A Fully-Unrolled LDPC Decoder Based on Quantized Message Passing

Alexios Balatsoukas-Stimming*, Michael Meidlinger†, Reza Ghanaatian*, Gerald Matz†, and Andreas Burg*

*EPFL, Switzerland
Email: {alexios.balatsoukas, reza.ghanaatian, andreas.burg}@epfl.ch

†Vienna University of Technology, Austria
Email: {mmeidlin, gmatz}@nt.tuwien.ac.at

***Abstract*—In this paper, we propose a finite alphabet message passing algorithm for LDPC codes that replaces the standard min-sum variable node update rule by a mapping based on generic look-up tables. This mapping is designed in a way that maximizes the mutual information between the decoder messages and the codeword bits. We show that our decoder can deliver the same error rate performance as the conventional decoder with a much smaller message bit-width. Finally, we use the proposed algorithm to design a fully unrolled LDPC decoder hardware architecture.**

I. Introduction

The excellent error correction performance of *low-density parity-check (LDPC)* codes, alongside with the availability of low-complexity and highly parallel decoding algorithms and hardware architectures makes them an attractive choice for many high throughput communication systems. LDPC codes are traditionally decoded using iterative *message passing (MP)* algorithms like the *sum-product (SP)* algorithm and variants thereof [1], most notably the *min-sum (MS)* algorithm. Those conventional algorithms rely on the exchange of continuous messages, which are usually quantized with resolutions of 4 to 7 bits in most in hardware implementations. Lower resolutions are possible but entail severe performance penalties, especially in the error-floor region [2].

Previous work on quantized MP algorithms for LDPC decoding has shown that decoders which are designed to operate directly on message alphabets of finite size can lead to improved performance. There are numerous different approaches towards the design of such decoders. For example, the authors of [3], [4] and [5] consider *look-up table (LUT)* based update rules that are designed such that the resulting decoders can correct most of the error events contributing to the error floor. However, their design is restricted to codes with column weight 3 and to binary output channels. In [6] a quasi-uniform quantization was proposed which extends the dynamic range of the messages at later iterations and improves the error floor performance. However, the design of [6] still relies on the conventional message update rules and therefore does not reduce the required message bit-width. Finally, the authors of [7], [8] consider message updates based on an information theoretic fidelity criterion. While [3], [4], and [6] analyze the performance of their decoding schemes by means of *frame error rate (FER)* simulations, [7] only provides density evolution results and [8] focuses solely on the algorithm for designing the message update rules. To the best of our knowledge, none of the above schemes have been assessed in terms of their impact on hardware implementations.

Funded by WWTF Grant ICT12-054.

Contribution: In this paper, we derive a low-complexity decoding algorithm that is designed to directly operate with a finite message alphabet and that manages to achieve better error-rate performance than conventional algorithms with message resolutions as low as 3 bits. Based on this algorithm, we synthesize a fully unrolled LDPC decoder and compare our results with our implementation of the only existing fully unrolled LDPC decoder [9]. Our approach for the design of the variable node update rule is similar to [7], [8], but we use a more sophisticated tree structure as well as a different check node update rule.

II. LDPC Codes and Min-Sum Decoding

A (d_v, d_c)-regular LDPC code is the set of codewords

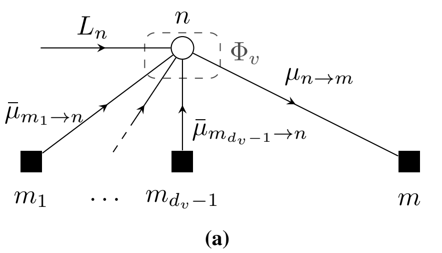
$$\big\{\boldsymbol{c} \in \{0,1\}^N \big| \boldsymbol{Hc} = \boldsymbol{0}\big\}, \tag{1}$$

where all operations are performed modulo 2. The parity check matrix $\boldsymbol{H} \in \{0,1\}^{M\times N}$ contains d_v ones per column and d_c ones per row and is sparse in the sense that $d_v < d_c \ll N$. The parity-check matrix forms an incidence matrix for a Tanner graph which contains N *variable nodes (VNs)* and M *check nodes (CNs)*. Variable node n is connected to check node m if and only if $\boldsymbol{H}_{mn} = 1$.

LDPC codes are traditionally decoded using MP algorithms, where information is exchanged between the VNs and the CNs over the course of several decoding iterations. Let the message alphabet be denoted by $\mathcal{M}$. For simplicity, in this work we assume that $\mathcal{M}$ does not change over the iterations. At each iteration the messages from VN n to CN m are computed using the mapping $\Phi_v : \mathcal{L} \times \mathcal{M}^{d_v-1} \to \mathcal{M}$, which is defined as

$$\mu_{n\to m} = \Phi_v\big(L_n, \bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\setminus m\to n}\big), \tag{2}$$

where $\mathcal{N}(n)$ denotes the neighbours of node n in the Tanner graph, $\bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\setminus m\to n} \in \mathcal{M}^{d_v-1}$, is a vector that contains the incoming messages from all neighboring CNs except m, and $L_n \in \mathcal{L}$ denotes the channel *log-likelihood ratio (LLR)* corresponding to VN n. Similarly, the CN-to-VN messages



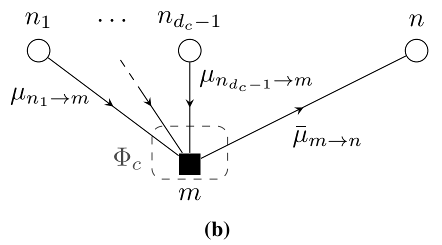


Fig. 1: VN update (a) and CN update (b) for $\mathcal{N}(n) = \{m, m_1, \ldots, m_{d_v-1}\}$ and $\mathcal{N}(m) = \{n, n_1, \ldots, n_{d_c-1}\}$

are computed using the mapping $\Phi_c : \mathcal{M}^{d_c-1} \to \mathcal{M}$, which is defined as

$$\bar{\mu}_{m\to n} = \Phi_c\big(\boldsymbol{\mu}_{\mathcal{N}(m)\setminus n\to m}\big), \tag{3}$$

Figure 1 illustrates the message updates in the Tanner graph. In addition to Φ_v and Φ_c, a third mapping $\Phi_d : \mathcal{L} \times \mathcal{M}^{d_v} \to \{0, 1\}$ is needed to provide an estimate of the transmitted codeword bit based on the incoming check node messages and the channel LLR L_n

$$\hat{c}_n = \Phi_d(L_n, \bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\to n}). \tag{4}$$

For the widely used MS algorithm, the mappings read

$$\Phi_v^{\mathrm{MS}}(L, \bar{\boldsymbol{\mu}}) = L + \sum_i \bar{\mu}_i, \tag{5}$$

$$\Phi_c^{\mathrm{MS}}(\boldsymbol{\mu}) = \operatorname{sign}\boldsymbol{\mu}\, \min|\boldsymbol{\mu}|, \tag{6}$$

where $\min|\boldsymbol{\mu}|$ denotes the minimum of the absolute values of the vector components, $\operatorname{sign}\boldsymbol{\mu} = \prod_j \operatorname{sign}\mu_j$ and $\mathcal{M} = \mathcal{L} = \mathbb{R}$. The decision mapping Φ_d is defined as

$$\Phi_d^{\mathrm{MS}}(L, \bar{\boldsymbol{\mu}}) = \frac{1}{2}\left(1 - \operatorname{sign}\left(L + \sum_i \bar{\mu}_i\right)\right). \tag{7}$$

III. Finite-Alphabet Decoder Design algorithm

The MS algorithm assumes that the message set $\mathcal{M}$ and the LLR set $\mathcal{L}$ are real numbers. However, it is impractical to use floating-point arithmetic in hardware implementations of such decoders and the message alphabets are usually discretized using a relatively low number of uniformly spaced quantization levels. This uniform quantization, together with the well-established two's complement and sign-magnitude binary encoding, leads to efficient arithmetic circuits, but it is not necessarily the best choice in terms of error-rate performance.

Recently, efforts have been made to devise decoders that are designed to work directly with finite message and LLR alphabets [3], [7]. Instead of arithmetic computations such as (5) and (6), the update rules for these decoders are implemented as look-up tables (LUTs). There are numerous approaches to the design of such LUTs. In the following, we provide an algorithm that is a mixture between the conventional MS algorithm and purely LUT-based decoders. More specifically, we only replace the VN update rules with LUTs, which are designed using an information theoretic metric. For the design of the CNs, we exploit the fact that the outputs of the LUT-based VNs, although not representing real numbers, can be ordered and for symmetric channels, the message sign can be directly inferred from the labels, cf. section III-B. This allows us to use the standard MS update rule, thereby avoiding the high hardware complexity that a LUT-based CN design would cause for codes with high CN degree. Our hybrid algorithm provides excellent performance even with very few message levels and leads to an efficient hardware architecture, which is described in detail in Section IV.

A. *Mutual Information Based VN LUT Design*

The key idea behind the LUT design method that we employ is that, given the CN-to-VN message distributions of the previous iterations, one can design the VN LUTs for each iteration in a way that maximizes the mutual information between the VN output messages and the codeword bit corresponding to the VN in question.

We first describe how the distribution of the CN-to-VN messages can be computed based on the distribution of the incoming CN-to-VN messages. If the Tanner graph is cycle-free, then the individual input messages of a CN at iteration i are iid conditioned on the transmitted bit x[1], and their distribution is denoted by $p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu|x)$. Then, the joint distribution of the $(d_c - 1)$ incident messages conditioned on the transmitted bit value corresponding to the recipient VN (cf. Fig. 1) reads

$$p^{(i)}_{\boldsymbol{\mathsf{m}}|\mathsf{x}}(\boldsymbol{\mu}|x) = \left(\frac{1}{2}\right)^{dc-2} \sum_{\boldsymbol{x}:\, \bigoplus \boldsymbol{x}=x} \prod_{j=1}^{d_c-1} p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu_j|x_j), \tag{8}$$

where $\bigoplus \boldsymbol{x}$ denotes the modulo-2 sum of the components of $\boldsymbol{x}$. Using the update rule (6), the distribution of the outgoing CN-to-VN message is then given by

$$p^{(i)}_{\bar{\mathsf{m}}|\mathsf{x}}(\bar{\mu}|x) = \sum_{\boldsymbol{\mu}\in\mathcal{M}_{\bar{\mu}}} p^{(i)}_{\boldsymbol{\mathsf{m}}|\mathsf{x}}(\boldsymbol{\mu}|x_n), \tag{9}$$

where $\mathcal{M}_{\bar{\mu}} \triangleq \left\{\boldsymbol{\mu} \;\middle|\; \min|\boldsymbol{\mu}| = |\bar{\mu}| \;\wedge\; \operatorname{sign}\boldsymbol{\mu} = \operatorname{sign}\bar{\mu}\right\}$. The output message values are given by

$$\mu \triangleq \log \frac{p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu|0)}{p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu|1)}. \tag{10}$$

Conventional decoding algorithms need a high dynamic range in order to represent the growing message magnitudes, as they are using the same message representation for every iteration. In our LUT-based decoder, the message representation changes from one iteration to the next and the message values grow

[1]In the following, random variables are denoted by sans-serif letters.

implicitly as the distributions $p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu|x)$ become more and more concentrated over the course of the iterations, thus providing an explanation for the good performance we can achieve with very low resolutions.

Let $\bar{\boldsymbol{\mu}} = \big(\bar{\mu}_1, \dots, \bar{\mu}_{d_v-1}\big)$ denote the $(d_v - 1)$ incident CN-to-VN messages that are involved in the update of a certain VN (one of which is always the channel LLR L) and let x be the transmit bit corresponding to this VN. Then, the joint distribution of the VN input messages is given by [7]

$$p^{(i)}_{\mathsf{L},\overline{\mathsf{m}}|\mathsf{x}}(L, \bar{\boldsymbol{\mu}}|x) = \sum_{\boldsymbol{x}:\, x_0 = \dots = x_{d_v-1} = x} p_{\mathsf{L}|\mathsf{x}}(L|x_0) \prod_{j=1}^{d_v-1} p^{(i)}_{\overline{\mathsf{m}}|\mathsf{x}}(\bar{\mu}_j|x_j). \tag{11}$$

Given this distribution, we can construct an update rule

$$\Phi_v^{(i)\,\mathrm{MI}} = \arg\max_{Q\in\mathcal{Q}} I(\mathsf{m};\mathsf{x}) = \arg\max_{Q\in\mathcal{Q}} I\big(Q(\mathsf{L},\overline{\mathsf{m}});\mathsf{x}\big), \tag{12}$$

where $\mathcal{Q}$ is the set of all deterministic mappings in the form of (2) and $I(\mathsf{m};\mathsf{x})$ denotes the mutual information between m and x. Hence, the resulting update rule locally maximizes the information flow between the CNs and the VNs.

An algorithm that solves (12) with complexity $O\big(|\mathcal{L}|^3|\mathcal{M}|^{3(d_v-1)}\big)$ was provided in [8]. Using the update rule (12), we can compute the message conditional distribution of the next iteration

$$p^{(i+1)}_{\mathsf{m}|\mathsf{x}}(\mu|x) = \sum_{(L,\bar{\boldsymbol{\mu}}):\, \Phi_v^{(i)\,\mathrm{MI}}(L,\bar{\boldsymbol{\mu}})=\mu} p^{(i)}_{\mathsf{L},\overline{\mathsf{m}}|\mathsf{x}}(L, \bar{\boldsymbol{\mu}}|x). \tag{13}$$

Given an initial message distribution $p^{(0)}_{\mathsf{m}|\mathsf{x}}(\mu|x)$ and a distribution of the channel LLRs $p_{\mathsf{L}|\mathsf{x}}(L|x)$, the repeated alternating application of (8), (9) and (11) to (13) produces a sequence of locally optimal VN update mappings $\Phi_v^{(i)\,\mathrm{MI}},\ i \in \{1,\dots,I\}$, where I denotes a pre-determined maximum number of performed iterations.

B. Discussion and Practical Considerations

1) LUT-based VN and Tree Structure: As the mappings Φ_v take $|\mathcal{L}|\cdot|\mathcal{M}|^{(d_v-1)}$ inputs, a direct application of the algorithm described in Section III-A is restricted to low weight codes. However, we can construct a hierarchy of mappings where each partial mapping only processes a subset of the inputs and the intermediate outputs of preceding stages.[2] The quantizer design for such a hierarchy follows directly by considering only the messages incident to the respective mapping in (11) and, for the intermediate nodes, replacing the distributions (9) of the incident CN messages by the distributions (13) of the previous stage.

2) Channel Output Quantization: So far we considered the initial distributions $p^{(0)}_{\mathsf{m}|\mathsf{x}}(\mu|x)$ and $p_{\mathsf{L}|\mathsf{x}}(L|x)$ as given. When designing practical decoders for communication applications, the initial distributions follow from the transmission channel and the LLR quantization of the preceding signal processing.

[2]In our simulations, we observe that the choice of LUT tree structure can significantly affect the FER performance of the decoder. It is an interesting open problem to identify the best possible tree structure under some complexity constraint (e.g., we could limit the number of LUT inputs to two).

Throughout the rest of the paper, we consider a *binary input additive white Gaussian noise (BI-AWGN)* channel followed by maximum mutual information quantization of the LLRs [10]. In this case, the initial distributions depend on the SNR, which renders the LUT design SNR-specific. Nevertheless, we observe in our simulations that the decoder generally performs well also for SNRs other than the design SNR.

3) Message Representation for Symmetric Channels: Consider the practically relevant case where $|\mathcal{M}|$ and $|\mathcal{L}|$ are even and the distributions $p^{(0)}_{\mathsf{m}|\mathsf{x}}(\mu|x)$ and $p_{\mathsf{L}|\mathsf{x}}(L|x)$ are symmetric in the sense that

$$p_{\mathsf{L}|\mathsf{x}}(L_k|0) \equiv p_{\mathsf{L}|\mathsf{x}}(L_{|\mathcal{L}|-k+1}|1), \qquad k = 1,\dots,\frac{|\mathcal{L}|}{2} \tag{14}$$

$$p^{(0)}_{\mathsf{m}|\mathsf{x}}(\mu_k|0) \equiv p^{(0)}_{\mathsf{m}|\mathsf{x}}(\mu_{|\mathcal{M}|-k+1}|1), \quad k = 1,\dots,\frac{|\mathcal{M}|}{2} \tag{15}$$

or equivalently, expressed in terms of the LLRs values

$$L_k \equiv -L_{|\mathcal{L}|-k+1} \qquad \mu_k \equiv -\mu_{|\mathcal{M}|-k+1}. \tag{16}$$

For that case, computing the CN update (6) is simplified as the sign follows immediately from the message labels. Thus, for the CN update the message values do not need to be stored and the entire decoder can be implemented based on the message labels.

4) Decision Stage: Since the discrete messages of our decoder do not represent real numbers but are labels, a simple arithmetic decision mapping such as (7) is not possible. Instead, Φ_d has to be implemented as a generic mapping as well. The construction of Φ_d is similar to the construction of Φ_v, with the difference that all d_v input messages and the channel LLR have to be processed and that the output is binary.

IV. LUT-Based Fully Unrolled Decoder Hardware Architecture

In the previous section, we have described an algorithm that can construct locally optimal variable node update rules in the form of LUTs for a given quantization bit-width for each iteration for any given (d_v, d_c)-regular LDPC code. Most conventional LDPC decoder architectures are either partially parallel, meaning that fewer than N VNs and M CNs are instantiated, or fully parallel, meaning that N VNs and M CNs are instantiated. Using a LUT-based decoder with a carefully designed quantization scheme can significantly reduce the memory required to store the messages exchanged by the VNs and CNs due to the reduced message bit-width required to achieve the same FER performance. However, both for partially parallel and for fully parallel decoders, separate LUTs would be required within each VN for each one of the performed decoding iterations, significantly increasing the size of each VN, and thus possibly outweighing the gain in the memory area.

An additional degree of parallelism was recently explored in [9], where a *fully unrolled* and fully parallel LDPC decoder was presented. This decoder instantiates N VNs and M CNs for each iteration of the decoding algorithm, leading to a total of NI VNs and MI CNs. While such a fully unrolled

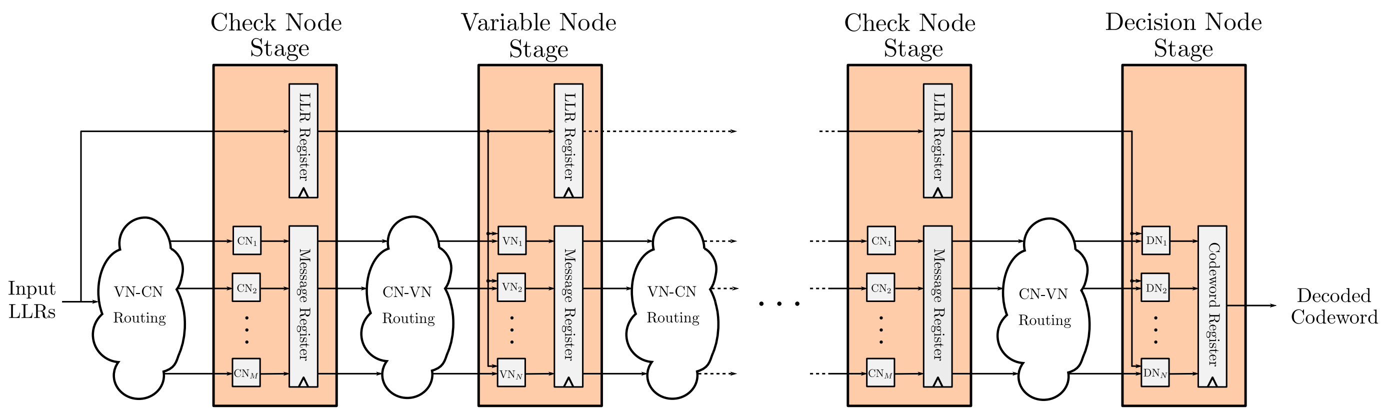


Fig. 2: Top level decoder architecture processing pipeline. The channel LLRs are the input of the left-hand side and the decoded codeword is obtained as the output of the right-hand side.

decoder requires significant hardware resources, it also has a very high throughput since one decoded codeword can be output in each clock cycle. Thus, the hardware efficiency (i.e., throughput per unit area) of the fully unrolled decoder presented in [9] turns out to be significantly better than the hardware efficiency of partially parallel and fully parallel (non-unrolled) approaches. Since in a fully unrolled LDPC decoder architecture VNs and CNs are instantiated for each iteration, it is a very suitable candidate for the application of our LUT-based decoding algorithm.

In this section, we describe the hardware architecture of our fully unrolled LUT-based LDPC decoder. Our hardware architecture is similar to the architecture used in [9], while the most important differences are the optimized LUT-based variable node and the significantly reduced bit-width of all quantities involved in the decoding process.

A. Decoder Architecture

An overview of our decoder architecture is shown in Fig. 2. Each decoding iteration is mapped to a distinct set of variable nodes and check nodes which then form a processing pipeline. In essence, a fully unrolled and fully parallel LDPC decoder is a systolic array in which data flows from left to right. A new set of N channel LLRs can be read in each clock cycle, and a new decoded codeword is output in each clock cycle. The decoding latency as well as the maximum frequency depend on the number of performed iterations as well as the number of pipeline registers present in the decoder. Our decoder consist of three types of stages, namely the CN stage, the VN stage, and the DN stage, which are described in detail in the sequel. As long as a steady flow of input channel LLRs can be provided to the decoder, there is no control logic required apart from the clock and reset signals.

1) Check Node Stage: Each CN stage contains M check node units, as well as Md_c Q_{msg}-bit registers which store the check node output messages, where Q_{msg} denotes the number of bits used to represent the internal decoder messages. Moreover, each CN stage contains N Q_{ch}-bit channel LLR registers which are used to forward the channel LLRs required by the following variable node stages, where Q_{ch} denotes the number of bits used to represent the channel LLRs.

Due to (16), we can use a check node architecture which is practically identical to the check node architecture used in [9]. More specifically, each check node consists of a sorting unit that identifies the two smallest messages among all d_c input messages and an output unit which selects the first or the second minimum for each output, along with the appropriate sign. The sorting unit contains 4-input compare-and-select (CS) units in a tree structure, which identify and output the two smallest values out of the four input values [9]. We use sign-magnitude (SM) to represent all message labels. The SM2TC unit used in the check node of [9] is not required in our architecture since the variable node does not perform any arithmetic operations where the two's complement representation could be favorable.

2) Variable Node Stage: Each VN stage contains N variable node units, as well as Nd_v Q_{msg}-bit registers that store the variable node output messages. Moreover, each VN stage contains N Q_{ch}-bit channel LLR registers which are used to forward the channel LLRs required by the following VN stages.

In the variable node architecture used in the adder-based decoder of [9], all input messages are added and then the input message corresponding to each output is subtracted from the sum in order to form the output message, thus implementing the conventional MS update rule given in (5). In order to avoid overflows, in our implementation of [9] the bit-width of the internal signals is increased by one bit for each addition.

For our LUT-based decoder the adder tree is replaced by d_v LUT trees, each of which computes one of the d_v outputs of the variable node. One possible LUT-tree structure is shown in Fig. 3a, where $\bar{\mu}$ denotes an internal message from a check node and L denotes the channel LLR. LUT sharing between the d_v LUT trees can be achieved by identifying the nodes that appear in more than one tree and instantiating them only once, thus significantly reducing the required hardware resources. Moreover, keeping the number of inputs of each LUT as low as possible ensures that the size of the LUTs, which grows exponentially with the number of inputs, is manageable for the automated logic synthesis process.

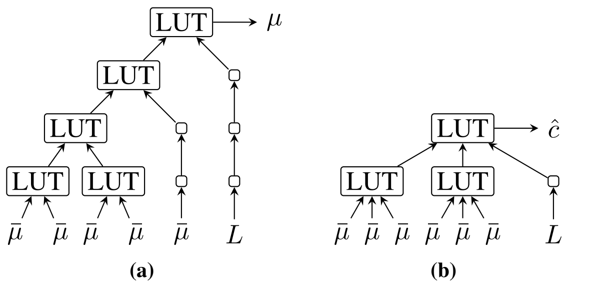


Fig. 3: (a) A variable node LUT tree for the calculation of one output of a variable node of degree $d_v = 6$. Each LUT-based variable node contains d_v such LUT trees, one for each of the $\binom{d_v}{d_v-1}$ possible combinations of input messages.
(b) A decision node LUT tree for a variable node of degree $d_v = 6$. Each LUT-based decision node contains a single decision tree.

3) Decision Node Stage: The variable node that corresponds to the final decoding iteration is called a *decision node* (DN). The DN stage contains N decision nodes, as well N single-bit registers that store the decoded codeword bits. The DN stage does not contain channel LLR registers, as there are no subsequent decoding stages where the channel LLRs would be used. The architecture of a decision node is generally simpler than that of a variable node, as a single output value (i.e., the decoded bit) is calculated instead of d_v distinct outputs.

More specifically, in the architecture of [9], the decision metric of (4) is already calculated as part of the variable node update rule. However, for the decision node, there is no need to subtract each input message from the sum in order to generate d_v distinct output messages. It suffices to check whether the sum is positive or negative, and output the corresponding decoded codeword bit.

In our LUT-based decoder, as discussed in Section III-B4, a LUT tree is designed whose tree node has an output bit-width of a single bit, which is the corresponding decoded codeword bit. An example of a decision LUT tree for a decision node that corresponds to a code with $d_v = 6$ is shown in Fig. 3b. Each decision node contains a single LUT tree, in contrast with the variable nodes which contain d_v LUT trees.

B. Decoding Latency and Throughput

Our LUT-based architecture contains pipeline registers at the output of each stage (VN, CN, and DN). Thus, for a given number of decoding iterations I, the decoding latency is $2I$ clock cycles. Since one decoded codeword is output in each clock cycle, the decoding throughput of the decoder, measured in Gbits/s, is given by $T = Nf$, where f denotes the operating frequency measured in GHz.

C. Memory Requirements

Each pipeline stage except the DN stage requires an NQ_{ch} channel LLR register. Moreover, each VN and CN stage requires Nd_vQ_{msg} (equivalently, Md_cQ_{msg}) registers to store the output messages. Finally, the DN stage requires N registers to store the decoded codeword bits. Thus, the total number of register bits required by our LUT-based decoder can be calculated as

$$B_{\text{tot}} = (2I-1)N(d_vQ_{\text{msg}} + Q_{\text{ch}}) + N. \qquad (17)$$

Naturally, (17) can also be used to calculate the register bits required by an adder-based MS architecture with the same pipeline register structure.

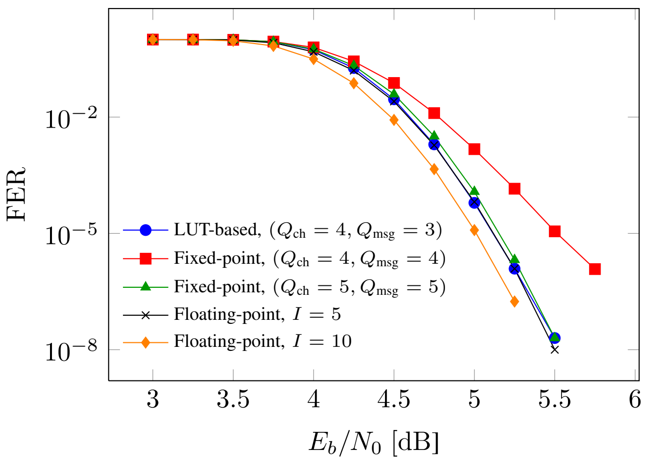


Fig. 4: FER vs E_b/N_0 for the $N = 2048$ $(6, 32)$-regular LDPC code defined in IEEE 802.3an.

V. Implementation Results

In this section, we present synthesis results for a fully unrolled LUT-based LDPC decoder and we compare it with synthesis results of our implementation of a fully unrolled adder-based MS LDPC decoder. We have used the parity-check matrix of the LDPC code defined in the IEEE 802.3an standard [11] (10 Gbit/s Ethernet), which is a $(6, 32)$-regular LDPC code of rate $R = 13/16$ and blocklength $N = 2048$. For the fixed point decoder and the LUT-based decoder, a total of $I = 5$ decoding iterations are performed, since from Fig. 4 we observe that increasing the number of iterations to, e.g., $I = 10$, does not lead to a significant improvement in performance for this LDPC code. All synthesis results are obtained by using a TSMC 90nm CMOS library under typical operating conditions.

A. Quantization Parameters

For the LUT-based decoder, we have used $Q_{\text{ch}} = 4$ bits for the representation of the channel LLRs and $Q_{\text{msg}} = 3$ bits for the representation of the internal messages, as this leads to an error correction performance that is very close the floating-point MS decoder (cf. Fig. 4). For the variable nodes, we use the LUT tree structure of Fig. 3a and for the decision nodes we use the LUT tree structure of Fig. 3b. The design SNR is set to 4.5 dB. For the adder-based MS decoder which serves as a reference, we use $Q_{\text{ch}} = 5$ bits for the representation of the channel LLRs and $Q_{\text{msg}} = 5$ bits for the representation of the internal messages, as this leads to practically the same FER performance for the LUT-based and the adder-based MS decoder, as can be seen in Fig. 4.

TABLE I: Synthesis Results

	Adder-based MS	LUT-based
Area	35.63 mm^2	33.79 mm^2
Frequency	495 MHz	813 MHz
Latency	20.20 ns	12.30 ns
Throughput	1014 Gbps	1665 Gbps
Area Efficiency	28.46 Gbps/mm^2	49.27 Gbps/mm^2

TABLE II: Area Breakdown

	Adder-based MS	LUT-based
	Check Node Stage	
Check Nodes	2.77 mm^2	1.11 mm^2
Pipeline Registers	1.11 mm^2	0.70 mm^2
Total	3.88 mm^2	1.81 mm^2
	Variable Node Stage	
Variable Nodes I_1	2.35 mm^2	4.62 mm^2
Variable Nodes I_2	2.35 mm^2	4.78 mm^2
Variable Nodes I_3	2.35 mm^2	4.64 mm^2
Variable Nodes I_4	2.35 mm^2	4.68 mm^2
Pipeline Registers	1.11 mm^2	0.57 mm^2
Total I_1	3.46 mm^2	5.32 mm^2
Total I_2	3.46 mm^2	5.48 mm^2
Total I_3	3.46 mm^2	5.34 mm^2
Total I_4	3.46 mm^2	5.38 mm^2
	Decision Node Stage	
Decision Nodes	2.035 mm^2	3.21 mm^2
Pipeline Registers	0.03 mm^2	0.03 mm^2
Total	2.38 mm^2	3.24 mm^2
	Top-Level Decoder	
Logic Area	25.58 mm^2	27.46 mm^2
Register Area	10.05 mm^2	6.33 mm^2
Total Area	35.63 mm^2	33.79 mm^2

B. Adder-based vs. LUT-based Decoder

We present synthesis results for the adder-based and the LUT-based decoders in Table I. For fair comparison, we synthesized both designs for various clock constraints and selected the result with the highest hardware efficiency for each design. These results should not be regarded in absolute terms, as the placement and routing of such a large design is highly non-trivial and will increase the area and the delay of both designs significantly. However, it is safe to make relative comparisons, especially when considering the fact that the LUT-based decoder will be easier to place and route due to the fact that it requires approximately 40% fewer wires for the interconnect between the VN, CN, and DN stages. We observe that the LUT-based decoder is approximately 8% smaller as well as 64% faster than the adder-based MS decoder. As a result, the area efficiency of the LUT-based decoder is 73% higher than that of the adder-based MS decoder. For both designs, the critical path goes through the CN, but in the LUT-based decoder the delay is smaller due to the reduced bit-width.

We show the area breakdown of the LUT-based and the adder-based decoders in Table II. We observe that the VN stage area of the LUT-based decoder varies significantly over the iterations, even though the LUT tree structures are identical. This is not unexpected, since the contents of the LUTs are different for different iterations and the resulting logic circuits can have very different complexities. Moreover, we see that the CN stage of the LUT-based decoder is approximately 53% smaller than the CN stage of the adder-based decoder due to the bit-width reduction enabled by the optimized LUT design. The VN stage of the LUT-based decoder, on the other hand, is larger than the VN stage of the adder-based decoder. However, the reduction in the CN stage is larger than the increase in the VN stage, leading to an overall reduction in area. From Table II we can see that this reduction stems mainly from the reduced number of required registers, as the area occupied by the logic of each decoder is similar.

VI. Conclusion

In this paper, we described a method that can be applied to design a discrete message-passing decoder for LDPC codes by replacing the standard VN update rules with locally optimal LUT-based update rules. Moreover, we presented a hardware architecture for a LUT-based fully unrolled LDPC decoder which can reduce the area and increase the operating frequency compared to a conventional adder-based MS decoder by 8% and 64%, respectively, due to the significantly reduced bit-width required to achieve identical error correction performance. Finally, the LUT-based decoder requires approximately 40% fewer wires, simplifying the routing step, which is a known problem in fully parallel architectures.

References

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

[2] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *Communications, IEEE Transactions on*, vol. 57, no. 11, pp. 3258–3268, Nov 2009.

[3] S. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders – part I: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Trans. on Communications*, Oct. 2013.

[4] D. Declercq, B. Vasic, S. Planjery, and E. Li, "Finite alphabet iterative decoders – Part II: Towards guaranteed error correction of LDPC codes via iterative decoder diversity," *IEEE Transactions on Communications*, vol. 61, no. 10, pp. 4046–4057, October 2013.

[5] F. Cai, X. Zhang, D. Declercq, S. Planjery, and B. Vasić, "Finite Alphabet Iterative Decoders for LDPC Codes: Optimization, Architecture and Analysis," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 5, pp. 1366–1375, May 2014.

[6] X. Zhang and P. Siegel, "Quantized Iterative Message Passing Decoders with Low Error Floor for LDPC Codes," *IEEE Trans. on Communications*, vol. 62, no. 1, pp. 1–14, Jan. 2014.

[7] B. Kurkoski, K. Yamaguchi, and K. Kobayashi, "Noise thresholds for discrete LDPC decoding mappings," in *Proc. IEEE Global Telecommunications Conf. (GLOBECOM)*, Nov. 2008.

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

[9] P. Schlafer, N. Wehn, M. Alles, and T. Lehnigk-Emden, "A new dimension of parallelism in ultra high throughput LDPC decoding," in *IEEE Workshop on Signal Processing Systems (SiPS)*, October 2013, pp. 153–158.

[10] A. Winkelbauer and G. Matz, "On quantization of log-likelihood ratios for maximum mutual information," in *Proc. 16th IEEE Int. Workshop on Signal Processing Advances in Wireless Communications (SPAWC 2015).* Stockholm, Sweden: accepted for publication, Jun. 2015.

[11] "IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific Requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," IEEE Std. 802.3an, Sep. 2006.