Hardware Architecture of Layered Decoders for PLDPC-Hadamard Codes
Peng-Wei Zhang, Sheng Jiang, Francis C. M. Lau, Fellow, IEEE, and Chiu-Wing 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 $E_b/N_0 = -1.14$ 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], [13]. Among different types of 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 [14], [15]. It has also been shown that well-designed QC-LDPC codes can achieve good decoding performance, low error floor and high throughput.

To decode QC-LDPC codes, layered decoding architectures are mostly used because they have relatively low hardware requirements and high throughputs. 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 [16]. In [17], it has been shown that memory access conflicts introduced by the pipeline process in layered decoding can be reduced by lowering the maximum available parallelism and efficient scheduling. In [18], a novel layered decoder architecture that supports QC-LDPC codes with any circulant weight is proposed. To resolve the access conflict issue, a block-serial scheduling algorithm, whose processing time is independent of the circulant weight, is further developed. Using the China Mobile Multimedia Broadcasting standard as an example, a decoder synthesized using 65-nm CMOS technology has shown to achieve a throughput of 1.1 Gb/s with 15 iterations. In [19], a block-level-parallel layered decoder for irregular QC-LDPC codes is proposed and a dynamic multi-frame processing schedule is developed to minimize pipeline stages and memory overheads. The decoder can also be reconfigured to support multiple block lengths and code rates of the WiFi standard. In [20], a RAM-based decoder architecture is proposed to decode cyclically-coupled QC-LDPC codes and obtains a throughput of 3.0 Gbps and an error floor of about $10^{-16}$. In [21], it is shown that a layered decoder throughput can be increased by reordering the layered decoding procedure and applying some optimization techniques. In [22], off-line mapping and scheduling algorithms have been proposed together with a novel residue-based layered QC-LDPC decoding to increase the resource usage of the layered decoder. Evaluation performed for six QC-LDPC codes shows up to 57% improvement in hardware utilization efficiency for a one-layer overlap. In [23], an efficient decoder architecture is proposed for highly irregular QC-LDPC codes. It normally works as the layered schedule. When a pipeline conflict is foreseen, the decoder changes to the flooding schedule. An offline parity-check matrix reordering method based on genetic algorithm is then further to optimized this hybrid schedule. Throughput increases between 30.8% and 109.1% are demonstrated for 5G NR codes. In [24], it is shown that 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. In [25], a logarithmic-likelihood-ratio compound (LLRC) segregation technique is proposed. Based on the technique and other architectural optimizations, a hardware-efficient QC-LDPC layered decoder architecture with reduced data-congestion and high throughput is presented. Comparison with other works shows that the proposed decoder achieves more than two times throughput improvement and eight times better hardware-efficiency.

Moreover, when both turbo and LDPC codes are used together with Hadamard codes, forming turbo-Hadamard codes [26] and LDPC-Hadamard codes (LDPC-HCs) [27], 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) [28]. Another ultimate-Shannon-limit-approaching code is the concatenated zigzag-Hadamard code [29]. 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 [27]. These ultimate-Shannon-limit-approaching codes can be applied to extreme communication environments such as deep-space communications and interleave division multiple access systems with many users [30].

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 [31], [32]. 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 quasi-cyclic structure which is beneficial to hardware implementation. To improve the convergence rate, a PLDPC-HC layered decoding algorithm has been proposed [33]. 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 [14]. 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 [31], [32]. Fig. 1 illustrates the base matrix $B_{m \times n}$ 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 $B_{m \times n}$, 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 $B_{m \times n}$. To obtain the adjacency matrix $H_{M \times N}$ of the PLDPC-HC, the base matrix $B_{m \times n}$ is lifted twice with factors $z_1$ and $z_2$ where $M = m z_1 z_2$ and $N = n z_1 z_2$ [34]. The first lifting replaces each non-zero $B(i, j)$ in $B_{m \times n}$ 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 and decoding [35]. 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 $H_{M \times N}$ 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 [31], [32]. 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

$$R = \frac{n - m}{m (2^r - r - 2) + n}.\quad (1)$$

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 [31], [32].

To speed up the convergence speed, a layered decoding algorithm has been proposed [33]. For $\alpha = 0, 1, \ldots, M - 1$ and $\beta = 0, 1, \ldots, N - 1$, we denote

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

After lifting the base matrix of a PLDPC-HC two times, the resultant adjacency matrix $H_{M \times N}$ is divided into $m z_1$ layers
where each layer is composed of $1 \times n_z 1$ CPMs each of size $z_2 \times z_2$. Hence, each layer corresponds to a $z_2 \times n_z 1$ matrix and contains $z_2$ H-CNs. Since each H-CN connects $d$ P-VNs and $2' - d$ D1H-VNs (when $r$ is even), the $z_2$ H-CNs in one layer connects $d z_2$ P-VNs and $(2' - d) z_2$ D1H-VNs. Table I summarizes the numbers of H-CNs, P-VNs and D1H-VNs contained in one layer.

Defining $k$ as the layer number ($k = 0, 1, \ldots, m z_2 - 1$) and $L(k) = \{a k z_2, a k z_2 + 1, \ldots, a k z_2 + z_2 - 1\}$ as the set of H-CNs in layer $k$, the layered decoding algorithm is described as follows [33].

1) Initialization: Set $L_{\text{app}}^P(\beta) = L_{\text{ch}}^P(\beta) \forall \beta$; and set $L_{\text{ex}}^H(\alpha, \beta) = 0 \forall \alpha, \beta$.

2) Symbol maximum-a-posterior Hadamard sub-decoder: Set $k = 0$.

   a) For the $a$-th H-CN in layer $k$ ($a \in L(k)$), perform the following computations.

      i) For $\beta \in \mathcal{P}(a)$, compute

         $L_{\text{ex}}^P(\alpha, \beta) = L_{\text{app}}^P(\beta) - L_{\text{ex}}^H(\alpha, \beta) \forall \beta \in \mathcal{P}(a). \ (2)$

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

         $L_{\text{app}}^H(\alpha) = \{L_{\text{app}}^H(\alpha, \beta) : \beta \in \mathcal{P}(a)\}$

         $= \mathcal{T} \left[ \{L_{\text{ex}}^P(\alpha, \beta) : \beta \in \mathcal{P}(a)\}, L_{\text{ch}}^{D1H}(\alpha) \right] \ (3)$

         where $\mathcal{T}$ is a transformation involving the fast Hadamard transform (FHT) and the dual FHT (DFHT) operations [27], [31], [32].

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

         $L_{\text{ex}}^H(\alpha, \beta) = L_{\text{app}}^H(\alpha, \beta) - L_{\text{ex}}^P(\alpha, \beta); \forall \beta \in \mathcal{P}(a) \ (4)$

         $L_{\text{app}}^P(\beta) = L_{\text{app}}^H(\alpha, \beta); \forall \beta \in \mathcal{P}(a). \ (5)$

b) If the last layer has not been reached, i.e., $k < m z_2 - 1$, 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 $L_{\text{app}}^P(\beta) \forall \beta$.

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)[5]. 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 $\{L_{\text{ex}}^P(\alpha, \beta)\}$ in (2) and $\{L_{\text{app}}^H(\alpha, \beta)\}$ in (3) are only temporary values in the computation process and need not to be stored, whereas the other four types of LLRs, i.e., $\{L_{\text{ch}}^P(\beta)\}$, $\{L_{\text{ch}}^P(\beta)\}$, $\{L_{\text{ch}}^H(\alpha, \beta)\}$ and $\{L_{\text{ch}}^{D1H}(\alpha)\}$, are not temporary and thus need to be stored in RAMs.

In a practical environment, the LLRs first are generated one-by-one by the demodulator at the receiver. They can then be passed to the next stage, i.e., decoder, one-by-one or in parallel (in a small number) and stored in the RAMs of the decoder. Th...
need to retrieve \(dz_2\) values of \(L_{\text{PVN}}^H(\beta)\) (during initialization) or \(dz_2\) values of \(L_{\text{PVN}}^H(\alpha, \beta)\) to process each layer, we therefore need to retrieve \(dz_2\) values of \(L_{\text{PVN}}^H(\beta)\) for each layer, respectively. The only differences are that the RAMs will have different depths and widths. Fig. 3 shows the storage arrangement of \(L_{\text{PVN}}^H(\alpha, \beta)\) and \(L_{\text{PVN}}^H(\alpha, \beta)\) (corresponding to the first layer) in \(Nh\) RAMs. In Fig. 3(a), \(\alpha = i : \beta_0\) (i.e., all the P-VNs connected to the \(i\)-th H-CN) are retrieved; at clock \(t = 1\), the \(Nh\) LLR values stored at Address \#1 (with indices \(1, G_1 + 1, \ldots, (Nh - 1)G_1 + 1\)) are retrieved; at clock \(t = G\), the \(Nh\) LLR values stored at Address \(G + 1\) (with indices \(G + 1, 2G + 1, \ldots, NhG + 1\)) 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 \(L_{\text{PVN}}^H(\alpha, \beta)\) or \(L_{\text{PVN}}^H(\alpha, \beta)\) for each layer requires \(dG\) clock cycles.

We use a similar storage arrangement for \(L_{\text{PVN}}^H(\alpha, \beta)\) and \(L_{\text{PVN}}^H(\alpha, \beta)\), 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 \(L_{\text{PVN}}^H(\alpha, \beta)\) and \(L_{\text{PVN}}^H(\alpha, \beta)\) (corresponding to the first layer) in \(Nh\) RAMs. In Fig. 3(b), each address stores the \(2^r - d\) channel LLRs corresponding to the \(2^r - d\) channel H-CN.

Remark: The aforementioned arrangement of the LLRs in the RAMs is valid regardless of single-port RAMs or dual-port RAMs being used. In other words, no read/write conflicts will occur whether single-port RAMs or dual-port RAMs are used. 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. The theoretical latency and...
throughput derived in Sect. III-D and the experimental results shown in Sect. IV are all based on the use dual-port RAMs.

Supposing we have retrieved $N_h$ values for $\{F_{ch}^{PVN}(\beta)\}$ or $\{F_{app}^{PVN}(\beta)\}$, we need to interleave them — a process similar to that used in QC-LDPC decoding [36]. For each layer, the exact connections between the H-CNs and the P-VNs are determined by the CPMs, and hence the 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 $q_a = \lfloor p/G \rfloor$ and the remainder $r_e = p \mod G$, where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to $x$ and “mod” denotes the modulus operation. When (address mod $G$) $< r_e$, the corresponding $N_h$ LLRs are cyclically shifted to the left by $(q_a + 1 \mod N_h)$; otherwise, these LLRs are cyclically shifted to the left by $q_a$.

**Example:** We assume that $z_2 = 16$. Fig. 4(a) shows a $16 \times 16$ identity matrix, i.e., a $16 \times 16$ CPM with $p = 0$; Fig. 4(b) depicts a $16 \times 16$ CPM with $p = 9$, which can be obtained by cyclically shifting the $16 \times 16$ identity matrix to the right by $p = 9$ columns.

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

After retrieving these LLRs from the RAMs and interleaving them, these indices are expected to be reordered into $[9 \, 10 \, 11 \, 12 \, 13 \, 14 \, 0 \, 1 \, 2 \, 3 \, 4 \, 5 \, 6 \, 7 \, 8]$. Suppose we use $N_h = 4$ RAMs to store this set of $L_{ch}^{PVN}(\beta)$. According to our aforementioned storage scheme, each set of $L_{ch}^{PVN}(\beta)$ 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 $q_a = \lfloor p/G \rfloor = 2$ and $r_e = p \mod G = 1$. 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 $q_a = 2$ 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 $q_a = 2$ 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 $q_a = 2$ 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 $q_a = 2$ and the order of the indices becomes $[11 \, 15 \, 3 \, 7]$.

Therefore, the expected interleaving effect $[9 \, 10 \, 11 \, 12 \, 13 \, 14 \, 0 \, 1 \, 2 \, 3 \, 4 \, 5 \, 6 \, 7 \, 8]$ 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.

**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

$$H_q = [\pm h_j, j = 0, 1, \ldots, q - 1]$$

$$= \left[ \pm H_{q/2} \pm H_{q/2} \right]$$

$$= \left[ \pm H_{q/2} \pm H_{q/2} \right]$$

where $q = 2^r$ equals the code length and $\pm H_1 = [\pm 1]$. Each column $\pm h_j$ of the Hadamard matrices corresponds to a Hadamard codeword, and hence there is a total of $2q = 2^{r+1}$ codewords in $\pm H_q$. In (7), as shown at the bottom of the next page, we show the $16 \times 16$ Hadamard matrices $\pm H_{16}$ corresponding to the order-$r = 4$ Hadamard code having $2^{4+1} = 32$ 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 [27], [31], [32], i.e.,

$$[\pm h_{0,j} \oplus \pm h_{1,j} \oplus \cdots \oplus \pm h_{2^{r-1},j} \oplus \cdots \oplus \pm h_{2^{r-1},j} \oplus h_{2^{r-1},j}] \oplus h_{2^{r-1},j} = 0,$$

where the symbol $\oplus$ represents the XOR operator. In (7), the length-6 SPC constraint is $\pm h_{0,j} \oplus \pm h_{1,j} \oplus \pm h_{2,j} \pm h_{3,j} \pm h_{4,j} \pm h_{5,j} = 0 \forall j$ 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 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 $2^4 - 6 = 10$ Hadamard parity-check bits.)

| Address | 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    |

![Table II](image)
To decode Hadamard codes, a symbol-MAP decoding algorithm has been proposed [27], [31], [32]. We define

\[ \mathbf{L}^H_{ch} = [L^H_{ch}(0), L^H_{ch}(1), \ldots, L^H_{ch}(2^r - 1)]^T, \]

(9)

\[ \mathbf{L}^H_{apr} = [L^H_{apr}(0), L^H_{apr}(1), \ldots, L^H_{apr}(2^r - 1)]^T, \]

(10)

\[ \mathbf{L}^H_{app} = [L^H_{app}(0), L^H_{app}(1), \ldots, L^H_{app}(2^r - 1)]^T, \]

(11)

as the channel, the \textit{a priori} and the \textit{a posteriori} LLR information of the coded bit, respectively. Note that \( \mathbf{L}^H_{ch} \) 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, \( \mathbf{L}^H_{apr} \) 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 [31, Section III-B] and [32, Section III-B] for the detailed arrangement of \( \mathbf{L}^H_{ch} \) and \( \mathbf{L}^H_{apr} \).) Based on \( \mathbf{L}^H_{ch} \) and \( \mathbf{L}^H_{apr} \), \( L^H_{app}(i) \) is computed using

\[ L^H_{app}(i) = \ln \left( \frac{\sum \gamma(\pm h_j) \mathbb{1}_{H(i,j)=1}}{\sum \gamma(\pm h_j) \mathbb{1}_{H(i,j)=-1}} \right), \]

(12)

where \( \gamma(\pm h_j) = \exp \left( \frac{\pm h_j, L^H_{ch} + L^H_{apr}}{2} \right) \) represents the \textit{a posteriori} “information” of the codeword \( \pm h_j \); and \( \langle i \rangle \) 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 [26], [27], [37].

1) We first use a FHT block to compute \( \pm h_j, L^H_{ch} + L^H_{apr} \). Using the structure of the FHT block for \( r = 4 \) shown in Fig. 5 as an example, the inputs are \( \text{In}_{-j} = L^H_{ch}(j) + L^H_{apr}(j) \) and the outputs are \( \text{Out}_{-j} = 2 \ln \left[ \gamma(\pm h_j) \right] (j = 0, 1, \ldots, 15) \). Then, \( \ln \left[ \gamma(\pm h_j) \right] \) is readily obtained from \( 2 \ln \left[ \gamma(\pm h_j) \right] \) by shifting the least significant bit out. Moreover, \( \ln \left[ \gamma(-h_j) \right] \) is readily available because \( \ln \left[ \gamma(-h_j) \right] = -\ln \left[ \gamma(h_j) \right] \). There are \( r = 4 \) stages in the FHT block and thus a latency of \( r = 4 \) clock cycles is required.

\[ \pm H_{16} = \]

\[
\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 & \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 & \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 & \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 & \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 & \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 & \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 & \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 & \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 & \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 & \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 & \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 & \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 & \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 \\
\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 & \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 & \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 & \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 & \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 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1 & \pm 1
\end{bmatrix}
\]

(7)

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 \( \ln \left[ \gamma(\pm h_j) \right] \) and \( \ln \left[ \gamma(-h_j) \right] \); and the outputs are \( \ln \left[ \sum \pm H(i,j)=1 \gamma(\pm h_j) \right] \) and \( \ln \left[ \sum \pm H(i,j)=-1 \gamma(\pm h_j) \right] \) (\( j = 0, 1, \ldots, 15 \)).
module $\max$ in the DFHT block represents the Jacobian logarithm, i.e.,

$$\max(a, b) = \ln[\exp(a) + \exp(b)]$$

$$= \max(a, b) + \ln \left[ 1 + \exp(-|a - b|) \right]$$

(13)

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 $\ln \left[ 1 + \exp(-|a - b|) \right]$ 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 $i = 0, 1, \ldots, 2^k-1, 2^{r-1}, \ldots, 2^r-1$, it takes another clock cycle to compute

- the $r + 2$ extrinsic LLR messages $L^H_{ex}(i)$ which is computed using (4).

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.\(^3\)

As can be seen in Fig. 7, one control logic is responsible for reading data from RAMs and loading them to the inputs of the Hadamard sub-decoders while the other control logic is responsible for writing the outputs of the Hadamard sub-decoders to the RAMs. These two control logics can be combined into one, but are shown as two for clarity.
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 $\{L_{app}^{PVN}(\beta)\}$, $N_h$ values of $\{L_{app}^{PVN}(\alpha, \beta)\}$ and $N_h$ vectors of $\{L_{app}^{PVN}(\alpha, \beta)\}$ 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 $\{L_{app}^{PVN}(\beta)\}$, $dN_h$ values of $\{L_{app}^{PVN}(\alpha, \beta)\}$ and $dN_h$ vectors of $\{L_{app}^{PVN}(\alpha, \beta)\}$ 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 $\{L_{app}^{PVN}(\beta) : \beta = 0, 1, \ldots, N - 1\}$. Each RAM has a width of $w_{app}$ bits (to represent the quantized LLR value) and a depth of $n_2 G$. Referring to Fig. 2, the $g$-th location ($g = 0, 1, \ldots, n_2 G - 1$) in the $t$-th RAM ($t = 0, 1, \ldots, N_h - 1$) stores $L_{app}^{PVN}(\beta)$ where $\beta = [g / G]z_2 + lG + (g \mod G)$. Note that $\{L_{app}^{PVN}(\beta)\}$ 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 $\{L_{app}^{PVN}(\beta) : \beta = 0, 1, \ldots, N - 1\}$. Each RAM has a width of $w_{app}$ bits and a depth of $n_2 G$. Data are stored in the same way as in PVN-CH-RAM, i.e., the $g$-th location ($g = 0, 1, \ldots, n_2 G - 1$) in the $t$-th RAM ($t = 0, 1, \ldots, N_h - 1$) stores $L_{app}^{PVN}(\beta)$ where $\beta = [g / G]z_2 + lG + (g \mod G)$.

- $N_h$ RAMs, denoted by H-EX-RAM, are used to store $\{L_{app}^{PVN}(\alpha, \beta) : \alpha = 0, 1, \ldots, M - 1; \beta \in \{0, 1, \ldots, M - 1\} \in P(\alpha)\}$. Each RAM has a width of $w_{app}$ bits and a depth of $n_2 G$. Referring to Fig. 3(a), the $g$-th location ($g = 0, 1, \ldots, n_2 G - 1$) in the $t$-th RAM ($t = 0, 1, \ldots, N_h - 1$) stores $L_{app}^{PVN}(\alpha, \beta)$ where $\alpha = [q / d]N_h + l$, $\beta = \beta_0$, and $\delta = q \mod d$.

- $N_h$ RAMs, denoted by D1H-CH-RAM, are used to store $\{L_{app}^{D1H}(\alpha) : \alpha = 0, 1, \ldots, M - 1\}$. Each RAM has a width of $w_{app} = w_{app} \times (2^r - 1)$ of $mz_1 G$. Each address stores all the $2^r - 1$-channel LLR values of D1H-VNs connected to a H-CN. Referring to Fig. 3(b), the $w$-th location ($w = 0, 1, \ldots, mz_1 G - 1$) in the $t$-th RAM ($t = 0, 1, \ldots, N_h - 1$) stores $L_{D1H}(\alpha)$ 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 $2mz_1 G$. We double the depth of D1H-CH-RAM to $2mz_1 G$ 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 $^4$ and it takes $d / 2$ clock cycles to retrieve the required $d$ sets of $L_{app}^{PVN}(\beta)$ or $L_{app}^{PVN}(\beta)$ and $L_{app}^{PVN}(\beta)$ values in each layer. Note that $\{L_{app}^{PVN}(\alpha, \beta)\}$ in (2) is computed in the same clock cycle as $\{L_{app}^{PVN}(\beta)\}$ or $L_{app}^{PVN}(\beta)$ and $L_{app}^{PVN}(\beta)$ are retrieved. At the $d / 2$-th clock cycle, we also load the required $z_2$ sets of $L_{app}^{PVN}(\beta)$ from one address location to the sub-decoders. Subsequently, $d z_2$ LLRs of $\{L_{app}^{PVN}(\alpha, \beta)\}$ and $z_2$ vectors of $\{L_{app}^{PVN}(\alpha, \beta)\}$ are passed to the $z_2$ FHT blocks in the $z_2$ Hadamard sub-decoders, i.e., $d$ LLRs of $\{L_{app}^{PVN}(\alpha, \beta)\}$ and one vector of $\{L_{app}^{PVN}(\alpha, \beta)\}$ to one FHT block in one Hadamard sub-decoder. Then, it takes $2r + 1$ clock cycles to compute $d z_2$ LLRs of $\{L_{app}^{PVN}(\beta)\}$ and $d z_2$ LLRs of $\{L_{app}^{PVN}(\beta)\}$ using (4) and (5), respectively. Finally, it takes another $d / 2$ clock cycles to write these updated $L_{app}^{PVN}(\beta)$ and $L_{app}^{PVN}(\alpha, \beta)$ values into the RAMs.

To summarize,

i) Clock cycle no. 1 to $d / 2$: read $\{L_{app}^{PVN}(\beta)\}$ or $\{L_{app}^{PVN}(\beta)\}$ and $\{L_{app}^{PVN}(\beta)\}$ from memory, and at the same time compute $\{L_{app}^{PVN}(\beta)\}$ using (2);

ii) Clock cycle no. $d / 2$ (in parallel with above): read $\{L_{app}^{PVN}(\alpha, \beta)\}$;

iii) Clock cycle no. $d / 2 + 1$ to $d / 2 + 2r$: process the inputs $\{L_{app}^{PVN}(\alpha, \beta)\}$ and $\{L_{app}^{PVN}(\alpha, \beta)\}$ by the Hadamard sub-decoders (consisting of FHT and DFHT blocks) using (3);

iv) Clock cycle no. $d / 2 + 2r + 1$: compute $\{L_{app}^{PVN}(\alpha, \beta)\}$ and $\{L_{app}^{PVN}(\alpha, \beta)\}$ using (4) and (5);

$^4$Note that single-port RAMs can also be used but then only one memory address can be accessed at one time. The derivations of latency and throughput would be similar to those described in this section but the results would be worse.
v) Clock cycle no. \( d/2 + 2r + 2 \) to \( d/2 + 2r + 1 + d/2 \): write \( \{L_{PVN}^H(\beta)\} \) and \( \{L_{ch}^H(\alpha, \beta)\} \) 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-CNIs (each consisting of \( N_h \) H-CNIs) are sequentially processed in each layer. Referring to the timing details in Section III-B and Section III-D.1 and with the use of our RAM designs, it takes \( d/2 \) clock cycles to load the data of one group of H-CNIs. (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., \( L_{PVN}^H(\beta) \) and \( L_{ch}^H(\alpha, \beta) \)) is available at the \( t_{1st\ output} = (d/2 + 2r + 1) \)-th clock cycle.

a) Case 1. \( t_{1loading} \leq t_{1st\ output} \): 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.,

\[
t_{11} = t_{1loading} + (2r + 1) + d/2 = (r/2 + 1)G + 5r/2 + 2
\]

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

\[
t_{c1} = 1mz_1t_{11}/f_c = 1mz_1[(r/2 + 1)G + 5r/2 + 2]/f_c,
\]

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 \( l = mz_1z_2 + mz_1z_2(2^r - r - 2) \), the throughput of the decoder is expressed as

\[
T_1 = \frac{1}{t_{c1}} = \frac{[mz_1z_2 + mz_1z_2(2^r - r - 2)]f_c}{Imz_1t_{11}} = \frac{n/m + (2^r - r - 2)z_2f_c}{I((r/2 + 1)G + (5r/2 + 2))}
\]

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 \( \{L_{PVN}^H(\alpha, \beta)\} \) into the Hadamard sub-decoders corresponding to the \( z_2 \) H-CNIs in the layer in \( dG/2 = 12 \) clock cycles. In each clock cycle, \( 2N_h = 256 \) LLRs of \( \{L_{PVN}^H(\beta)\} \) (or \( L_{PVN}^H(\beta) \)) and 256 LLRs of \( \{L_{ch}^H(\alpha, \beta)\} \) are read from RAMs, and at the same time 256 LLRs of \( \{L_{PVN}^H(\alpha, \beta)\} \) are computed using \( (2) \) 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 \( \{L_{PVN}^H(\beta)\} \) (or \( L_{PVN}^H(\beta) \)) and 768 LLRs of \( \{L_{ch}^H(\alpha, \beta)\} \), and to compute and load 768 LLRs of \( \{L_{PVN}^H(\alpha, \beta)\} \) 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, “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 \( \{L_{ch}^{D1H(a)}\} \) into the Hadamard sub-decoders. At clock
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 $\{L_{\text{app}}^H(\beta)\}$ and $\{L_{\text{ex}}^H(\alpha, \beta)\}$. The representations of other symbols are the same as in Fig. 8.

no. 3, we load the channel LLRs for D1H-VNs in Group #1 into the $N_h = 128$ Hadamard sub-decoders. Referring to “D1H_ch” in 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 $Z = 2, 3, 4, 5$.

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 $t_{11} = 24$: 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 $\{L_{\text{app}}^H(\beta)\}$ and 768 LLR values of $\{L_{\text{ex}}^H(\alpha, \beta)\}$. 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 $\{L_{\text{app}}^H(\beta)\}$ are stored into $N_h = 128$ PVN-APP-RAMS and $2 \times 128 = 256$ updated LLR values of $\{L_{\text{ex}}^H(\alpha, \beta)\}$ 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 (14).

b) Case II $t_{\text{loading}} > t_{\text{first output}}$: 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., $L_{\text{app}}^H(\beta)$ and $L_{\text{ex}}^H(\alpha, \beta)$) 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.,

$$t_{l2} = dG/2 + dG/2 = (r + 2)G.$$  \hspace{1cm} (17)

The latency to decode one codeword equals

$$t_{c2} = Imz_2G(r + 2)/f_c,$$ \hspace{1cm} (18)

and the throughput equals

$$T_2 = \frac{n/m + (2^r - r - 2)}{G(r + 2)}f_cz_2$$ \hspace{1cm} (19)

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 layers.

Note that here for convenience, we use only 1 clock cycle to load one group of $\{L_{\text{ch}}^H(\alpha)\}$ into the Hadamard sub-decoders. Thus the four groups of $\{L_{\text{ch}}^H(\alpha)\}$ are loaded during clock cycle nos. 3, 6, 9 and 12, leaving some “blank regions” between these clock cycles in Fig. 8 (and also in Fig. 9).

Another design is to load each group of $\{L_{\text{ch}}^H(\alpha)\}$ using multiple clock cycles (a maximum of 3 clock cycles in this example) so as to minimize the span of the “blank regions”.

5Note that it is possible to start processing the next layer before the current one is entirely completed. Memory access conflicts as in conventional QC-LDPC decoding architectures will occur but can also be resolved by methods introduced in Sect. I. As this paper mainly focuses on realizing the PLDPC-HC layered decoder and estimates its fixed-point error performance, improving the throughput of the decoder would be left for our future work.
codewords at the same time. The extra requirement would be $d/2$ times increase in memory storage and a bit more control logics [19].

**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 [31] and [32] on the Xilinx VCU118 FPGA board. The maximum operating frequency is $f_c = 130$ MHz and true dual-port RAMs are used. Binary phase-shift-keying (BPSK) modulation and an additive white Gaussian noise channel are assumed. To compare with the floating-point results in [33], 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. First, we consider the bit-widths setting $S1$ shown in Fig. 10(a) that has been implemented for both designs. Fig. 11 plots the FER/BER results of the PLDPC-Hadamard code when the number of iterations $I$ of 20 and 150. It can be observed that the two designs (i.e., $N_h = 128$ and $N_h = 64$) with bit-widths setting $S1$ produce almost the same FER/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 also show that at a BER of $10^{-5}$, the fixed-point decoder with bit-widths setting $S1$ suffers from a degradation of 0.08 dB compared with the floating-point computation when $I = 150$; and a degradation of 0.10 dB when $I = 20$. While no FER/BER error floors appear for the floating-point simulations; for fixed-point results, error floors start to emerge (i) at a BER of $3 \times 10^{-6}$ (FER around $1.05 \times 10^{-2}$) for $I = 150$ iterations and (ii) at a BER of $10^{-7}$ (FER around $1.05 \times 10^{-2}$) for $I = 20$ iterations.

To investigate the effect of bit-widths setting on the error performance of the fixed-point decoder, we increase the integer part for all the types of LLRs (except for channel observations) in setting $S1$ by one bit and form the bit-widths setting $S2$ shown in Fig. 10(b). We implement the setting $S2$ for the design with $N_h = 128$ Hadamard sub-decoders and plot the FER/BER results in Fig. 11 with $I = 20$ iterations. We can observe that increasing the bit-widths can effectively remove the FER/BER error floors when $I = 20$ iterations. Based on the setting $S2$, we increase the fractional part of (i) output of FHT, (ii) input of DFHT, and (iii) 4 (internal) stages in DFHT, are increased by one bit, i.e., change from “1 sign + 7 int + 2 frac” to “1 sign + 7 int + 3 frac”, bit-widths setting $S3$ is formed.

### Footnotes

1. We start the experiments by setting the clock frequency of the FPGA to 80 MHz. After confirming that the experimental results (i.e., error rates) are the same as those given by fixed-point computer simulations, we continually increase the clock frequency of the FPGA. When the clock frequency exceeds 130 MHz, the experimental results are no longer the same as those given by fixed-point computer simulations. Thus we claim a maximum operating frequency of $f_c = 130$ MHz.

2. Frequency is the Xilinx VCU118 FPGA board. The maximum operating graph are shown in Fig. 1) optimized in [31] and [32] on the Xilinx VCU118 FPGA board. The maximum operating frequency is $f_c = 130$ MHz and true dual-port RAMs are used. Binary phase-shift-keying (BPSK) modulation and an additive white Gaussian noise channel are assumed. To compare with the floating-point results in [33], we use the same lifting factors, i.e., $z_1 = 32$ and $z_2 = 512$, and the same code length $l = 1327104$.

3. 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. First, we consider the bit-widths setting $S1$ shown in Fig. 10(a) that has been implemented for both designs. Fig. 11 plots the FER/BER results of the PLDPC-Hadamard code when the number of iterations $I = 20$ and 150. It can be observed that the two designs (i.e., $N_h = 128$ and $N_h = 64$) with bit-widths setting $S1$ produce almost the same FER/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 also show that at a BER of $10^{-5}$, the fixed-point decoder with bit-widths setting $S1$ suffers from a degradation of 0.08 dB compared with the floating-point computation when $I = 150$; and a degradation of 0.10 dB when $I = 20$. While no FER/BER error floors appear for the floating-point simulations; for fixed-point results, error floors start to emerge (i) at a BER of $3 \times 10^{-6}$ (FER around $1.05 \times 10^{-2}$) for $I = 150$ iterations and (ii) at a BER of $10^{-7}$ (FER around $1.05 \times 10^{-2}$) for $I = 20$ iterations.

To investigate the effect of bit-widths setting on the error performance of the fixed-point decoder, we increase the integer part for all the types of LLRs (except for channel observations) in setting $S1$ by one bit and form the bit-widths setting $S2$ shown in Fig. 10(b). We implement the setting $S2$ for the design with $N_h = 128$ Hadamard sub-decoders and plot the FER/BER results in Fig. 11 with $I = 20$ iterations. We can observe that increasing the bit-widths can effectively remove the FER/BER error floors when $I = 20$ iterations. Based on the setting $S2$, we increase the fractional part of (i) output of FHT, (ii) input of DFHT, and (iii) 4 (internal) stages in DFHT, by one bit and form the bit-widths setting $S3$. In other words, all the above three categories are represented by 11 bits, i.e., change from “1 sign + 7 int + 2 frac” to “1 sign + 7 int + 3 frac”. We implement the setting $S3$ for the design with $N_h = 128$ Hadamard sub-decoders and plot the FER/BER results in Fig. 11 with $I = 150$ iterations. Comparing the results for setting $S3$ and those for setting $S1$ shows that no BER error floor is observed down to $2 \times 10^{-8}$ (for setting $S3$) and the FER error floor is lowered from $10^{-2}$ (for setting $S1$) to $5 \times 10^{-5}$ (for setting $S3$).

Finally, we consider the latency and hardware implementation of the decoders. For $r = 4$ (hence $d = r + 2 = 6$),...
Fig. 11. Floating-point and fixed-point BER/FER performance of the layered PLDPC-Hadamard decoders. Floating-point results are obtained by computer simulations and the fixed-point results are generated via the FPGA platform with bit-widths setting $S_1$, $S_2$ or $S_3$. $r = 4$, $l = 1327104$, $I = 20$, 150 and $N_h = 64$ or 128.

| Available LUT | 1,182,240 |
|---------------|-----------|
| Available BRAM | 2,160 |

**TABLE III**

COMPARISON OF IMPLEMENTATION RESULTS FOR PLDPC-HADAMARD DECODER WITH 64 AND 128 HADAMARD SUB-DECODERS.

**BIT-WIDTHS SETTINGS $S_1$ AND $S_2$ ARE USED. 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** | $N_h = 64$ | $N_h = 128$ |
| **S1: LUT Utilization** | 485,738 (41.09%) | 968,538 (81.92%) |
| **S1: BRAM Utilization** | 718.5 (33.26%) | 715 (33.10%) |
| **S2: LUT Utilization** | NA | 1,102,958 (93.29%) |
| **S2: BRAM Utilization** | NA | 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 |
| **Coded throughput** | 0.10 Gbps, 0.77 Gbps, 0.20 Gbps, 1.48 Gbps |

$t_{1st\; output} = (d/2 + 2r + 1) = 12$ 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 $t_{1} = 24$ cycles. $^8$ Similarly when $G = 8$, $t_{loading} = dG/2 = 24 > t_{1st\; output}$ which belongs to Case II. The decoding latency per layer equals $t_{2} = 48$ cycles. Table III lists the hardware implementation results of the proposed layered decoder for $N_h = 64$ ($G = 8$) and $N_h = 128$ ($G = 4$). $^9$

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 and under the same bit widths setting $S_1$, 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). For the decoder with $N_h = 128$ sub-decoders, increasing the bit widths from setting $S_1$ to setting $S_2$ increases the LUT utilization from 81.92% to 93.29% but does not change the amount of BRAMs used. Note that the hardware utilization of bit widths setting $S_3$ is almost the same as that of bit widths setting $S_2$ and is therefore not shown in Table III.

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 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. Moreover, increasing the bit widths of the data in the decoder has shown to be an effective way of eliminating/lowering the FER/BER error floors.

$^8$In practice, there is a fixed delay $t_d$ when operating RAMs. In our designs, $t_d = 2$ cycles and are included in deriving the latency and throughput in Table III.

$^9$The vast majority of the hardware resources are used in (i) storing the LLRs (with BRAMs) and (ii) implementing the parallel Hadamard sub-decoders (with LUTs). The usage of the resources for other purposes is relatively very small.
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.

To eliminate/lower the error floors, we have increased the bit widths in our decoder design. Another future direction of research is to design variable bit widths (i.e., quantization schemes) in the decoder [38] so as to reduce the complexity of the decoder and to remove the error floor issue.

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, May 1993, pp. 1064–1070.

[2] R. G. Gallager, “Low-density parity-check codes,” Ph.D. thesis, Dept. Elect. Eng., Massachusetts Inst. Technol., Cambridge, MA, USA, 1963.

[3] D. Divsalar, H. Jin, and R. McEliece, “Coding theorems for ‘turbo-like’ codes,” in Proc. Allerton Conf., 1998, pp. 201–210.

[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. Breijer, L. Hanzo, G. M. Maunder, I. Al-Hashimi, C. Berrou, and L. Hanzo, “20 years of turbo coding and energy-aware design guidelines for energy-constrained wireless applications,” IEEE Commun. Surveys Tuts., vol. 18, no. 1, pp. 8–28, 1st Quart., 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, Jun. 2015.

[7] Y. Liu, P. M. Ooms, and G. M. Mitchell, “On generalized LDPC codes for 5G ultra reliable communication,” in Proc. IEEE Int. Conf. Commun. (ICC), 2018, pp. 1–5.

[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. Surveys Tuts., vol. 20, no. 2, pp. 1574–1606, 2nd Quart., 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] C.-C. Cheng, J.-D. Yang, H.-C. Lee, C.-Y. Yang, and Y.-L. Ueng, “A fully parallel LDPC decoder architecture using probabilistic min-sum algorithm for high-throughput applications,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 9, pp. 2738–2746, Sep. 2014.

[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, Jan. 2015.

[12] Y. Lee, M. Li, and R. V. Saranovac, and A. Hadadlevic, “Flexible high-throughput QC-LDPC decoder with perfect pipeline conflicts resolution and efficient hardware utilization,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 67, no. 12, pp. 5454–5467, Dec. 2020.

[13] G. Yue, L. Ping, and X. Wang, “Generalized low-density parity-check codes based on Hadamard constraints,” IEEE Trans. Inf. Theory, vol. 53, no. 10, pp. 3968–3983, 2007.

[14] D. J. Costello and G. D. Forney, “Channel coding: The road to channel capacity,” Proc. IEEE, vol. 95, no. 6, pp. 1150–1177, Jun. 2007.

[15] W. R. Leung, G. Yue, L. Ping, and X. Wang, “Concatenated zigzag Hadamard codes,” IEEE Trans. Inf. Theory, vol. 52, no. 4, pp. 1711–1723, Apr. 2006.

[16] L. Ping, L. Liu, K. Wu, and W. K. Leung, “Interleave division multiple-access,” IEEE Trans. Wireless Commun., vol. 4, no. 4, pp. 938–947, Jul. 2005.

[17] P. W. Zhang, F. C. M. Lau, and C.-W. Sham, “Protagonist-based low-density parity-check codes,” 2020, arXiv:2010.08285.

[18] P. W. Zhang, F. C. M. Lau, and C.-W. Sham, “Protagonist-based LDPC Hadamard codes,” IEEE Commun. Lett., vol. 69, no. 8, pp. 998–5013, Aug. 2021.

[19] 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, Jun. 2021.

[20] 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. 4532–4583, Jul. 2013.

[21] M. P. C. 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.

[22] 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. 1837–1869, Jul. 2013.

[23] 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, Oct. 2020.

[24] X. Zhang and P. H. Siegel, “Quantized iterative message passing decoders with low error floor for LDPC codes,” IEEE Trans. Commun., vol. 62, no. 1, pp. 1–14, Jan. 2014.
Peng-Wei Zhang received the B.E. degree in electronics and information engineering and the M.E. degree in electronics and communication engineering from the Chongqing University of Posts and Telecommunications, China, in 2013 and 2016, respectively, and the Ph.D. degree from the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong, SAR, China, in 2021. He is currently with Huawei Technologies Ltd., Chengdu, China.

Sheng Jiang received the B.E. degree in microelectronics from Shanghai Jiao Tong University, China, and the M.E. degree in electronic engineering from The Hong Kong University of Science and Technology, Hong Kong, and the Ph.D. degree from The Hong Kong Polytechnic University, Hong Kong. He is currently a Post-Doctoral Fellow with The Hong Kong Polytechnic University.

Francis C. M. Lau (Fellow, IEEE) received the B.Eng. degree (Hons.) in electrical and electronic engineering and the Ph.D. degree from King’s College London, University of London, U.K. He is currently a Professor with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong. He is the coauthor of two research monographs. He is also a co-holder of six U.S. patents. He has published more than 330 papers. His main research interests include channel coding, cooperative networks, wireless sensor networks, chaos-based digital communications, applications of complex-network theories, and wireless communications.

Prof. Lau is a Fellow of IET. He is a co-recipient of one Natural Science Award from the Guangdong Provincial Government, China; eight best/outstanding conference paper awards; one Technology Transfer Award; two young scientist awards from the International Union of Radio Science; and one FPGA Design Competition Award. He was the General Co-Chair of International Symposium on Turbo Codes and Iterative Information Processing in 2018 and the Chair of Technical Committee on Nonlinear Circuits and Systems, IEEE Circuits and Systems Society (2012–2013). He served as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS (2004–2005 and 2015–2019), IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS (2006–2007), and IEEE Circuits and Systems Magazine (2012–2015). Since 2010, he has been a Guest Associate Editor of International Journal of Bifurcation and Chaos.

Chiu-Wing Sham (Senior Member, IEEE) received the bachelor’s degree in computer engineering and the M.Phil. and Ph.D. degrees from The Chinese University of Hong Kong, in 2000, 2002, and 2006, respectively. He was worked as an Electronic Engineer on the FPGA applications of the motion-control system and system security with cryptography at ASM Pacific Technology Ltd., Hong Kong. During the years at The Hong Kong Polytechnic University, he engaged in various university projects for the commercialization of technology, in particular, a few optical communication projects which were in collaboration with Huawei. He also worked on the physical design of VLSI design automation. He was invited to work at Synopsys, Inc., Shanghai, in Summer 2005, as a Visiting Research Engineer. He is currently working with The University of Auckland as a Senior Lecturer. He has also been an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS in 2017.