An FPGA Implementation of Two-Input LUT Based Information Bottleneck LDPC Decoders

Bo-Yu Tseng, Brian M. Kurkoski
School of Information Science
Japan Advanced Institute of Science and Technology
Email: {tseng, kurkoski}@jaist.ac.jp

Philipp Mohr, Gerhard Bauch
Institute of Communications
Hamburg University of Technology
Email: {philipp.mohr, bauch} @tuhh.de

***Abstract*—A lookup table-based check and variable node are considered for designing low-density parity check (LDPC) decoder architectures, using the principle of the information bottleneck method. It has been shown that an information bottleneck LUT operation can outperform conventional min-sum arithmetic operation in terms of error-correction capability. This paper presents a cost-efficient hardware implementation of LUT-based node processing units in the decoder architecture. It exploits the symmetry of the communication channel and multi-input LUT decomposition to generate a reduced-size LUT structure. The LUT operations are designed as a two-level memory subsystem enabling LUT mappings reconfiguration at runtime. As a case study, a rate-7/10 7650-bit regular QC-LDPC decoder is implemented on an FPGA, achieving a throughput of up to 1.345 Gbps at 10 iterations. Compared with the conventional offset min-sum decoder, the proposed decoder increases the throughput-to-area ratio up to 39.22% at a cost of no more than 0.08 dB decoding performance loss. In addition, the hardware complexities of node design variants are investigated.**

I. Introduction

Low-density parity-check (LDPC) codes are a class of forward error correcting codes widely used in modern communication systems, e.g. non-volatile memories, fibre optic networks and wireless communication.

The LDPC codes can be decoded using the state-of-the-art min-sum (MS) algorithm, which updates and exchanges two types of messages: check-to-variable and variable-to-check messages between check and variable nodes [1]. The message updates are conventionally based on arithmetic operations where operands and calculated results directly represent probabilistic information, e.g., log-likelihood ratio. Another class of novel update methods replaces those arithmetic operations with simple lookup tables (LUTs) where equivalent operands and results are integer values encoding exact probabilistic information [2]–[5]. The LUT compresses input messages such that the mutual information between the relevant codeword bits and corresponding message is maximized. In [3] a *sequential information bottleneck* (IB) algorithm is proposed to generate deterministic LUTs which reaches a lower bit-error rate than MS decoding. However, there is still a lack of insight into the required hardware design. In particular, exponential growth of LUT size with message bit widths, and the required number of LUTs for parallelism of node processing units can be the challenges for practical designs. Regarding the existing hardware implementation of information-maximizing LUTs, an ASIC solution based on combinatorial circuits has been proposed [4]. Despite present advantage of area efficiency and ultrahigh decoding throughput, such hardwired LUTs maintain no flexibility for post-fabrication modifications. Furthermore, the area saving of hardwired LUTs highly depends on the actual data patterns of lookup input-output mapping values. Thus a deterministic analysis of hardware complexity is difficult.

The aim of this work is to develop reconfigurable LUT-based node processing units for layered LDPC decoder architecture using the information bottleneck method. As a result, the hardware utilization and error correction are comparable to those of MS decoder architecture. For this purpose, the multi-input LUT decomposition [2] is considered. A reduced-size LUT structure is generated by exploiting the symmetric communication channel, and the LUT sharing is introduced to reduce the number of LUT components inside one node processing unit. Furthermore, node processing units and corresponding LUT structures are implemented as a memory subsystem with two-level hierarchy. It efficiently distributes the large number of mapping values to large-sized ROMs and small-sized RAMs. In this way, not only is the storage required for mapping values reduced, but also LUT mapping changes without hardware modification are possible.

The proposed architecture is implemented on Xilinx Zynq UltraScale+ MPSoC ZCU104 FPGA platform in a partially parallel manner. In particular, a rate-7/10 regular quasi-cyclic code is evaluated under 4 bits and 3 bits message quantization. The implemented decoder architecture can achieve maximum 24.4% area savings and 1.39x throughput-to-area ratio over the conventional MS decoders at a cost of no more than 0.08 dB performance loss in terms of bit error rate. In particular, the 3-bit quantized decoders achieved gigabit throughput even when the maximum number of iterations is 10. Moreover, an investigation into the hardware utilization of node design variants is carried out, which provides a design guideline for satisfying the required hardware constraint and specifications.

II. LDPC Codes and Decoding Processes

A regular LDPC code is specified by an $M \times N$ parity-check matrix $\mathbf{H}$ which can be graphically represented by a bipartite graph [6]. The two disjoint sets of a given parity-check matrix represent: variable nodes (VNs) associated to N codeword bits $b_i \in \{0, 1\}, i = \{0, 1, \cdots, N-1\}$, and check nodes (CNs) corresponding to M parity-check operations. The jth CN is adjacent to ith VN if and only if $H_{j,i} = 1$ indicating a message exchange between them. Moreover, a set of VNs adjacent to jth CN is denoted by $\mathcal{N}_j$, whereas a set of CNs adjacent to ith VN is denoted by $\mathcal{M}_i$. The degrees of CN and VN are $d_c = |\mathcal{N}_j|$ and $d_v = |\mathcal{M}_i|$. In addition, a well structured LDPC code, *quasi-cyclic* (QC) codes [7] parameterized by a *lifting degree* $Z \in \mathbb{Z}^+$, is widely utilized in practical systems [8]–[10] which partitions H matrix into $M_Z \cdot N_Z$ number of Z-by-Z sub-matrices. The sub-matrix is a cyclic shifted identity matrix I_Z by a factor ranging in $\{0, 1, ..., Z-1\}$.

A. Offset min-sum decoding algorithm

The typical message update operations behind CN and VN are based on the offset min-sum algorithm [11] described by the following equations. First, let $L_{c_j \to v_i}$ and $L_{v_i \to c_j}$ denote extrinsic messages sent from jth CN to ith VN, and ith VN to jth CN, respectively. Any $L_{c_j \to v_i}$ and $L_{v_i \to c_j}$ are iteratively updated and exchanged according to

$$L_{c_j \to v_i} = \prod_{\forall i' \in \mathcal{N}_j \setminus i} \operatorname{sgn}(L_{v_{i'}}) \cdot \max(\min_{\forall i' \in \mathcal{N}_j \setminus i} |L_{v_{i'}}| - \beta, 0), \tag{1}$$

$$L_{v_i \to c_j} = L_i^{ch} + \sum_{\forall j' \in \mathcal{M}_i \setminus j} L_{c_{j'}}, \tag{2}$$

where L_i^{ch} is the channel log-likelihood ratio (LLR) to measure how likely codeword bit c_i is 0 or 1, whilst β is an offset factor. Here, $\operatorname{sgn}(x)$ is the sign function extracting the sign bit of x. For each CN and VN, a total of d_c and d_v messages are processed from above equations. In addition, the *a-posterior probability* (APP) for each codeword bit is calculated at the end of every iteration, i.e.

$$L_i^{APP} = L_i^{ch} + \sum_{\forall j \in \mathcal{M}_i} L_{c_j \to v_i}, \tag{3}$$

Finally, the hard decision for each codeword bit is taken that $\hat{c}_i = 1$ if $L_i^{APP} \leq 0$, otherwise 0. The message updates repeat until $H \cdot \hat{C} = 0$ where $\hat{C} = [\hat{c}_0 \cdots \hat{c}_{\mathcal{N}-1}]^T$, or a predefined maximum number of iterations i_{max} has been reached.

III. On the Design of IB-LDPC Decoders

The information bottleneck LDPC (IB-LDPC) decoders have been proposed to replace conventional arithmetic operations corresponding to (1)-(3) with simple lookup tables (LUTs). Thus, only integer-valued messages from a small alphabet are handled during the entire decoding process, as opposed to a floating-point representation. The LUT design for the node processing units is based on discrete density evolution [2] [12] and application of the IB method [3]. The IB method was introduced as a mathematical framework for preserving relevant information about the variable X, while performing compression of an observed random variable Y which is mapped to the compressed variable T [13].

In this work, the IB decoder design with layered schedule proposed in [5], is used. The channel is assumed to be a symmetric BPSK AWGN channel $\tilde{y}_k^{ch} = f(b_k) + n_k$ with $f : \{0, 1\} \to \{+1, -1\}$ and n_k is a normal distributed random variable with variance $\sigma^2 = N_0/2$. A threshold quantizer $t_k^{ch} = Q(\tilde{y}_k^{ch})$ is designed to maximize the mutual information $I(B_k; T_k^{ch})$ with respect to each code bit b_k, using the IB method. All compressed messages are w bits wide.

Multiple options exist to perform the node operations when working with IB-LUT decoders. One variant is to use a two-input lookup table structure in the check node and in the variable node, denoted by the 'LUT-LUT' combination in [5].

In the 'LUT-LUT' configuration, the CN LUT exploits the underlying parity check equation $b_0 \oplus \ldots \oplus b_{d_c-1} = 0$ and the input messages $y_0, \ldots, y_{d_c-1} \in \{0, 1, \ldots, 2^w - 1\}^{d_c-1}$ to compute extrinsic information about each of the participating bits b_n. For example, extrinsic information about bit b_{d_c-1} can be obtained from the messages $y_0, \ldots, y_{d_c-2}$. Since the number of CN LUT entries has an exponential complexity in w bits and d_c, which leads to $2^{w(d_c-1)}$ entries. It is suggested to decompose that multiple-input LUT into d_c-2 number of cascade two-input partial LUTs. Thus, the corresponding complexity is scaled to $2^{2w} \cdot (d_c-2)$ entries. The update operation for the extrinsic information about b_0 is:

$$t_{k+1} = \Phi_{i,l,k}^c(t_k, y_{k+1}) \text{ for } k = 0, \ldots, d_c - 3, \tag{4}$$

where $t_0 := y_0$ and $i \in \{0, \ldots, i_{\max}-1\}$, $l \in \{0, \ldots, d_v-1\}$ and k denote the different lookup tables for the individual iteration, layer and cascade level of two-input LUT datapath, respectively. Each LUT is designed to $\max_{\Phi_{i,l,k}^c} I(X_{k+1}; T_{k+1})$, where $x_{k+1} = x_k \oplus b_{k+1}$ with $x_0 = b_0$. The message t_{d_c-2} from the last two-input LUT is sent back to the corresponding variable node.

The variable node, representing some codeword bit b_k, observes the inputs y^{ch} and extrinsic check node messages $y_0, \ldots, y_{d_v-2} \in \{0, 1, \ldots, 2^w - 1\}^{d_v-1}$. The two-input LUT decomposition yields a mapping complexity of $2^{2w} \cdot (d_v - 1)$ entries, unlike a d_v-input LUT which requires $2^{w \cdot d_v}$ entries. The corresponding update operation producing one extrinsic message is performed as

$$t_{k+1} = \Phi_{i,l,k}^v(t_k, y_k) \text{ for } k = 0, \ldots, d_v - 2, \tag{5}$$

where $t_0 := y^{ch}$. Each LUT is designed to maximize the mutual information according to $\max_{\Phi_{i,l,k}^v} I(B_k; T_{k+1})$. Finally, t_{d_v-1} is sent back to the check node.

The $\Phi_{i,l,k}(\cdot)$ for different layers and/or iterations are based on distinct sets of input-output mapping values. For example, a message update at (d_v-1)th layer of 0th iteration followed by the message update at 0th layer of $(0+1)$th iteration, use mapping sets of $\Phi_{i=0,l=d_v-1,k}(\cdot)$ and respective $\Phi_{i=1,l=0,k}(\cdot)$. Those mapping sets are not identical.

Another option is the 'min-LUT' configuration, where the minimum approximation is used in the check node [14]. The check node inputs $y_n \in \{-2^{w-1}, \ldots, -1, +1, \ldots, +2^{w-1}\}$, are assumed to have labels sorted according to the underlying log-likelihood ratio $L(b_n|y_n)$. This way the LUT mapping in (4) can be simplified to

$$\Phi_{i,l,k}^c(t_k, y_{k+1}) = \operatorname{sgn}(t_k) \operatorname{sgn}(y_{k+1}) \min(|t_k|, |y_{k+1}|). \tag{6}$$

To avoid any confusion in later sections, the LUTs designed with the IB method and the 6-input LUT of FPGA resources are called *IB-LUT* and *FPGA LUT*, respectively.

IV. Proposed Hardware Architecture

Fig. 1 shows the top module of the proposed LDPC decoder. The received N-bit codeword is represented by a vector form of w-bit soft decisions. The codeword is stored in the *channel message FIFO* as the input source to the decoder. Regarding the data I/Os of node processing units, the updated extrinsic messages $y_{c\to v}$ and $y_{v\to c}$ are kept in the *message-passing buffer* for iterative message exchanges between variable and check node processes. The buffering locations are decided by the *routing network*, with the input from the node processing units. In the parallel node processing configuration, P^c check node units and $P^c{\cdot}s^{row}$ variable node units are parallelized. The $s^{row} \in \{1,\dots,d_c\}$ denotes the row split factor and determines the bandwidth of each check node, as the number of output extrinsic messages per clock cycle.

In this work, any LUT mapping function is designed as a memory structure that stores a certain number of input-output mapping values for the iterative message update. The overall sizes of $d_v \cdot i_{max}$ sets of mapping values is enormous, specially due to the parallelism of node processing units which is a multiple of $d_v \cdot i_{max}$ sets. To limit such overhead from every node processing unit, a two-level memory hierarchy is proposed to provide an efficient data allocation (mapping values) over the first-level read-only memories (ROMs) and the second-level random access memories (RAMs). In the first level, the ROMs are devised to centrally store all $d_v \cdot i_{max}$ sets of mapping values. In the second level, each node processing unit is represented in RAM that caches one set of mapping values for the current lth layer of the current ith iteration. The *LUT configurator* in the left of Fig. 1, periodically makes copies of a mapping value set corresponding to the next layer of the next iteration and write the copied sets into all RAMs in a broadcast fashion. Overall, the read operation of the ROMs and write operation of the RAMs are controlled by the shown memory controller. The precise operation timings are also determined by the update control logic in the LUT configurator. The RAM based node processing unit as well as the control updating are detailed below. Due to the space limitation, the detaild of the routing network and message-passing buffer will appear in a forthcoming extended version of this paper.

A. Cascade datapath of decomposed IB-LUTs

An extrinsic message is generated from a node processing unit through a series of two-input IB-LUTs in a cascade datapath. To implement this cascade datapath with less hardware overhead, the symmetry of communication channel is exploited. Thus, the IB-LUT mappings imply an input-output relation as shown in Fig. 2. For the input-output mapping of check nodes, ignoring the sign bits, all four combinations of inputs lead to exactly the identical magnitudes of mapping results. On the other hand, for the input-output

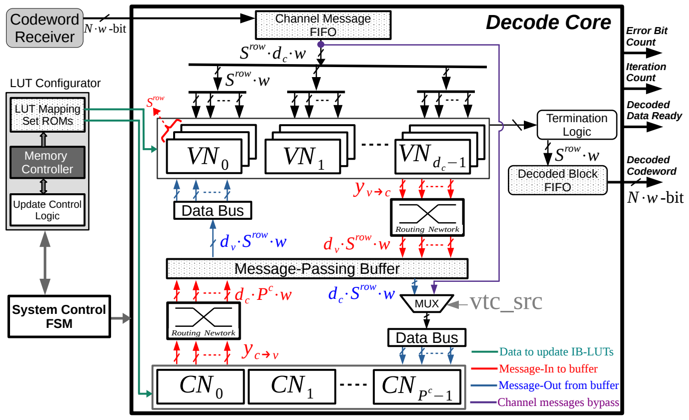
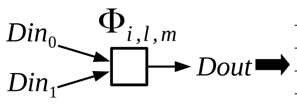


Fig. 1: LDPC decoder top-level architecture.



Din$_0$	Din$_1$	CN: Dout	VN: Dout
$+\alpha$	$+\beta$	$+\gamma$	$+\delta$
$+\alpha$	$-\beta$	$-\gamma$	$+\epsilon$
$-\alpha$	$+\beta$	$-\gamma$	$-\epsilon$
$-\alpha$	$-\beta$	$+\gamma$	$-\delta$

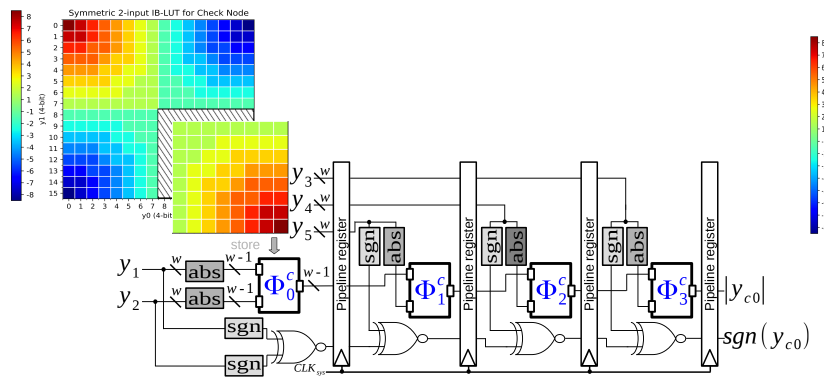
Fig. 2: Symmetric distributions over IB-LUT mappings.

mapping of variable nodes, only if both inputs are flipped on sign bits, the magnitude of that mapping result will be unchanged. Accordingly, it is sufficient to preserve only partial mapping entries from both CN IB-LUTs and VN IB-LUTs, i.e. $|\Phi^c_{i,l,k}(|t_k|,|y_{k+1}|)|$ and $\Phi^v_{i,l,k}(|t_k|,y_k)$, respectively. This way the IB-LUT mapping complexities of the whole cascade datapaths are further reduced to $(d_c-2)\cdot 2^{2(w-1)}$ and $(d_v-1)\cdot 2^{2w-1}$ entries, at a cost of additional logic circuits to reconstruct the sign-valued mapping result t_{k+1}. Those logic circuits can be simplified by applying DeMorgan's theorem to their corresponding Boolean expressions. An example of cascade datapath for updating a single extrinsic message, is depicted in Fig. 3 which implemented the (4) and (5).

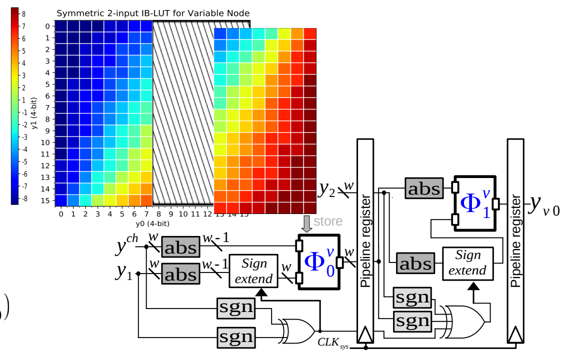
B. Implementation of IB-LUT-based node processing units

A node with a degree of d updates d number of extrinsic messages, the construction of a complete node processing unit and its memory based implementation on FPGAs are discussed herein. Consider the case of a check node with node degree d_c, where the corresponding d_c extrinsic messages are updated in parallel. An efficient construction reuses some of the intermediate results $t_{k+1} = \Phi^c_{i,l,k}(\cdot)$. The reusable t_{k+1} can be identified by using a recursive search. An exemplary data flow graph of 6-degree check node processing unit is illustrated in Fig. 4 where every dashed edge accounts for a data flow of a shared intermediate result. The variable-to-check messages (as incoming messages) and check-to-variable messages (as outgoing messages) are denoted by y_n and y_{c_n}, respectively, for all $n \in \{0,\dots,d_c{-}1\}$. Moreover, the partial data flow including red-coloured vertices and edges is equivalent to a case of cascade datapath in Fig 3a. The same recursive search can be applied in variable nodes.

The next step is to address the FPGA implementation of complete node processing units. As for the aforementioned

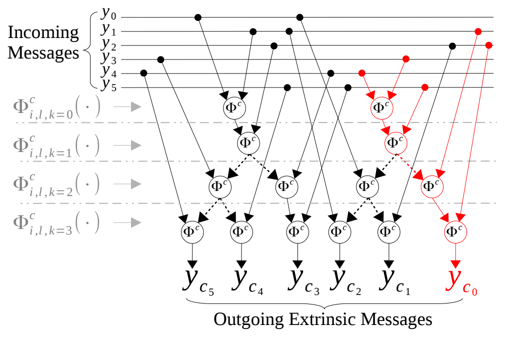


(a) Check-to-variable message update from degree-6 check node.



(b) Variable-to-check message update from degree-3 variable node.

Fig. 3: Diagram of cascade datapaths with $w = 4$ bits, where the $\Phi^c_{i,l,k}(\cdot)$ and $\Phi^v_{i,l,k}(\cdot)$ are rewritten by Φ^c_k and Φ^v_k, respectively.



Incoming Messages
y_0
y_1
y_2
y_3
y_4
y_5
$\Phi^c_{i,l,k=0}(\cdot)$
$\Phi^c_{i,l,k=1}(\cdot)$
$\Phi^c_{i,l,k=2}(\cdot)$
$\Phi^c_{i,l,k=3}(\cdot)$
y_{c_5}
y_{c_4}
y_{c_3}
y_{c_2}
y_{c_1}
y_{c_0}
Outgoing Extrinsic Messages

Fig. 4: Data flow of a check node reusing intermediate results.

two-memory hierarchy in the decoder architecture, the $d_v \cdot i_{max}$ sets of mapping values are stored across a batch of central ROMs. Multiple RAMs keep only one mapping value set associated with $\Phi_{i,l,k}(\cdot)$ of certain indices i and l for all k. Each RAM is updated once in every layer, such RAM update is realized as a LUT remapping operation. To satisfy the memory access granularity of the ROM and RAMs, the *Distributed RAM* (LUTRAM) is employed as the target memory solution to implement each RAM whilst the ROM is implemented by Block RAM (BRAM). The LUTRAM is based on a 6-input logic LUT component with capacity of 64 bits which is sufficient to store all mapping values of one two-input LUT, and it supports an on-the-fly lookup table remapping. On the other hand, a BRAM supports a true dual-port mode with capacity of 36 kibibits where a batch of BRAMs is feasible to store the overall $d_v \cdot i_{max}$ sets of mapping values.

The implementation of ROMs and RAMs is demonstrated in Fig. 5 as an example for a 4-bit quantized decoders. According to the symmetric distribution of LUT (refer to Fig. 2), the number of necessarily cached mapping values, i.e. $|\Phi^c_{i,l,k}(|t_k|, |y_{k+1}|)|$ for all t_k and $y_{k+1} \in \{0, \ldots 2^w - 1\}$, is 64, each of which is 3 bits wide. To improve the memory access bandwidth, the RAM is designed as a bank interleaving structure. So that any two consecutively indexed mapping values can be read and written from/to a RAM in one clock cycle, this is considered a read/write of one RAM page data. More specifically, every memory bit of mapping value is constructed by two LUTRAMs, each of which from one of interleaving banks. As for the ROM implementation, the output ports of every BRAM are configured to match the bit width of RAM page, and therefore each page data of BRAM is a concatenation of two consecutively indexed mapping values.

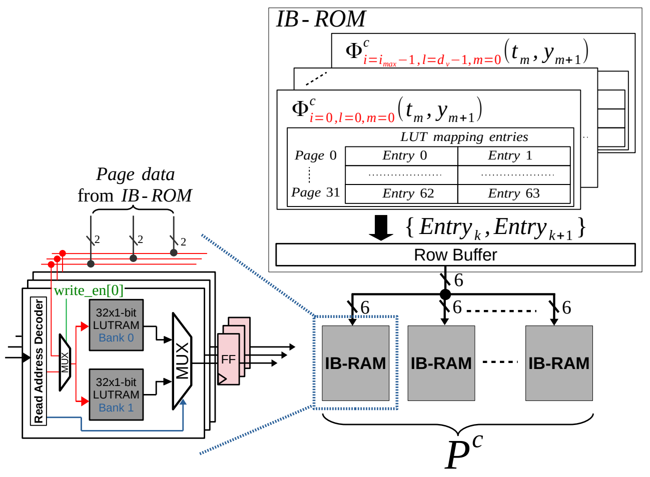


Fig. 5: Diagram of IB-RAMs and IB-ROM for CN IB-LUTs where the control signal *write_en[x]* for $x = 0, 1, 2$, switches the read/write mode of LUTRAMs.

C. Memory controller for IB-LUT remappings

The ROM read operations and RAM write operations dictated by the memory controller generate the read/write addresses with their enable signals to perform the LUT remapping operation over node processing units. To easily describe an arbitrary LUT remapping operation, let $l' = (l+1) \bmod d_v$ and $i' = \lfloor (l+1)/d_v \rfloor + i$ denote the layer and iteration indices of the next message update operation, respectively. Thus, an l' with i' indicates either $(l+1)$th layer of ith iteration or 0th layer of $(i+1)$th iteration.

At the end of current $l$the layer of ith iteration, the LUT remapping operation overwrites RAM pages with the mapping values of $\Phi_{i',l',k}(\cdot)$. During the LUT remapping operation, the involved node processing units are temporarily inaccessible for extrinsic message update. To accelerate the completion of LUT remapping, the clock rate of the write clock CLK_{wr} driving the memory controller, is configured to be higher than the clock rate of system clock CLK_{sys} driving the rest of decoder functionalities. The default settings are $CLK_{wr} = 2 \cdot CLK_{sys}$. Nevertheless, the remapping duration cannot always guarantee not to stall the decoding operation of the next layer or next iteration due to the maximum frequency limit and routing congestion in FPGA synthesis. To further shorten the remapping duration, this work proposes a base-delta ($B+\Delta$) compression scheme inspired by [15]. It is based on an observation that any pair of $\Phi_{i,l,k}(\cdot)$ and $\Phi_{i',l',k}(\cdot)$ generally shows quite slight variation in their mapping values. It enables the use of the proposed $B+\Delta$ scheme that calculates *base* values and *delta* values between identically-indexed mapping values of $\Phi_{i,l,k}(\cdot)$ and $\Phi_{i',l',k}(\cdot)$, where the base and delta values indicate the common value and variance, respectively. Moreover, every nonzero delta value and its corresponding mapping index are recorded in the *delta value ROM* and *delta address ROM*, respectively. The workflow of the compression (in offline mode) and decompression is depicted in Fig. 6.

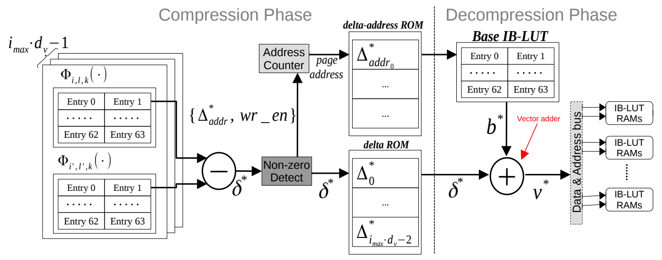


Fig. 6: The $B+\Delta$ based compression and decompression.

During the IB-LUT remapping, a decompressor recovers the original mapping values by adding base values to delta values. Consequently, only the LUT entries marked by delta address ROM are remapped, reducing the write cycles for the corresponding RAMs. According to a preliminary study of the target QC code, the RAM write latency for 3-bit quantized decoder can be hidden within the message-passing duration. On the other hand, for a 4-bit quantized decoder, the write latency only introduces one clock cycle stall to the message update for l'th layer of i'th iteration.

V. Evaluation

The proposed decoders were implemented on a Xilinx ZCU104 development board with Zynq UltraScale+ MPSoC device. The RTL design was written in VerilogHDL and synthesised using Vivado 2019.2. The evaluation considers a QC code with code rate-7/10, codeword length of $N = 7650$ bits, node degrees of $d_v = 3, d_c = 10$ and lifting degree $Z = 765$. The decoder implementation is based on a partially parallel architecture operated in a layered schedule, where only 255 out of 765 CN units and 510 out of 7650 VN units are explicitly built. As for the proposed architecture, both LUT-LUT and min-LUT node operation setups are evaluated. In addition, an 4-bit offset min-sum (OMS) decoding experiment with an offset factor of $\beta = 1$ is considered as the baseline comparison. The messages in the evaluated OMS decoder are quantized via the uniform thresholds of which the distance between any two consecutive thresholds is $\Delta = 0.14$. In Fig. 7, the bit error rate (BER) performance of all decoder settings are presented. The maximum decoding iteration i_{max} of 10 are configured for decoding every codeword. Once the received codeword has been decoded i_{max} iterations but the error bits still exists, it will be counted as one frame error. For every signal-to-noise ratio (SNR) instance, at least 1000 frame errors present. As observed in the results, the 3-bit LUT decoders show the performance loss of up to 0.22 dB and 0.08 dB w.r.t. the 4-bit LUT and 4-bit OMS decoders, respectively.

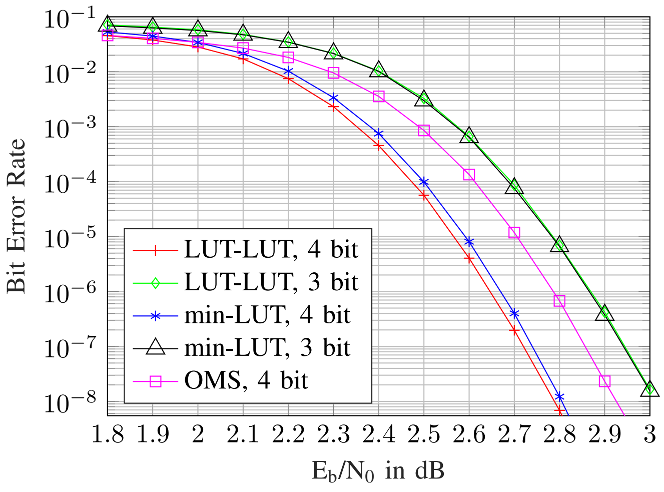


Fig. 7: Bit error rate results.

For a fair comparison, the implementation of all decoders have identical structures of message-passing buffers and routing networks. The CLK_{sys} was set to a maximum frequency f_{max} according to each place and route result. The decoding throughput is calculated as $(N \times f_{max})/(latency \times \text{number of iterations})$, where the N and $latency$ denote codeword length and the latency of one decoding iteration at unit of clock cycle, respectively. Due to the fact that every received codeword with variant error pattern may take different iterations to fully correct error bits, the normalized throughput T_{norm} is more preferable as the evaluation metric, accounting for the throughput per iteration. Another evaluation metric to address hardware efficiency is *throughput-to-area ratio* (TAR), expressing the fraction of normalized throughput contributed by every equal amount of hardware resources. The implementation results of the decoder architectures are summarized in Table I.

Comparing the min-LUT decoder to LUT-LUT decoder, the former shows 17.2% and 20.2% less area overhead (FPGA slices) than the latter one, in 4 and 3 bits, respectively. Regarding throughput-to-area ratio 3 bit setup can achieve 2.1x and 2.0x hardware efficiency under the LUT-LUT decoder and min-LUT decoder, respectively. Another appealing option of utilizing 3-bit min-LUT decoder is its 24.2% area saving and 1.39x hardware efficiency compared to the 4-bit offset min-

TABLE I: Implementation results of the decoder architectures

Decoder type	This work: LUT-LUT		This work: min-LUT		Offset min-sum [11]
Codeword [bit]	7650 ($z = 765$, $d_v = 3$, $d_c = 10$, code rate: 7/10)				
CN & VN parallelism	(255, 510)				
Quantization [bit]	4	3	4	3	4
Resource utilization					
FPGA Slice[1][kCLB]	19.85	13.48	16.44	10.76	14.20
Slice LUT [kLUT]	98.14	51.93	81.32	43.36	59.21
Register [kFF]	125.58	84.27	126.92	83.47	114.93
BRAM [36KiBit]	68	51	60	43	57
f_{max} [MHz]	92.5	132.8	121.5	160	150
Latency [cycle/Iter.]	94	91	94	91	90
T_{norm} [Gbps/Iter.]	7.53	11.16	9.89	13.45	12.75
TAR [Mbps/kCLB]	379.35	827.95	601.62	1250	897.89

[1] A configurable logic block (CLB) contains eight 6-input LUTs and 16 flip-flops.

sum decoder design. Such area saving and hardware efficiency is because of the reduced bit width over all messages, and less routing complexity inside the node processing units.

Apart from evaluations of decoder architectures, it is preferable to provide a clear guideline for designers to consider which type of node processing units is well suited for required decoder specifications and hardware constraint. Table II summarizes the memory and logic overheads of variant node designs to implement node processing units. It includes individual evaluations of offset minimum sorter, minimum sorter, summation, check node LUT and variable node LUT. This way, any particular logic optimization from the FPGA synthesis tool can be disassociated. Since OMS setup does not need any memory, it only costs logic LUTs to implement node operations. As for the BRAM usages in LUT-based node designs, they are actually shared with all parallel nodes as the central ROMs. As observed, the overhead of 4-bit min-LUT configuration requires overhead of 1.83x slice LUTs compared to that of 4-bit offset min-sum configuration, it gains better decoding performance up to 0.134 dB. Thus, it can be a good candidate satisfying requirement of high error correction capability at a reasonable hardware cost. As for the trade-off between hardware efficiency and decoding performance, the 3-bit min-LUT configuration can be a recommended choice that especially costs 12.9%-72.7% less slice LUTs than those of the other configurations.

TABLE II: Logic and memory overheads of node variants

Node design		Logic LUT	LUTRAM[1]	BRAM
Offset minimum sorter	4 bit	41	0	0
Minimum sorter	4 bit	35	0	0
	3 bit	33	0	0
Check node LUT	4 bit	54	54	8
	3 bit	26	18	8
Summation (accumulator)	4 bit	13	0	0
Variable node LUT	4 bit	28	36	3
	3 bit	6	8	3

[1] LUTRAM is a dynamically reconfigurable type of logic LUT, which is based on the same 6-input Slice LUT inside Xilinx FPGA devices.

VI. Conclusion

In this work, an FPGA implementation of LUT-based node processing units was demonstrated in both 3-bit and 4-bit layered LDPC decoder architectures. To mitigate the large storage overhead of LUT mapping values, a reduced-size LUT structure and a memory subsystem with two-level hierarchy are proposed. That provides efficient data allocation of mapping values over the FPGA memory resources, as well as reconfiguability for new patterns of LUT mapping values without modifying the underlying hardware. A study of rate-7/10 QC-LDPC code has been implemented. It achieves a normalized throughput up to 13.45 Gbps with the maximum throughput-to-area ratio of 1250 Mbps/area in addition to 5.1%-24.2% area saving. According to the comprehensive evaluation, the 3-bit min-LUT decoder showed decoding performance close to 4-bit OMS decoding, but with less hardware complexity. Furthermore, since a 3-bit decoder requires fewer bits to represent messages throughout the entire architecture, it is more beneficial for designs targeting low power and low routing complexity. Thus, it is a candidate for cost-efficient applications of reliable error correction using LDPC codes.

References

[1] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of ldpc codes," *IEEE transactions on communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[2] F. J. Cuadros Romero and B. M. Kurkoski, "LDPC decoding mappings that maximize mutual information," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 9, pp. 2391–2401, 2016.

[3] J. Lewandowsky and G. Bauch, "Information-optimum LDPC decoders based on the information bottleneck method," *IEEE Access*, vol. 6, pp. 4054–4071, 2018.

[4] R. Ghanaatian, A. Balatsoukas-Stimming, T. C. Müller, M. Meidlinger, G. Matz, A. Teman, and A. Burg, "A 588-Gb/s LDPC decoder based on finite-alphabet message passing," vol. 26, no. 2. IEEE, 2018, pp. 329–340.

[5] P. Mohr, G. Bauch, F. Yu, and M. Li, "Coarsely Quantized Layered Decoding Using the Information Bottleneck Method," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[6] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.

[7] M. P. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE transactions on information theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[8] O. Boncalo, G. Kolumban-Antal, A. Amaricai, V. Savin, and D. Declercq, "Layered LDPC Decoders With Efficient Memory Access Scheduling and Mapping and Built-In Support for Pipeline Hazards Mitigation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 4, pp. 1643–1656, 2018.

[9] Q. Lu, J. Fan, C.-W. Sham, W. M. Tam, and F. C. Lau, "A 3.0 Gb/s throughput hardware-efficient decoder for cyclically-coupled QC-LDPC codes," vol. 63, no. 1. IEEE, 2016, pp. 134–145.

[10] V. L. Petrović, M. M. Marković, D. M. El Mezeni, L. V. Saranovac, and A. Radošević, "Flexible high throughput QC-LDPC decoder with perfect pipeline conflicts resolution and efficient hardware utilization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 5454–5467, 2020.

[11] J. Chen and M. P. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on communications*, vol. 50, no. 3, pp. 406–414, 2002.

[12] B. M. Kurkoski and H. Yagi, "Quantization of binary-input discrete memoryless channels," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4544–4552, 2014.

[13] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *Proceedings of the 37-th Annual Allerton Conference on Communication, Control and Computing, 1999*, p. 368–377.

[14] M. Meidlinger, A. Balatsoukas-Stimming, A. Burg, and G. Matz, "Quantized message passing for LDPC codes," in *2015 49th Asilomar Conference on Signals, Systems and Computers*. IEEE, 2015, pp. 1606–1610.

[15] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *2012 21st international conference on parallel architectures and compilation techniques (PACT)*. IEEE, 2012, pp. 377–388.