

Hardware Architecture of Layered Decoders for PLDPC-Hadamard Codes

Peng W. Zhang, Francis C.M. Lau, *Fellow, IEEE*, and
Chiu-W. Sham, *Senior Member, IEEE*

Abstract

Protograph-based low-density parity-check Hadamard codes (PLDPC-HCs) are a new type of ultimate-Shannon-limit-approaching codes. In this paper, we propose a hardware architecture for the PLDPC-HC layered decoders. The decoders consist mainly of random address memories, Hadamard sub-decoders and control logics. Two types of pipelined structures are presented and the latency and throughput of these two 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.

Index Terms

hardware design, layered decoding, PLDPC-Hadamard code

I. INTRODUCTION

Both turbo codes [1] and low-density parity-check (LDPC) codes [2] have been demonstrated to be capacity-approaching channel codes [3], [4]. They have been used in a wide variety of communication and data storage systems [5], including 3G/4G/5G cellular communications, optical communications, and magnetic recording systems, [6], [7], [8]; and various encoder/decoder designs have been proposed [9], [10], [11], [12]. In particular, turbo codes can employ the serial Bahl-Cocke-Jelinek-Raviv (BCJR) computational method in the iterative decoding algorithm to

P.W. Zhang and F.C.M. Lau are with the Future Wireless Networks and IoT Focusing Area, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong (e-mail: pengwei.zhang@connect.polyu.hk and francis-cm.lau@polyu.edu.hk).

C.-W. Sham is with the Department of Computer Science, The University of Auckland, New Zealand (e-mail: b.sham@auckland.ac.nz).

approach the channel capacity [13]. When implementing the BCJR decoder on hardware, state metric normalization and ungrouped backward recursion techniques are developed to reduce the critical path delay and to enhance the clock frequency [14]. Furthermore, by eliminating the data dependencies from the computations in the BCJR algorithm, a fully-parallel turbo decoding algorithm has been proposed [15]. The proposed method not only reduces the complexity of the original algorithm, but also increases the parallelism of the decoder and improves the throughput.

As to the LDPC codes, the structured quasi-cyclic (QC) LDPC codes allow easy realization of linear encoding and parallel decoding. QC-LDPC codes can be constructed from the perspective of a protograph. By lifting a protograph containing a small number of variable nodes and check nodes, QC-LDPC codes called protograph-based LDPC (PLDPC) codes are formed [16], [17]. It has also been shown that well-designed QC-LDPC codes can achieve good decoding performance, low error floor and high throughput [18], [19], [20], [21], [22]. For example, a rate-compatible layered decoding architecture that allows parallel decoding of QC-LDPC codes has been shown to achieve a throughput of 1.28 Gbps [19]. By reordering the layered decoding procedure and applying other optimization techniques, the layered decoder throughput is shown to increase to 4.67 Gbps [20]. With a multi-core architecture and a full row-parallel layered decoder, a throughput of 860 Gbps is achievable at a maximum of 2 decoding iterations [21]. Moreover, a RAM-based decoder architecture is also proposed to decode cyclically-coupled QC-LDPC codes and obtains a throughput of 3.0 Gbps and an error floor of about 10^{-16} [22].

When both turbo and LDPC codes are used together with Hadamard codes, forming turbo-Hadamard codes [23] and LDPC-Hadamard codes (LDPC-HCs) [24], respectively, very good error performance can be achieved even when operating close to the ultimate Shannon limit (i.e., bit-energy-to-noise-power-spectral-density ratio (E_b/N_0) equals -1.59 dB) [25]. Another ultimate-Shannon-limit-approaching code is the concatenated zigzag-Hadamard code [26]. Among these three types of codes, LDPC-HCs have been shown to produce the best error performance. For example, a rate-0.05 LDPC-HC with a theoretical threshold of -1.35 dB can achieve a bit error rate (BER) of 10^{-5} at $E_b/N_0 = -1.18$ dB [24]. These ultimate-Shannon-limit-approaching codes can be applied to extreme communication environments such as deep-space communications and interleaved division multiple access systems with many users [27].

Recently, a new type of LDPC-HCs called protograph-based LDPC Hadamard codes (PLDPC-HCs) have been proposed, and a new technique is developed to enable the analysis of PLDPC-HCs which possess degree-1 and/or punctured variable nodes [28], [29]. PLDPC-HCs perform

as good as traditional LDPC-HCs. For instance, a rate-0.0494 PLDPC-HC with a theoretical threshold of -1.42 dB is found to achieve a BER of 10^{-5} at $E_b/N_0 = -1.19$ dB. In addition, PLDPC-HC possesses a semi-regular ¹ quasi-cyclic structure which is beneficial to hardware implementation. To improve the convergence rate, a PLDPC-HC layered decoding algorithm has been proposed [30]. In this paper, we propose a hardware architecture for PLDPC-HC layered decoders. The proposed architecture is generic and can be readily modified to decode other PLDPC-derived codes when the Hadamard constraint in the PLDPC-HC is replaced by other coding constraints.

The paper is organized as follows. Section II reviews the structure of a PLDPC-HC and its layered decoding algorithm. Section III first introduces the read and write operations of a random access memory and the pipeline structure of a Hadamard sub-decoder. Then it presents a hardware architecture of PLDPC-HC layered decoders, and derives its latency and throughput. Section IV shows the implementation results and finally Section V gives some concluding remarks.

II. REVIEW OF PLDPC-HADAMARD CODES

The structure of a PLDPC-HC can be constructed from a PLDPC code [16]. When the check nodes in a PLDPC code are replaced by Hadamard check-nodes (H-CNs) to which an appropriate number of degree-1 Hadamard variable nodes (D1H-VNs) are connected, a PLDPC-HC is formed [28], [29]. Fig. 1 illustrates the base matrix $\mathbf{B}_{m \times n}$ of a PLDPC-HC and its corresponding protograph. As can be observed, there are $n = 11$ protograph variable nodes (P-VNs) and $m = 7$ H-CNs. Moreover, each H-CN is connected to a number of D1H-VNs. The (i, j) -th entry in $\mathbf{B}_{m \times n}$, represented by $B(i, j)$, denotes the number of edges connected between the i -th H-CN and the j -th P-VN. In this example, each H-CN is connected to $d = 6$ P-VNs, where d also equals the row weight of the base matrix $\mathbf{B}_{m \times n}$. To obtain the adjacency matrix $\mathbf{H}_{M \times N}$ of the PLDPC-HC, the base matrix $\mathbf{B}_{m \times n}$ is lifted twice with factors z_1 and z_2 where $M = mz_1z_2$ and $N = nz_1z_2$ [31]. The first lifting replaces each non-zero $B(i, j)$ in $\mathbf{B}_{m \times n}$ with a summation of $B(i, j)$ different $z_1 \times z_1$ permutation matrices, and each $B(i, j) = 0$ with $z_1 \times z_1$ zero matrix. The aim is to remove parallel edges between P-VNs and H-CNs. The second lifting then replaces each “1” with a circulant permutation matrix (CPM) of size $z_2 \times z_2$ and each “0” with the $z_2 \times z_2$ zero matrix. The aim is to construct a quasi-cyclic code structure for easy encoding

¹In the protograph of a PLDPC-HC [28], [29], the degrees of the protograph variable nodes can be different while the degrees of Hadamard check nodes are kept the same.

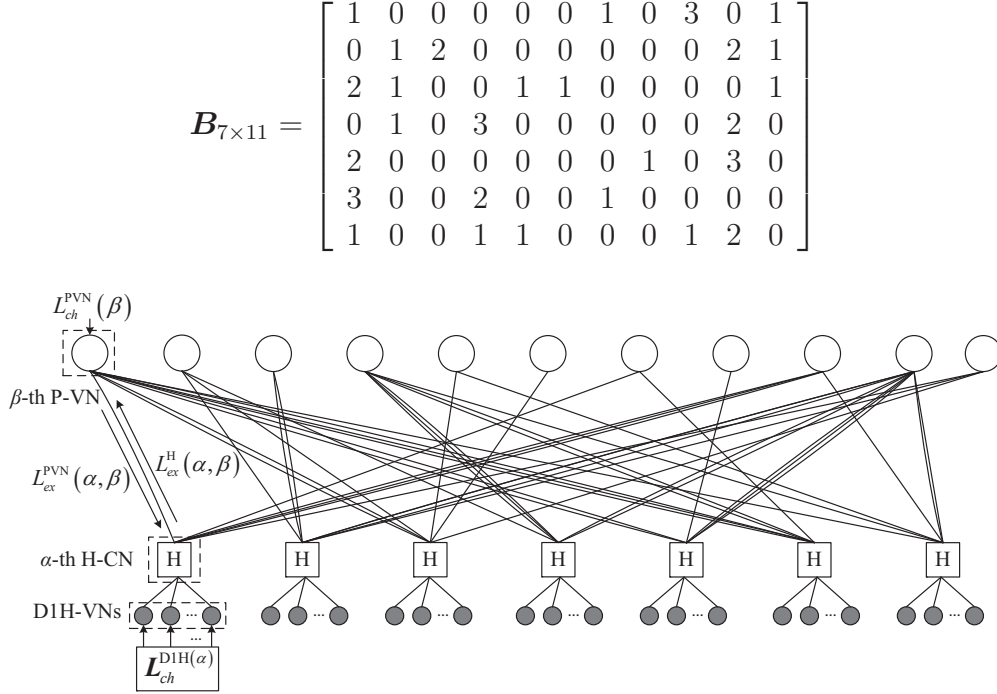


Fig. 1. The base matrix and corresponding protograph of a PLDPC-Hadamard code [28], [29]. A circle denotes a protograph variable node (P-VN), a square with “H” denotes a Hadamard check node (H-CN), and a filled circle denotes a degree-1 Hadamard variable node (D1H-VN). Row weight $d = 6$, Hadamard order $r = d - 2 = 4$, and $2^r - r - 2 = 10$ D1H-VNs are attached to each H-CN. Code rate $R = 0.0494$.

and decoding [32]. After the double-lifting process, the lifted graph, which corresponds to the adjacency matrix, contains M H-CNs and N P-VNs.

Based on the adjacency matrix $\mathbf{H}_{M \times N}$ obtained, $N - M$ information bits are first encoded into a length- N LDPC code. Then for each H-CN, the d incoming messages from the P-VNs are used to encode an order r ($= d - 2$) Hadamard code [28], [29]. Supposing r is even, $2^r - r - 2$ Hadamard parity-bits are generated and attached to each H-CN as D1H-VNs. The overall code rate of the PLDPC-Hadamard code is therefore

$$R = \frac{n - m}{m(2^r - r - 2) + n}.$$

Throughout this paper, we assume that d is even. When d is odd, $2^r - 2$ Hadamard parity-bits are generated, and the encoding and decoding algorithms become slightly different [28], [29].

To speed up the convergence speed, a layered decoding algorithm has been proposed [30]. For $\alpha = 0, 1, \dots, M - 1$ and $\beta = 0, 1, \dots, N - 1$, we denote

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

TABLE I
NUMBERS OF H-CNS, P-VNS AND D1H-VNS CONTAINED IN ONE LAYER WHEN r IS EVEN. $r = d - 2$.

No. of H-CNs	No. of P-VNs	No. of D1H-VNs
z_2	dz_2	$(2^r - d)z_2$

- $\mathcal{H}(\beta)$ as the set of H-CNs connected to the β -th P-VN;
- $L_{ch}^{PVN}(\beta)$ as the channel log-likelihood-ratio (LLR) value of the β -th P-VN;
- $\mathbf{L}_{ch}^{D1H(\alpha)}$ as a vector consisting of the channel LLR values of the D1H-VNs connected to the α -th H-CN;
- $L_{app}^{PVN}(\beta)$ as the *a posteriori* probability (APP) LLR value of the β -th P-VN;
- $L_{ex}^{PVN}(\alpha, \beta)$ as the extrinsic LLR value from the β -th P-VN to the α -th H-CN;
- $L_{app}^H(\alpha, \beta)$ as the APP LLR value computed by the α -th H-CN for the β -th P-VN;
- $L_{ex}^H(\alpha, \beta)$ as the extrinsic LLR value sent from the α -th H-CN to the β -th P-VN.

After lifting the base matrix of a PLDPC-HC two times, the resultant adjacency matrix $\mathbf{H}_{M \times N}$ is divided into mz_1 layers (also called block rows), where each layer is composed of $1 \times nz_1$ CPMs each of size $z_2 \times z_2$. Hence, each layer corresponds to a $z_2 \times nz_1z_2$ matrix and contains z_2 H-CNs. Since each H-CN connects d P-VNs and $2^r - d$ D1H-VNs (when r is even), the z_2 H-CNs in one layer connects dz_2 P-VNs and $(2^r - d)z_2$ D1H-VNs. Table I summarizes of the numbers of H-CNs, P-VNs and D1H-VNs contained in one layer.

Defining k as the layer number ($k = 0, 1, \dots, mz_1 - 1$) and $\mathcal{L}(k) = \{\alpha_{kz_2}, \alpha_{kz_2+1}, \dots, \alpha_{kz_2+z_2-1}\}$ as the set of H-CNs in layer k , the layered decoding algorithm is described as follows [30].

- 1) Initialization: Set $L_{app}^{PVN}(\beta) = L_{ch}^{PVN}(\beta) \forall \beta$; and set $L_{ex}^H(\alpha, \beta) = 0 \forall \alpha, \beta$.
- 2) Symbol maximum-a-posterior Hadamard sub-decoder: Set $k = 0$.
 - a) For the α -th H-CN in layer k ($\alpha \in \mathcal{L}(k)$), perform the following computations.
 - i) For $\beta \in \mathcal{P}(\alpha)$, compute

$$L_{ex}^{PVN}(\alpha, \beta) = L_{app}^{PVN}(\beta) - L_{ex}^H(\alpha, \beta) \quad \forall \beta \in \mathcal{P}(\alpha). \quad (1)$$

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

$$\mathbf{L}_{app}^H(\alpha) = \{L_{app}^H(\alpha, \beta) : \beta \in \mathcal{P}(\alpha)\}$$

$$= \mathcal{T} \left[\{L_{ex}^{PVN}(\alpha, \beta) : \beta \in \mathcal{P}(\alpha)\}, \mathbf{L}_{ch}^{D1H(\alpha)} \right] \quad (2)$$

where \mathcal{T} is a transformation involving the fast Hadamard transform (FHT) and the dual FHT (DFHT) operations [24], [28], [29].

iii) Update $L_{ex}^H(\alpha, \beta)$ and $L_{app}^{PVN}(\beta)$ using

$$L_{ex}^H(\alpha, \beta) = L_{app}^H(\alpha, \beta) - L_{ex}^{PVN}(\alpha, \beta); \quad \forall \beta \in \mathcal{P}(\alpha) \quad (3)$$

$$L_{app}^{PVN}(\beta) = L_{app}^H(\alpha, \beta); \quad \forall \beta \in \mathcal{P}(\alpha). \quad (4)$$

b) If the last layer has not been reached, i.e., $k < mz_1 - 1$, increment k by 1 and go to Step 2a).

3) Repeat Step 2) I times and make decisions on the P-VNs based on the sign of $L_{app}^{PVN}(\beta)$ $\forall \beta$.

Note that the layered decoding algorithm neither returns any extrinsic information to the D1H-VNs nor makes hard decisions on the D1H-VNs. The algorithm only makes use of the channel information provided by the D1H-VNs to aid the decoding of the PLDPC code and hence the P-VNs.

III. HARDWARE DESIGN OF THE LAYERED DECODER

This section presents and analyzes a hardware implementation of the layered decoding algorithm for PLDPC-HC. First, we present the read and write operations of LLR values in random access memories (RAMs). Second, we describe the pipeline structure of the symbol-maximum-a-posterior (symbol-MAP) Hadamard sub-decoder, which is composed mainly of FHT and DFHT components. Third, we combine the RAMs and Hadamard sub-decoders and propose a layered decoder architecture for PLDPC-HC. Fourth, we analyze the decoding timing, latency and throughput of the proposed architecture.

A. Read and Write Operations of RAMs

As described in the layered decoding algorithm for the PLDPC-HC, there are six types of LLRs. Among them $\{L_{ex}^{PVN}(\alpha, \beta)\}$ in (1) and $\{\mathbf{L}_{app}^H(\alpha, \beta)\}$ in (2) are only temporary values in the computation process and need not to be stored, whereas the other four types of LLRs,

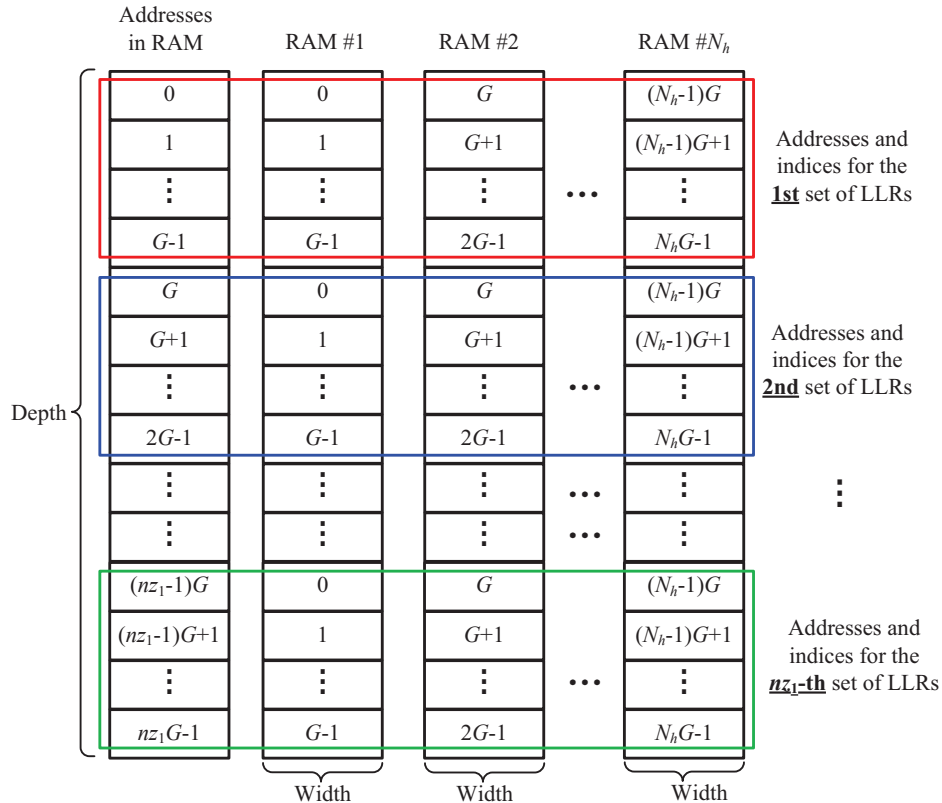


Fig. 2. RAM arrangement for $\{L_{ch}^{PVN}(\beta)\}$ or $\{L_{app}^{PVN}(\beta)\}$ corresponding P-VNs.

i.e., $\{L_{ch}^{PVN}(\beta)\}$, $\{L_{app}^{PVN}(\beta)\}$, $\{L_{ex}^H(\alpha, \beta)\}$ and $\{L_{ch}^{D1H(\alpha)}\}$, are not temporary and thus need to be stored in RAMs.

Referring to Table I, the z_2 H-CNs in each layer connect dz_2 P-VNs and $(2^r - d)z_2$ D1H-VNs (when r is even). Using the layered decoding algorithm to process each layer, we therefore need to retrieve dz_2 values of $\{L_{ch}^{PVN}(\beta)\}$ (during initialization) or dz_2 values of $\{L_{app}^{PVN}(\beta)\}$ (to be used in (1)); and z_2 vectors of $\{L_{ch}^{D1H(\alpha)}\}$. Note that each vector of $\{L_{ch}^{D1H(\alpha)}\}$ contains $2^r - d$ LLR values. According to (1), we also need to retrieve dz_2 values of $\{L_{ex}^H(\alpha, \beta)\}$ in order to compute the dz_2 values of $\{L_{ex}^{PVN}(\alpha, \beta)\}$. In our design, we form sets of LLRs where each set has a size of z_2 — the same size as the second lifting factor. For a PLDPC-HC with an $m \times n$ protomatrix and lifting factors z_1 and z_2 , $\{L_{ch}^{PVN}(\beta)\}$ will be divided into $N/z_2 = n_{z_1}$ sets, $\{L_{app}^{PVN}(\beta)\}$ into $N/z_2 = n_{z_1}$ sets, $\{L_{ex}^H(\alpha, \beta)\}$ into $Md/z_2 = mdz_1$ sets, and $\{L_{ch}^{D1H(\alpha)}\}$ into $M/z_2 = mz_1$ sets.

To achieve reading/retrieving N_h data from memories in one clock cycle, we use N_h RAMs to store each type of LLRs, where $0 < N_h \leq z_2$ and $G = z_2/N_h$ is an integer and represents the

number of groups. Taking $\{L_{ch}^{\text{PVN}}(\beta)\}$ which is related to the P-VNs as an example, each set of LLRs, i.e., a total of z_2 LLR values, is further divided into G groups. Referring to Fig. 2, the addresses $0, 1, \dots, G-1$ in the N_h RAMs are to store the first set of $L_{ch}^{\text{PVN}}(\beta)$. In particular, RAM #1 stores the first group of LLRs, i.e., LLRs with indices $0, 1, \dots, G-1$; RAM #2 stores the second group of LLRs, i.e., LLRs with indices $G, G+1, \dots, 2G-1$; \dots ; and RAM # N_h stores the N_h -th group of LLRs, i.e., LLRs with indices $(N_h-1)G, (N_h-1)G+1, \dots, N_hG-1$. Using a similar fashion, the addresses $G, G+1, \dots, 2G-1$ in the N_h RAMs are to store the second set of $L_{ch}^{\text{PVN}}(\beta)$. The arrangement is repeated until all nz_1 sets of $L_{ch}^{\text{PVN}}(\beta)$ are stored in the N_h RAMs.

With the above storage arrangement, in each clock cycle N_h values from the same LLR set can be retrieved from the N_h RAMs. Using Fig. 2 as an example, at clock $t = 1$ the N_h LLR values stored at Address #0 (with indices $0, G, \dots, (N_h-1)G$) are retrieved; at clock $t = 2$, the N_h LLR values stored at Address #1 (with indices $1, G+1, \dots, (N_h-1)G+1$) are retrieved; \dots ; at clock $t = G$, the N_h LLR values stored at Address # $G-1$ (with indices $G-1, 2G-1, \dots, N_hG-1$) are retrieved. Thus one set of LLR values (i.e., z_2 LLR values) can be retrieved in G clock cycles. Hence, reading or writing d sets of $L_{ch}^{\text{PVN}}(\beta)$ or $L_{app}^{\text{PVN}}(\beta)$ for each layer requires dG clock cycles.

We use a similar storage arrangement for $\{L_{ex}^{\text{H}}(\alpha, \beta)\}$ and $\{\mathbf{L}_{ch}^{\text{D1H}(\alpha)}\}$, which correspond to H-CNs and D1H-VNs, respectively. The only differences are that the RAMs will have different depths and widths. Fig. 3 shows the storage arrangement of $\{L_{ex}^{\text{H}}(\alpha, \beta)\}$ and $\{\mathbf{L}_{ch}^{\text{D1H}(\alpha)}\}$ (corresponding to the first layer) in N_h RAMs. In Fig. 3(a), “ $\alpha = i : \beta_j$ ” ($i = 0, \dots, GN_h-1$; $j = 0, \dots, d-1$) denotes the j -th H-CN connected to the i -th P-VN; and hence “ $\alpha = i : \beta_0$ ” to “ $\alpha = i : \beta_{d-1}$ ” represent $\mathcal{P}(\alpha = i)$, i.e., all the P-VNs connected to the i -th H-CN. In Fig. 3(b), each address stores the $2^r - d$ channel LLRs corresponding to the $2^r - d$ D1H-VNs connected to the same H-CN.

Remark: In the actual hardware implementation, we use dual-port RAMs instead of single-port ones. Since two memory locations in each dual-port RAM can be accessed (read and/or write) at the same time, the number of clock cycles required to read/write one set of LLRs can be further reduced by half compared with the discussion above.

Supposing we have retrieved N_h values for $\{L_{ch}^{\text{PVN}}(\beta)\}$ or $\{L_{app}^{\text{PVN}}(\beta)\}$, we need to interleave them — a process similar to that used in QC-LDPC decoding [33]. For each layer, the exact connections between the H-CNs and the P-VNs are determined by the CPMs, and hence the

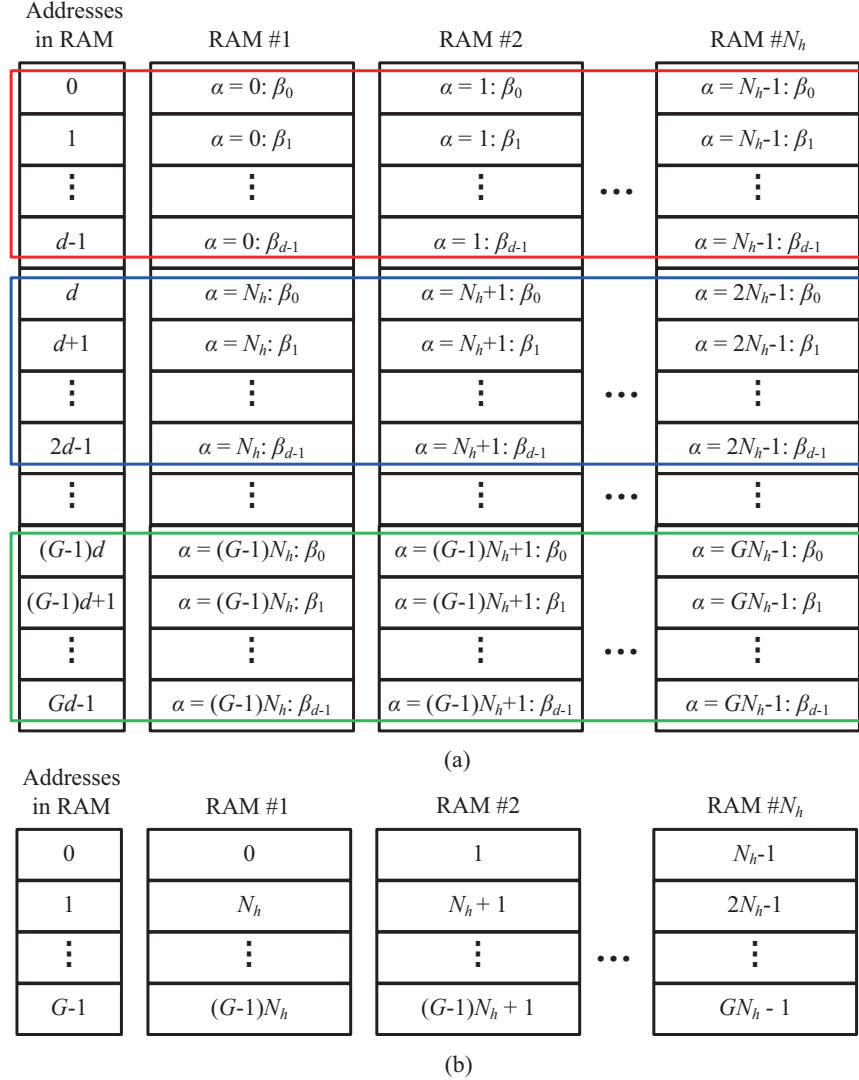


Fig. 3. Storage arrangement of (a) $\{L_{ex}^H(\alpha, \beta)\}$ and (b) $\{L_{ch}^{D1H(\alpha)}\}$ in N_h RAMs. The first layer of H-CN is being considered.

interleaver can be realized by a simple cyclic shifter. Assuming that the offset value of a CPM equals p ($0 \leq p < z_2$), we calculate the quotient $q_u = \lfloor p/G \rfloor$ and the remainder $r_e = p \bmod G$, where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x and “mod” denotes the modulus operation. When $(address \bmod G) < r_e$, the corresponding N_h LLRs are cyclically shifted to the left by $(q_u + 1 \bmod N_h)$; otherwise, these LLRs are cyclically shifted to the left by q_u .

Example: We assume that $z_2 = 16$. Fig. 4(a) shows a 16×16 identity matrix, i.e., a 16×16 CPM with $p = 0$; Fig. 4(b) depicts a 16×16 CPM with $p = 9$, which can be obtained by cyclically shifting the 16×16 identity matrix to the right by 9 columns. Assume that the CPM with $p = 9$ corresponds to one set of $L_{ch}^{PVN}(\beta)$ with indices $[0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15]$.

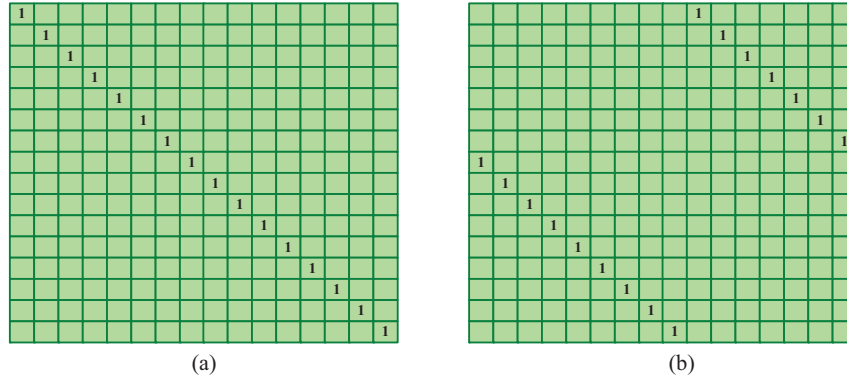


Fig. 4. (a) A 16×16 identity matrix, i.e., a 16×16 circulant permutation matrix (CPM) with $p = 0$; (b) A 16×16 CPM with $p = 9$, which can be obtained by cyclically shifting the 16×16 identity matrix to the right by $p = 9$ columns.

After retrieving these LLRs from the RAMs and interleaving them, these indices are expected to be re-ordered into $[9\ 10\ 11\ 12\ 13\ 14\ 15\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8]$. Suppose we use $N_h = 4$ RAMs to store this set of $L_{ch}^{\text{PVN}}(\beta)$. According to our aforementioned storage scheme, each set of $L_{ch}^{\text{PVN}}(\beta)$ is divided into $G = z_2/N_h = 4$ groups; and each RAM would use the first $G = 4$ addresses, i.e., Addresses #0, #1, #2, #3, to store 4 LLR values. The storage arrangement is shown in Table II. As $p = 9$, we have $q_u = \lfloor p/G \rfloor = 2$ and $r_e = p \bmod G = 1$. Once the $N_h = 4$ LLRs are retrieved, we process them as follows.

- Cyclically shift the LLRs stored at Address #0 ($< r_e = 1$), i.e., LLRs with indices $[0\ 4\ 8\ 12]$, to the left by $q_u + 1 \bmod N_h = 3$ and the order of the indices becomes $[12\ 0\ 4\ 8]$;
- Cyclically shift the LLRs stored at Address #1 ($\geq r_e = 1$) to the left by $q_u = 2$ and the order of the indices becomes $[9\ 13\ 1\ 5]$;
- Cyclically shift the LLRs stored at Address #2 ($\geq r_e = 1$) to the left by $q_u = 2$ and the order of the indices becomes $[10\ 14\ 2\ 6]$;
- Cyclically shift the LLRs stored at Address #3 ($\geq r_e = 1$) to the left by $q_u = 2$ and the order of the indices becomes $[11\ 15\ 3\ 7]$.

Therefore, the expected interleaving effect $[9\ 10\ 11\ 12\ 13\ 14\ 15\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8]$ can be achieved by such a process. Note that the “write” operation can be regarded as the reverse process of the “read” operation. Hence the procedures are similar and are omitted here.

TABLE II
EXAMPLE FOR STORING A SET OF LLRS IN $N_h = 4$ RAMS. $z_2 = 16$ AND $G = z_2/N_h = 4$.

Address	Indices			
	RAM #1	RAM #2	RAM #3	RAM #4
0	0	4	8	12
1	1	5	9	13
2	2	6	10	14
3	3	7	11	15

B. Operation of A Symbol-MAP Hadamard Sub-decoder

The Hadamard sub-decoder can be considered as the kernel of the PLDPC-HC layered decoder in our implementation and hence will be described in detail. For an order- r Hadamard code, the corresponding Hadamard matrices of size $q \times q$ can be recursively constructed by

$$\begin{aligned} \pm \mathbf{H}_q &= \{\pm \mathbf{h}_j, j = 0, 1, \dots, q-1\} \\ &= \begin{bmatrix} \pm \mathbf{H}_{q/2} & \pm \mathbf{H}_{q/2} \\ \pm \mathbf{H}_{q/2} & \mp \mathbf{H}_{q/2} \end{bmatrix} \end{aligned} \quad (5)$$

where $q = 2^r$ equals the code length and $\pm \mathbf{H}_1 = [\pm 1]$. Each column $\pm \mathbf{h}_j$ of the Hadamard matrices corresponds to a Hadamard codeword, and hence there is a total of $2q = 2^{r+1}$ codewords in $\pm \mathbf{H}_q$. In (6), we show the 16×16 Hadamard matrices $\pm \mathbf{H}_{16}$ corresponding to the order- $r = 4$ Hadamard code having $2^{r+1} = 32$ codewords. Note that the codewords are formed by mapping each $+1$ in the Hadamard matrices to bit “0” and each -1 to bit “1”.

When the Hadamard order r is even, it has been proven that there always exists a length- $d = r + 2$ single-parity-check (SPC) codeword “embedded” in each Hadamard codeword [24], [28], [29], i.e.,

$$\begin{aligned} [\pm h_{0,j} \oplus \pm h_{1,j} \oplus \dots \oplus \pm h_{2^{k-1},j} \oplus \dots \oplus \\ \pm h_{2^{r-1},j}] \oplus \pm h_{2^r-1,j} = 0, \end{aligned} \quad (7)$$

where the symbol \oplus represents the XOR operator. (In (6), the length-6 SPC constraint is $\pm h_{0,j} \oplus \pm h_{1,j} \oplus \pm h_{2,j} \oplus \pm h_{4,j} \oplus \pm h_{8,j} \oplus \pm h_{15,j} = 0 \forall j$ and the corresponding 6 bits are marked in red color.) In each H-CN of the PLDPC-HC described in Section II, the length- d SPC codeword is

$$\pm \mathbf{H}_{16} = \begin{bmatrix} \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 \\ \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 \\ \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 \\ \pm 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 \\ \pm 1 & \pm 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 \\ \pm 1 & \mp 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 \\ \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 \\ \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 \\ \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 \\ \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 \\ \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 \\ \pm 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 \\ \pm 1 & \pm 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 \\ \pm 1 & \mp 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 \\ \pm 1 & \pm 1 & \mp 1 & \mp 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 \\ \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \pm 1 & \mp 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 & \mp 1 & \pm 1 \end{bmatrix} \quad (6)$$

formed by the d P-VNs to which the H-CN is connected. Using these d bits as inputs to the Hadamard encoder, $2^r - d$ Hadamard parity-check bits corresponding to the D1H-VNs attached to the H-CN can be generated. (In the case of an order-4 Hadamard code, $d = 6$ bits are input to the Hadamard encoder which generates $2^r - d = 10$ Hadamard parity-check bits.)

To decode Hadamard codes, a symbol-MAP decoding algorithm has been proposed [24], [28], [29]. We define

$$\mathbf{L}_{ch}^H = [L_{ch}^H(0) L_{ch}^H(1) \cdots L_{ch}^H(2^r - 1)]^T, \quad (8)$$

$$\mathbf{L}_{apr}^H = [L_{apr}^H(0) L_{apr}^H(1) \cdots L_{apr}^H(2^r - 1)]^T, \quad (9)$$

$$\mathbf{L}_{app}^H = [L_{app}^H(0) L_{app}^H(1) \cdots L_{app}^H(2^r - 1)]^T, \quad (10)$$

as the channel, the *a priori* and the *a posteriori* LLR information of the coded bit, respectively. Note that \mathbf{L}_{ch}^H contains only $2^r - d$ channel observations coming from the D1H-VNs while the remaining d values are set to 0. On the other hand, \mathbf{L}_{apr}^H has d non-zero values coming from P-VNs (i.e., repeat decoder) while the remaining $2^r - d$ values are set to 0. (Please refer to [28, Section III-B] and [29, Section III-B] for the detailed arrangement of \mathbf{L}_{ch}^H and \mathbf{L}_{apr}^H .) Based on

\mathbf{L}_{ch}^H and \mathbf{L}_{apr}^H , $L_{apr}^H(i)$ is computed using

$$L_{apr}^H(i) = \ln \frac{\sum_{\pm H[i,j]=+1} \gamma(\pm \mathbf{h}_j)}{\sum_{\pm H[i,j]=-1} \gamma(\pm \mathbf{h}_j)}, \quad (11)$$

where $\gamma(\pm \mathbf{h}_j) = \exp(\langle \pm \mathbf{h}_j, \mathbf{L}_{ch}^H + \mathbf{L}_{apr}^H \rangle / 2)$ represents the *a posteriori* “information” of the codeword $\pm \mathbf{h}_j$; and $\langle \cdot \rangle$ denotes the inner-product operator. Since the Hadamard matrix has a butterfly-like structure, our Hadamard decoder design is based on the fast Hadamard transform (FHT) block and the dual FHT (DFHT) block [23], [24], [34].

- 1) We first use a FHT block to compute $\langle +\mathbf{h}_j, \mathbf{L}_{ch}^H + \mathbf{L}_{apr}^H \rangle$. Using the structure of the FHT block for $r = 4$ shown in Fig. 5 as an example, the inputs are $\text{In}_{-j} = L_{ch}^H(j) + L_{apr}^H(j)$ and the outputs are $\text{Out}_{-j} = 2 \ln[\gamma(+\mathbf{h}_j)]$ ($j = 0, 1, \dots, 15$). Then, $\ln[\gamma(+\mathbf{h}_j)]$ is readily obtained from $2 \ln[\gamma(+\mathbf{h}_j)]$ by shifting the least significant bit out. Moreover, $\ln[\gamma(-\mathbf{h}_j)]$ is readily available because $\ln[\gamma(-\mathbf{h}_j)] = -\ln[\gamma(+\mathbf{h}_j)]$. There are $r = 4$ stages in the FHT block and thus a latency of $r = 4$ clock cycles is required.
- 2) The structure of a DFHT block is similar to that of a FHT block, but with twice the number of inputs and outputs. Using the structure of the DFHT block for $r = 4$ shown in Fig. 6 as an example, the inputs to the DFHT block are $\ln[\gamma(+\mathbf{h}_j)]$ and $\ln[\gamma(-\mathbf{h}_j)]$; and the outputs are $\ln\left[\sum_{\pm H[i,j]=+1} \gamma(\pm \mathbf{h}_j)\right]$ and $\ln\left[\sum_{\pm H[i,j]=-1} \gamma(\pm \mathbf{h}_j)\right]$ ($j = 0, 1, \dots, 15$). The module max^* in the DFHT block represents the Jacobian logarithm, i.e.,

$$\begin{aligned} \text{max}^*(a, b) &= \ln[\exp(a) + \exp(b)] \\ &= \max(a, b) + \ln[1 + \exp(-|a - b|)] \end{aligned} \quad (12)$$

where $\max(a, b)$ returns the greater value between a and b . In our design, we use a comparison operation to realize $\max(a, b)$, a look-up-table to realize $\ln[1 + \exp(-|a - b|)]$ and an addition operation to sum the above outputs.

As we only need to feedback values related to the $r + 2$ information bits, the structure of DFHT block can be further simplified to minimize resources requirement. Same as the FHT block, the DFHT block contains r stages and thus has a latency of r clock cycles.

Finally, for $i = 0, 1, \dots, 2^{k-1}, \dots, 2^{r-1}, 2^r - 1$, it takes another clock cycle to compute

- the $r + 2$ LLR values $L_{app}^H(i)$ which equals

$$\ln \left[\sum_{\pm H[i,j]=+1} \gamma(\pm \mathbf{h}_j) \right] - \ln \left[\sum_{\pm H[i,j]=-1} \gamma(\pm \mathbf{h}_j) \right],$$

- the $r + 2$ extrinsic LLR messages $L_{ex}^H(i)$ which is computed using (3).

Overall, it takes $2r + 1$ clock cycles to complete one set of computation. Note that the FHT and DFHT blocks have pipeline structures, and the results computed in each stage will be stored in registers. To simplify the presentations of the structures of the FHT block (Fig. 5) and the DFHT block (Fig. 6), we omit all connections to the clock in the figures.

C. Layered Decoder Architecture

Referring to Fig. 7, we propose an architecture of PLDPC-Hadamard layered decoder based on RAMs and Hadamard sub-decoders. Moreover, we assume that there are N_h Hadamard sub-decoders. In addition to RAMs and sub-decoders, the architecture contains control logics. The control logics are dependent on the structure of the 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.

Using the read/write operations described in Section III-A, each set of LLRs, i.e., z_2 LLRs or z_2 vectors, is first divided into G groups and then each group of LLRs is stored in one of the N_h RAMs, where $G = z_2/N_h$. With this storage method, we can retrieve N_h values of $\{L_{app}^{PVN}(\beta)\}$ (or $\{L_{ch}^{PVN}(\beta)\}$), N_h values of $\{L_{ex}^H(\alpha, \beta)\}$ and N_h vectors of $\{\mathbf{L}_{ch}^{D1H(\alpha)}\}$ from the N_h RAMs in each clock cycle when single-port RAMs are used; and twice the number of LLRs values/vectors when dual-port RAMs are used. Once dN_h values of $\{L_{app}^{PVN}(\beta)\}$ (or $\{L_{ch}^{PVN}(\beta)\}$), dN_h values of $\{L_{ex}^H(\alpha, \beta)\}$ and dN_h vectors of $\{\mathbf{L}_{ch}^{D1H(\alpha)}\}$ are retrieved, the N_h Hadamard sub-decoders can operate on these N_h individual batches of independent data. To ensure that no conflict of memory access occurs during the decoding process, we design the size and storage of RAMs as follows.

- N_h RAMs, denoted by PVN-CH-RAM, are used to store $\{L_{ch}^{PVN}(\beta) : \beta = 0, 1, \dots, N - 1\}$. Each RAM has a width of w_{ch}^{PVN} bits (to represent the quantized LLR value) and a depth of nz_1G . Referring to Fig. 2, the g -th location ($g = 0, 1, \dots, nz_1G - 1$) in the l -th RAM ($l = 0, 1, \dots, N_h - 1$) stores $L_{ch}^{PVN}(\beta)$ where $\beta = \lfloor g/G \rfloor z_2 + lG + (g \bmod G)$. 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 $\{L_{app}^{PVN}(\beta) : \beta = 0, 1, \dots, N-1\}$. 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 ($g = 0, 1, \dots, nz_1G - 1$) in the l -th RAM ($l = 0, 1, \dots, N_h - 1$) stores $L_{app}^{PVN}(\beta)$ where $\beta = \lfloor g/G \rfloor z_2 + lG + (g \bmod G)$.
- N_h RAMs, denoted by H-EX-RAM, are used to store $\{L_{ex}^H(\alpha, \beta) : \alpha = 0, 1, \dots, M - 1; \beta \in \{\beta_0, \beta_1, \dots, \beta_{d-1}\} = \mathcal{P}(\alpha)\}$. Each RAM has a width of w_{ex}^H bits and a depth of mdz_1G . Referring to Fig. 3(a), the q -th location ($q = 0, 1, \dots, mdz_1G - 1$) in the l -th RAM ($l = 0, 1, \dots, N_h - 1$) stores $\{L_{ex}^H(\alpha, \beta)\}$ where $\alpha = \lfloor q/d \rfloor N_h + l$, $\beta = \beta_\delta$, and $\delta = q \bmod d$.
- N_h RAMs, denoted by D1H-CH-RAM, are used to store $\{\mathbf{L}_{ch}^{D1H(\alpha)} : \alpha = 0, 1, \dots, M - 1\}$. 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 D1H-VNs connected to a H-CN. Referring to Fig. 3(b), the w -th location ($w = 0, 1, \dots, mz_1G - 1$) in the l -th RAM ($l = 0, 1, \dots, N_h - 1$) stores $\{\mathbf{L}_{ch}^{D1H(\alpha)}\}$ where $\alpha = wN_h + l$. (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, we use dual-port RAMS — 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.

D. Latency and Throughput

1) $N_h = z_2$: We first consider a special case in which maximum parallelism is designed for each layer. In other words, we consider the case where $N_h = z_2$ and $G = z_2/N_h = 1$. We also assume dual-port RAMs are used and hence two memory addresses can be accessed at the same time and it takes $d/2$ clock cycles to retrieve the required d sets of $L_{app}^{PVN}(\beta)$ (or $L_{ch}^{PVN}(\beta)$) and $L_{ex}^H(\alpha, \beta)$ values in each layer. Note that $\{L_{ex}^{PVN}(\alpha, \beta)\}$ in (1) is computed in the same clock cycle as $L_{app}^{PVN}(\beta)$ (or $L_{ch}^{PVN}(\beta)$) and $L_{ex}^H(\alpha, \beta)$ are retrieved. At the $d/2$ -th clock cycle, we also load the required z_2 sets of $\mathbf{L}_{ch}^{D1H(\alpha)}$ from one address location to the sub-decoders. Subsequently, dz_2 LLRs of $\{L_{ex}^{PVN}(\alpha, \beta)\}$ and z_2 vectors of $\{\mathbf{L}_{ch}^{D1H(\alpha)}\}$ are passed to the z_2 FHT blocks in the z_2 Hadamard sub-decoders, i.e., d LLRs of $\{L_{ex}^{PVN}(\alpha, \beta)\}$ and one vector of $\{\mathbf{L}_{ch}^{D1H(\alpha)}\}$ to

one FHT block in one Hadamard sub-decoder. Then, it takes $2r + 1$ clock cycles to compute dz_2 LLRs of $\{L_{ex}^H(\alpha, \beta)\}$ and dz_2 LLRs of $\{L_{app}^{PVN}(\beta)\}$ using (3) and (4), respectively. Finally, it takes another $d/2$ clock cycles to write these updated $L_{app}^{PVN}(\beta)$ and $L_{ex}^H(\alpha, \beta)$ values into the RAMs.

To summarize,

- i) Clock cycle no. 1 to $d/2$: read $\{L_{app}^{PVN}(\beta)\}$ (or $L_{ch}^{PVN}(\beta)$) and $\{L_{ex}^H(\alpha, \beta)\}$ from memory, and at the same time compute $\{L_{ex}^{PVN}(\alpha, \beta)\}$ using (1);
- ii) Clock cycle no. $d/2$ (in parallel with above): read $\{\mathbf{L}_{ch}^{D1H(\alpha)}\}$;
- iii) Clock cycle no. $d/2 + 1$ to $d/2 + 2r$: process the inputs $\{L_{ex}^{PVN}(\alpha, \beta)\}$ and $\{\mathbf{L}_{ch}^{D1H(\alpha)}\}$ by the Hadamard sub-decoders (consisting of FHT and DFHT blocks) using (2);
- iv) Clock cycle no. $d/2 + 2r + 1$: compute $\{L_{ex}^H(\alpha, \beta)\}$ and $\{L_{app}^{PVN}(\beta)\}$ using (3) and (4);
- v) 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.

When $N_h = z_2$, maximum parallelism for each layer is achieved. The latency is minimized and the throughput of the decoder is maximized. However, such a design consumes a lot of hardware resources (a large number of RAMs and N_h Hadamard sub-decoders) and may not be practical. In the next section, we consider the cases when N_h is smaller than z_2 .

2) $N_h < z_2$: We consider the case when $N_h < z_2$ and $G = z_2/N_h$ is an integer. Using the proposed decoder architecture, $G(> 1)$ groups of H-CNs (each consisting of N_h H-CNs) are sequentially processed in each layer. Referring to the timing details in Section III-B and Section III-D1 and with the use of our RAM designs, it takes $d/2$ clock cycles to load the data of one group of H-CNs. (Recall that dual-port RAMs are used.) 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 $t_{loading} = dG/2$ clock cycles. Moreover, the first set of outputs (i.e., $L_{app}^{PVN}(\beta)$ and $L_{ex}^H(\alpha, \beta)$) is available at the $t_{1st\ output} = (d/2 + 2r + 1)$ -th clock cycle.

a) *Case I* $t_{loading} \leq t_{1st\ output}$: It means that 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.,

$$\begin{aligned} t_{l1} &= t_{loading} + (2r + 1) + d/2 \\ &= (r/2 + 1)G + 5r/2 + 2 \end{aligned} \quad (13)$$

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

$$\begin{aligned} t_{c1} &= Imz_1 t_{l1} / f_c \\ &= Imz_1 [(r/2 + 1)G + 5r/2 + 2] / f_c, \end{aligned} \quad (14)$$

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

$$\begin{aligned} T_1 &= \frac{l}{t_{c1}} = \frac{[nz_1z_2 + mz_1z_2(2^r - r - 2)] f_c}{Imz_1 t_{l1}} \\ &= \frac{[n/m + (2^r - r - 2)] z_2 f_c}{I[(r/2 + 1)G + (5r/2 + 2)]}. \end{aligned} \quad (15)$$

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

Example: Taking $r = 4$, $d = r + 2 = 6$, $z_2 = 512$ and $N_h = 128$ as an example, we have $t_{loading} = dG/2 = 12$ and $t_{1st\ output} = (d/2 + 2r + 1) = 12$. Fig. 8 shows the timing diagram for the decoding of one layer, in which the LLR data is divided into $G = z_2/N_h = 4$ groups.

- i) Clock cycle no. 1 to $dG/2 = 12$: We load $G = 4$ groups of $\{L_{ex}^{PVN}(\alpha, \beta)\}$ into the Hadamard sub-decoders corresponding to the z_2 H-CNs in the layer in $dG/2 = 12$ clock cycles. In each clock cycle, $2N_h = 256$ LLRs of $\{L_{app}^{PVN}(\beta)\}$ (or $L_{ch}^{PVN}(\beta)$) and 256 LLRs of $\{L_{ex}^H(\alpha, \beta)\}$ are read from RAMs, and at the same time 256 LLRs of $\{L_{ex}^{PVN}(\alpha, \beta)\}$ are computed using (1) and loaded into the $N_h = 128$ Hadamard sub-decoders. Therefore, it takes $d/2 = 3$ clock cycles to completely retrieve all LLR values belonging to the first group, i.e., $dN_h = 6 \times 128 = 768$ LLRs of $\{L_{app}^{PVN}(\beta)\}$ (or $L_{ch}^{PVN}(\beta)$) and 768 LLRs of $\{L_{ex}^H(\alpha, \beta)\}$, and to compute and load 768 LLRs of $\{L_{ex}^{PVN}(\alpha, \beta)\}$ into the $N_h = 128$ Hadamard sub-decoders. Referring to Fig. 8, we use the symbol “1g12” to represent the LLRs corresponding to the first and second P-VNs in Group #1, “1g34” to represent the

LLRs corresponding to the third and fourth P-VNs in Group #1, and “1g56” to represent the LLRs corresponding to the fifth and sixth P-VNs in Group #1. Moreover, “Zg12”, “Zg34” and “Zg56” where $Z = 2, 3, 4$ are defined in a similar fashion. Thus, LLR values belonging to Group #1 are retrieved during Clock cycle no. #1 to #3; Group #2 during Clock cycle no. #4 to #6; Group #3 during Clock cycle no. #7 to #9; and Group #4 during Clock cycle no. #10 to #12.

- ii) Clock cycle no. 3, 6, 9 and 12: We load $G = 4$ groups of $\{\mathbf{L}_{ch}^{D1H(\alpha)}\}$ into the Hadamard sub-decoders. At clock no. 3, we load the channel LLRs for D1H-VNs in Group #1 into the $N_h = 128$ Hadamard sub-decoders. Referring to Fig. 8, we use the symbol “1gllr” to represent these LLRs in Group #1. Similarly, at clock no. 6, 9 and 12, we load the channel LLRs for D1H-VNs in Group #2, Group #3 and Group #4, respectively, into the $N_h = 128$ Hadamard sub-decoders. They are represented by “Zgllr” in Fig. 8 where $Z = 2, 3, 4$.
- iii) Clock cycle no. 4 to 21: We decode one layer consisting of z_2 H-CNs in a pipeline manner. At Clock cycle no. 4, the Hadamard sub-decoders starts processing the LLRs belonging to Group #1 which has completed its LLR loading at Clock cycle no. 3. Similarly, at Clock cycle no. 7, 10 and 13, the Hadamard sub-decoders starts processing the LLRs belonging to Group #2, Group #3 and Group #4, respectively. Since it takes $2r + 1 = 9$ clock cycles to process each group of LLRs and the groups of LLRs are processed in a pipeline manner, the last group of LLRs will be processed completely at Clock cycle no. $13 + 9 - 1 = 21$.
- iv) Clock cycle no. 13 to $t_{l1} = 24$: We write the updated $G = 4$ groups of data into the corresponding RAMs. Referring to the step above, at Clock cycle no. 4, 7, 10 and 13, the Hadamard sub-decoders starts processing the LLRs belonging to Group #1, Group #2, Group #3 and Group #4, respectively. Moreover, at Clock cycle no. 12, 15, 18 and 21, the Hadamard sub-decoders has completed processing the LLRs belonging to Group #1, Group #2, Group #3 and Group #4, respectively; and has each time generated a group of LLRs consisting of 768 LLR values of $\{L_{app}^{PVN}(\beta)\}$ and 768 LLR values of $\{L_{ex}^H(\alpha, \beta)\}$. In a similar fashion as in Step i), it takes $d/2$ clock cycles to store/write the LLRs belonging to one group. Thus, LLRs belonging to Group #1 are stored during Clock cycle no. 13 to 15; Group #2 stored during Clock cycle no. 16 to 18; Group #3 stored during Clock cycle no. 19 to 21; Group #4 stored during Clock cycle no. 22 to 24. In other words, from Clock cycle no. 13 to 24, $2 \times 128 = 256$ updated LLR values of $\{L_{app}^{PVN}(\beta)\}$ are stored into $N_h = 128$ PVN-APP-RAMs and $2 \times 128 = 256$ updated LLR values of $\{L_{ex}^H(\alpha, \beta)\}$ are

stored to $N_h = 128$ H-EX-RAMs during each clock cycle.

Note that the total time taken to complete updating one layer is 24 clock cycles, which is the same as the theoretical result computed using (13).

b) Case II $t_{loading} > t_{1st\ output}$: It means that the Hadamard sub-decoders start to output the updated results before all the required data have been 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., $L_{app}^{PVN}(\beta)$ and $L_{ex}^H(\alpha, \beta)$) 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.,

$$t_{l2} = dG/2 + dG/2 = (r + 2)G. \quad (16)$$

The latency to decode one codeword equals

$$t_{c2} = Imz_1G(r + 2)/f_c, \quad (17)$$

and the throughput equals

$$T_2 = \frac{[n/m + (2^r - r - 2)] f_c z_2}{IG(r + 2)} \quad (18)$$

which can be improved by (a) increasing f_c and/or z_2 ; or (b) decreasing I and/or G . Fig. 9 shows the timing diagram when decoding one layer with parameters $z_2 = 512$, $N_h = 64$ and $G = z_2/N_h = 8$. The difference between this case and the previous one 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 into the N_h Hadamard sub-decoders. Thus, the N_h Hadamard sub-decoders are idle most of the time. Therefore the throughput can potentially be increased by a factor of $d/2$ 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.

IV. IMPLEMENTATION RESULTS

We implement the $r = 4$ and $R = 0.0494$ PLDPC-Hadamard decoder (whose base matrix and protograph are shown in Fig. 1) optimized in [28], [29] on the Xilinx VCU118 FPGA board. The maximum operating frequency is $f_c = 130$ MHz. Binary phase-shift-keying (BPSK) modulation and an additive white Gaussian noise channel are assumed. To compare with the floating-point results in [30], we use the same lifting factors, i.e., $z_1 = 32$ and $z_2 = 512$, and the same code length $l = 1327104$.

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 III-D. Fig. 10 shows the quantization schemes used and Fig. 11 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. 11 also show that at a BER of 10^{-5} , the fixed-point decoder suffers 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 $I = 20$. While no error floor appears for the floating-point simulations; for fixed-point results, error floors start to emerge at around 3×10^{-6} for $I = 150$ iterations and 10^{-6} for $I = 20$ iterations.

For $r = 4$ (hence $d = r + 2 = 6$), $t_{1st\ output} = (d/2 + 2r + 1) = 12$ cycles. When $G = 4$, $t_{loading} = dG/2 = 12 = t_{1st\ output}$ which belongs to Case I in Section III-D. The decoding latency per layer equals $t_{l1} = 24$ cycles.² Similarly when $G = 8$, $t_{loading} = dG/2 = 24 > t_{1st\ output}$ which belongs to Case II. The decoding latency per layer equals $t_{l2} = 48$ cycles. Table III 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).

V. CONCLUSION

A hardware architecture of the PLDPC-Hadamard layered decoder has been designed and implemented onto an FPGA. The architecture consists of control logics, BRAMs and Hadamard

²In 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 III.

TABLE III
 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.

Available LUT	1, 182, 240			
Available BRAM	2, 160			
No. of sub-decoders	$N_h = 64$		$N_h = 128$	
LUT Utilization	485, 738 (41.09%)		968, 538 (81.92%)	
BRAM Utilization	718.5 (33.26%)		715 (33.10%)	
No. of iterations	$I = 150$	$I = 20$	$I = 150$	$I = 20$
E_b/N_0 at BER of 10^{-5}	-1.11 dB	-0.40 dB	-1.11 dB	-0.40 dB
Latency	12.92 ms	1.72 ms	6.72 ms	0.896 ms
Throughput	0.10 Gbps	0.77 Gbps	0.20 Gbps	1.48 Gbps

sub-decoders. The latency and throughput of the design have been derived in the terms of the code parameters and the amount of parallel sub-decoders deployed. A throughput of 1.48 Gbps is achieved when 20 decoding iterations are used. Error floors are observed when the BER is around 3×10^{-6} .

In our current decoder design, the Hadamard sub-decoders are not fully utilized in the time domain. When these sub-decoders are fully utilized, the decoder can decode $d/2$ ($= 3$ in the example used) codewords simultaneously and hence increase the throughput by the same factor (i.e., to almost 4.5 Gbps). To decode more codewords, more BRAMs would be needed though. Our decoder architecture is generic and can be readily modified to decode LDPC-Hadamard codes with the order of the Hadamard code being odd, i.e., r is odd. Moreover, it can be modified and applied to decode other LDPC-derived codes when the Hadamard constraints LDPC-HC are replaced by other code constraints. Another future direction of research is to investigate the cause of the error floors when fixed-point computation is used and then to design novel ways to overcome it.

REFERENCES

- [1] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, vol. 2, pp. 1064–1070, May 1993.
- [2] R. G. Gallager, *Low-Density Parity-Check Codes*. PhD thesis, MIT Press, Cambridge, USA, 1963.
- [3] D. Divsalar, H. Jin, and R. McEliece, “Coding theorems for turbo-like codes,” in *Proc. Allerton Conf.*, pp. 201–210, 1998.
- [4] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, “Design of capacity-approaching irregular low-density parity-check codes,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.
- [5] M. F. Brejza, L. Li, R. G. Maunder, B. M. Al-Hashimi, C. Berrou, and L. Hanzo, “20 years of turbo coding and energy-aware design guidelines for energy-constrained wireless applications,” *IEEE Commun. Surv. Tut.*, vol. 18, no. 1, pp. 8–28, 2016.
- [6] A. Ardakani and M. Shabany, “A novel area-efficient VLSI architecture for recursion computation in LTE turbo decoders,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 6, pp. 568–572, 2015.
- [7] Y. Liu, P. M. Olmos, and D. G. M. Mitchell, “On generalized LDPC codes for 5G ultra reliable communication,” in *Proc. 2018 IEEE Inf. Theory Workshop (ITW)*, pp. 1–5, 2018.
- [8] Y. Fang, G. Han, G. Cai, F. C. M. Lau, P. Chen, and Y. L. Guan, “Design guidelines of low-density parity-check codes for magnetic recording systems,” *IEEE Commun. Surv. Tut.*, vol. 20, no. 2, pp. 1574–1606, 2018.
- [9] Z. W. Li, L. Chen, L. Q. Zeng, S. Lin, and W. H. Fong, “Efficient encoding of quasi-cyclic low-density parity-check codes,” *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 670–678, Apr. 2004.
- [10] Y. Lee, M. Li, and L. Van der Perre, “Memory-reduced turbo decoding architecture using NII metric compression,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 63, no. 2, pp. 211–215, 2016.
- [11] M. Zhao, X. Zhang, L. Zhao, and C. Lee, “Design of a high-throughput QC-LDPC decoder with TDMP scheduling,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 1, pp. 56–60, 2015.
- [12] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, “A survey of FPGA-based LDPC decoders,” *IEEE Commun. Surv. Tut.*, vol. 18, no. 2, pp. 1098–1122, 2016.
- [13] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rate,” *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, 1974.
- [14] R. Shrestha and R. P. Paily, “High-throughput turbo decoder with parallel architecture for LTE wireless communication standards,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 9, pp. 2699–2710, 2014.
- [15] R. G. Maunder, “A fully-parallel turbo decoding algorithm,” *IEEE Trans. Commun.*, vol. 63, no. 8, pp. 2762–2775, 2015.
- [16] J. Thorpe, “Low-density parity-check (LDPC) codes constructed from protographs,” in *Proc. IPN Progr. Rep.*, pp. 1–7, Aug. 2003.
- [17] Y. Fang, G. A. Bi, Y. L. Guan, and F. C. M. Lau, “A survey on protograph LDPC codes and their applications,” *IEEE Commun. Surv. Tut.*, vol. 17, no. 4, pp. 1989–2016, 2015.
- [18] K. Zhang, X. Huang, and Z. Wang, “High-throughput layered decoder implementation for quasi-cyclic LDPC codes,” *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 985–994, 2009.
- [19] K. Zhang, X. Huang, and Z. Wang, “A high-throughput LDPC decoder architecture with rate compatibility,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 4, pp. 839–847, 2011.
- [20] H.-C. Lee, M.-R. Li, J.-K. Hu, P.-C. Chou, and Y.-L. Ueng, “Optimization techniques for the efficient implementation of high-rate layered QC-LDPC decoders,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 2, pp. 457–470, 2017.
- [21] M. Li, V. Derudder, K. Bertrand, C. Desset, and A. Bourdoux, “High-speed LDPC decoders towards 1 Tb/s,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 5, pp. 2224–2233, 2021.

- [22] Q. Lu, J. Fan, C. Sham, W. M. Tam, and F. C. M. Lau, "A 3.0 Gb/s throughput hardware-efficient decoder for cyclically-coupled QC-LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 1, pp. 134–145, 2016.
- [23] L. Ping, W. K. Leung, and K. Y. Wu, "Low-rate turbo-Hadamard codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 12, pp. 3213–3224, Dec. 2003.
- [24] G. Yue, L. Ping, and X. Wang, "Generalized low-density parity-check codes based on Hadamard constraints," *IEEE Trans. Inf. Theory*, vol. 53, no. 3, pp. 1058–1079, 2007.
- [25] D. J. Costello and G. D. Forney, "Channel coding: The road to channel capacity," *Proc. IEEE*, vol. 95, no. 6, pp. 1150–1177, 2007.
- [26] W. K. R. Leung, G. Yue, L. Ping, and X. Wang, "Concatenated zigzag Hadamard codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1711–1723, 2006.
- [27] L. Ping, L. Liu, Keying Wu, and W. K. Leung, "Interleave division multiple-access," *IEEE Trans. Wireless Commun.*, vol. 5, no. 4, pp. 938–947, 2006.
- [28] 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.
- [29] P.-W. Zhang, F. C. M. Lau, and C.-W. Sham, "Protograph-based LDPC Hadamard codes," *IEEE Trans. Commun.*, vol. 69, no. 8, pp. 4998–5013, 2021.
- [30] P. W. Zhang, F. C. M. Lau, and C.-W. Sham, "Layered decoding for protograph-based low-density parity-check Hadamard codes," *IEEE Commun. Lett.*, vol. 25, no. 6, pp. 1776–1780, 2021.
- [31] 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.
- [32] M. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.
- [33] 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 Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 7, pp. 1857–1869, 2013.
- [34] S. Jiang, P. W. Zhang, F. C. M. Lau, and C.-W. Sham, "An ultimate-Shannon-limit-approaching Gbps throughput encoder/decoder system," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 10, pp. 2169–2173, 2020.

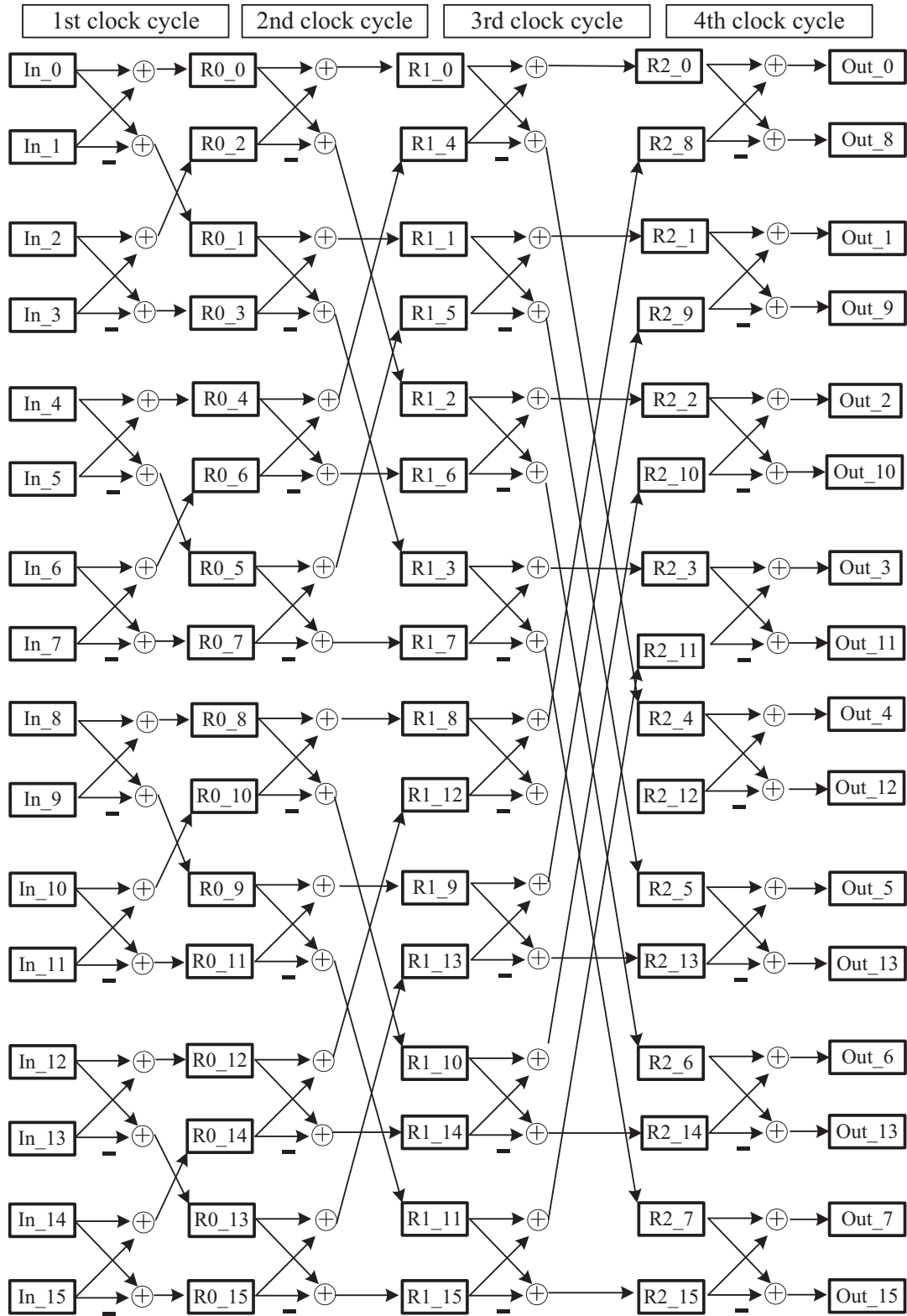


Fig. 5. Pipeline structure of a FHT block for $r = 4$ [24]. Connections to the clock are omitted for clarity.

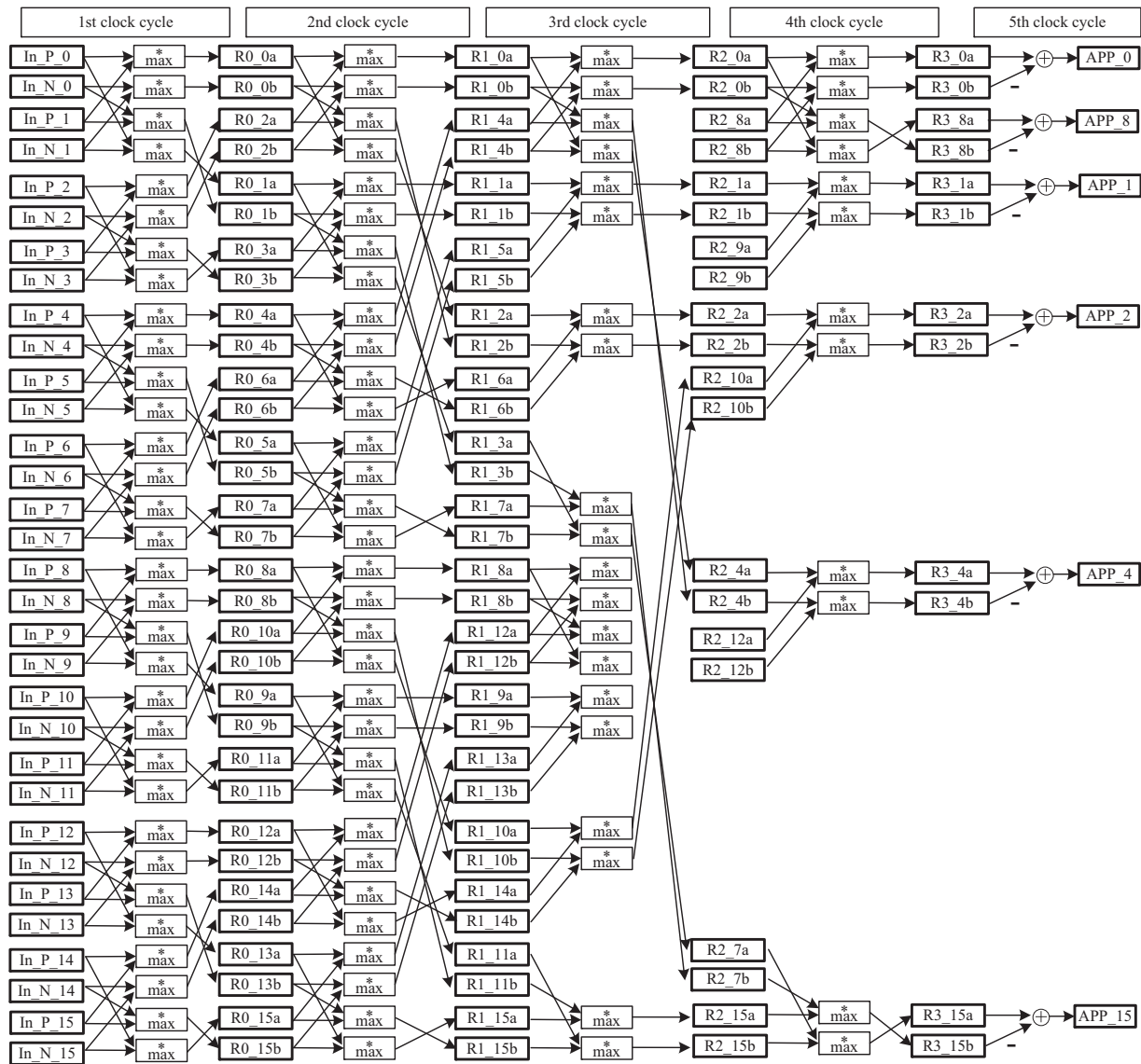


Fig. 6. Pipeline structure of a reduced DFHT block for $r = 4$. An additional clock cycle (5-th clock cycle) is shown for the computation of APP LLRs $L_{app}^H(i)$ [24]. Connections to the clock are omitted for clarity.

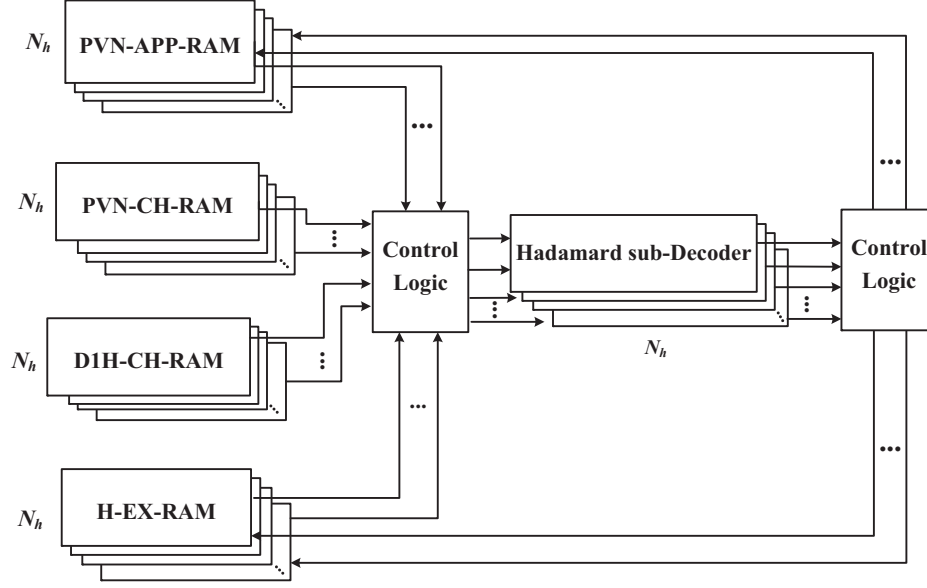


Fig. 7. Proposed layered PLDPC-Hadamard decoder with four types of RAMs, Hadamard sub-decoders and control logics.

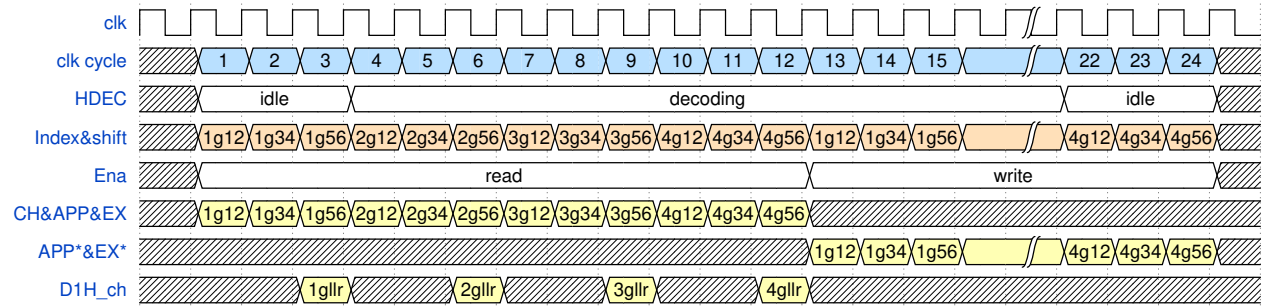


Fig. 8. Timing diagram for the decoding of one layer of PLDPC-Hadamard code. $r = 4$, $z_2 = 512$, $N_h = 128$ and $G = z_2/N_h = 4$. HDEC represents the state of the Hadamard sub-decoder; Index&shift represents the shift values of the corresponding CPMs; Ena represents the state of PVN-CH-RAMs, PVN-APP-RAMs and H-EX-RAMs; CH&APP&EX represent the channel LLR values of $\{L_{ch}^{PVN}(\beta)\}$, the *a posteriori* LLRs of $\{L_{app}^{PVN}(\beta)\}$ and the extrinsic LLRs of $\{L_{ex}^H(\alpha, \beta)\}$; APP*EX* represents the updated LLRs for $\{L_{app}^H(\beta)\}$ and the updated extrinsic LLRs for $\{L_{ex}^H(\alpha, \beta)\}$; DIH_ch represents the channel LLRs of $\{L_{ch}^{D1H}(\alpha)\}$.

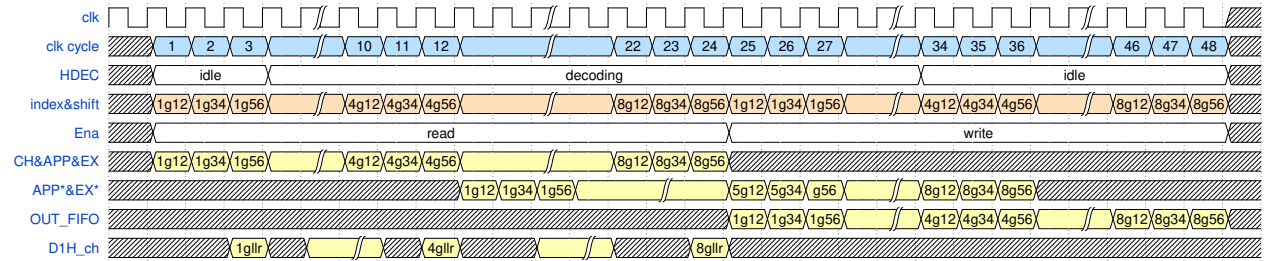


Fig. 9. Timing diagram for the decoding of one layer of PLDPC-Hadamard code. $r = 4$, $z_2 = 512$, $G = 8$ and $N_h = 64$. OUT_FIFO represents the output LLRs for $\{L_{app}^H(\beta)\}$ and $\{L_{ex}^H(\alpha, \beta)\}$. The representations of other symbols are the same as in Fig. 8.

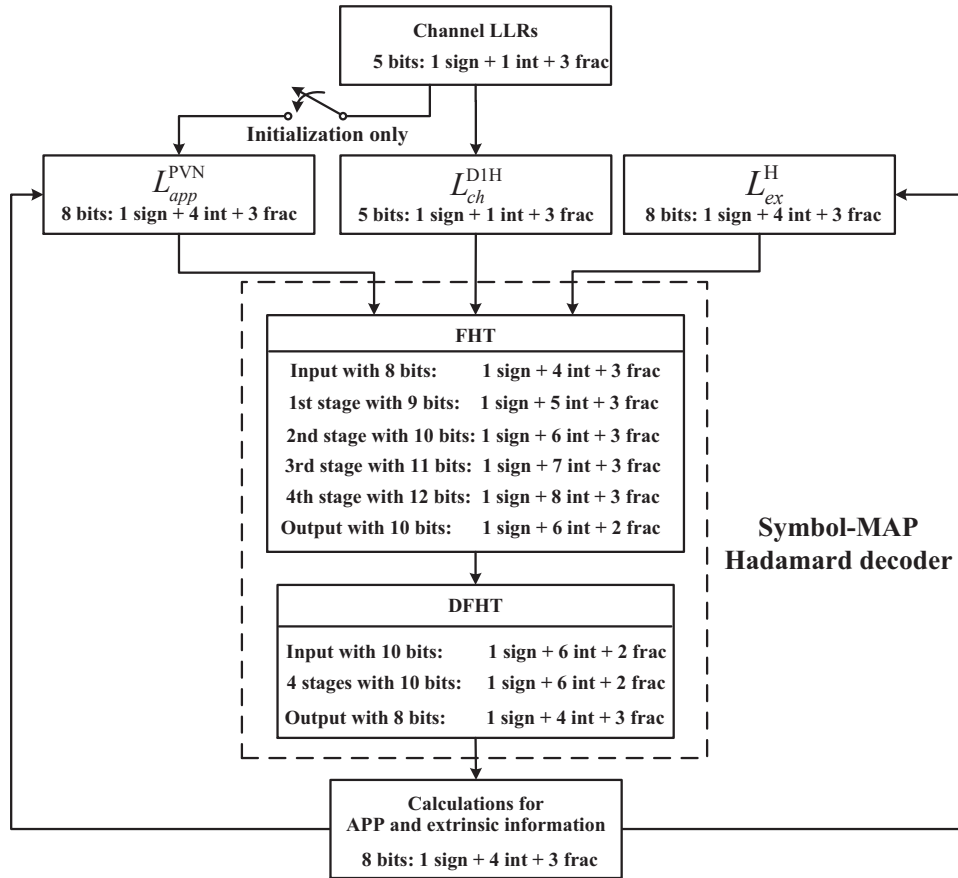


Fig. 10. Data transformation among different modules for a $r = 4$ PLDPC-Hadamard code. “1 sign + y int + z frac” denotes 1 bit to represent sign, y bits to represent the integral part, and z bits to represent the fractional part.

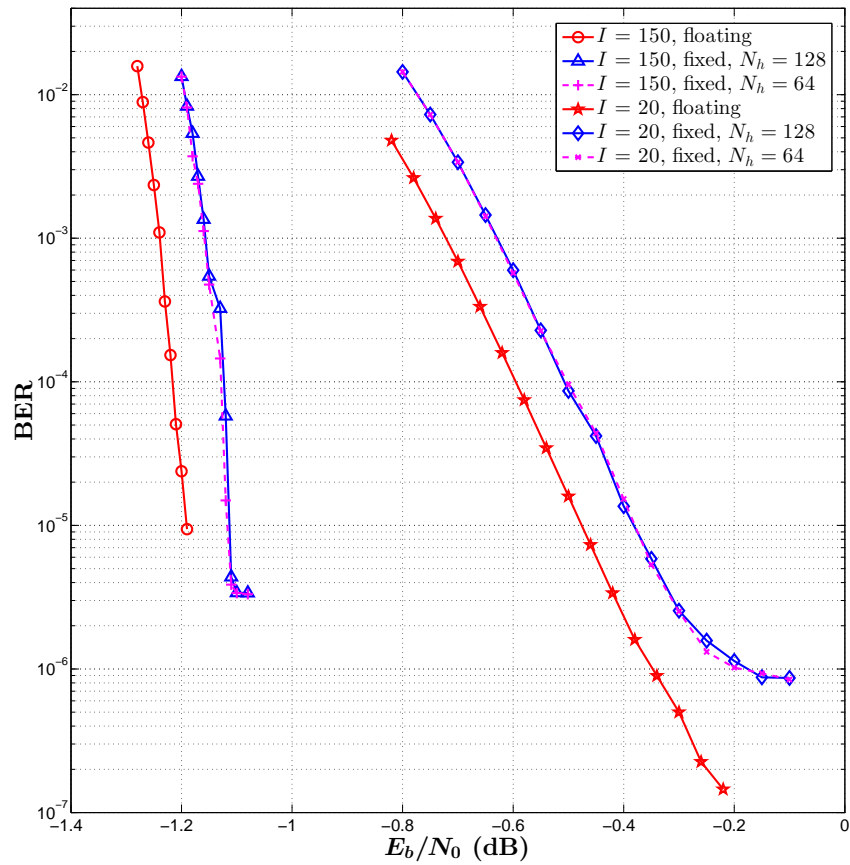


Fig. 11. Floating-point and fixed-point BER performance of the layered PLDPC-Hadamard decoders. $r = 4$ and $l = 1327104$.