ (or 0 in the first iteration) and $\{\boldsymbol{L}_{ch}^{D1H(\alpha)}\}$ while the outputs of the decoder are updated $\{L_{app}^{PVN}(\beta)\}$ and $\{L_{ex}^{H}(\alpha,\beta)\}$. We suppose each H-CN connects to $d = 6$ P-VNs. Thus, $|\mathcal{P}(\alpha)| = 6$ and 6 sets of $L_{app}^{PVN}(\beta)$ and $L_{ex}^{H}(\alpha,\beta)$ need to be read from the RAMs and input to decoder according to (67). Since dual-port RAMs are used, two memory addresses can be accessed at the same time and it takes $d/2$ clock cycles to retrieve the required $L_{app}^{PVN}(\beta)$ and $L_{ex}^{H}(\alpha,\beta)$ values. Note that $\{L_{ex}^{PVN}(\alpha,\beta)\}$ in (67) is computed in the same clock cycle as $L_{app}^{PVN}(\beta)$ and $L_{ex}^{H}(\alpha,\beta)$ are retrieved. At the $d/2$-th clock cycle, we also load the required $\boldsymbol{L}_{ch}^{D1H(\alpha)}$ vector from one address location to the decoder.

Subsequently, $L_{ex}^{PVN}(\alpha,\beta)$ and $\boldsymbol{L}_{ch}^{D1H(\alpha)}$ are passed to the transformation block $\mathcal{T}$. The transformation block $\mathcal{T}$ is based on the FHT block and the DFHT block [94]. First, there are r stages in the FHT block and hence a latency of r clock cycles is incurred. The structure of a DFHT block is similar to that of a FHT, but with twice the number of inputs and outputs. Same as the FHT block, the DFHT block contains r stages and has a latency of r clock cycles. Thus the transformation block $\mathcal{T}$ has a latency of $2r$ clock cycles. Then, it takes one clock cycle to compute $L_{ex}^{H}(\alpha,\beta)$ and $L_{app}^{PVN}(\beta)$ using (69) and (70), respectively. Finally, it takes another $d/2$ clock cycles to write the updated $L_{app}^{PVN}(\beta)$ and $L_{ex}^{H}(\alpha,\beta)$ values to the RAMs.

To summarize,

1. Clock cycle no. 1 to $d/2$: read $L_{app}^{PVN}(\beta)$ and $L_{ex}^{H}(\alpha,\beta)$ from memory, and at the same time compute $\{L_{ex}^{PVN}(\alpha,\beta)\}$ using (67);
2. Clock cycle no. $d/2$ (in parallel with above): read $\boldsymbol{L}_{ch}^{D1H(\alpha)}$;
3. Clock cycle no. $d/2+1$ to $d/2+2r$: process the inputs $\{L_{ex}^{PVN}(\alpha,\beta)\}$ and $\boldsymbol{L}_{ch}^{D1H(\alpha)}$ by the transformation block $\mathcal{T}$ using (63);
4. Clock cycle no. $d/2+2r+1$: compute $L_{ex}^{H}(\alpha,\beta)$ and $L_{app}^{PVN}(\beta)$ using (69) and (70);

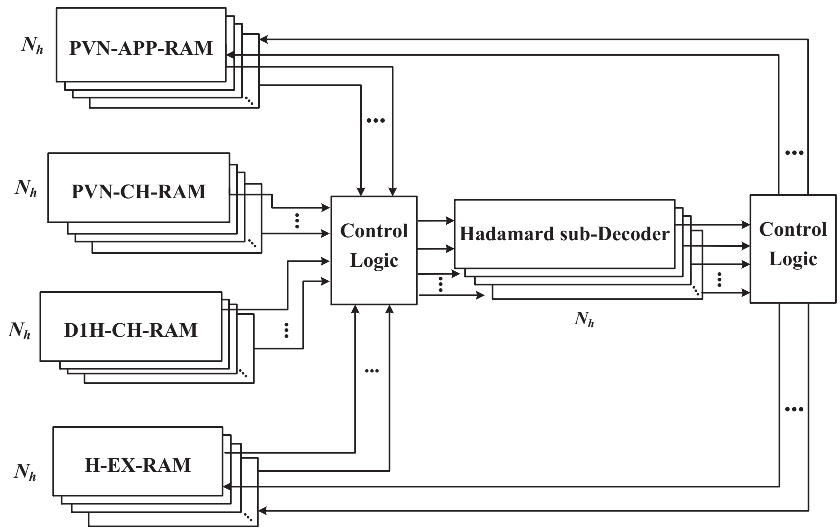

Figure 45: Proposed layered PLDPC-Hadamard decoder with N_h sub-decoders.

5. Clock cycle no. $d/2+2r+2$ to $d/2+2r+1+d/2$: write $L_{app}^{PVN}(\beta)$ and $L_{ex}^{H}(\alpha,\beta)$ to memory.

Since $d = r+2$, the whole process takes $d/2+2r+1+d/2 = 3r+3$ clock cycles. To minimize the latency and maximize the throughput of the decoder, one can employ z_2 symbol-MAP Hadamard sub-decoders to process all the H-CNs in each layer simultaneously because these H-CNs are independent of one another. However, it consumes a lot of hardware resources and may not be practical. As in the decoding of other LDPC codes [102], we propose here dividing the H-CNs in each layer into G groups, where $N_h = z_2/G$ is an integer. Then each group of H-CNs are processed in parallel at the same time by N_h individual Hadamard sub-decoders.

4.3.2 *Decoder Architecture*

Fig. 45 shows the architecture of our proposed PLDPC-Hadamard layered decoder which operates with four types of RAM. The control logics are dependent on the structure of adjacency matrix which has a relatively simple quasi-cyclic format. They are used to ensure that the correct data are loaded into the individual Hadamard sub-decoder and the updated data are written to the correct memory locations. Moreover, each Hadamard sub-decoder can be realized with additions and look-up tables, which can reduce the implementation complexity.

To ensure that no conflict of memory access occurs when the N_h Hadamard sub-decoders are operating on N_h individual sets of independent data, we design the size and storage of RAMs as follows.

- N_h RAMs, denoted by PVN-CH-RAM, are used to store $\{L_{ch}^{PVN}(\beta) : \beta = 0, \ldots, N-1\}$. Each RAM has a width of w_{ch}^{PVN} bits (to represent the quantized LLR value) and a depth of nz_1G. The g-th location ($g = 0, 1, \ldots, nz_1G-1$) in the l-th RAM ($l = 0, 1, \ldots, N_h - 1$) stores $L_{ch}^{PVN}(\beta)$ where $\beta = \lfloor g/G \rfloor z_2 + lG + (g \mod G)$, $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x and "mod" denotes the modulus operation. Note that $\{L_{ch}^{PVN}(\beta)\}$ is needed only once during the first decoding iteration. After the first iteration, the content in PVN-CH-RAM is overwritten by the incoming channel LLR values of the next codeword.

- N_h RAMs, denoted by PVN-APP-RAM, are used to store $\{L_{app}^{PVN}(\beta) : \beta = 0, \ldots, N-1\}$. Each RAM has a width of w_{app}^{PVN} bits and a depth of nz_1G. Data are stored in the same way as in PVN-CH-RAM, i.e., the g-th location ($g = 0, 1, \ldots, nz_1G - 1$) in the l-th RAM ($l = 0, 1, \ldots, N_h - 1$) stores $L_{app}^{PVN}(\beta)$ where $\beta = \lfloor g/G \rfloor z_2 + lG + (g \mod G)$.

- N_h RAMs, denoted by H-EX-RAM, are used to store $\{L_{ex}^{H}(\alpha, \beta) : \alpha = 0, \ldots, M-1; \beta \in \{\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5\} = \mathcal{P}(\alpha)\}$. Each RAM has a width of w_{ex}^{H} bits and a depth of mdz_1G. The p-th location ($p = 0, 1, \ldots, mdz_1G - 1$) in the l-th RAM ($l = 0, 1, \ldots, N_h - 1$) stores $\{L_{ex}^{H}(\alpha, \beta)\}$ where $\alpha = \lfloor p/(dG) \rfloor z_2 + lG + \lfloor \Delta/d \rfloor$, $\beta = \beta_\delta$, $\Delta = p \mod (dG)$ and $\delta = \Delta \mod d$.

- N_h RAMs, denoted by D1H-CH-RAM, are used to store $\{\boldsymbol{L}_{ch}^{D1H(\alpha)} : \alpha = 0, \ldots, M-1\}$. Each RAM has a width of $w_{ch}^{D1H} = w_{ch}^{PVN} \times (2^r - r - 2)$ bits and a depth of mz_1G. Each address stores all the $2^r - r - 2$ channel LLR values for D1HVNs connected to a H-CN. The q-th location ($q = 0, 1, \ldots, mz_1G - 1$) in the l-th RAM ($l = 0, 1, \ldots, N_h - 1$) stores $\{\boldsymbol{L}_{ch}^{D1H(\alpha)}\}$ where $\alpha = \lfloor q/G \rfloor z_2 + lG + (q \mod G)$. (To allow the decoding to proceed while receiving the incoming channel LLR values of the next codeword, either two sets of D1H-CH-RAM are used or the depth of D1H-CH-RAM is doubled to $2mz_1G$. We double the depth of D1H-CH-RAM to $2mz_1G$ in our design.) Moreover, one port reads the data in D1H-CH-RAM used for decoding and the other port writes incoming channel LLR values into the same RAM.

4.3.3 *Latency and Throughput*

Using the proposed decoder architecture, G groups of H-CNs (each consisting of N_h H-CNs) are sequentially processed in each layer. Referring to the timing details in Section 4.3.1 and with the use of our RAM designs, it takes $d/2$ clock cycles to load the data of one group of H-CNs. We use a pipelined structure and load the G groups of data to the sub-decoders in a consecutive manner. To complete loading all

G groups of data, it takes $t_{\mathrm{loading}} = dG/2$ clock cycles. Moreover, the first set of outputs (i.e., $L_{\mathrm{app}}^{\mathrm{PVN}}(\beta)$ and $L_{\mathrm{ex}}^{\mathrm{H}}(\alpha, \beta)$) is available at the $t_{1\mathrm{st\ output}} = (d/2 + 2r + 1)$-th clock cycle.

4.3.3.1 *Case I:* $t_{\mathrm{loading}} \leqslant t_{1\mathrm{st\ output}}$

When $t_{\mathrm{loading}} \leqslant t_{1\mathrm{st\ output}}$, all the required data are read from the RAMs before the Hadamard sub-decoders generate the updated results. The total time taken to complete updating one layer equals "loading time of all groups + processing time of last group + writing time of last group", i.e.,

$$t_{l1} = t_{\mathrm{loading}} + (2r + 1) + d/2 = (r/2 + 1)G + 5r/2 + 2 \tag{73}$$

using $d = r + 2$. Supposing I iterations are needed and the clock frequency is f_c, the latency for decoding each codeword equals

$$t_{c1} = Imz_1 t_{l1}/f_c = Imz_1[(r/2 + 1)G + 5r/2 + 2]/f_c, \tag{74}$$

where mz_1 is the number of layers in layered decoding. For a given $m \times n$ base matrix, the latency t_{c1} can be reduced by (a) lowering I and/or z_1 and/or G; or (b) increasing f_c. As the codeword length is $l = nz_1z_2 + mz_1z_2\,(2^r - r - 2)$, the throughput of the decoder is expressed as

$$\begin{aligned} T_1 &= \frac{l}{t_{c1}} = \frac{[nz_1z_2 + mz_1z_2\,(2^r - r - 2)]\,f_c}{Imz_1t_{l1}} \\ &= \frac{[n/m + (2^r - r - 2)]\,z_2f_c}{I[(r/2 + 1)G + (5r/2 + 2)]}. \end{aligned} \tag{75}$$

To improve the throughput, we can (a) increase z_2 and/or f_c; or (b) decrease I and/or G.

4.3.3.2 *Case II:* $t_{\mathrm{loading}} > t_{1\mathrm{st\ output}}$

When $t_{\mathrm{loading}} > t_{1\mathrm{st\ output}}$, the Hadamard sub-decoders start to output the updated results while all the required data are being read from the RAMs. In this case, we need to use first-in-first-out (FIFO) RAMs to temporarily store the updated results (i.e., $L_{\mathrm{app}}^{\mathrm{PVN}}(\beta)$ and $L_{\mathrm{ex}}^{\mathrm{H}}(\alpha, \beta)$) from the Hadamard sub-decoders. Once all the required data are read from the RAMs, the updated results stored in the FIFO RAMs are written to the RAMs. The total time taken to complete updating one layer equals "loading time of all groups + writing time of all groups", i.e.,

$$t_{l2} = dG/2 + dG/2 = (r + 2)G. \tag{76}$$

Table 10: Quantization schemes used to represent different LLR values; at the input, different stages and output of the FHT and DFHT blocks for a $r = 4$ PLDPC-Hadamard layered decoder.

Width	w_{ch}^{PVN}	w_{ex}^{H}	w_{app}^{PVN}	FHT block						DFHT block		
				In	Stage 1	2	3	4	Out	In	Stage 1 ~ 4	Out
No. of sign bits	1	1	1	1	1	1	1	1	1	1	1	1
No. of int. bits	1	4	4	4	5	6	7	8	6	6	6	4
No. of frac. bits	3	3	3	3	3	3	3	3	2	2	2	3

The latency to decode one codeword equals

$$t_{c2} = I m z_1 G(r+2)/f_c, \tag{77}$$

and the throughput equals

$$T_2 = \frac{[n/m + (2^r - r - 2)]\, f_c z_2}{IG(r+2)} \tag{78}$$

which can be improved by (a) increasing f_c and/or z_2; or (b) decreasing I and/or G. The difference from the first case is that we use FIFO RAMs to temporarily store the “updated" LLR values until all the required data are loaded into Hadamard sub-decoders.

Note that in both Case I and Case II, it requires $d/2$ clock cycles to complete loading one group of data to the Hadamard sub-decoders. Thus the throughput can potentially be increased by $d/2$ times if the Hadamard sub-decoders are allowed to process $d/2$ different codewords at the same time. The extra requirement would be $d/2$ times increase in memory storage and a bit more control logics.

4.3.4 *Implementation Results*

We implement the PLDPC-Hadamard decoder for the $r = 4$ and $R = 0.0494$ PLDPCH-BC optimized in Section 3.4.1.1 [76] (whose base matrix is shown in (72) and protograph is shown in Fig. 42) on a Xilinx VCU118 FPGA board. The maximum operating frequency is $f_c = 130$ MHz. All-zero codewords, binary phase shift keying (BPSK) modulation and an additive white Gaussian noise channel are assumed. To compare with the floating-point results in Section 4.2.3 [96], we use the same lifting factors, i.e., $z_1 = 32$ and $z_2 = 512$, and the same code length $l = 1,327,104$.

Table 11: Total number of four types of LLRs required in RAMs and width of four types of RAMs. $d = r + 2 = 6$, $N = nz_1z_2 = 11 \times 32 \times 512$ and $M = mz_1z_2 = 7 \times 32 \times 512$ for $r = 4$ PLDPCH-BC.

LLRs	$\{L_{ch}^{PVN}(\beta)\}$	$\{L_{app}^{PVN}(\beta)\}$	$\{L_{ex}^{H}(\alpha,\beta)\}$	$\{\boldsymbol{L}_{ch}^{D1H(\alpha)}\}$
Number	N	N	Md	M
width	5 bits	8 bits	8 bits	50 bits

Table 12: Comparison of implementation results for PLDPC-Hadamard decoder with 64 and 128 Hadamard sub-decoders. Hadamard order $r = 4$, code rate $R = 0.0494$, code length $l = 1327104$, and clock frequency $f_c = 130$ MHz. LUT: Look-up Table; BRAM: Block RAM.

No. of sub-decoders	$N_h = 64$		$N_h = 128$	
LUT Utilization	41.10%		81.76%	
BRAM Utilization	33.26%		33.10%	
No. of iterations	$I = 150$	$I = 20$	$I = 150$	$I = 20$
Latency	12.92 ms	1.72 ms	6.72 ms	0.896 ms
Throughput	0.10 Gb/s	0.77 Gb/s	0.20 Gb/s	1.48 Gb/s

We implement two designs with $N_h = 128$ ($G = 4$) and $N_h = 64$ ($G = 8$) Hadamard sub-decoders, respectively, which belong to Case I and Case II in Section 4.3.3. Table 10 shows the quantization schemes used, and Table 11 lists the total number of four types of LLRs required in RAMs and width of four types of RAMs. Note that $\{\boldsymbol{L}_{ch}^{D1H}(\beta)\}$ is a vector corresponding to $2^r - r - 2 = 10$ D1H-VNs and hence the width for its RAM equals $10W_{ch}^{PVN} = 50$ bits. Section 4.2.2 have mentioned that the standard decoding algorithm needs another set of memory to store $\{L_{ch}^{PVN}(\beta)\}$ for the next codeword. If implementing this algorithm on hardware, based on Table 11, we need to use one more memory with size of $5N = 880$ kb (i.e. use $\frac{5N}{5N+8N+48M+50M} \approx 6.63\%$ more memory compared with layered decoder).

Fig. 46 plots the BER results. It can be observed that the two designs produce almost the same BER curves. The minute difference arises only because the same noise samples generated have been assigned to different code bits in the two different designs. The results in Fig. 46 also show that a BER of 10^{-5}, the fixed-point decoder suffers

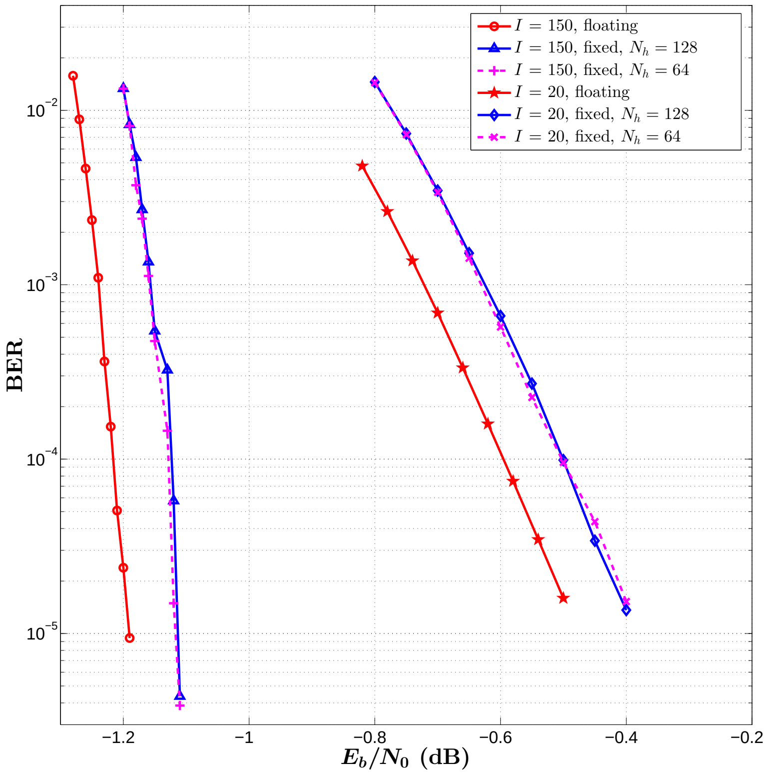

Figure 46: Floating-point and fixed-point BER performance of the layered PLDPC-Hadamard decoders. $r = 4$ and $l = 1327104$.

from a small degradation of 0.08 dB compared with the floating-point computation when $I = 150$ iterations are used; and a degradation of 0.10 dB when $I = 20$. Moreover, no error floor appears at a BER of 10^{-5} for both floating-point or fixed-point results.

For $r = 4$ (hence $d = r + 2 = 6$), $t_{1\text{st output}} = (d/2 + 2r + 1) = 12$ cycles. When $G = 4$, $t_{\text{loading}} = dG/2 = 12 = t_{1\text{st output}}$ which belongs to Case I in Section 4.3.3. The decoding latency per layer equals $t_{l1} = 24$ cycles. [1] Similarly when $G = 8$, $t_{\text{loading}} = dG/2 = 24 > t_{1\text{st output}}$ which belongs to Case II. The decoding latency per layer equals $t_{l2} = 48$ cycles. Table 12 lists the hardware implementation results of the proposed layered decoder for $N_h = 64$ ($G = 8$) and $N_h = 128$ ($G = 4$). Since the code lengths are identical, the two designs consume almost the same amount of block RAMs (BRAMs). Compared with the decoder with $N_h = 64$ Hadamard sub-decoders, the one with $N_h = 128$ sub-decoders produces about twice the throughput, reduces the latency by about half, and utilizes about twice the amount of look-up tables (LUTs).

1 In practice, there is a fixed delay t_δ when operating RAMs. In our designs, $t_\delta = 2$ cycles and are included in deriving the latency and throughput in Table 12.

4.4 SUMMARY

In this chapter, we have proposed a layered decoding algorithm for the ultimate-Shannon-limit-approaching PLDPC-Hadamard block code. Simulation results have verified that the layered decoding method can speed up the PLDPC-Hadamard decoder by about two times compared with the standard decoder. Though an even-order PLDPCH-BC is illustrated, the proposed algorithm can be readily applied to odd-order PLDPCH-BCs and other generalized LDPC codes by making appropriate modifications.

Based on the layered decoding algorithm, we have designed a hardware architecture of the PLDPC-Hadamard layered decoder and implemented it onto an FPGA. A throughput of 1.48 Gbps is achieved when 20 decoding iterations are used. If the Hadamard sub-decoders in the decoder are fully utilized, the throughput will be increased by $d/2 = 3$ times to almost 4.5 Gbps in the example used. Our decoder architecture is generic and can be readily modified to decode LDPC-Hadamard codes with the order r being odd and to decode other LDPC-derived codes when the Hadamard constraints LDPC-HC are replaced by other code constraints.

In the next chapter, we proceed to introducing and evaluating a derivative of PLDPCH-BC, namely spatially coupled PLDPC-Hadamard convolutional codes.

Chapter 5

SPATIALLY COUPLED PLDPC-HADAMARD CONVOLUTIONAL CODES

In this chapter, we show the details of our proposed spatially coupled PLDPC-Hadamard convolutional codes (SC-PLDPCH-CC). First, we show the way of constructing SC-PLDPCH codes, including SC-PLDPCH tail-biting code (SC-PLDPCH-TBC), SC-PLDPCH terminated code (SC-PLDPCH-TDC) and SC-PLDPCH-CC, from its block code counterpart. Second, we briefly explain the encoding process of SC-PLDPCH-CCs. Third, we describe an efficient decoding algorithm for SC-PLDPCH-CC, which combines the layered decoding used for decoding PLDPCH-BC [96] and the pipeline decoding used for decoding SC-PLDPC-CC [43]. Finally, we compare the bit error rate (BER) performance of SC-PLDPCH-CCs with their block code counterparts.

5.1 CODE CONSTRUCTION

Spatially coupled PLDPC-Hadamard codes are constructed in a similar way as the SC-PLDPC codes shown in Section 2.1.3. We also denote the coupling width as W and coupling length as L in a SC-PLDPCH code. Given a PLDPC-Hadamard block code with a protomatrix $\boldsymbol{B}$, we apply the edge spreading procedure to split $\boldsymbol{B}$ into $W+1$ protomatrices $\boldsymbol{B}_i$ ($i = 0, 1, \ldots, W$) under the constraint $\boldsymbol{B} = \sum_{i=0}^{W} \boldsymbol{B}_i$. Then we couple L sets of these matrices to construct the protomatrix of a spatially coupled PLDPC-Hadamard code. Similar to the SC-PLDPC codes described in Section 2.1.3, a SC-PLDPCH-TDC is formed if the coupled matrices are directly terminated; a SC-PLDPCH-TBC is formed if the coupled matrices are connected end-to-end; and a SC-PLDPCH-CC is formed if the coupling length L becomes infinite. Since the constructed protomatrices only represent the connections between P-VNs and H-CNs, SC-PLDPC-Hadamard codes have protomatrix structures similar to those of SC-PLDPC codes, i.e., (79) for SC-PLDPCH-TDC; (80) for SC-PLDPCH-TBC; and (81) for SC-PLDPCH-CC.

$$\boldsymbol{B}_{\text{SC-PLDPCH-TDC}} = \left.\overbrace{\begin{bmatrix} \boldsymbol{B}_0 & & & \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & & \\ \vdots & \boldsymbol{B}_1 & \ddots & \\ \boldsymbol{B}_W & \vdots & \ddots & \boldsymbol{B}_0 \\ & \boldsymbol{B}_W & \ddots & \boldsymbol{B}_1 \\ & & \ddots & \vdots \\ & & & \boldsymbol{B}_W \end{bmatrix}}^{nL}\right\} m(L+W). \tag{79}$$

$$\boldsymbol{B}_{\text{SC-PLDPCH-TBC}} = \\ \left.\overbrace{\begin{bmatrix} \boldsymbol{B}_0 & & & & & \boldsymbol{B}_W & \cdots & \boldsymbol{B}_1 \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & & & & & \ddots & \vdots \\ \vdots & \boldsymbol{B}_1 & \boldsymbol{B}_0 & & & & & \boldsymbol{B}_W \\ \boldsymbol{B}_W & \vdots & \boldsymbol{B}_1 & \ddots & & & & \\ & \boldsymbol{B}_W & \vdots & \ddots & \boldsymbol{B}_0 & & & \\ & & \boldsymbol{B}_W & \ddots & \boldsymbol{B}_1 & \boldsymbol{B}_0 & & \\ & & & \ddots & \vdots & \ddots & \boldsymbol{B}_0 & \\ & & & & \boldsymbol{B}_W & \cdots & \boldsymbol{B}_1 & \boldsymbol{B}_0 \end{bmatrix}}^{nL}\right\} mL \tag{80}$$

$$\boldsymbol{B}_{\text{SC-PLDPCH-CC}} = \begin{bmatrix} \boldsymbol{B}_0 & & & & \\ \boldsymbol{B}_1 & \boldsymbol{B}_0 & & & \\ \vdots & \boldsymbol{B}_1 & \ddots & & \\ \boldsymbol{B}_W & \vdots & \ddots & \boldsymbol{B}_0 & \\ & \boldsymbol{B}_W & \ddots & \boldsymbol{B}_1 & \ddots \\ & & \ddots & \vdots & \ddots \\ & & & \boldsymbol{B}_W & \ddots \\ & & & & \ddots \end{bmatrix}. \tag{81}$$

Unlike the protographs of SC-PLDPC codes which consist of P-VNs and SPC-CNs, the protographs of SC-PLDPCH codes contains P-VNs and H-CNs connected with some appropriate D1H-VNs.

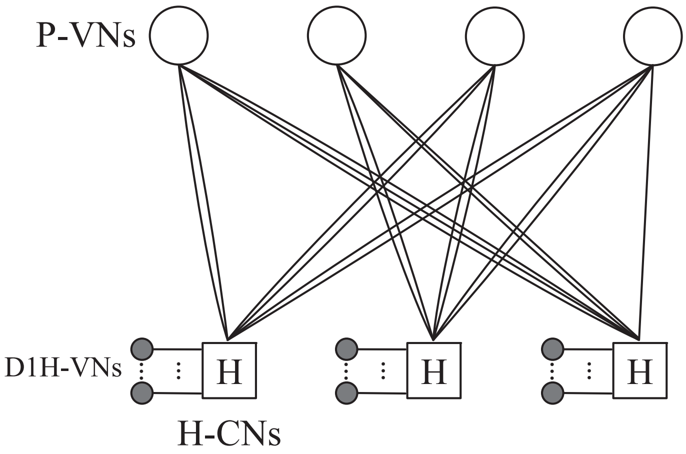

Figure 47: A protograph of PLDPC-Hadamard code. Number of D1H-VNs connected to each HCN is $2^r - d = 10$ using order-$r = d - 2 = 4$ Hadamard code.

Example: Assuming that

$$\boldsymbol{B} = \begin{bmatrix} 2 & 0 & 2 & 2 \\ 0 & 2 & 2 & 2 \\ 3 & 2 & 0 & 1 \end{bmatrix} \tag{82}$$

represents the 3×4 protomatrix of a PLDPC-BC, Fig. 47 illustrates the corresponding protograph consisting of $m = 3$ H-CNs and $n = 4$ P-VNs. The inputs to each H-CN fulfills the Hadamard constraint which is denoted by a box with the letter "H" inside. In Fig. 47, each H-CN connects $d = 6$ P-VNs. Thus, the Hadamard code has an order of $r = d - 2 = 4$ and generates $2^r - d = 10$ Hadamard parity-check bits, which are denoted as D1H-VNs and depicted as filled circles.

Assuming that $W = 1$, Fig. 48 shows the protograph of a SC-PLDPCH-CC which is derived from the PLDPCH-BC in Fig. 47. Two other types of terminated protographs of SC-PLDPCH codes, i.e., protographs of SC-PLDPCH-TDC and SC-PLDPCH-TBC, are shown in Fig. 49 and Fig. 50, respectively. In Figs. 48, 49 and 50, blue connections between P-VNs and H-CNs correspond to split protomatrix $\boldsymbol{B}_0$ (83) and red ones correspond to split protomatrix $\boldsymbol{B}_1$ (84), satisfying the constraint $\boldsymbol{B}_0 + \boldsymbol{B}_1 = \boldsymbol{B}$.

$$\boldsymbol{B}_0 = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix} \tag{83}$$

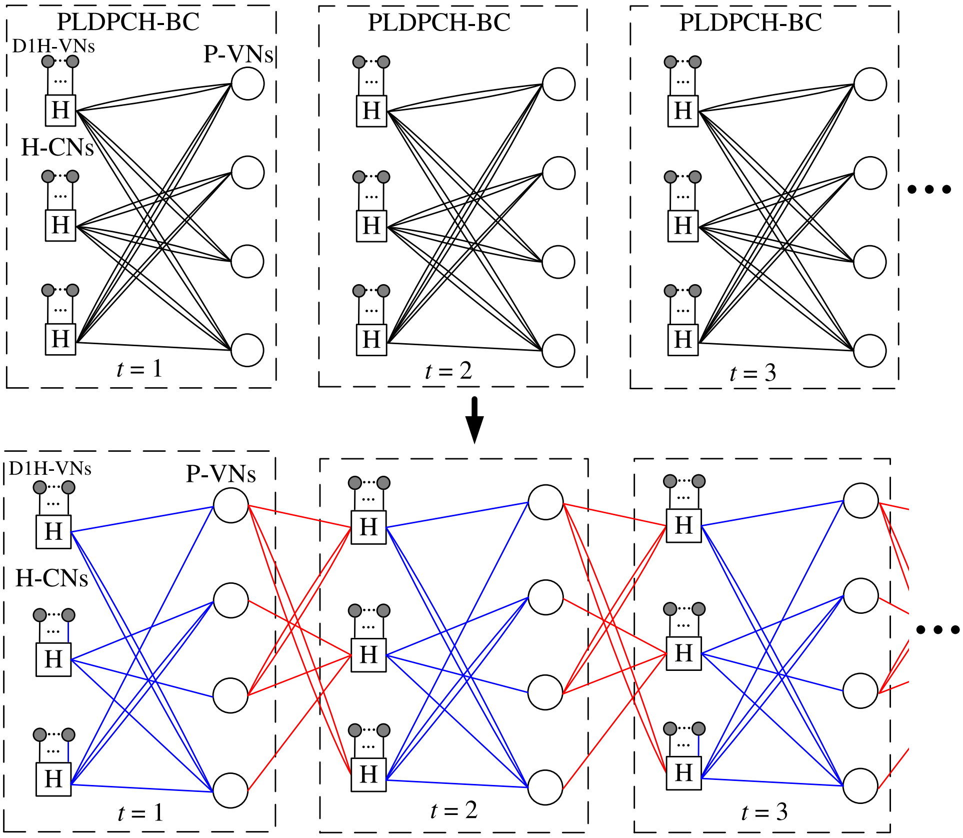

Figure 48: Protograph of a SC-PLDPCH-CC derived from the PLDPCH-BC in Fig. 47. $W = 1$.

and

$$\boldsymbol{B}_1 = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 1 & 1 \\ 2 & 0 & 0 & 0 \end{bmatrix}. \tag{84}$$

Using a similar two-step lifting process as that in Appendix D, SC-PLDPCH codes can be constructed from the coupled protographs. Assuming that $\boldsymbol{B}$ has a constant row weight of d and hence an order-r $(= d - 2)$ Hadamard code is used, it can be readily shown that the code rates of the SC-PLDPCH codes are as follows. For SC-PLDPCH-TDCs, the code rate equals

$$\begin{aligned} R^{\text{even}}_{\text{SC-PLDPCH-TDC}} &= \frac{nL - m(L+W)}{nL + m(L+W)(2^r - d)} \\ &= \frac{n - m\left(1 + \frac{W}{L}\right)}{n + m\left(1 + \frac{W}{L}\right)(2^r - d)} \end{aligned} \tag{85}$$

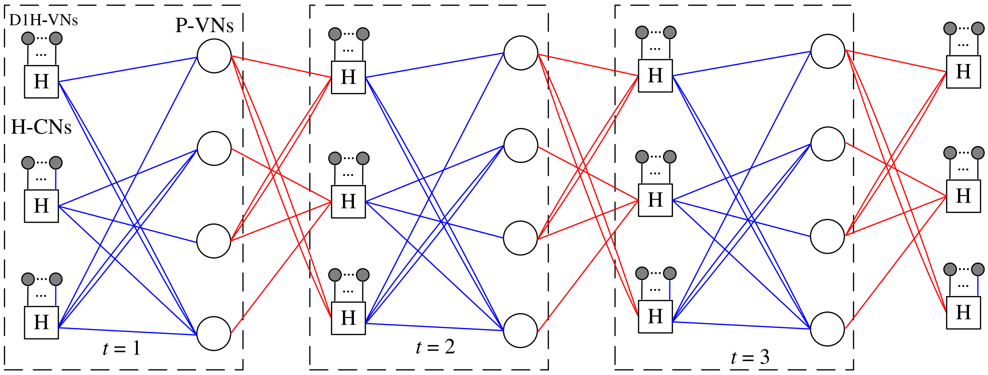

Figure 49: Protograph of a SC-PLDPCH-TDC derived by terminating SC-PLDPCH-CC protograph in Fig. 48. $W = 1$ and $L = 3$.

when r is even, and

$$\begin{aligned} R^{\text{odd}}_{\text{SC-PLDPCH-TDC}} &= \frac{nL - m(L+W)}{nL + m(L+W)(2^r - 2)} \\ &= \frac{n - m\left(1 + \frac{W}{L}\right)}{n + m\left(1 + \frac{W}{L}\right)(2^r - 2)} \end{aligned} \tag{86}$$

when r is odd. For SC-PLDPCH-TBCs and SC-PLDPCH-CCs, their code rates are the same as the block code counterparts, i.e.,

$$\begin{aligned} R^{\text{even}}_{\text{SC-PLDPCH-TBC}} &= R^{\text{even}}_{\text{SC-PLDPCH-CC}} \\ = R^{\text{even}}_{\text{PLDPCH-BC}} &= \frac{n - m}{n + m(2^r - r - 2)} \end{aligned} \tag{87}$$

when r is even, and

$$\begin{aligned} R^{\text{odd}}_{\text{SC-PLDPCH-TBC}} &= R^{\text{odd}}_{\text{SC-PLDPCH-CC}} \\ = R^{\text{odd}}_{\text{PLDPCH-BC}} &= \frac{n - m}{n + m(2^r - 2)} \end{aligned} \tag{88}$$

when r is odd.

5.2 ENCODING OF SC-PLDPCH-CC

From this point forward and unless otherwise stated, we focus our study on SC-PLDPCH-CC. We also assume that the row weight of $\boldsymbol{B}$ equals $d = r + 2$ and is even. After performing a two-step lifting process on (81), we obtain the semi-infinite parity-check matrix of a SC-PLDPCH-CC in Fig. 51.

Denoting the two lifting factors by z_1 and z_2, each $\boldsymbol{H}_i$ ($i = 0, 1, \ldots, W$) has a size of $M \times N = mz_1z_2 \times nz_1z_2$. At time t, $M - N$ information bits denoted by $\boldsymbol{b}(t) \in \{0, 1\}^{M-N}$ are input to the SC-PLDPCH-CC encoder. The output of the SC-PLDPCH-CC encoder contains N coded bits corresponding to P-VNs, which are denoted by

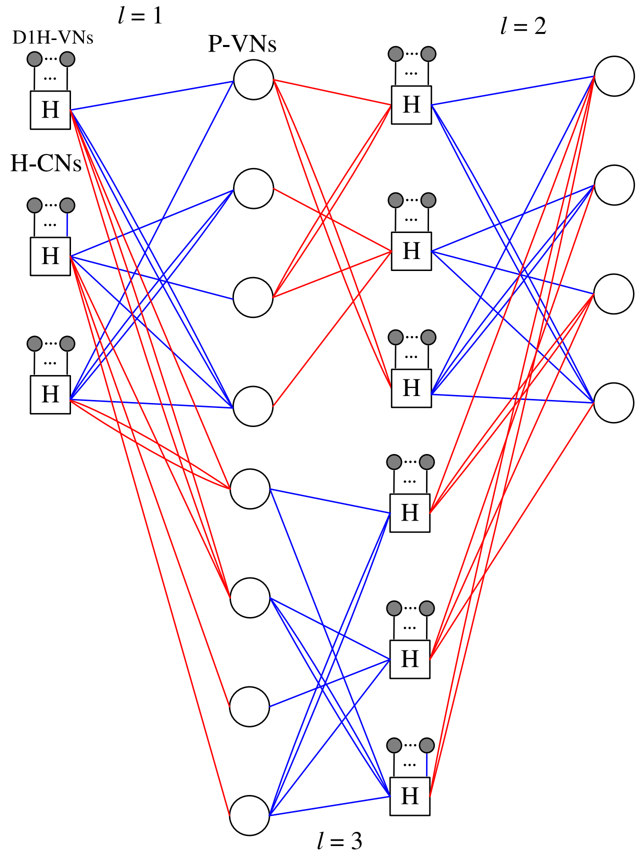

Figure 50: Protograph of a SC-PLDPCH-TBC derived by terminating SC-PLDPCH-CC protograph in Fig. 48. $W = 1$ and $L = 3$.

$\boldsymbol{P}(t)$; and $M(2^r - r - 2)$ Hadamard parity-check bits corresponding to D1H-VNs, which are denoted by $\boldsymbol{D}(t)$. Referring to Fig. 51, we generate the output bits as follows.

1. $t = 1$: Given $\boldsymbol{b}(1)$, $\boldsymbol{P}(1)$ is generated based on the first block row of $\boldsymbol{H}_{\mathrm{SC-PLDPC-CC}}$, i.e., $\boldsymbol{H}_0$. Moreover, $\boldsymbol{D}(1)$ is computed based on $[\overbrace{\mathbf{0} \cdots \mathbf{0}}^{W}\ \boldsymbol{P}(1)]$ and the structure $[\boldsymbol{H}_W\ \cdots\ \boldsymbol{H}_1\ \boldsymbol{H}_0]$, where each $\mathbf{0}$ is a length-N zero vector.

2. $t = 2$: Given $\boldsymbol{b}(2)$ and $\boldsymbol{P}(1)$, $\boldsymbol{P}(2)$ is generated based on the second block row of $\boldsymbol{H}_{\mathrm{SC-PLDPC-CC}}$, i.e., $[\boldsymbol{H}_1\ \boldsymbol{H}_0]$. Moreover, $\boldsymbol{D}(2)$ is computed based on $[\overbrace{\mathbf{0} \cdots \mathbf{0}}^{W-1}\ \boldsymbol{P}(1)\ \boldsymbol{P}(2)]$ and the structure $[\boldsymbol{H}_W\ \cdots\ \boldsymbol{H}_1\ \boldsymbol{H}_0]$.

3. $t \leqslant W$: Given $\boldsymbol{b}(t)$ and $[\boldsymbol{P}(1)\ \boldsymbol{P}(2)\ \cdots\ \boldsymbol{P}(t-1)]$, N coded bits $\boldsymbol{P}(t)$ are generated based on the t-th block row of $\boldsymbol{H}_{\mathrm{SC-PLDPC-CC}}$, i.e., $[\boldsymbol{H}_{t-1}\ \cdots\ \boldsymbol{H}_1\ \boldsymbol{H}_0]$. $\boldsymbol{D}(t)$ corresponding to the $M(2^r - r - 2)$ D1H-VNs are computed based on $[\overbrace{\mathbf{0} \cdots \mathbf{0}}^{W+1-t}\boldsymbol{P}(1)\ \cdots\ \boldsymbol{P}(t)]$ and the structure $[\boldsymbol{H}_W\ \boldsymbol{H}_{W-1}\ \cdots\ \boldsymbol{H}_0]$.

$$\begin{array}{r} \text{Coded bits of P-VNs} \\ \text{Coded bits of D1H-VNs} \end{array} \begin{bmatrix} \boldsymbol{P}(1) \\ \boldsymbol{D}(1) \end{bmatrix} \begin{bmatrix} \boldsymbol{P}(2) \\ \boldsymbol{D}(2) \end{bmatrix} \begin{array}{c} \cdots \\ \cdots \end{array} \begin{bmatrix} \boldsymbol{P}(t-1) \\ \boldsymbol{D}(t-1) \end{bmatrix} \begin{bmatrix} \boldsymbol{P}(t) \\ \boldsymbol{D}(t) \end{bmatrix} \begin{array}{c} \cdots \\ \cdots \end{array}$$

$$\boldsymbol{H}_{\text{SC-PLDPCH-CC}} = \begin{bmatrix} \boldsymbol{H}_0 & & & & & \\ \boldsymbol{H}_1 & \boldsymbol{H}_0 & & & & \\ \vdots & \boldsymbol{H}_1 & \ddots & & & \\ \boldsymbol{H}_W & \vdots & \ddots & \boldsymbol{H}_0 & & \\ & \boldsymbol{H}_W & \ddots & \boldsymbol{H}_1 & \boldsymbol{H}_0 & \\ & & \ddots & \vdots & \boldsymbol{H}_1 & \ddots \\ & & & \boldsymbol{H}_W & \vdots & \ddots \\ & & & & \boldsymbol{H}_W & \ddots \\ & & & & & \ddots \end{bmatrix}$$

Figure 51: Encoding of a SC-PLDPCH-CC. Coded bits $\boldsymbol{P}(1)$, $\boldsymbol{P}(2)$, ..., $\boldsymbol{P}(t-1)$, $\boldsymbol{P}(t)$, ... correspond to P-VNs at time $1, 2, \ldots, t-1, t, \ldots$. Hadamard parity-check bits $\boldsymbol{D}(1), \boldsymbol{D}(2), \ldots, \boldsymbol{D}(t-1), \boldsymbol{D}(t), \ldots$ correspond to D1H-VNs at time $1, 2, \ldots, t-1, t, \ldots$.

4. $t > W$: Given $\boldsymbol{b}(t)$ and $[\boldsymbol{P}(t-W)\ \boldsymbol{P}(t-W+1)\ \cdots\ \boldsymbol{P}(t-1)]$, $\boldsymbol{P}(t)$ is generated based on the t-th block row of $\boldsymbol{H}_{\text{SC-PLDPC-CC}}$, i.e., $[\boldsymbol{H}_W\ \cdots\ \boldsymbol{H}_1\ \boldsymbol{H}_0]$. Then, $\boldsymbol{D}(t)$ is computed based on $[\boldsymbol{P}(t-W)\ \cdots\ \boldsymbol{P}(t-1)\ \boldsymbol{P}(t)]$ and the structure $[\boldsymbol{H}_W\ \boldsymbol{H}_{W-1}\ \cdots\ \boldsymbol{H}_0]$.

Remarks: The values of $\boldsymbol{D}(t)$ are generated during the encoding corresponding to the t-th block row. They are not needed for generating other $\boldsymbol{D}(t')$ where $t \neq t'$. When $t \leqslant W$, $W+1-t$ length-N zero vectors are inserted in front of $\boldsymbol{P}(1)$ for computing $\boldsymbol{D}(t)$. But these zero vectors are not transmitted through the channel.

5.3 PIPELINE DECODING

At the receiving end, we receive channel observations regarding the coded bits $\boldsymbol{P}(t)$ (corresponding to P-VNs) and Hadamard parity-check bits $\boldsymbol{D}(t)$ (corresponding to D1H-VNs). We denote the log-likelihood-ratio (LLR) values corresponding to $\boldsymbol{P}(t)$ by $\boldsymbol{L}^{\text{P}}_{\text{ch}}(t)$ and the LLR values corresponding to $\boldsymbol{D}(t)$ by $\boldsymbol{L}^{\text{D}}_{\text{ch}}(t)$. We consider a pipeline decoder which consists of I identical message-passing processors [41, 43, 102]. Each processor is a PLDPC-Hadamard block sub-decoder corresponding to $[\boldsymbol{H}_W\ \boldsymbol{H}_{W-1}\ \cdots\ \boldsymbol{H}_0]$. Thus, each processor operates on $W+1$ sets of P-VNs and one set of D1H-VNs each time, i.e., a total of $N(W+1)$ P-VNs and $M(2^r-r-2)$ D1H-VNs (when r is even). Hence the pipeline decoder operates on $(W+1)I$ sets of P-VNs and I sets of D1H-VNs each time. Each processor (sub-

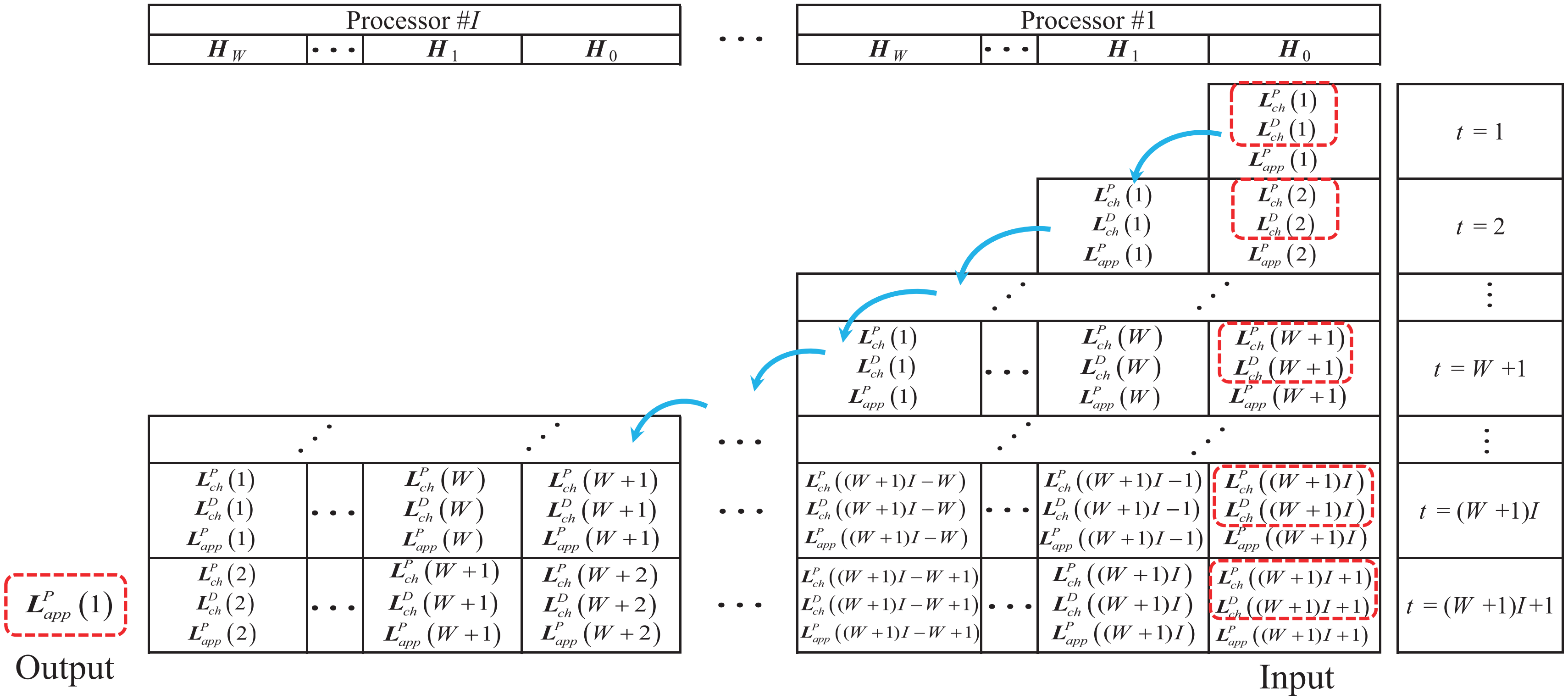

Figure 52: Structure of a pipeline SC-PLDPCH-CC decoder consisting of I processors (PLDPC-Hadamard block sub-decoders). $\{\boldsymbol{L}^{\mathrm{P}}_{\mathrm{ch}}(t), \boldsymbol{L}^{\mathrm{D}}_{\mathrm{ch}}(t)\}$ $(t = 1, 2, \ldots)$ are input into the decoder one set by one set. Every time, all sets of LLRs inside the decoder are shifted to the left, and all APP-LLRs of all P-VNs inside the different I processors are updated. When $\{\boldsymbol{L}^{\mathrm{P}}_{\mathrm{ch}}((W+1)I+t'), \boldsymbol{L}^{\mathrm{D}}_{\mathrm{ch}}((W+1)I+t')\}$ $(t' = 1, 2, \ldots)$ is input to the decoder, the APP-LLRs $\boldsymbol{L}^{\mathrm{P}}_{\mathrm{app}}(t')$ are output and the values of the coded bits $\boldsymbol{P}(t')$ are determined.

decoder) can apply either the standard decoding algorithm or the layered decoding algorithm to compute/update the *a posteriori* LLR (APP-LLR) values of the coded bits $\boldsymbol{P}(t)$ and the related extrinsic LLR information. Here, we apply the layered decoding algorithm (See the details in Section 4.2.1) [96] in each of these PLDPC-Hadamard block sub-decoders.

We denote the APP-LLR values of the coded bits $\boldsymbol{P}(t)$ by $\boldsymbol{L}^{\mathrm{P}}_{\mathrm{app}}(t)$. Referring to Fig. 52, $\{\boldsymbol{L}^{\mathrm{P}}_{\mathrm{ch}}(1), \boldsymbol{L}^{\mathrm{D}}_{\mathrm{ch}}(1)\}$ is first input to the pipeline decoder and Processor #1 updates the APP-LLR of all P-VNs inside, i.e., $\boldsymbol{L}^{\mathrm{P}}_{\mathrm{app}}(1)$. In addition, extrinsic LLR information is updated and stored in the processor but is not depicted in the figure. Then, $\{\boldsymbol{L}^{\mathrm{P}}_{\mathrm{ch}}(2), \boldsymbol{L}^{\mathrm{D}}_{\mathrm{ch}}(2)\}$ is input to the pipeline decoder while $\{\boldsymbol{L}^{\mathrm{P}}_{\mathrm{ch}}(1), \boldsymbol{L}^{\mathrm{D}}_{\mathrm{ch}}(1), \boldsymbol{L}^{\mathrm{P}}_{\mathrm{app}}(1)\}$ and related extrinsic LLR information are shifted to the left in the decoder. Processor #1 updates the APP-LLRs of all P-VNs inside, i.e., $\boldsymbol{L}^{\mathrm{P}}_{\mathrm{app}}(1)$ and $\boldsymbol{L}^{\mathrm{P}}_{\mathrm{app}}(2)$. Again, extrinsic LLR information is updated and stored in the processor but is not depicted. Subsequently, $\{\boldsymbol{L}^{\mathrm{P}}_{\mathrm{ch}}(t), \boldsymbol{L}^{\mathrm{D}}_{\mathrm{ch}}(t)\}$ $(t = 3, 4, \ldots)$ are input into the decoder one set by one set. Every time, all sets of LLRs inside the decoder are shifted to the left by one "$\boldsymbol{H}_i$" block, and all APP-LLRs of all P-VNs inside the different I processors are updated. Referring to Fig. 52, when $\{\boldsymbol{L}^{\mathrm{P}}_{\mathrm{ch}}((W+1)I+1), \boldsymbol{L}^{\mathrm{D}}_{\mathrm{ch}}((W+1)I+1)\}$ is input to the pipeline decoder, the APP-LLRs $\boldsymbol{L}^{\mathrm{P}}_{\mathrm{app}}(1)$ have gone through the iterative process and are output from the decoder. Hard decisions are made based on these APP-LLRs to determine the values of the coded bits $\boldsymbol{P}(1)$. The process continues and every time $\{\boldsymbol{L}^{\mathrm{P}}_{\mathrm{ch}}((W+1)I+t'), \boldsymbol{L}^{\mathrm{D}}_{\mathrm{ch}}((W+1)I+t')\}$ $(t' = 1, 2, \ldots)$ is input to the decoder, the APP-LLRs $\boldsymbol{L}^{\mathrm{P}}_{\mathrm{app}}(t')$ are output and the values of the coded bits $\boldsymbol{P}(t')$ are determined.

5.4 SIMULATION RESULTS

We set $W = 1$ in our simulations. We use the edge spreading procedure to randomly split the optimized protomatrix $\boldsymbol{B}$ of PLDPCH-BCs (obtained in Section 3.4) into $\boldsymbol{B}_0$ and $\boldsymbol{B}_1$, where $\boldsymbol{B}_0 + \boldsymbol{B}_1 = \boldsymbol{B}$. Following Section 5.1, we use $\boldsymbol{B}_0$ and $\boldsymbol{B}_1$ to construct the protomatrix of a SC-PLDPCH-CC. Using the two-step lifting method, we lift the protomatrix to obtain a convolutional parity-check matrix, where the two lifted factors are denoted as z_1 and z_2, respectively. We use binary phase-shift-keying (BPSK) modulation over an AWGN channel. Based on the lifted matrix, we apply the pipeline decoder with the layered decoding algorithm to evaluate the error performance of the constructed SC-PLDPCH-CC.

5.4.1 *Rate*-0.0494 *and* $r = 4$

Based on the 7×11 protomatrix

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \\ 2 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 \\ 3 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 2 & 0 \end{bmatrix} \tag{89}$$

of the optimized rate-0.0494 PLDPCH-BC in Section 3.4.1.1 [93, 94], we find two 7×11 protomatrices

$$\boldsymbol{B}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{90}$$

and

$$\boldsymbol{B}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}. \tag{91}$$

We use the lifting factors $z_1 = 16$ and $z_2 = 1024$ to expand the protomatrix such that the sub-block length of the SC-PLDPCH-CC equals $1,327,104$, which is identical to the code length of the PLDPCH-BC with $z_1 = 32$ and $z_2 = 512$, i.e., $N + M(2^r - r - 2) = 1,327,104$. Table 14 in Appendix D shows the details of lifted matrix corresponding to $[\boldsymbol{B}_1\ \boldsymbol{B}_0]$. The BER performance of the SC-PLDPCH-CC with different number of processors I contained in pipeline decoding is shown in Fig. 53. We observe that the decoder with $I = 80$ processors in pipeline decoding achieves a BER of 10^{-5} at about $E_b/N_0 = -1.235$ dB, which outperforms that with $I = 70$ by about 0.03 dB, and that with $I = 60$ by about 0.06 dB. In the same figure, we also see that the SC-PLDPCH-CC outperforms the PLDPCH-BC using

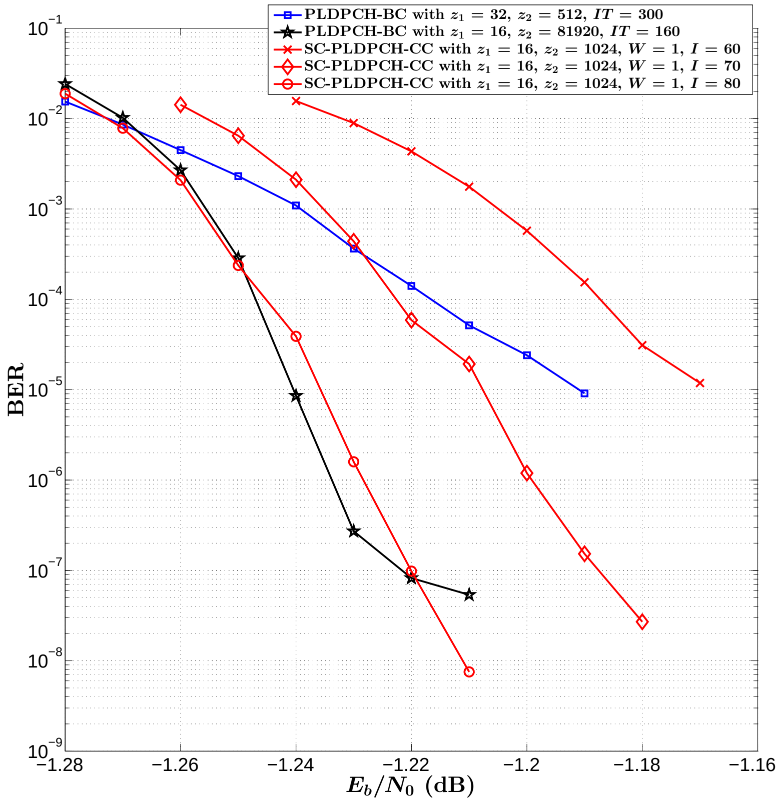

Figure 53: BER performance comparison between the rate-0.0494 PLDPCH-BC and rate-0.0494 SC-PLDPCH-CC. $r = 4$.

300 standard decoding iterations (equivalent to 150 layered decoding iterations) by about 0.045 dB at a BER of 10^{-5}. The gaps of the SC-PLDPCH-CC (with $I = 80$ and BER of 10^{-5}) to the Shannon capacity (-1.44 dB) of $R = 0.05$ and to the ultimate Shannon limit (-1.59 dB) are about 0.205 dB and 0.355 dB, respectively.

As mentioned in Section 5.3, pipeline decoding with I processors operates on $(W + 1)I$ sets of P-VNs and I sets of D1H-VNs. When the product of the two lifting factors ($z_1 \times z_2$) is the same, pipeline decoding involves more P-VNs and D1H-VNs than PLDPCH-BC decoding. To increase the number of P-VNs and D1H-VNs for the rate-0.0494 PLDPCH-BC, we increase the code length of PLDPCH-BC by 80 times, i.e., $z_1 = 16$ and $z_2 = 81920$ ($= 1024 \times 80$). The BER result of the lengthened PLDPCH-BC using 160 standard decoding iterations (equivalent to 80 layered decoding iterations) is shown in Fig. 53. The lengthened PLDPCH-BC slightly outperforms the SC-PLDPCH-CC with $I = 80$ but suffers from an error floor at a BER of 10^{-7}. However, no error floor is observed at a BER of 10^{-8} for the proposed SC-PLDPCH-CC. We further study the lifted matrices corresponding to the PLDPCH-BC and the SC-PLDPCH-CC.

We find that both matrices have a girth of 10 (girth refers to the minimum cycle length and plays an important role in the error-floor performance of LDPC-based codes). Thus the SC-PLDPCH-CC has an advantage over the PLDPCH-BC in terms of error floor even though both codes have the same girth.

5.4.2 *Rate*-0.021 *and* $r = 5$

Based on the 6×10 protomatrix

$$\boldsymbol{B} = \begin{bmatrix} 3 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 2 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 1 & 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 3 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & 2 & 0 & 2 \\ 2 & 0 & 1 & 1 & 0 & 0 & 0 & 2 & 0 & 1 \end{bmatrix} \tag{92}$$

of the optimized rate-0.021 PLDPCH-BC in Section 3.4.1.2 [93, 94], we find two 6×10 protomatrices

$$\boldsymbol{B}_0 = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{93}$$

and

$$\boldsymbol{B}_1 = \begin{bmatrix} 1 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 2 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{94}$$

We use the same lifting factors, i.e., $z_1 = 32$ and $z_2 = 512$, as those used in the PLDPCH-BC to expand the protomatrix such that the sub-block length of the SC-PLDPCH-CC equals $3,112,960$. The BER performance of the SC-PLDPCH-CC with different number of processors I is shown in Fig. 54. The pipeline decoder with $I = 80$ processors achieves a BER of 10^{-5} at about $E_b/N_0 = -1.30$ dB, which outperforms that with $I = 70$ by about 0.02 dB, and that with $I = 60$ by about 0.05 dB. In the same figure, we also see that the SC-PLDPCH-CC outperforms the PLDPCH-BC using 300 standard

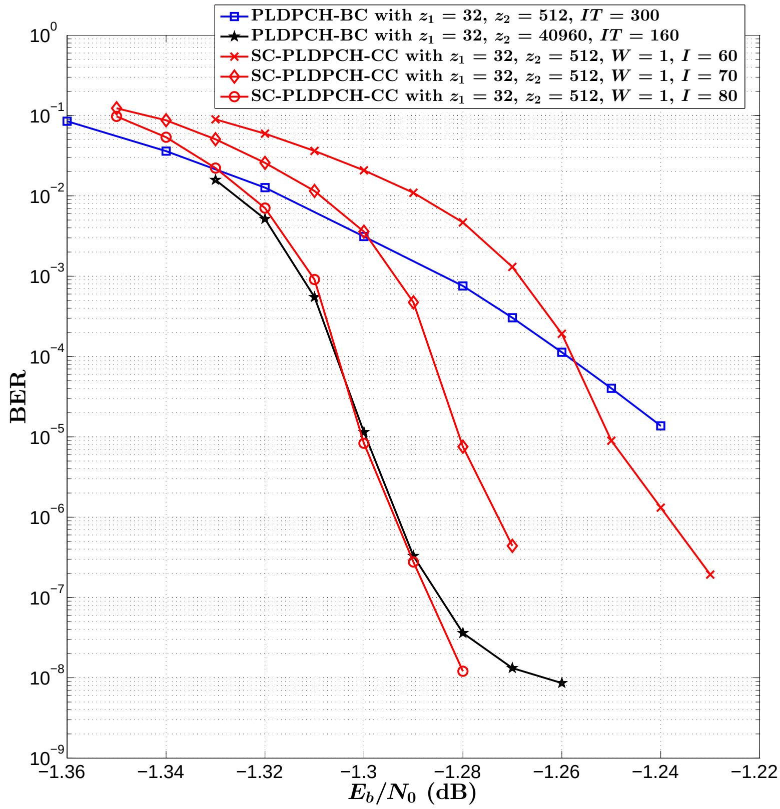

Figure 54: BER performance comparison between the rate-0.021 PLDPCH-BC and rate-0.021 SC-PLDPCH-CC. $r = 5$.

decoding iterations by about 0.06 dB at a BER of 10^{-5}. The gaps of the SC-PLDPCH-CC (with $I = 80$ and BER of 10^{-5}) to the Shannon capacity (-1.53 dB) of $R = 0.02$ and to the ultimate Shannon limit (-1.59 dB) are about 0.23 dB and 0.29 dB, respectively.

We again increase the code length of PLDPCH-BC by 80 times, i.e., $z_1 = 32$ and $z_2 = 40960$ ($= 512 \times 80$). The BER result of the lengthened PLDPCH-BC using 160 standard decoding iterations is shown in Fig. 54. The lengthened PLDPCH-BC performs similarly as the SC-PLDPCH-CC with $I = 80$ but suffers from an error floor at a BER of 4×10^{-8}. However, no error floor is observed at a BER of 10^{-8} for the proposed SC-PLDPCH-CC. Both the lifted matrices corresponding to the PLDPCH-BC and the SC-PLDPCH-CC are found to have a girth of 10.

5.4.3 *Rate*-0.008 *and* $r = 8$

The 5×15 protomatrix $\boldsymbol{B}$ of the optimized rate-0.008 PLDPCH-BC in Section 3.4.1.3 [93, 94] and the two protomatrices $\boldsymbol{B}_0$ and $\boldsymbol{B}_1$ are shown in (95), (96) and (97), respectively.

$$\boldsymbol{B} = \begin{bmatrix} 2 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 2 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 3 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 0 & 0 & 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 2 & 3 & 0 & 0 \end{bmatrix} \tag{95}$$

$$\boldsymbol{B}_0 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{96}$$

$$\boldsymbol{B}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 2 & 3 & 0 & 0 \end{bmatrix} \tag{97}$$

We lift the SC-PLDPCH-CC with factors $z_1 = 16$ and $z_2 = 1280$ such that its sub-block length is the same as that of the PLDPCH-BC in Section 3.4.1.3 [93, 94]. Fig. 55 shows the BER performance of the two codes. SC-PLDPCH-CC pipeline decoder with $I = 100$ processors achieves a BER of 10^{-5} at about $E_b/N_0 = -1.40$ dB, which outperforms that with $I = 90$ by about 0.01 dB, and that with $I = 80$ by about 0.03 dB. It does not suffer from any error floor down to a BER of 10^{-8}. It also outperforms the PLDPCH-BC using 300 standard decoding iterations by about 0.05 dB at a BER of 10^{-5}. At a BER of 10^{-5}, the gaps (for the SC-PLDPCH-CC with $I = 100$ iterations) to the Shannon capacity (−1.57 dB) of $R = 0.008$ and to the ultimate Shannon limit (−1.59) dB are 0.17 dB and 0.19 dB, respectively.

5.4.4 *Rate*-0.00295 *and* $r = 10$

The 6×24 protomatrix $\boldsymbol{B}$ of the optimized rate-0.00295 PLDPCH-BC in Section 3.4.1.4 [93, 94], and the two split protomatrices $\boldsymbol{B}_0$ and $\boldsymbol{B}_1$ are shown in (98), (99) and (100), respectively. We lift the SC-PLDPCH-CC with factors $z_1 = 20$ and $z_2 = 1280$ such that its sub-block length is the same as that of the PLDPCH-BC in Section 3.4.1.4 [93, 94]. Fig. 56 shows the BER performance of the two codes. SC-PLDPCH-CC decoder with $I = 140$ processors achieves a BER of 10^{-5} at about

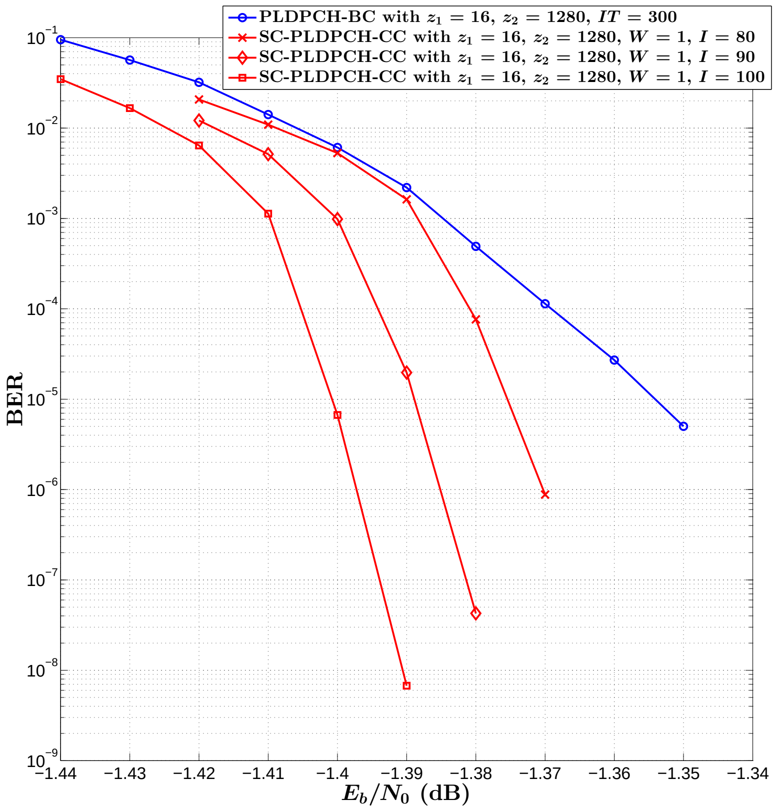

Figure 55: BER performance comparison between the rate-0.008 PLDPCH-BC and rate-0.008 SC-PLDPCH-CC. $r = 8$.

$E_b/N_0 = -1.46$ dB, which outperforms that with $I = 120$ by about 0.01 dB, and that with $I = 100$ by about 0.03 dB. It does not suffer from any error floor down to a BER of 2×10^{-7}. It also outperforms the PLDPCH-BC using 300 standard decoding iterations by about 0.03 dB at a BER of 10^{-5}. At a BER of 10^{-5}, the gaps (for the SC-PLDPCH-CC with $I = 140$ iterations) to the Shannon capacity (−1.58 dB) of $R = 0.003$ and to the ultimate Shannon limit (−1.59) dB are 0.12 dB and 0.13 dB, respectively.

$$\boldsymbol{B} = \begin{bmatrix} 1&0&0&0&0&0&3&0&0&0&0&0&0&0&2&0&0&0&0&1&4&0&1&0 \\ 0&0&0&3&2&0&0&0&1&1&0&0&1&0&0&0&3&1&0&0&0&0&0&0 \\ 0&1&2&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&3&0&4&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&3&3&0&0&0&0&0&0&0&0&2&2&0&1 \\ 2&2&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&3&0&0&1&1 \\ 0&0&0&0&0&1&0&3&3&2&0&0&1&1&0&0&0&0&0&0&0&1&0&0 \end{bmatrix} \quad (98)$$

$$
\boldsymbol{B}_0 =
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 &
\end{bmatrix}
\tag{99}
$$

$$
\boldsymbol{B}_1 =
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 \\
0 & 0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 2 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\tag{100}
$$

Remark: Compared with long length PLDPCH-BC, our proposed convolutional codes possess lower error floor. The error floor for channel codes is related to the minimum distance. Paper [45] shows that when using spatially coupled method to construct spatially coupled LDPC ensembles, the minimum distance can grow linearly with block length such that this property can suppress error floor. We think our spatially coupled PLDPCH convolutional codes also inherit such property and hence promise low error floor in high BER region.

5.5 summary

In this chapter, we have derived another type of ultimate-Shannon-limit-approaching code called spatially coupled PLDPC-Hadamard convolutional codes (SC-PLDPCH-CCs). As the name implies, SC-PLDPCH-CCs are formed by spatially coupling PLDPCH block codes (PLDPCH-BCs). We develop a pipeline decoding with the layered decoding algorithm to efficiently and effectively decode SC-PLDPCH-CCs. Based on the protograph of a PLDPCH-BC, we have found the protomatrices of good SC-PLDPCH-CCs with rates 0.0494, 0.021, 0.008 and 0.00295. When the product of two lifting factors is the same, these found SC-PLDPCH-CCs outperform their block code counterparts in terms of bit error performance. Moreover, at a BER

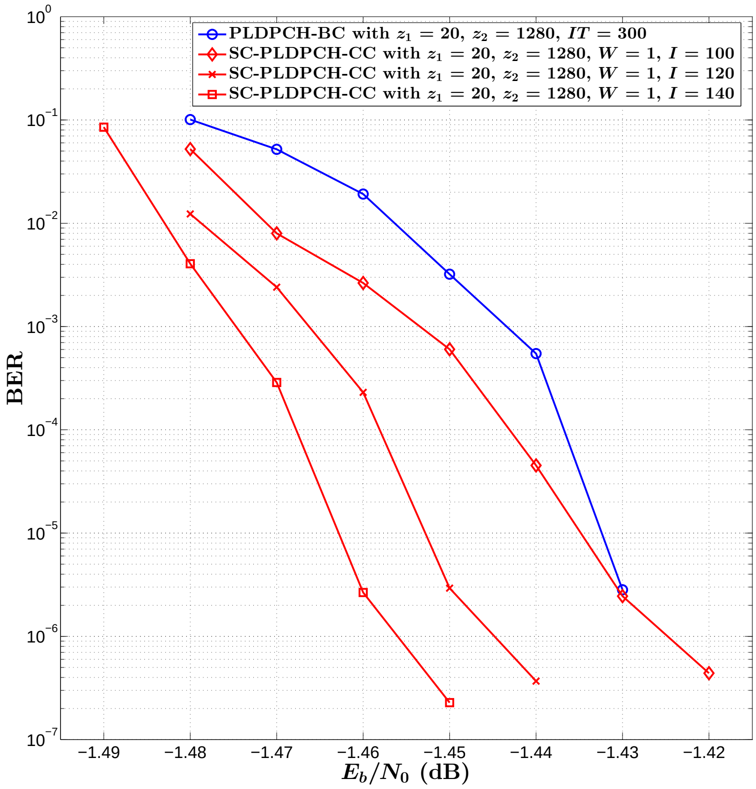

Figure 56: BER performance comparison between the rate-0.00295 PLDPCH-BC and rate-0.00295 SC-PLDPCH-CC. $r = 10$.

of 10^{-5}, the SC-PLDPCH-CCs of rates 0.0494, 0.021, 0.008 and 0.00295 are only 0.36 dB, 0.29 dB, 0.19 dB and 0.13 dB from the ultimate Shannon limit, i.e., −1.59 dB. They are also the closest to ultimate Shannon limit in these four code rates compared with other published results.

Part IV

CONCLUSIONS AND FUTURE WORK

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

The ultimate-Shannon-limit approaching channel codes can be applied in space communications, multiple access with severe inter-user interferences. Among the existing channel codes with error performance close to this limit, LDPC-Hadamard codes are one of the competitive candidate codes because they allow parallel processing and hence can achieve low decoding latency. However, the traditional LDPC-Hadamard block codes are unstructured and the corresponding EXIT chart analysis method is not valid for codes with degree-1 or/and punctured variable nodes. To analyze and design LDPC-Hadamard codes more comprehensively, we have proposed protograph-based LDPC-Hadamard block codes in Chapter 3. Moreover, we have proposed an efficient layered decoding algorithm for PLDPCH-BCs and the corresponding hardware architecture in Chapter 4. To further approach the ultimate Shannon limit, we have proposed the spatially coupled PLDPC-Hadamard convolutional codes in Chapter 5. We conclude our contributions as follows.

- In Chapter 3, we have proposed a new type of ultimate-Shannon-limit-approaching channel codes, i.e., protograph-based LDPC-Hadamard block codes (PLDPCH-BCs). Unlike traditional LDPC-Hadamard block codes designed by degree distributions, we design LDPC-Hadamard codes from the perspective of protographs. We have proposed a low-complexity PEXIT chart method to evaluate the threshold of PLDPCH-BCs, which can effectively analyze protographs containing degree-1 or/and punctured P-VNs. Based on the analysis method, we have proposed optimization criterion to deign the PLDPCH-BCs. Using the proposed method, we have found good PLDPCH-BCs with different code rates and very low thresholds (< -1.40 dB). We have shown that our proposed PLDPCH-BCs can achieve comparable error performance to the tradition LDPC-Hadamard block codes. We have also studied punctured PLDPCH-BCs. We have observed that puncturing the P-VNs with different degrees produces different BER/FER performance while puncturing extra D1H-VNs (when $r = 5$) degrades the error performance quite significantly.

- In Chapter 4, we have proposed a layered decoding algorithm to fast decode PLDPCH-BCs and have implemented the layered decoder of PLDPCH-BCs onto an FPGA board. Compared with the standard decoding algorithm, our proposed layered decoding algorithm not only consumes less memory storage and computational logic, but also improves the convergence rate by about two times. Based on the layered decoding algorithm, we have proposed a hardware architecture of the PLDPC-Hadamard layered decoder, and analyzed the latency and throughput of the decoder. We have shown that a throughput of 1.48 Gbps is achieved when 20 decoding iterations are used. If we fully utilize the Hadamard sub-decoders, the throughput will be increased to almost 4.5 Gbps. Moreover, the proposed even-order PLDPCH-BC decoder architecture is generic and can be readily applied to odd-order PLDPCH-BC decoders.

- In Chapter 5, we make use of PLDPCH-BCs proposed in Chapter 3 to design another type of ultimate-Shannon-limit-approaching channel codes, i.e., spatially coupled PLDPC-Hadamard convolutional codes (SC-PLDPCH-CCs). We have described the code constructions for three types of SC-PLDPCH codes, i.e., spatially coupled PLDPC-Hadamard terminated codes (SC-PLDPCH-TDCs), spatially coupled PLDPC-Hadamard tail-biting codes (SC-PLDPCH-TBCs) and SC-PLDPCH-CCs. We introduce the encoding of a SC-PLDPCH-CC based on its convolutional parity-check matrix, which is derived by lifting the protomatrix. We have proposed an effective pipeline decoder with layered decoding processors to evaluate the error performance of SC-PLDPCH-CCs. We have shown that our SC-PLDPCH-CCs can outperform their block code counterparts in terms of bit error performance. The BER performance of these SC-PLDPCH-CCs (with rates of 0.0494, 0.021, 0.008 and 0.00295) is the closest to ultimate Shannon limit compared with other published results.

6.2 FUTURE WORK

Based on the research in this thesis, we list the following future work that can be performed.

- In Chapter 3, by directly puncturing our proposed PLDPC-Hadamard codes, punctured codes are obtained and evaluated. However, these punctured codes, strictly speaking, are not optimized. In the future, we plan to apply the proposed analytical technique to find optimal PLDPC-Hadamard codes with punctured VNs and compare their results with those presented in this thesis.

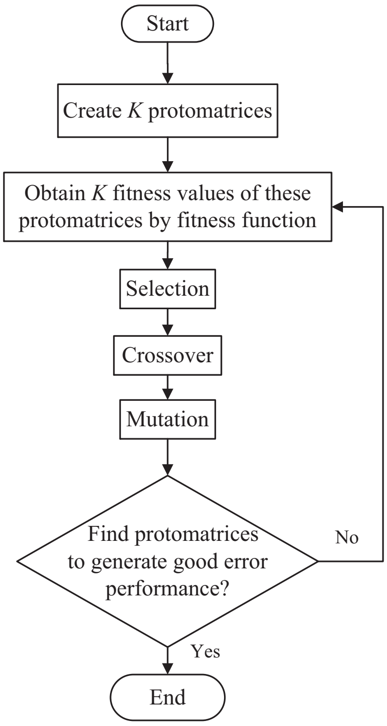

Figure 57: The flowchart of genetic algorithm for finding optimal protomatrices.

- In Chapter 4, the PLDPCH-BC layered decoder uses 128 symbol-MAP Hadamard sub-decoders in parallel in order to obtain a high throughput. The design also consumes a lot of look-up tables of the FPGA board. Another possible future work is therefore to simplify the hardware of the Hadamard sub-decoder with an aim to reducing the look-up-table utilization. During this simplification, however, minimal performance loss should be allowed.

- In Chapter 5, the BER performance of SC-PLDPCH-CCs are simulated but the theoretical thresholds are not derived. In the future, we can propose analytical techniques to derive the thresholds of SC-PLDPCH-CCs and compare their accuracies with the simulation results.

- In Chapters 3 and 5, the optimized protomatrices of PLDPCH-BCs and SC-PLDPCH-CCs are found through random searches. In the future, we can investigate annealing approaches or genetic algorithms to systematically search for optimal protomatrices under some given constraints. Fig. 57 is a basic procedure of genetic algorithm to find optimal protomatrices based on generation group, fitness function, selection, crossover and mutation.

Part V

APPENDICES

Appendix A

TWO OTHER TYPES OF LDPC-HADAMARD CODES

As mentioned in Section 3.1, $d_{c_i} = r+1$ bits from P-VNs need to fulfill the SPC constraint. However, if the inputs to the H-CNs are not required to satisfy the SPC constraint, two other types of codes can be formed.

Fig. A1 shows the first type, in which the information bits (information VNs) are first encoded into an LDPC codeword (with the generation of the parity-check VNs) based on the SPC constraints (SPC-CNs). Subsequently, these VNs (including both information VNs and parity-check VNs) are repeated and interleaved. Then they are used as inputs to the Hadamard check nodes (H-CNs) and to generate the Hadamard parity-check bits (D1H-VNs). Suppose the order of the Hadamard codes used is r and hence there are 2^{r+1} possible Hadamard codewords. As the inputs to the Hadamard check nodes may not satisfy the SPC constraint, the number of inputs would be $r+1$ (instead of $r+2$ in our PLDPC-Hadamard code) and the number of Hadamard parity-check bits (D1H-VNs) generated in each H-CN equals $2^r-(r+1)$ (instead of $2^r-(r+2)$ in our PLDPC-Hadamard code when r is even). Compared with our PLDPC-Hadamard code, the code in Fig. A1 will have a lower code rate when r is even. The decoder structure of the code in Fig. A1 will also be different from ours. The decoder structure of the code in Fig. A1 will consist of a traditional LDPC decoder and a Hadamard decoder, which will iteratively exchange the extrinsic information of the variable nodes (i.e., the VNs shown in the middle layer of Fig. A1). There will be also two interleavers in the code in Fig. A1, as opposed to only one interleaver in our PLDPC-Hadamard code. Thus the decoder is more complicated compared with ours.

Fig. A2 depicts the second type of code in which the SPC constraints are not required. In this case, the information bits (shown as VNs at the top) are repeated and interleaved. Then they are used as inputs to the Hadamard check nodes (H-CNs) and to generate the Hadamard parity-check bits (D1H-VNs). The code can be viewed as a concatenation of repeat codes and Hadamard codes, and the code structure is very different from our PLDPC-Hadamard code.

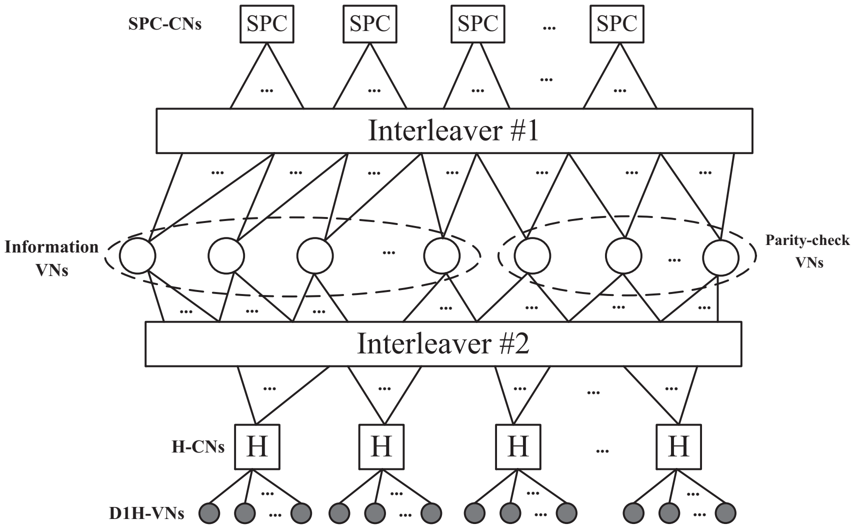

Figure A1: First type of code in which the inputs to the H-CNs do not need to satisfy the SPC constraint.

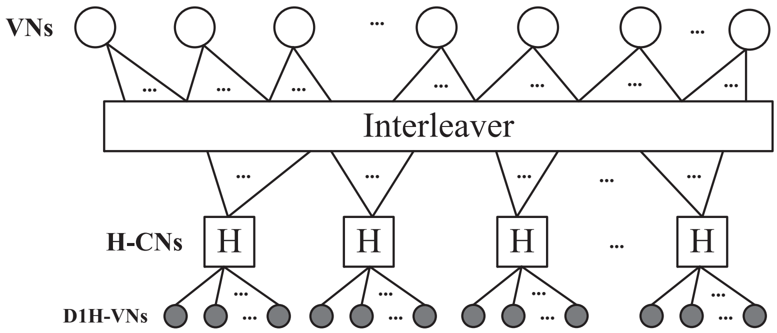

Figure A2: Second type of code in which the inputs to the H-CNs do not need to satisfy the SPC constraint.

Appendix B

COMPUTING APP LLRS OF INFORMATION BITS IN A NON-SYSTEMIC HADAMARD CODE

We convert the extrinsic LLR values $\boldsymbol{L}_{ex}^{H}$ of the SPC code bits $\boldsymbol{c}_{\mu}$ to the *a priori* LLR values of the information bits $\boldsymbol{c}'_{\mu}$ in $\boldsymbol{c}^{H}$. In (45), $c_0^H = c'_{\mu_0} = c_{\mu_0}$ and $c^H_{2^r-1} = c'_{\mu_{r+1}} = c_{\mu_{r+1}}$. Hence, the *a priori* information for $c_0^H = c'_{\mu_0}$ and $c^H_{2^r-1} = c'_{\mu_{r+1}}$ equals the extrinsic information for c_{μ_0} and $c_{\mu_{r+1}}$, that is,

$$\begin{aligned} L_{apr}^{H}(0) &= L_{ex}^{R}(0); \\ L_{apr}^{H}(2^r-1) &= L_{ex}^{R}(r+1). \end{aligned} \tag{B1}$$

For $k = 1, 2, \dots, r$, (45) shows that $c^H_{2^{k-1}} = c'_{\mu_k} = c_{\mu_k} \oplus c_{\mu_0}$. $L_{ex}^{R}(k)$ is the *a priori* information for c_{μ_k}, i.e.,

$$L_{ex}^{R}(k) = \ln \frac{\Pr(c_{\mu_k} = \text{“}0\text{"})}{\Pr(c_{\mu_k} = \text{“}1\text{"})} = \ln \frac{\Pr(c'_{\mu_k} \oplus c_{\mu_0} = \text{“}0\text{"})}{\Pr(c'_{\mu_k} \oplus c_{\mu_0} = \text{“}1\text{"})}. \tag{B2}$$

Alternatively,

$$\begin{aligned} L_{apr}^{H}(2^{k-1}) &= \ln \frac{\Pr(c^H_{2^{k-1}} = \text{“}0\text{"})}{\Pr(c^H_{2^{k-1}} = \text{“}1\text{"})} \\ &= \ln \frac{\Pr(c'_{\mu_k} = \text{“}0\text{"})}{\Pr(c'_{\mu_k} = \text{“}1\text{"})} = \ln \frac{\Pr(c_{\mu_k} \oplus c_{\mu_0} = \text{“}0\text{"})}{\Pr(c_{\mu_k} \oplus c_{\mu_0} = \text{“}1\text{"})} \\ &= \begin{cases} L_{ex}^{R}(k) & \text{if } c_{\mu_0} = \text{“}0\text{"} \\ -L_{ex}^{R}(k) & \text{if } c_{\mu_0} = \text{“}1\text{"} \end{cases}. \end{aligned} \tag{B3}$$

The $2^r - r - 2$ remaining L_{apr}^{H} values should be 0. Thus, the assignment of $\boldsymbol{L}_{apr}^{+H}$ (if $c_{\mu_0} = \text{“}0\text{"}$) and $\boldsymbol{L}_{apr}^{-H}$ (if $c_{\mu_0} = \text{“}1\text{"}$) is as follows:

$$\begin{aligned} & L_{apr}^{\pm H}(k) = L_{ex}^{R}(0) && \text{for } k = 0; \\ & \begin{cases} L_{apr}^{+H}(k) = L_{ex}^{R}(i) \\ L_{apr}^{-H}(k) = -L_{ex}^{R}(i) \end{cases} && \text{for } k = 1, 2, \cdots, 2^{i-1}, \cdots, 2^{r-1}; \\ & L_{apr}^{\pm H}(k) = L_{ex}^{R}(r+1) && \text{for } k = 2^r - 1; \\ & L_{apr}^{\pm H}(k) = 0 && \text{for the } 2^r - r - 2 \text{ remaining } k. \end{aligned} \tag{B4}$$

The $2^r - 2$ channel observations corresponding to the code bits c_1^H to $c_{2^r-2}^H$ are received and the assignment of $\boldsymbol{L}_{ch}^H$ is as follows:

$$\begin{cases} L_{ch}^H(k) = \frac{2y_{ch}^H(k)}{\sigma_{ch}^2} & \text{for } k = 1, 2, \cdots, 2^r - 2; \\ L_{ch}^H(k) = \frac{2y_{ch}^H(k)}{\sigma_{ch}^2} = 0 & \text{for } k = 0, 2^r - 1. \end{cases} \tag{B5}$$

c_{μ_0} and $c_{\mu_{r+1}}$: Since $c_{\mu_0} = c'_{\mu_0} = c_0^H$ and $c_{\mu_{r+1}} = c'_{\mu_{r+1}} = c_{2^r-1}^H$ in (45), we can apply DFHT directly to (38) to obtain the *a posteriori* LLR values $L_{app}^H(0)$ for c_{μ_0} and $L_{app}^H(2^r-1)$ for $c_{\mu_{r+1}}$.
c_{μ_k} for $k = 1, 2, \cdots, r$: Based on the relationship between c_{μ_k} and c'_{μ_k}, we derive (B6) for computing $L_{app}^H(2^{k-1})$. Note that the first element in $+\boldsymbol{h}_j$ is always $+1$ (corresponds to bit "c_0^H = "0""). Thus, the term $\sum_{+H[2^{k-1},j]=+1} \Pr(\boldsymbol{c}^H = +\boldsymbol{h}_j \mid \boldsymbol{y}_{ch}^H)$ can be used to compute $\Pr(c_{2^{k-1}}^H = \text{"0"}, c_0^H = \text{"0"} \mid \boldsymbol{y}_{ch}^H)$ in (B6). Using a similar argument, we arrive at the other three summation terms.

$$\begin{aligned}
L_{app}^H(2^{k-1}) &= \ln \frac{\Pr(c_{\mu_k} = \text{"0"} \mid \boldsymbol{y}_{ch}^H)}{\Pr(c_{\mu_k} = \text{"1"} \mid \boldsymbol{y}_{ch}^H)} = \ln \frac{\Pr(c'_{\mu_k} \oplus c_{\mu_0} = \text{"0"} \mid \boldsymbol{y}_{ch}^H)}{\Pr(c'_{\mu_k} \oplus c_{\mu_0} = \text{"1"} \mid \boldsymbol{y}_{ch}^H)} \\
&= \ln \frac{\Pr(c'_{\mu_k} = \text{"0"}, c_{\mu_0} = \text{"0"} \mid \boldsymbol{y}_{ch}^H) + \Pr(c'_{\mu_k} = \text{"1"}, c_{\mu_0} = \text{"1"} \mid \boldsymbol{y}_{ch}^H)}{\Pr(c'_{\mu_k} = \text{"1"}, c_{\mu_0} = \text{"0"} \mid \boldsymbol{y}_{ch}^H) + \Pr(c'_{\mu_k} = \text{"0"}, c_{\mu_0} = \text{"1"} \mid \boldsymbol{y}_{ch}^H)} \\
&= \ln \frac{\Pr(c_{2^{k-1}}^H = \text{"0"}, c_0^H = \text{"0"} \mid \boldsymbol{y}_{ch}^H) + \Pr(c_{2^{k-1}}^H = \text{"1"}, c_0^H = \text{"1"} \mid \boldsymbol{y}_{ch}^H)}{\Pr(c_{2^{k-1}}^H = \text{"1"}, c_0^H = \text{"0"} \mid \boldsymbol{y}_{ch}^H) + \Pr(c_{2^{k-1}}^H = \text{"0"}, c_0^H = \text{"1"} \mid \boldsymbol{y}_{ch}^H)} \\
&= \ln \frac{\sum\limits_{+H[2^{k-1},j]=+1} \Pr(\boldsymbol{c}^H = +\boldsymbol{h}_j \mid \boldsymbol{y}_{ch}^H) + \sum\limits_{-H[2^{k-1},j]=-1} \Pr(\boldsymbol{c}^H = -\boldsymbol{h}_j \mid \boldsymbol{y}_{ch}^H)}{\sum\limits_{+H[2^{k-1},j]=-1} \Pr(\boldsymbol{c}^H = +\boldsymbol{h}_j \mid \boldsymbol{y}_{ch}^H) + \sum\limits_{-H[2^{k-1},j]=+1} \Pr(\boldsymbol{c}^H = -\boldsymbol{h}_j \mid \boldsymbol{y}_{ch}^H)} \\
&= \ln \frac{\sum\limits_{+H[2^{k-1},j]=+1} \gamma\left(+\boldsymbol{h}_j\right) + \sum\limits_{-H[2^{k-1},j]=-1} \gamma\left(-\boldsymbol{h}_j\right)}{\sum\limits_{+H[2^{k-1},j]=-1} \gamma\left(+\boldsymbol{h}_j\right) + \sum\limits_{-H[2^{k-1},j]=+1} \gamma\left(-\boldsymbol{h}_j\right)}.
\end{aligned} \tag{B6}$$

In (38), the numerator only needs to consider the case $\pm H[i,j] = +1$ while the denominator only needs to consider the case $\pm H[i,j] = -1$. However, in (B6), both the numerator and denominator need to consider both $\pm H[i,j] = +1$ and $\pm H[i,j] = -1$; and thus DFHT cannot be used directly to compute $L_{app}^H(2^{k-1})$. To apply DFHT, the following simple transformation is required.

Considering the 2^{k-1}-th row ($k = 1, 2, \cdots, r$) of an order-r Hadamard matrix, there are 2^{r-1} entries with $-H[2^{k-1}, j] = +1$ and 2^{r-1} entries with $-H[2^{k-1}, j] = -1$. (In other words, there are 2^{r-1} $-\boldsymbol{h}_j$'s in which the 2^{k-1}-th entry ($k = 1, 2, \cdots, r$) equals $+1$; and there are 2^{r-1} $-\boldsymbol{h}_j$'s in which the 2^{k-1}-th entry ($k = 1, 2, \cdots, r$) equals -1.) We denote

- $\boldsymbol{J}_{+1}^{k}$ as the set of column indexes s.t. the element $-H[2^{k-1}, j] = +1$
- $\boldsymbol{J}_{-1}^{k}$ as the set of column indexes s.t. the element $-H[2^{k-1}, j] = -1$

It can also be readily proven that if $-H[2^{k-1}, j] = \pm 1$ in $-\boldsymbol{h}_j$ ($j = 0, 1, \ldots, 2^r - 1$), then $-H[2^{k-1}, 2^r - 1 - j] = \mp 1$ in $-\boldsymbol{h}_{2^r-1-j}$. Thus we have

$$\boldsymbol{J}_{+1}^{k} = \{2^r - 1 - j \mid j \in \boldsymbol{J}_{-1}^{k}\}$$

and

$$\boldsymbol{J}_{-1}^{k} = \{2^r - 1 - j' \mid j' \in \boldsymbol{J}_{+1}^{k}\}.$$

It means that

$$\sum_{j \in \boldsymbol{J}_{+1}^{k}} \gamma\left(-\boldsymbol{h}_j\right) = \sum_{j' \in \boldsymbol{J}_{-1}^{k}} \gamma\left(-\boldsymbol{h}_{2^r-1-j'}\right) \tag{B7}$$

and

$$\sum_{j \in \boldsymbol{J}_{-1}^{k}} \gamma\left(-\boldsymbol{h}_j\right) = \sum_{j' \in \boldsymbol{J}_{+1}^{k}} \gamma\left(-\boldsymbol{h}_{2^r-1-j'}\right). \tag{B8}$$

By using the simple transformation

$$\gamma'(-\boldsymbol{h}_j) = \gamma(-\boldsymbol{h}_{2^r-1-j}), \;\; j = 0, 1, \ldots, 2^r - 1; \tag{B9}$$

(B6) can be rewritten as

$$\begin{aligned}
&\ln \frac{\sum\limits_{+H[2^{k-1},j]=+1} \gamma\left(+\boldsymbol{h}_j\right) + \sum\limits_{-H[2^{k-1},j]=+1} \gamma\left(-\boldsymbol{h}_{2^r-1-j}\right)}{\sum\limits_{+H[2^{k-1},j]=-1} \gamma\left(+\boldsymbol{h}_j\right) + \sum\limits_{-H[2^{k-1},j]=-1} \gamma\left(-\boldsymbol{h}_{2^r-1-j}\right)} \\
&= \ln \frac{\sum\limits_{+H[2^{k-1},j]=+1} \gamma\left(+\boldsymbol{h}_j\right) + \sum\limits_{-H[2^{k-1},j]=+1} \gamma'\left(-\boldsymbol{h}_j\right)}{\sum\limits_{+H[2^{k-1},j]=-1} \gamma\left(+\boldsymbol{h}_j\right) + \sum\limits_{-H[2^{k-1},j]=-1} \gamma'\left(-\boldsymbol{h}_j\right)}.
\end{aligned} \tag{B10}$$

As the numerator only needs to consider the case $\pm H[i, j] = +1$ while the denominator only needs to consider the case $\pm H[i, j] = -1$, DFHT can be readily applied to compute (B10).

Appendix C

MONTE CARLO METHOD FOR FORMING THE $m \times d$ MI MATRIX $\{I_{eh}(i,k)\}$

We define the following symbols:

- $\boldsymbol{\sigma}_\mu = [\sigma_{\mu_0}\ \sigma_{\mu_1}\ \ldots\ \sigma_{\mu_{d-1}}]$: $d\ (= r+2)$ noise standard deviations;
- $\boldsymbol{c}_\mu = [c_{\mu_0}\ c_{\mu_1}\ \ldots\ c_{\mu_{d-1}}]$: a length-$d$ SPC codeword;
- $\boldsymbol{c}_p = [c_{p_0}\ c_{p_1}\ \ldots\ c_{p_{g-1}}]$: g Hadamard parity bits generated based on the SPC $\boldsymbol{c}_\mu$; $g = 2^r - d$ and $g = 2^r - 2$, respectively, for systematic (r =even) and non-systematic coding (r =odd);
- $\boldsymbol{n}_\mu = [n_{\mu_0}\ n_{\mu_1}\ \ldots\ n_{\mu_{d-1}}]$: d samples following a normal distribution;
- $\boldsymbol{n}_p = [n_{p_0}\ n_{p_1}\ \ldots\ n_{p_{g-1}}]$: g samples following a normal distribution;
- $\boldsymbol{L}_\mu = [L_{\mu_0}\ L_{\mu_1}\ \ldots\ L_{\mu_{d-1}}]$: d LLR values corresponding to the SPC codeword $\boldsymbol{c}_\mu$;
- $\boldsymbol{L}_p = [L_{p_0}\ L_{p_1}\ \ldots\ L_{p_{g-1}}]$: g channel LLR values corresponding to the Hadamard parity bits $\boldsymbol{c}_p$;
- $\boldsymbol{L}_e = [L_{e_0}\ L_{e_1}\ \ldots\ L_{e_{d-1}}]$: d extrinsic LLR values generated by the Hadamard decoder;
- $\boldsymbol{U}$: a $w \times d$ matrix in which each row represents a length-d SPC codeword; and the k-th column ($k = 0, 1, \ldots, d-1$) corresponds to the k-th bit (c_{μ_k}) of the SPC codeword;
- $\boldsymbol{V}$: a $w \times d$ matrix in which each row represents a set of (d) extrinsic LLR values generated by the Hadamard decoder; and the k-th column ($k = 0, 1, \ldots, d-1$) corresponds to the extrinsic LLR value for the k-th bit (c_{μ_k}) of the SPC codeword;
- $\boldsymbol{p}_{e_0} = [p_e(\xi|c_{\mu_0} = \text{“}0\text{"})\ p_e(\xi|c_{\mu_1} = \text{“}0\text{"})\ \cdots\ p_e(\xi|c_{\mu_{d-1}=\text{“}0\text{"}})]$: PDFs for $c_{\mu_k} = \text{“}0\text{"}$ ($k = 0, 1, \ldots, d-1$);
- $\boldsymbol{p}_{e_1} = [p_e(\xi|c_{\mu_0} = \text{“}1\text{"})\ p_e(\xi|c_{\mu_1} = \text{“}1\text{"})\ \cdots\ p_e(\xi|c_{\mu_{d-1}=\text{“}1\text{"}})]$: PDFs for $c_{\mu_k} = \text{“}1\text{"}$ ($k = 0, 1, \ldots, d-1$).

The $m \times d$ MI matrix $\{I_{eh}(i,k)\}$ is updated with following steps.

i) Given the standard deviation $\sigma_{L_{ch}}$.

ii) Set $i = 0$.

iii) For the i-th row in the MI matrix $\{I_{ah}(i,k)\}$, use the J-function in [23] to compute the standard deviation $\sigma_{\mu_k} = J^{-1}(I_{ah}(i,k))$ for $k = 0, 1, \ldots, d-1$.

iv) Set $j = 0$.

v) Randomly generate a length-d SPC codeword $\boldsymbol{c}_\mu$; further encode $\boldsymbol{c}_\mu$ into a Hadamard codeword using systematic (when $r = d-2$ is even) or non-systematic (when r is odd) coding and generate the g Hadamard parity bits $\boldsymbol{c}_p$.

vi) Randomly generate a sample vector $\boldsymbol{n}_\mu$ where each n_{μ_k} ($k = 0, 1, \ldots, d-1$) follows a different normal distribution $\mathcal{N}(\sigma^2_{\mu_k}/2, \sigma^2_{\mu_k})$.

vii) Randomly generate a sample vector $\boldsymbol{n}_p$ where all $n_{p_{k'}}$'s ($k' = 0, 1, \ldots, g-1$) follow the same normal distribution $\mathcal{N}(\sigma^2_{L_{ch}}/2, \sigma^2_{L_{ch}})$.

viii) For $k = 0, 1, \ldots, d-1$, set $L_{\mu_k} = +n_{\mu_k}$ if $c_{\mu_k} =$ "0"; otherwise set $L_{\mu_k} = -n_{\mu_k}$ if $c_{\mu_k} =$ "1".

ix) For $k' = 0, 1, \ldots, g-1$, set $L_{p_{k'}} = +n_{p_{k'}}$ if $c_{p_{k'}} =$ "0"; otherwise set $L_{p_{k'}} = -n_{p_{k'}}$ if $c_{p_{k'}} =$ "1".

x) Input $\boldsymbol{L}_\mu$ and $\boldsymbol{L}_p$, respectively, as the *a priori* and channel LLRs to the Hadamard decoder. Use the decoding algorithm described in Section 3.2 to compute the d output extrinsic LLR values $\boldsymbol{L}_e$.

xi) Assign $\boldsymbol{c}_\mu$ to the j-th row of $\boldsymbol{U}$ and assign $\boldsymbol{L}_e$ to the j-th row of $\boldsymbol{V}$.

xii) Set $j = j+1$. If $j < w$, go to Step v). (We set $w = 10,000$.)

xiii) The k-th columns ($k = 0, 1, \ldots, d-1$) of both $\boldsymbol{U}$ and $\boldsymbol{V}$ correspond to bit c_{μ_k}. Obtain the PDFs $p_e(\xi|c_{\mu_k} = \text{"0"})$ and $p_e(\xi|c_{\mu_k} = \text{"1"})$ ($k = 0, 1, \ldots, d-1$) based on $\boldsymbol{U}$ and $\boldsymbol{V}$.

xiv) Use $p_e(\xi|c_{\mu_k} = \text{"0"})$ and $p_e(\xi|c_{\mu_k} = \text{"1"})$ to compute (56) and hence $I_{eh}(i,k)$ ($k = 0, 1, \ldots, d-1$).

xv) Set $i = i+1$. If $i < m$, go to step iii).

Appendix D

TWO-STEP LIFTING OF A BASE MATRIX

In the first step, we "lift" a base matrix $\{b(i,j)\}$ by replacing each non-zero entry $b(i,j)$ with a summation of $b(i,j)$ different $z_1 \times z_1$ permutation matrices and replacing each zero entry with the $z_1 \times z_1$ zero matrix. After the first lifting process, all entries in the lifted matrix are either "0" or "1". In the second step, we lift the resultant matrix again by replacing each entry "1" with a $z_2 \times z_2$ circulant permutation matrix (CPM), and replacing each entry "0" with the $z_2 \times z_2$ zero matrix. As can be seen, the final connection matrix can be easily represented by a series of CPMs. Note that in each lifting step, the permutation matrices and CPMs are selected using the PEG algorithm [28] such that the girth (shortest cycle) in the resultant matrix can be maximized.

Take the PLDPC-Hadamard code with code rate $R = 0.0494$ and Hadamard code order $r = 4$ as an example, i.e.,

$$\boldsymbol{B}_{7\times 11} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \\ 2 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 \\ 3 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 2 & 0 \end{bmatrix}. \tag{D1}$$

The size of the optimized base matrix is 7×11. After lifting the base matrix twice with factors $z_1 = 32$ and $z_2 = 512$, respectively, we can obtain a $114,688(= 7 \times 32 \times 512)$ by $180,224(= 11 \times 32 \times 512)$ connection matrix between the variable nodes and the Hadamard check nodes. The $114,688$ by $180,224$ connection matrix can simply be represented by a $224(= 7 \times 32)$ by $352(= 11 \times 32)$ matrix whose entries are CPMs. Such a connection matrix is a structured quasi-cyclic (QC) matrix which greatly facilitates parallel encoding/decoding and enhances the throughput. In this example, there are only $6(= r + 2)$ non-zero CPMs in each row. To simplify the representation, we only record the positions of these non-zero CPMs in each row and their "cyclic-shift" values.

In Table 13, we show the details of the structured QC matrix of the rate-0.0494 PLDPC-Hadamard block code. Besides the header row, there is a total of 224 rows. In each row, there are 6 entries each represented as (c, s). The symbol c denotes the column index where the non-zero CPM locates and it ranges from 1 to 352; while the symbol s denotes the "cyclic-shift" value of this non-zero CPM and ranges from 0 to 511. For example, the entry $(20, 379)$ in the first row shows that in the first row, (i) there is a non-zero CPM in the 20-th column and (ii) this CPM is constructed by cyclically left-shifting the 512×512 identity matrix by 379 columns.

To construct the convolutional protomatrix of SC-PLDPCH-CC, we split (D1) of rate-0.0494 PLDPC-Hadamard block code into two 7×11 protomatrices $\boldsymbol{B}_0$ and $\boldsymbol{B}_1$, where $\boldsymbol{B}_0 + \boldsymbol{B}_1 = \boldsymbol{B}_{7\times 11}$. We lift the convolutional protomatrix with factors $z_1 = 16$ and $z_2 = 1024$ so as the sub-block length also equals $1,327,104$. Since the convolutional protomatrix is obtained by repeating $[\boldsymbol{B}_1\ \boldsymbol{B}_0]$ in a row-wise manner, we only need to record the lifted matrix of $[\boldsymbol{B}_1\ \boldsymbol{B}_0]$. The size of $[\boldsymbol{B}_1\ \boldsymbol{B}_0]$ is 7×22 and its lifted matrix can be denoted by a $112 (= 7 \times 16) \times 352 (= 22 \times 16)$ matrix whose entries are CPMs with size of 1024×1024. Table 14 shows the details of the lifted matrix and has a total of $7 \times 16 = 112$ rows except for the header row. Other descriptions are similar to Table 13.

$$\boldsymbol{B}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{D2}$$

and

$$\boldsymbol{B}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}. \tag{D3}$$

Table 13: QC matrix for rate-0.0494 PLDPC-Hadamard block code

ROW	(COL, CPM)	(COL, CPM)	(COL, CPM)	(COL, CPM)	(COL, CPM)	(COL, CPM)
1	(20, 379)	(211, 194)	(261, 380)	(267, 266)	(278, 320)	(345, 449)
2	(5, 85)	(210, 114)	(263, 47)	(275, 313)	(282, 78)	(335, 369)
3	(21, 220)	(224, 422)	(258, 237)	(268, 304)	(287, 344)	(348, 120)
4	(4, 212)	(222, 342)	(258, 435)	(273, 374)	(286, 164)	(343, 269)
5	(32, 40)	(218, 107)	(259, 41)	(271, 216)	(283, 337)	(339, 226)
6	(10, 205)	(200, 292)	(262, 156)	(278, 94)	(280, 501)	(340, 57)
7	(13, 390)	(217, 294)	(266, 214)	(276, 26)	(286, 46)	(326, 176)
8	(14, 102)	(219, 122)	(260, 356)	(269, 363)	(287, 230)	(323, 464)
9	(3, 7)	(198, 71)	(264, 323)	(276, 218)	(277, 307)	(338, 206)
10	(11, 364)	(199, 230)	(258, 475)	(274, 101)	(283, 82)	(346, 238)
11	(15, 388)	(221, 109)	(263, 284)	(272, 475)	(284, 459)	(333, 75)
12	(8, 436)	(214, 193)	(262, 385)	(272, 509)	(281, 412)	(349, 491)
13	(18, 236)	(202, 472)	(265, 372)	(273, 62)	(285, 438)	(330, 324)
14	(2, 353)	(207, 431)	(259, 132)	(278, 446)	(279, 272)	(336, 328)
15	(28, 173)	(212, 297)	(262, 349)	(271, 250)	(288, 44)	(324, 475)
16	(27, 30)	(194, 170)	(267, 448)	(269, 405)	(282, 453)	(342, 493)
17	(29, 33)	(196, 319)	(267, 46)	(274, 453)	(285, 113)	(321, 102)
18	(19, 84)	(193, 318)	(260, 84)	(270, 127)	(286, 63)	(329, 388)
19	(24, 332)	(223, 28)	(265, 18)	(271, 320)	(287, 149)	(327, 174)
20	(7, 99)	(195, 444)	(263, 15)	(277, 297)	(283, 183)	(337, 148)
21	(25, 387)	(204, 465)	(257, 276)	(277, 43)	(280, 474)	(347, 490)
22	(22, 338)	(215, 362)	(261, 48)	(273, 253)	(284, 195)	(331, 413)
23	(12, 50)	(209, 231)	(266, 227)	(272, 159)	(285, 384)	(344, 279)
24	(16, 383)	(216, 91)	(259, 69)	(275, 115)	(288, 265)	(351, 180)
25	(31, 37)	(201, 143)	(265, 223)	(276, 81)	(282, 450)	(325, 442)
26	(17, 212)	(205, 37)	(257, 453)	(269, 231)	(281, 376)	(350, 507)
27	(6, 436)	(203, 110)	(261, 272)	(270, 275)	(288, 197)	(332, 280)
28	(30, 252)	(206, 5)	(266, 85)	(268, 347)	(279, 379)	(341, 432)
29	(1, 247)	(213, 143)	(264, 125)	(270, 325)	(279, 465)	(322, 16)
30	(23, 245)	(208, 246)	(264, 469)	(274, 75)	(281, 373)	(328, 429)
31	(26, 115)	(197, 275)	(260, 142)	(275, 250)	(280, 172)	(334, 156)
32	(9, 509)	(220, 28)	(257, 246)	(268, 414)	(284, 251)	(352, 164)
33	(53, 9)	(68, 451)	(94, 230)	(289, 323)	(307, 367)	(345, 381)
34	(40, 226)	(76, 35)	(95, 234)	(290, 287)	(309, 174)	(327, 97)
35	(39, 31)	(69, 138)	(87, 234)	(289, 445)	(309, 67)	(328, 74)
36	(35, 79)	(71, 1)	(84, 264)	(292, 29)	(320, 133)	(347, 426)
37	(52, 2)	(78, 194)	(89, 474)	(302, 215)	(311, 255)	(329, 125)
38	(34, 79)	(75, 299)	(93, 272)	(293, 177)	(315, 89)	(340, 383)
39	(37, 254)	(67, 254)	(86, 127)	(291, 228)	(306, 132)	(349, 278)
40	(50, 172)	(73, 403)	(92, 500)	(303, 367)	(317, 126)	(337, 198)
41	(44, 388)	(79, 413)	(85, 94)	(292, 95)	(315, 177)	(325, 309)

42	(49, 206)	(72, 233)	(87, 489)	(301, 220)	(313, 174)	(344, 86)
43	(58, 247)	(66, 351)	(90, 231)	(295, 3)	(305, 85)	(348, 411)
44	(43, 248)	(71, 476)	(91, 160)	(296, 232)	(311, 208)	(326, 60)
45	(54, 151)	(66, 218)	(84, 403)	(298, 160)	(313, 72)	(336, 6)
46	(61, 219)	(76, 441)	(94, 417)	(298, 326)	(306, 54)	(333, 371)
47	(46, 462)	(80, 117)	(82, 63)	(295, 507)	(318, 431)	(339, 268)
48	(60, 443)	(72, 83)	(81, 508)	(291, 164)	(307, 354)	(343, 413)
49	(55, 163)	(80, 474)	(92, 335)	(290, 429)	(312, 41)	(350, 140)
50	(47, 462)	(70, 120)	(96, 346)	(301, 428)	(318, 495)	(342, 108)
51	(59, 312)	(67, 324)	(95, 277)	(299, 16)	(314, 270)	(322, 189)
52	(41, 79)	(68, 219)	(83, 456)	(304, 507)	(316, 283)	(338, 182)
53	(57, 361)	(65, 229)	(83, 212)	(294, 358)	(310, 463)	(324, 10)
54	(56, 203)	(77, 264)	(82, 57)	(303, 488)	(304, 201)	(341, 419)
55	(48, 75)	(70, 237)	(91, 332)	(302, 217)	(310, 489)	(330, 128)
56	(45, 288)	(74, 319)	(96, 324)	(297, 367)	(320, 209)	(335, 52)
57	(63, 468)	(73, 266)	(85, 286)	(300, 67)	(308, 276)	(334, 159)
58	(38, 355)	(65, 464)	(89, 142)	(305, 360)	(319, 342)	(351, 168)
59	(42, 207)	(69, 31)	(93, 50)	(297, 273)	(314, 100)	(352, 24)
60	(36, 286)	(77, 454)	(88, 442)	(296, 154)	(308, 457)	(323, 164)
61	(51, 319)	(78, 444)	(81, 316)	(294, 368)	(312, 288)	(321, 190)
62	(64, 162)	(79, 398)	(90, 507)	(293, 48)	(316, 207)	(331, 139)
63	(62, 493)	(74, 119)	(86, 314)	(300, 15)	(317, 186)	(346, 303)
64	(33, 336)	(75, 237)	(88, 325)	(299, 119)	(319, 161)	(332, 5)
65	(6, 54)	(27, 239)	(51, 398)	(140, 119)	(187, 166)	(341, 493)
66	(10, 336)	(22, 303)	(47, 82)	(147, 332)	(177, 119)	(328, 252)
67	(16, 259)	(24, 241)	(61, 115)	(144, 176)	(162, 59)	(352, 252)
68	(13, 498)	(17, 8)	(33, 426)	(159, 429)	(191, 453)	(346, 378)
69	(12, 460)	(22, 283)	(55, 283)	(158, 334)	(184, 430)	(330, 367)
70	(7, 283)	(20, 73)	(56, 497)	(133, 415)	(161, 373)	(322, 481)
71	(14, 368)	(26, 277)	(50, 493)	(157, 490)	(164, 217)	(324, 217)
72	(3, 485)	(27, 283)	(54, 426)	(154, 499)	(186, 261)	(321, 332)
73	(12, 458)	(30, 386)	(38, 334)	(141, 408)	(180, 225)	(332, 131)
74	(4, 189)	(23, 467)	(64, 154)	(142, 43)	(169, 485)	(344, 63)
75	(7, 31)	(25, 262)	(44, 173)	(150, 474)	(165, 183)	(351, 171)
76	(15, 129)	(17, 478)	(40, 477)	(149, 43)	(171, 440)	(334, 38)
77	(9, 56)	(29, 331)	(63, 413)	(136, 322)	(163, 300)	(342, 499)
78	(9, 198)	(20, 194)	(43, 467)	(160, 501)	(189, 242)	(349, 457)
79	(3, 159)	(32, 427)	(62, 304)	(138, 1)	(176, 19)	(329, 410)
80	(11, 77)	(18, 413)	(45, 367)	(153, 378)	(175, 377)	(338, 507)
81	(8, 30)	(19, 166)	(59, 204)	(139, 151)	(178, 62)	(336, 343)
82	(15, 272)	(28, 142)	(53, 193)	(146, 163)	(174, 421)	(326, 92)
83	(13, 114)	(18, 221)	(48, 484)	(155, 254)	(190, 315)	(348, 202)
84	(1, 315)	(28, 204)	(46, 467)	(143, 485)	(172, 13)	(345, 271)

85	(2, 159)	(24, 355)	(37, 199)	(134, 480)	(183, 250)	(331, 309)
86	(5, 43)	(19, 83)	(49, 252)	(148, 273)	(181, 271)	(327, 483)
87	(4, 2)	(31, 15)	(36, 157)	(156, 417)	(179, 438)	(335, 419)
88	(8, 487)	(31, 308)	(39, 259)	(129, 405)	(185, 178)	(339, 75)
89	(5, 444)	(29, 450)	(41, 286)	(137, 385)	(167, 271)	(343, 87)
90	(2, 222)	(32, 341)	(52, 191)	(130, 201)	(170, 270)	(350, 143)
91	(14, 397)	(21, 350)	(57, 134)	(151, 62)	(166, 485)	(347, 464)
92	(16, 101)	(23, 319)	(35, 166)	(132, 195)	(173, 234)	(323, 401)
93	(11, 41)	(30, 256)	(34, 61)	(131, 193)	(192, 403)	(333, 206)
94	(6, 239)	(26, 343)	(58, 311)	(145, 133)	(188, 312)	(340, 96)
95	(1, 83)	(25, 416)	(60, 447)	(152, 461)	(182, 407)	(337, 265)
96	(10, 411)	(21, 36)	(42, 278)	(135, 442)	(168, 179)	(325, 467)
97	(54, 145)	(102, 200)	(115, 140)	(123, 341)	(297, 438)	(307, 12)
98	(37, 272)	(106, 100)	(109, 414)	(120, 103)	(290, 69)	(319, 3)
99	(57, 68)	(97, 491)	(113, 466)	(125, 207)	(291, 314)	(305, 151)
100	(33, 214)	(101, 103)	(112, 485)	(118, 94)	(304, 413)	(317, 236)
101	(35, 384)	(106, 201)	(110, 83)	(128, 348)	(300, 106)	(312, 447)
102	(53, 260)	(103, 218)	(117, 285)	(126, 416)	(294, 274)	(311, 85)
103	(40, 191)	(101, 209)	(110, 488)	(123, 215)	(295, 464)	(320, 30)
104	(49, 51)	(106, 404)	(114, 400)	(127, 311)	(292, 261)	(320, 279)
105	(47, 447)	(99, 468)	(117, 17)	(119, 337)	(303, 213)	(314, 352)
106	(56, 393)	(103, 19)	(118, 60)	(120, 418)	(292, 206)	(308, 185)
107	(45, 30)	(100, 53)	(115, 496)	(126, 199)	(303, 464)	(305, 480)
108	(41, 423)	(98, 443)	(115, 280)	(124, 129)	(289, 100)	(310, 179)
109	(62, 433)	(104, 215)	(113, 246)	(123, 283)	(304, 301)	(309, 336)
110	(60, 130)	(99, 132)	(110, 85)	(122, 92)	(297, 74)	(315, 509)
111	(51, 414)	(100, 326)	(113, 230)	(121, 375)	(302, 283)	(318, 48)
112	(52, 157)	(103, 360)	(114, 50)	(124, 429)	(291, 478)	(317, 104)
113	(34, 403)	(98, 285)	(108, 263)	(119, 446)	(290, 491)	(313, 467)
114	(43, 93)	(105, 308)	(111, 505)	(121, 109)	(301, 435)	(312, 282)
115	(50, 139)	(100, 39)	(116, 486)	(127, 347)	(293, 161)	(306, 350)
116	(58, 292)	(104, 397)	(112, 104)	(119, 289)	(296, 413)	(310, 95)
117	(59, 390)	(107, 201)	(111, 298)	(118, 259)	(295, 46)	(307, 178)
118	(46, 486)	(99, 338)	(107, 451)	(124, 247)	(293, 385)	(319, 397)
119	(42, 448)	(97, 294)	(111, 161)	(127, 211)	(298, 491)	(308, 506)
120	(44, 126)	(98, 459)	(109, 331)	(122, 136)	(299, 307)	(311, 481)
121	(39, 259)	(102, 44)	(112, 459)	(122, 126)	(298, 344)	(316, 407)
122	(38, 149)	(102, 346)	(117, 130)	(128, 408)	(301, 213)	(306, 408)
123	(55, 387)	(104, 312)	(116, 333)	(120, 172)	(302, 192)	(316, 438)
124	(61, 302)	(107, 267)	(109, 18)	(125, 452)	(294, 126)	(315, 192)
125	(36, 295)	(105, 348)	(114, 413)	(125, 139)	(289, 372)	(318, 197)
126	(64, 178)	(101, 340)	(108, 270)	(121, 464)	(300, 272)	(309, 271)
127	(63, 8)	(105, 246)	(116, 311)	(128, 153)	(299, 16)	(313, 56)

128	(48, 424)	(97, 58)	(108, 296)	(126, 77)	(296, 471)	(314, 98)
129	(3, 182)	(21, 262)	(242, 13)	(291, 188)	(304, 43)	(310, 270)
130	(5, 399)	(25, 106)	(238, 371)	(295, 352)	(306, 159)	(310, 254)
131	(7, 251)	(26, 77)	(244, 260)	(299, 312)	(306, 366)	(316, 223)
132	(3, 383)	(22, 484)	(240, 132)	(292, 429)	(307, 322)	(318, 197)
133	(5, 276)	(23, 75)	(252, 182)	(294, 153)	(307, 225)	(319, 40)
134	(14, 89)	(30, 245)	(233, 416)	(289, 95)	(304, 426)	(314, 258)
135	(9, 437)	(21, 38)	(232, 196)	(297, 480)	(302, 454)	(317, 169)
136	(8, 46)	(32, 119)	(245, 104)	(294, 363)	(305, 26)	(314, 468)
137	(9, 431)	(28, 144)	(254, 499)	(291, 257)	(303, 75)	(311, 345)
138	(6, 460)	(19, 123)	(227, 151)	(298, 426)	(310, 128)	(320, 393)
139	(16, 114)	(32, 358)	(229, 285)	(297, 154)	(308, 84)	(319, 443)
140	(4, 477)	(17, 157)	(243, 440)	(293, 69)	(300, 189)	(318, 158)
141	(4, 43)	(27, 228)	(248, 296)	(298, 234)	(303, 226)	(315, 449)
142	(12, 14)	(16, 372)	(256, 217)	(289, 189)	(300, 309)	(320, 250)
143	(17, 49)	(19, 327)	(225, 193)	(297, 210)	(309, 78)	(312, 408)
144	(13, 240)	(29, 292)	(236, 481)	(292, 461)	(301, 387)	(311, 360)
145	(15, 102)	(20, 141)	(228, 362)	(293, 330)	(299, 345)	(312, 139)
146	(8, 357)	(29, 148)	(239, 304)	(290, 270)	(308, 487)	(315, 209)
147	(2, 304)	(26, 135)	(249, 398)	(292, 113)	(305, 450)	(317, 463)
148	(6, 39)	(22, 223)	(247, 487)	(289, 74)	(303, 2)	(313, 165)
149	(10, 119)	(31, 362)	(230, 464)	(294, 105)	(309, 50)	(317, 468)
150	(12, 171)	(20, 284)	(231, 119)	(295, 306)	(308, 210)	(313, 507)
151	(11, 207)	(31, 78)	(234, 177)	(293, 58)	(301, 240)	(320, 222)
152	(11, 149)	(27, 474)	(241, 274)	(296, 42)	(306, 80)	(313, 306)
153	(15, 116)	(24, 318)	(255, 306)	(290, 100)	(304, 298)	(311, 259)
154	(13, 139)	(24, 20)	(253, 297)	(296, 196)	(302, 77)	(312, 63)
155	(2, 184)	(25, 30)	(250, 303)	(291, 156)	(301, 240)	(316, 342)
156	(1, 235)	(30, 377)	(226, 430)	(296, 337)	(307, 511)	(316, 489)
157	(14, 162)	(28, 468)	(251, 323)	(299, 193)	(302, 307)	(315, 361)
158	(7, 244)	(18, 342)	(237, 458)	(295, 10)	(309, 167)	(319, 191)
159	(10, 252)	(23, 112)	(246, 305)	(290, 4)	(305, 265)	(318, 351)
160	(1, 294)	(18, 485)	(235, 370)	(298, 6)	(300, 272)	(314, 447)
161	(8, 389)	(18, 442)	(23, 246)	(110, 47)	(126, 74)	(215, 125)
162	(12, 482)	(21, 195)	(31, 398)	(100, 241)	(119, 141)	(199, 85)
163	(5, 62)	(13, 66)	(28, 49)	(111, 5)	(128, 370)	(204, 223)
164	(9, 334)	(14, 140)	(32, 183)	(109, 419)	(117, 440)	(209, 481)
165	(2, 390)	(11, 299)	(29, 18)	(99, 494)	(113, 508)	(201, 360)
166	(1, 492)	(19, 75)	(32, 206)	(97, 471)	(120, 46)	(211, 356)
167	(10, 451)	(16, 334)	(28, 68)	(100, 134)	(118, 71)	(219, 308)
168	(7, 256)	(16, 74)	(29, 469)	(112, 360)	(128, 52)	(213, 202)
169	(6, 221)	(17, 403)	(25, 167)	(107, 317)	(123, 216)	(202, 258)
170	(11, 111)	(16, 421)	(26, 299)	(101, 489)	(127, 457)	(198, 489)

171	(3,96)	(15,30)	(30,418)	(108,86)	(116,178)	(218,7)
172	(8,273)	(12,62)	(27,372)	(111,337)	(117,209)	(203,369)
173	(8,202)	(20,321)	(24,283)	(112,36)	(125,243)	(216,465)
174	(3,495)	(18,292)	(28,80)	(102,150)	(114,189)	(195,11)
175	(1,265)	(15,268)	(22,157)	(105,277)	(122,495)	(207,459)
176	(5,32)	(12,342)	(26,399)	(105,446)	(123,408)	(222,447)
177	(9,93)	(11,421)	(32,440)	(98,266)	(118,430)	(196,397)
178	(7,332)	(19,183)	(31,189)	(108,452)	(114,77)	(194,294)
179	(1,64)	(20,127)	(26,187)	(106,259)	(119,296)	(205,252)
180	(10,196)	(13,51)	(27,81)	(98,325)	(127,490)	(224,454)
181	(4,455)	(21,122)	(22,60)	(106,215)	(115,392)	(210,121)
182	(6,178)	(20,150)	(29,122)	(103,230)	(122,289)	(223,197)
183	(7,260)	(17,421)	(30,171)	(104,268)	(126,179)	(212,171)
184	(5,200)	(21,91)	(27,207)	(99,112)	(120,402)	(220,113)
185	(4,390)	(14,91)	(25,46)	(104,92)	(124,429)	(208,227)
186	(3,199)	(13,312)	(23,216)	(103,440)	(113,88)	(221,351)
187	(10,446)	(15,239)	(25,2)	(109,455)	(121,483)	(197,152)
188	(6,58)	(23,443)	(24,85)	(97,510)	(115,228)	(206,279)
189	(4,347)	(19,359)	(24,43)	(102,71)	(116,202)	(217,65)
190	(9,511)	(17,299)	(22,289)	(110,43)	(125,468)	(214,135)
191	(2,89)	(14,242)	(31,81)	(107,170)	(121,99)	(200,295)
192	(2,248)	(18,4)	(30,163)	(101,352)	(124,128)	(193,73)
193	(25,106)	(108,407)	(151,412)	(273,425)	(297,464)	(305,267)
194	(19,133)	(113,45)	(131,70)	(259,491)	(293,151)	(311,266)
195	(11,47)	(117,254)	(154,194)	(265,215)	(300,208)	(316,431)
196	(27,68)	(128,9)	(145,34)	(276,238)	(291,231)	(309,409)
197	(2,481)	(111,373)	(144,360)	(286,457)	(289,101)	(315,311)
198	(31,83)	(104,185)	(150,405)	(266,21)	(291,510)	(308,261)
199	(21,280)	(97,357)	(146,129)	(275,129)	(303,501)	(306,295)
200	(18,32)	(121,95)	(135,71)	(281,362)	(304,373)	(315,253)
201	(16,46)	(116,246)	(129,48)	(263,50)	(295,295)	(312,185)
202	(32,120)	(127,145)	(132,103)	(272,126)	(294,36)	(320,267)
203	(3,369)	(120,508)	(160,381)	(269,321)	(289,445)	(311,328)
204	(6,509)	(106,431)	(159,45)	(279,122)	(302,463)	(308,359)
205	(14,283)	(122,464)	(136,319)	(283,189)	(292,155)	(313,321)
206	(15,354)	(114,216)	(138,67)	(288,429)	(295,147)	(314,170)
207	(30,178)	(102,324)	(140,53)	(262,376)	(290,510)	(310,192)
208	(4,56)	(103,65)	(139,488)	(274,33)	(301,180)	(314,391)
209	(22,301)	(124,103)	(134,292)	(257,11)	(298,84)	(309,0)
210	(28,180)	(107,108)	(149,7)	(258,197)	(296,278)	(320,179)
211	(8,49)	(119,277)	(152,87)	(261,373)	(304,255)	(319,493)
212	(20,327)	(105,474)	(143,273)	(260,33)	(290,390)	(317,371)
213	(29,51)	(125,31)	(153,200)	(284,60)	(303,91)	(319,471)

214	(24, 426)	(98, 230)	(141, 71)	(277, 219)	(297, 121)	(306, 73)
215	(9, 63)	(112, 5)	(148, 323)	(267, 426)	(293, 409)	(307, 108)
216	(1, 294)	(115, 470)	(147, 357)	(280, 464)	(301, 221)	(317, 373)
217	(5, 447)	(110, 106)	(133, 360)	(285, 73)	(302, 435)	(313, 58)
218	(10, 239)	(123, 73)	(130, 286)	(264, 30)	(292, 496)	(316, 412)
219	(7, 408)	(99, 40)	(155, 114)	(270, 454)	(296, 49)	(318, 327)
220	(17, 361)	(118, 2)	(156, 195)	(271, 508)	(299, 343)	(305, 319)
221	(23, 221)	(109, 415)	(158, 323)	(287, 423)	(300, 467)	(310, 207)
222	(26, 64)	(100, 470)	(137, 39)	(278, 87)	(298, 490)	(312, 123)
223	(12, 226)	(126, 194)	(142, 122)	(282, 260)	(299, 272)	(307, 105)
224	(13, 451)	(101, 146)	(157, 69)	(268, 154)	(294, 153)	(318, 267)

Table 14: Part of the QC matrix for rate-0.0494 SC-PLDPCH-CC with $W = 1$, i.e., $[\boldsymbol{B}_1 \ \boldsymbol{B}_0]$

ROW	(COL, CPM)	(COL, CPM)	(COL, CPM)	(COL, CPM)	(COL, CPM)	(COL, CPM)
1	(100, 0)	(133, 0)	(135, 0)	(144, 0)	(191, 737)	(343, 257)
2	(110, 0)	(132, 0)	(135, 0)	(142, 0)	(192, 189)	(350, 595)
3	(106, 0)	(130, 0)	(137, 0)	(141, 0)	(184, 979)	(341, 807)
4	(111, 0)	(129, 0)	(136, 0)	(143, 0)	(182, 366)	(348, 474)
5	(102, 0)	(131, 0)	(137, 0)	(143, 2)	(183, 560)	(349, 868)
6	(105, 0)	(131, 0)	(138, 0)	(143, 3)	(177, 218)	(338, 740)
7	(108, 0)	(132, 0)	(134, 0)	(140, 0)	(178, 106)	(351, 429)
8	(103, 0)	(132, 0)	(136, 0)	(139, 0)	(189, 123)	(352, 973)
9	(109, 0)	(133, 0)	(137, 0)	(142, 2)	(185, 849)	(340, 7)
10	(112, 0)	(134, 0)	(135, 1)	(141, 2)	(186, 822)	(345, 221)
11	(97, 0)	(133, 0)	(136, 1)	(141, 4)	(179, 576)	(344, 156)
12	(104, 0)	(130, 0)	(138, 1)	(140, 1)	(188, 1014)	(339, 675)
13	(101, 0)	(129, 0)	(139, 1)	(142, 4)	(181, 820)	(342, 700)
14	(98, 0)	(129, 0)	(138, 0)	(140, 2)	(190, 52)	(346, 607)
15	(107, 0)	(131, 0)	(139, 0)	(144, 2)	(187, 100)	(347, 29)
16	(99, 0)	(130, 0)	(134, 0)	(144, 4)	(180, 853)	(337, 414)
17	(26, 999)	(40, 0)	(151, 0)	(173, 0)	(216, 505)	(333, 782)
18	(32, 327)	(35, 0)	(160, 0)	(170, 0)	(211, 325)	(325, 701)
19	(19, 151)	(46, 0)	(154, 0)	(176, 0)	(222, 997)	(332, 901)
20	(24, 236)	(45, 0)	(155, 0)	(165, 0)	(221, 783)	(322, 851)
21	(23, 775)	(41, 0)	(156, 0)	(164, 0)	(217, 949)	(328, 531)
22	(17, 713)	(33, 0)	(159, 0)	(171, 0)	(209, 468)	(324, 607)
23	(20, 670)	(48, 0)	(152, 0)	(174, 0)	(224, 144)	(329, 287)
24	(30, 246)	(38, 0)	(145, 0)	(161, 0)	(214, 846)	(323, 241)
25	(21, 735)	(47, 0)	(146, 0)	(162, 0)	(223, 669)	(336, 165)
26	(28, 205)	(36, 0)	(149, 0)	(168, 0)	(212, 656)	(327, 994)
27	(27, 881)	(42, 0)	(153, 0)	(163, 0)	(218, 955)	(335, 516)
28	(18, 83)	(43, 0)	(147, 0)	(167, 0)	(219, 476)	(334, 612)
29	(25, 372)	(44, 0)	(148, 0)	(169, 0)	(220, 425)	(330, 166)
30	(22, 866)	(37, 0)	(158, 0)	(175, 0)	(213, 122)	(321, 64)
31	(29, 926)	(34, 0)	(150, 0)	(172, 0)	(210, 595)	(331, 179)
32	(31, 879)	(39, 0)	(157, 0)	(166, 0)	(215, 885)	(326, 586)
33	(6, 0)	(26, 0)	(66, 947)	(85, 791)	(167, 0)	(192, 0)
34	(3, 0)	(17, 0)	(75, 285)	(82, 340)	(163, 0)	(189, 0)
35	(9, 0)	(28, 0)	(79, 149)	(95, 982)	(165, 0)	(183, 0)
36	(13, 0)	(31, 0)	(69, 486)	(84, 560)	(168, 0)	(179, 0)
37	(14, 0)	(22, 0)	(67, 760)	(83, 755)	(166, 0)	(186, 0)
38	(12, 0)	(27, 0)	(80, 665)	(90, 180)	(169, 0)	(178, 0)
39	(10, 0)	(30, 0)	(77, 330)	(87, 337)	(171, 0)	(180, 0)
40	(4, 0)	(18, 0)	(78, 171)	(96, 803)	(164, 0)	(187, 0)
41	(8, 0)	(20, 0)	(70, 853)	(88, 371)	(161, 0)	(188, 0)

42	(5, 0)	(19, 0)	(72, 158)	(89, 968)	(176, 0)	(184, 0)
43	(2, 0)	(21, 0)	(65, 188)	(91, 2)	(175, 0)	(190, 0)
44	(15, 0)	(29, 0)	(68, 205)	(92, 556)	(162, 0)	(177, 325)
45	(16, 0)	(32, 0)	(73, 85)	(94, 591)	(173, 0)	(181, 0)
46	(7, 0)	(25, 0)	(71, 840)	(81, 94)	(170, 0)	(191, 0)
47	(1, 508)	(23, 0)	(76, 755)	(86, 764)	(174, 0)	(185, 0)
48	(11, 0)	(24, 0)	(74, 687)	(93, 103)	(172, 0)	(182, 0)
49	(25, 0)	(231, 971)	(234, 400)	(240, 746)	(327, 0)	(330, 0)
50	(24, 0)	(225, 195)	(230, 602)	(231, 0)	(328, 0)	(333, 0)
51	(32, 0)	(229, 135)	(232, 760)	(240, 0)	(324, 0)	(336, 0)
52	(26, 0)	(228, 466)	(237, 269)	(240, 0)	(321, 0)	(334, 0)
53	(30, 0)	(226, 707)	(238, 507)	(239, 652)	(322, 0)	(333, 1)
54	(21, 0)	(226, 0)	(233, 945)	(238, 0)	(322, 0)	(331, 0)
55	(18, 0)	(225, 0)	(237, 0)	(239, 0)	(323, 0)	(332, 0)
56	(29, 0)	(231, 0)	(236, 487)	(238, 2)	(326, 0)	(331, 3)
57	(20, 0)	(229, 0)	(234, 0)	(235, 428)	(326, 0)	(330, 1)
58	(22, 0)	(227, 468)	(236, 0)	(237, 0)	(328, 0)	(332, 9)
59	(23, 0)	(230, 0)	(233, 0)	(234, 2)	(325, 0)	(335, 0)
60	(27, 0)	(232, 0)	(235, 0)	(239, 0)	(323, 1)	(329, 0)
61	(19, 0)	(230, 0)	(235, 2)	(236, 0)	(325, 6)	(329, 5)
62	(28, 0)	(225, 0)	(226, 1)	(227, 0)	(321, 0)	(335, 15)
63	(17, 0)	(228, 0)	(232, 0)	(233, 0)	(327, 1)	(336, 3)
64	(31, 0)	(227, 0)	(228, 0)	(229, 1)	(324, 1)	(334, 6)
65	(5, 0)	(11, 1)	(127, 474)	(146, 0)	(155, 7)	(156, 20)
66	(1, 111)	(10, 0)	(119, 342)	(146, 0)	(151, 1)	(157, 5)
67	(2, 0)	(13, 0)	(123, 492)	(147, 12)	(155, 20)	(158, 15)
68	(8, 0)	(9, 0)	(122, 719)	(147, 14)	(154, 8)	(156, 31)
69	(3, 0)	(12, 2)	(120, 389)	(145, 10)	(154, 21)	(159, 27)
70	(6, 0)	(15, 1)	(118, 179)	(146, 10)	(151, 0)	(160, 21)
71	(9, 0)	(15, 3)	(116, 349)	(148, 6)	(153, 4)	(155, 27)
72	(1, 59)	(12, 0)	(115, 602)	(149, 0)	(150, 5)	(159, 28)
73	(6, 0)	(16, 2)	(117, 129)	(145, 5)	(154, 27)	(159, 53)
74	(4, 0)	(8, 0)	(125, 977)	(149, 11)	(151, 16)	(157, 23)
75	(5, 0)	(14, 1)	(128, 416)	(147, 17)	(152, 9)	(158, 27)
76	(2, 0)	(10, 0)	(121, 663)	(149, 19)	(152, 15)	(156, 55)
77	(7, 0)	(16, 4)	(113, 170)	(148, 9)	(152, 5)	(158, 34)
78	(7, 0)	(14, 0)	(124, 489)	(148, 11)	(153, 24)	(160, 44)
79	(4, 0)	(11, 2)	(126, 441)	(150, 4)	(153, 19)	(157, 30)
80	(3, 0)	(13, 1)	(114, 380)	(145, 4)	(150, 13)	(160, 40)
81	(6, 0)	(8, 0)	(186, 0)	(230, 1)	(236, 6)	(288, 881)
82	(4, 0)	(7, 0)	(187, 2)	(233, 0)	(237, 3)	(275, 887)
83	(4, 0)	(8, 1)	(182, 0)	(234, 5)	(240, 0)	(279, 503)
84	(11, 3)	(16, 8)	(180, 0)	(229, 0)	(239, 5)	(276, 970)

85	$(12,0)$	$(16,6)$	$(181,0)$	$(231,0)$	$(240,7)$	$(285,646)$
86	$(11,0)$	$(15,3)$	$(178,0)$	$(231,2)$	$(235,6)$	$(281,92)$
87	$(5,0)$	$(9,0)$	$(188,1)$	$(227,1)$	$(228,2)$	$(287,853)$
88	$(1,215)$	$(13,2)$	$(189,0)$	$(233,3)$	$(238,0)$	$(274,575)$
89	$(3,0)$	$(12,3)$	$(183,0)$	$(232,4)$	$(237,10)$	$(280,161)$
90	$(2,0)$	$(14,2)$	$(191,0)$	$(225,0)$	$(230,4)$	$(282,322)$
91	$(5,0)$	$(7,0)$	$(190,1)$	$(227,5)$	$(235,12)$	$(278,63)$
92	$(3,0)$	$(15,2)$	$(192,2)$	$(226,2)$	$(238,7)$	$(283,962)$
93	$(10,0)$	$(14,3)$	$(177,972)$	$(225,4)$	$(226,6)$	$(286,301)$
94	$(1,835)$	$(9,0)$	$(179,0)$	$(228,7)$	$(232,10)$	$(277,941)$
95	$(6,0)$	$(13,1)$	$(184,0)$	$(229,1)$	$(239,9)$	$(273,546)$
96	$(2,0)$	$(10,1)$	$(185,0)$	$(234,10)$	$(236,1)$	$(284,806)$
97	$(8,0)$	$(58,6)$	$(79,0)$	$(147,2)$	$(156,44)$	$(309,311)$
98	$(6,0)$	$(54,6)$	$(71,0)$	$(146,8)$	$(153,31)$	$(308,339)$
99	$(2,0)$	$(64,1)$	$(77,0)$	$(150,8)$	$(158,46)$	$(305,222)$
100	$(13,0)$	$(60,0)$	$(78,0)$	$(152,13)$	$(159,49)$	$(320,835)$
101	$(7,0)$	$(55,6)$	$(75,0)$	$(148,16)$	$(157,41)$	$(313,246)$
102	$(1,226)$	$(53,0)$	$(76,0)$	$(147,12)$	$(158,52)$	$(315,860)$
103	$(11,0)$	$(62,0)$	$(70,0)$	$(149,8)$	$(156,59)$	$(310,642)$
104	$(14,0)$	$(57,0)$	$(69,0)$	$(152,23)$	$(154,34)$	$(319,732)$
105	$(4,0)$	$(59,0)$	$(80,0)$	$(146,5)$	$(153,27)$	$(312,863)$
106	$(16,0)$	$(51,1)$	$(74,0)$	$(150,15)$	$(159,72)$	$(317,131)$
107	$(15,0)$	$(49,6)$	$(66,0)$	$(151,12)$	$(154,42)$	$(314,545)$
108	$(9,0)$	$(50,3)$	$(72,0)$	$(145,12)$	$(157,3)$	$(307,824)$
109	$(10,0)$	$(56,2)$	$(65,0)$	$(151,15)$	$(160,10)$	$(306,1017)$
110	$(5,0)$	$(63,8)$	$(67,0)$	$(145,3)$	$(155,29)$	$(311,214)$
111	$(3,0)$	$(61,3)$	$(68,0)$	$(149,17)$	$(160,63)$	$(316,961)$
112	$(12,0)$	$(52,3)$	$(73,0)$	$(148,16)$	$(155,31)$	$(318,679)$

Part VI

BIBLIOGRAPHY

Bibliography

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

[2] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, Jul. 1948.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in *Proceedings of IEEE International Conference on Communications (ICC)*, vol. 2, pp. 1064–1070, May 1993.

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

[5] R. G. Gallager, *Low-density parity-check codes*. PhD thesis, Cambridge, MA, USA, 1963.

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

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

[8] J. Hou, P. Siegel, and L. Milstein, "Performance analysis and code optimization of low density parity-check codes on Rayleigh fading channels," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 5, pp. 924–934, 2001.

[9] K. S. Andrews, D. Divsalar, S. Dolinar, J. Hamkins, C. R. Jones, and F. Pollara, "The development of turbo and LDPC codes for deep-space applications," *Proceedings of the IEEE*, vol. 95, no. 11, pp. 2142–2156, 2007.

[10] G. P. Calzolari, M. Chiani, F. Chiaraluce, R. Garello, and E. Paolini, "Channel coding for future space missions: New requirements and trends," *Proceedings of the IEEE*, vol. 95, no. 11, pp. 2157–2170, 2007.

[11] I. B. Djordjevic, M. Arabaci, and L. L. Minkov, "Next generation FEC for high-capacity communication in optical transport

networks," *Journal of Lightwave Technology*, vol. 27, no. 16, pp. 3518–3530, 2009.

[12] H. Song, R. Todd, and J. Cruz, "Low density parity check codes for magnetic recording channels," *IEEE Transactions on Magnetics*, vol. 36, no. 5, pp. 2183–2186, 2000.

[13] B. Kurkoski, P. Siegel, and J. Wolf, "Joint message-passing decoding of LDPC codes and partial-response channels," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1410–1422, 2002.

[14] S. Lin and D. J. Costello, *Error control coding: Fundamentals and Applications, Second Edition*. Upper Saddle River, NJ, USA: Prentice hall, 2001.

[15] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge, U.K.: Cambridge university press, 2008.

[16] N. Bonello, S. Chen, and L. Hanzo, "Low-density parity-check codes and their rateless relatives," *IEEE Communications Surveys Tutorials*, vol. 13, no. 1, pp. 3–26, 2011.

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

[18] S. Shao, P. Hailes, T. Wang, J. Wu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Survey of turbo, LDPC, and polar decoder ASIC implementations," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2309–2333, 2019.

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

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

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

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

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

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

[25] A. Ashikhmin, G. Kramer, and S. ten Brink, "Extrinsic information transfer functions: Model and erasure channel properties," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2657–2673, Nov. 2004.

[26] H. Xiao, E. Eleftheriou, and D.-M. Arnold, "Irregular progressive edge-growth (PEG) tanner graphs," in *Proceedings of IEEE International Symposium on Information Theory,*, pp. 480–480, 2002.

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

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

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

[30] D. Divsalar, H. Jin, and R. McEliece, "Coding theorems for turbo-like codes," in *Proceedings of Allerton Conference on Communication, Control, and Computing*, pp. 201–210, 1998.

[31] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proceedings of 2nd International Symposium on Turbo Codes and Related Topics*, pp. 1–8, 2000.

[32] A. Roumy, S. Guemghar, G. Caire, and S. Verdu, "Design methods for irregular repeat-accumulate codes," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1711–1727, 2004.

[33] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate-repeat-accumulate codes," *IEEE Transactions on Communications*, vol. 55, no. 4, pp. 692–702, Apr. 2007.

[34] D. Divsalar, C. Jones, S. Dolinar, and J. Thorpe, "Protograph based LDPC codes with minimum distance linearly growing with block size," in *Proceedings of IEEE Global Telecommunications Conference*, vol. 3, pp. 1152–1156, 2005.

[35] D. Divsalar, S. Dolinar, C. R. Jones, and K. Andrews, "Capacity-approaching protograph codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 876–888, 2009.

[36] J. Thorpe, "Low-density parity-check (LDPC) codes constructed from protographs," in *IPN progress report*, pp. 1–7, Aug. 2003.

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

[38] D. Divsalar, S. Dolinar, and C. Jones, "Low-rate LDPC codes with simple protograph structure," in *Proceedings of International Symposium on Information Theory (ISIT)*, pp. 1622–1626, 2005.

[39] Y. Fang, G. Bi, and Y. L. Guan, "Design and analysis of root-protograph LDPC codes for non-ergodic block-fading channels," *IEEE Transactions on Wireless Communications*, vol. 14, no. 2, pp. 738–749, 2015.

[40] Y. Fang, P. Chen, G. Cai, F. C. M. Lau, S. C. Liew, and G. Han, "Outage-limit-approaching channel coding for future wireless communications: root-protograph low-density parity-check codes," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 85–93, 2019.

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

[42] A. R. Iyengar, M. Papaleo, P. H. Siegel, J. K. Wolf, A. Vanelli-Coralli, and G. E. Corazza, "Windowed decoding of protograph-based LDPC convolutional codes over erasure channels," *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2303–2320, 2012.

[43] D. J. Costello, L. Dolecek, T. E. Fuja, J. Kliewer, D. G. Mitchell, and R. Smarandache, "Spatially coupled sparse codes on graphs: Theory and practice," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 168–176, 2014.

[44] M. Stinner and P. M. Olmos, "Finite-length performance of multi-edge protograph-based spatially coupled LDPC codes," in *Proceedings of 2015 IEEE International Symposium on Information Theory (ISIT)*, pp. 889–893, 2015.

[45] D. G. M. Mitchell, M. Lentmaier, and D. J. Costello, "Spatially coupled LDPC codes constructed from protographs," *IEEE*

Transactions on Information Theory, vol. 61, no. 9, pp. 4866–4889, 2015.

[46] S. Kudekar, T. J. Richardson, and R. L. Urbanke, "Threshold saturation via spatial coupling: why convolutional LDPC ensembles perform so well over the BEC," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 803–834, 2011.

[47] S. Kudekar, T. Richardson, and R. L. Urbanke, "Spatially coupled ensembles universally achieve capacity under belief propagation," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 7761–7813, 2013.

[48] K. Takeuchi, T. Tanaka, and T. Kawabata, "Improvement of BP-based CDMA multiuser detection by spatial coupling," in *Proceedings of 2011 IEEE International Symposium on Information Theory*, pp. 1489–1493, 2011.

[49] A. Yedla, P. S. Nguyen, H. D. Pfister, and K. R. Narayanan, "Universal codes for the Gaussian MAC via spatial coupling," in *Proceedings of 2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1801–1808, 2011.

[50] S. Kudekar and K. Kasai, "Spatially coupled codes over the multiple access channel," in *Proceedings of 2011 IEEE International Symposium on Information Theory*, pp. 2816–2820, 2011.

[51] D. Truhachev, "Achieving AWGN multiple access channel capacity with spatial graph coupling," *IEEE Communications Letters*, vol. 16, no. 5, pp. 585–588, 2012.

[52] C. Schlegel and D. Truhachev, "Multiple access demodulation in the lifted signal graph with spatial coupling," *IEEE Transactions on Information Theory*, vol. 59, no. 4, pp. 2459–2470, 2013.

[53] M. Stinner and P. M. Olmos, "On the waterfall performance of finite-length SC-LDPC codes constructed from protographs," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 345–361, 2016.

[54] L. Lentmaier and K. S. Zigangirov, "On generalized low-density parity-check codes based on Hamming component codes," *IEEE Communications Letters*, vol. 3, no. 8, pp. 248–250, Aug. 1999.

[55] J. Boutros, O. Pothier, and G. Zemor, "Generalized low-density (Tanner) codes," in *Proceedings of IEEE International Conference on Communications (ICC)*, vol. 1, pp. 441–445, Jun. 1999.

[56] Y. Min, F. C. M. Lau, and C. K. Tse, "Generalized LDPC code with single-parity-check product constraints at super check nodes," in *Proceedings of 2012 7th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pp. 165–169, 2012.

[57] S. Abu-Surra, G. Liva, and W. E. Ryan, "Low-floor Tanner codes via Hamming-node or RSCC-node doping," in *Proceedings of International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pp. 245–254, Feb. 2006.

[58] G. Liva, W. E. Ryan, and M. Chiani, "Quasi-cyclic generalized LDPC codes with low error floors," *IEEE Transactions on Communications*, vol. 56, no. 1, pp. 49–57, Jan. 2008.

[59] S. Abu-Surra, D. Divsalar, and W. Ryan, "Enumerators for protograph-based ensembles of LDPC and generalized LDPC codes," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 856–886, Feb. 2011.

[60] E. Sharon, A. Ashikhmin, and S. Litsyn, "EXIT functions for binary input memoryless symmetric channels," *IEEE Transactions on Communications*, vol. 54, no. 7, pp. 1207–1214, 2006.

[61] E. Sharon, A. Ashikhmin, and S. Litsyn, "Analysis of low-density parity-check codes based on EXIT functions," *IEEE Transactions on Communications*, vol. 54, no. 8, pp. 1407–1414, 2006.

[62] E. Sharon, R. Zamir, and D. Avraham, "Generalized low density parity check codes," in *Proceedings of 10th Annual Non-Volatile Memories Workshop*, Mar. 2019.

[63] G. Yue, L. Ping, and X. Wang, "Low-rate generalized low-density parity-check codes with Hadamard constraints," in *Proceedings of International Symposium on Information Theory (ISIT)*, pp. 1377–1381, 2005.

[64] D. J. Costello and G. D. Forney, "Channel coding: The road to channel capacity," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1150–1177, 2007.

[65] R. M. Buehrer, "Code division multiple access (CDMA)," *Synthesis Lectures on Communications*, vol. 1, no. 1, pp. 1–192, 2006.

[66] L. Ping, L. Liu, K. Y. Wu, and W. K. Leung, "Approaching the capacity of multiple access channels using interleaved low-rate codes," *IEEE Communications Letters*, vol. 8, no. 1, pp. 4–6, 2004.

[67] L. Ping, "Interleave-division multiple access and chip-by-chip iterative multi-user detection," *IEEE Communications Magazine*, vol. 43, no. 6, pp. S19–S23, 2005.

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

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

[70] H. H. Xu and Z. S. Bie, "Gbps turbo-Hadamard codes," in *Proceedings of 2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pp. 1–5, 2018.

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

[72] S. Jiang, P. W. Zhang, F. C. M. Lau, and C.-W. Sham, "An ultimate-Shannon-limit-approaching Gbps throughput encoder/decoder system," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 10, pp. 2169–2173, 2020.

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

[74] S. Jiang, F. C. M. Lau, and C.-W. Sham, "Hardware design of concatenated zigzag Hadamard encoder/decoder system with high throughput," *IEEE Access*, vol. 8, pp. 165298–165306, 2020.

[75] Y. Liu, P. M. Olmos, and D. G. M. Mitchell, "On generalized LDPC codes for 5G ultra reliable communication," in *Proceedings of IEEE Information Theory Workshop (ITW)*, pp. 1–5, 2018.

[76] P. W. Zhang, F. C. M. Lau, and C.-W. Sham, "Protograph-based LDPC-Hadamard codes," in *Proceedings of 2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, 2020.

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

[78] L. Chen, S. Mo, D. J. Costello, D. G. M. Mitchell, and R. Smarandache, "A protograph-based design of quasi-cyclic

spatially coupled LDPC codes," in *Proceedings of 2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 1683–1687, 2017.

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

[80] S. Myung, K. Yang, and J. Kim, "Quasi-cyclic LDPC codes for fast encoding," *IEEE Transactions on Information Theory*, vol. 51, no. 8, pp. 2894–2901, 2005.

[81] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," in *Proceedings of Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 8–15, 2002.

[82] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, 2005.

[83] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 985–994, 2009.

[84] Z. Cui, Z. Wang, and Y. Liu, "High-throughput layered LDPC decoding architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 4, pp. 582–587, 2009.

[85] E. Boutillon, F. Guillou, and J.-L. Danger, "Lambda-min decoding algorithm of regular and irregular LDPC codes," in *Proceedings of 3nd International Symposium on Turbo Codes and Related Topics*, 2003.

[86] J. Chen, R. Tanner, C. Jones, and Y. Li, "Improved min-sum decoding algorithms for irregular LDPC codes," in *Proceedings of International Symposium on Information Theory*, pp. 449–453, 2005.

[87] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

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

[89] D. V. Nguyen and B. Vasic, "Two-bit bit flipping algorithms for LDPC codes and collective error correction," *IEEE Transactions on Communications*, vol. 62, no. 4, pp. 1153–1163, 2014.

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

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

[92] T. Xia, T. Koike-Akino, D. S. Millar, K. Kojima, K. Parsons, Y. Miyata, K. Sugihara, and W. Matsumoto, "Dynamic window decoding for LDPC convolutional codes in low-latency optical communications," in *Proceedings of 2015 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–3, 2015.

[93] P. W. Zhang, F. C. M. Lau, and C.-W. Sham, "Protograph-based low-density parity-check Hadamard codes," *IEEE Transactions on Communications*, vol. 69, no. 8, pp. 4998–5013, 2021.

[94] P. W. Zhang, F. C. M. Lau, and C.-W. Sham, "Protograph-based low-density parity-check Hadamard codes," *https://arxiv.org/abs/2010.08285*, 2020.

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

[96] P. W. Zhang, F. C. M. Lau, and C.-W. Sham, "Layered decoding for protograph-based low-density parity-check Hadamard codes," *IEEE Communications Letters*, vol. 25, no. 6, pp. 1776–1780, 2021.

[97] P. W. Zhang, F. C. M. Lau, and C.-W. Sham, "Hardware architecture of layered decoders for PLDPC-Hadamard codes," in preparation.

[98] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proceedings of IEEE Workshop on Signal Processing Systems, 2004*, pp. 107–112, 2004.

[99] J. Ha, D. Klinc, J. Kwon, and S. W. Mclaughlin, "Layered BP decoding for rate-compatible punctured LDPC codes," *IEEE Communications Letters*, vol. 11, no. 5, pp. 440–442, 2007.

[100] G. Han and X. Liu, "An efficient dynamic schedule for layered belief-propagation decoding of LDPC codes," *IEEE Communications Letters*, vol. 13, no. 12, pp. 950–952, 2009.

[101] M. Ferrari, S. Bellini, and A. Tomasoni, "Safe early stopping for layered LDPC decoding," *IEEE Communications Letters*, vol. 19, no. 3, pp. 315–318, 2015.

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