INVITED PAPER *Special Section on Technology Trials and Proof-of-Concept Activities for 5G and Beyond*

Design and Evaluation of Information Bottleneck LDPC Decoders for Digital Signal Processors

Jan LEWANDOWSKY[†,††a)], **Gerhard BAUCH**[††b)], **Matthias TSCHAUNER**[†c)], *and* **Peter OPPERMANN**[†††d)], *Nonmembers*

SUMMARY Receiver implementations with very low quantization resolution will play an important role in 5G, as high precision quantization and signal processing are costly in terms of computational resources and chip area. Therefore, low resolution receivers with quasi optimum performance will be required to meet complexity and latency constraints. The Information Bottleneck method allows for a novel, information centric approach to design such receivers. The method was originally introduced by Naftali Tishby et al. and mostly used in the machine learning field so far. Interestingly, it can also be applied to build surprisingly good digital communication receivers which work fundamentally different than state-of-the-art receivers. Instead of minimizing the quantization error, receiver components with maximum preservation of relevant information for a given bit width can be designed. All signal processing in the resulting receivers is performed using only simple lookup operations. In this paper, we first provide a brief introduction to the design of receiver components with the Information Bottleneck method. We keep referring to decoding of low-density parity-check codes as a practical example. The focus of the paper lies on practical decoder implementations on a digital signal processor which illustrate the potential of the proposed technique. An Information Bottleneck decoder with 4 bit message passing decoding is found to outperform 8 bit implementations of the well-known min-sum decoder in terms of bit error rate and to perform extremely close to an 8 bit belief propagation decoder, while offering considerably higher net decoding throughput than both conventional decoders.

key words: *mutual information, channel decoding, quantization, information bottleneck method*

1. Introduction

5G receivers will have to satisfy very high requirements concerning power consumption and data throughput. The precision of the analog-to-digital conversion is well known to have a very huge impact on these quantities. Small bit width of the analog-to-digital converter, that is, a small number of quantization levels, is desirable for low implementation complexity and high throughput. Moreover, the applied receiver sided signal processing algorithms which work on the quantized samples from the analog-to-digital converter should be as simple and efficient as possible. One popular bottleneck in this context is the decoding of low-density parity-check (LDPC) codes which is computationally demanding [1], [2]. State-of-the-art LDPC decoders perform iterative message passing using log-likelihood ratios (LLRs). While message passing of continuous LLRs is required in the sum-product algorithm in theory, practical implementations work with quantized messages in fixed point format to reduce complexity [1], [2]. The precision of the messages has a strong impact on the decoder performance and its implementation complexity. Thus, receiver development comprises finding the best trade-off between complexity and acceptable performance degradation of a quantized decoder implementation. In this paper, we present and investigate a novel approach to development of receiver components which automatically yields the desired low quantization resolution and results in low implementation complexity. We apply the Information Bottleneck method [3] which has originally been introduced by Naftali Tishby et al. without any specific application, to development of an LDPC decoder. While having many very interesting applications in learning and classification (for an example see [4]), the Information Bottleneck method is rather unknown in the communications community. However, as we will see, it offers remarkable possibilities for development of quantized receiver implementations with low resolution and very simple, completely homogenous operations. The proposed receiver design is information centric and results in receivers which perform all signal processing using only simple lookup operations in static lookup tables (LUTs). These LUTs are designed in a relevant information preserving manner with the Information Bottleneck method. Works using similar approaches to a number of signal processing problems have been published so far, e.g., in [5]–[10]. Despite in this paper, we stick with LDPC decoding, these works show that the described method is generic and can also be applied to other receiver sided signal processing tasks, such that finally receivers which implement all signal processing with concatenated information preserving LUTs can be built.

In this paper, our most important aim is to give *practical* evidence of advantages of the proposed technique over state-of-the-art approaches. For this purpose, we compare bit error rates, net decoding throughputs and memory demands of practical implementations of Information Bottleneck LDPC

Manuscript received August 30, 2018.
Manuscript revised November 21, 2018.
Manuscript publicized February 20, 2019.

†The authors are with the Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE), 53348 Wachtberg, Germany.
††The authors are with the Institute of Communications of the Hamburg University of Technology, 21073 Hamburg, Germany.
†††The author is with the Fraunhofer Institute for Medical Image Computing (MEVIS), 28359 Bremen, Germany.
a) E-mail: jan.lewandowsky@fkie.fraunhofer.de
b) E-mail: bauch@tuhh.de
c) E-mail: matthias.tschauner@fkie.fraunhofer.de
d) E-mail: peter.oppermann@mevis.fraunhofer.de
DOI: 10.1587/transcom.2018TTI0001

decoders and state-of-the-art decoders on a digital signal processor (DSP).

The paper is structured as follows: In the next section, we introduce preliminaries on the Information Bottleneck method and state-of-the-art LDPC decoding. Section 3 describes in brief how both can be connected according to our preliminary work [9]. Section 4 afterwards describes implementation aspects of the proposed LUT based decoding. Section 5 presents results from DSP implementations, before Sect. 6 concludes the paper.

We use the following notation: Random variable X, realization x, probability distribution $p(x)$, mutual information $\mathrm{I}(X;Y)$, joint and conditional probability distributions $p(x, y)$ and $p(x|y)$, respectively, Kronecker delta $\delta(x)$.

2. Preliminaries

2.1 Information Bottleneck Method

The Information Bottleneck was introduced by Tishby et al. in [3] and is related to rate-distortion theory [11]. The fundamental idea can be summarized as follows. Given an observation of a random variable Y, a mapping of Y onto another variable T shall be found. This mapping is formally described by the conditional probability distribution $p(t|y)$. Typically, the event space of Y is larger than the one of T, such that mapping $p(t|y)$ introduces a compression. The compression relation can be formalized by choosing $p(t|y)$ such that the mutual information $\mathrm{I}(Y;T)$ is minimized. This idea is well known from rate-distortion theory [11]. Other than in rate-distortion theory, however, the mapping $p(t|y)$ is at the same time designed to also maximize the mutual information $\mathrm{I}(T;X)$ between the compression variable T and a so called *relevant* variable X. Thereby it is assumed that the variables $X \to Y \to T$ form a Markov chain and that joint distribution $p(x, y)$ is known. In rate-distortion theory, the idea is instead to keep a distortion measure which needs to be defined in advance below a certain threshold. Finally, in the Information Bottleneck setup, $p(t|y)$ can be considered to be a *relevant information preserving* compression mapping. The variable Y is pressed through a compact bottleneck variable T. T is designed to be informative about X and the cardinality of T influences the width of the bottleneck and, therefore, how much information $\mathrm{I}(T;X) \leq \mathrm{I}(X;Y)$ can be preserved. The authors of [3] explain how to find suitable $p(t|y)$ by making use of Lagrangian multipliers. Several other algorithms which also find suitable mappings $p(t|y)$ have meanwhile appeared in the literature [12]. These algorithms typically input $p(x, y)$ and deliver the desired $p(t|y)$. Moreover, they deliver $p(x,t) = p(x|t)p(t)$ as a side product. The conditional distribution $p(t|y)$ in general can introduce a non-deterministic mapping. In this paper, however, only deterministic mappings $p(t|y)$ will play a role. Such mappings map a certain realization y onto t without probabilistic uncertainty. Therefore, they can be implemented in a simple LUT $t = f(y)$. Mathematically, $p(t|y) = \delta(t - f(y))$. It is important, that the output alphabet of variable T is arbitrary. The mutual information $\mathrm{I}(T;X)$ does not depend on the particular elements of this alphabet, but is only influenced by its cardinality. We will therefore simply consider $t \in \{0, 1, \ldots, 2^q - 1\}$ which is the set of unsigned q bit integers.

2.2 State-of-the-Art LDPC Decoding

LDPC codes are typically decoded using sum-product update rules with iterative message passing on the Tanner graph of a parity-check matrix. The sum-product algorithm is usually implemented in the log-domain with LLRs as exchanged messages. Unfortunately, the required processing is computationally expensive. The involved Tanner graph consists of variable and check nodes. Utilizing the knowledge that in a valid codeword the sum of all bits in the neighborhood of a check node is zero, the check nodes process incoming messages to extract extrinsic information on bit $b_j = \sum_{i \neq j} b_i$. The check nodes in belief propagation LDPC decoders calculate LLRs reflecting the desired extrinsic information as

$$L^{\mathrm{ext}}(b_j) = \sum_{i \neq j}^{\boxplus} L(b_i), \tag{1}$$

where $\sum\boxplus$ denotes the sum notation using the famous box-plus operation

$$L(b_i) \boxplus L(b_j) = \log \frac{1 + \exp(L(b_i) + L(b_j))}{\exp(L(b_i)) + \exp(L(b_j))}. \tag{2}$$

Evaluation of this operation is the most expensive part of the decoding. However, one can rewrite

$$\begin{aligned} &L(b_i) \boxplus L(b_j) = \\ &\mathrm{sign}(L(b_i))\mathrm{sign}(L(b_j)) \min(|L(b_i)|, |L(b_j)|) + \\ &\qquad f_c(|L(b_i) + L(b_j)|) - f_c(|L(b_i) - L(b_j)|). \end{aligned} \tag{3}$$

The function $f_c(x)$ is a Jacobian logarithm correction term which can be implemented in a LUT, such that the computational efforts are reduced in comparison to a straightforward evaluation of (2) [1]. However, in practice the terms including $f_c(x)$ are often simply neglected in Eq. (3). This approach is known as min-sum decoding of LDPC codes. While lowering the computational efforts of iterative message passing decoding, the min-sum approximation degrades the decoding performance. The variable nodes simply sum up incoming LLRs from the check nodes and also channel LLRs for generation of extrinsic information. Message passing between both node types is repeated iteratively until a certain exit criterion, for example, a maximum allowed number of iterations is met.

3. Information Bottleneck LDPC Decoding

The Information Bottleneck method can be applied to design iterative message passing decoders for LDPC codes. The fundamental idea is sketched for an elementary check node

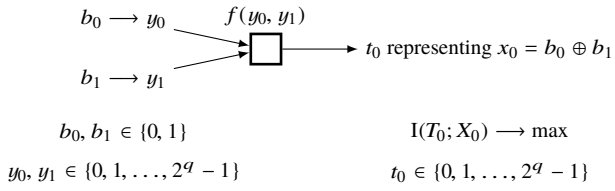

Fig. 1 Illustration of two-input LUT to implement a box-plus equivalent which works on unsigned integers with the Information Bottleck method.

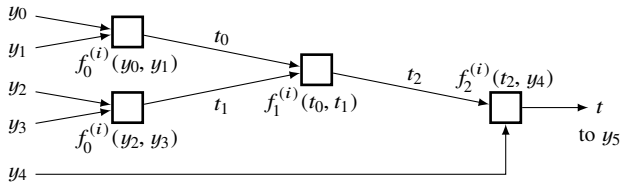

Fig. 2 Illustration of a decomposition implementing a degree $d_c = 6$ check node operation.

operation shown in Fig. 1 in the following. More details can be found in [9]. The shown block inputs two input messages (y_0, y_1) and delivers an outgoing message t_0. In contrast to state-of-the-art LDPC decoding which processes LLRs, we consider incoming messages (y_0, y_1) and the output t_0 to be quantization indices from the set $\{0, 1, \ldots, 2^q - 1\}$. When a q bit quantizer is applied, there exist 2^{2q} distinct combinations (y_0, y_1) and $2^q << 2^{2q}$ possible outputs of the two-input operation. Therefore, the shown system essentially implements a compression. Let $x_0 = b_0 \oplus b_1$ denote the outcome of the exclusive OR operation of two codeword bits b_0 and b_1. Now consider the mentioned compression to be designed with the Information Bottleneck method, such that it preserves the relevant information between the system output T_0 and the random variable X_0 with realization $x_0 = b_0 \oplus b_1$. In this case the shown two-input system serves as an information-optimum integer based equivalent to the box-plus operation of LLRs. The reasoning is that the output $t_0 \in \{0, 1, \ldots, 2^q - 1\}$ is chosen such that the mutual information shared with $x_0 = b_0 \oplus b_1$ is maximized. The most significant difference to the state-of-the-art is that inputs and outputs are from the set of integers representable using q bit in the hardware and, therefore, other than in the conventional decoding algorithms, no processing of any real numbers is required. No LLRs are used in the entire decoding process and determination of t_0 from (y_0, y_1) can simply be performed as a simple lookup operation, $t_0 = \mathbf{lut}[\mathrm{idx}(y_0, y_1)]$. The vector **lut** simply implements mapping $p(t_0|y_0, y_1)$ which is designed using an Information Bottleneck algorithm with two dimensional observation $y = (y_0, y_1)$. The vector holds the respective index $t_0 \in \{0, 1, \ldots, 2^q - 1\}$ which maximizes $\mathrm{I}(T_0; X_0)$ for the particular inputs (y_0, y_1) at position $\mathrm{idx}(y_0, y_1)$. Therefore, only calculation of the indexing function $\mathrm{idx}(y_0, y_1)$ from (y_0, y_1) followed by a lookup at the calculated address is required for generation of extrinsic information on x_0. We use the very simple indexing function

$$\mathrm{idx}(y_0, y_1) = y_0 \cdot 2^q + y_1 \tag{4}$$

Evaluating the indexing function (4) simply corresponds to writing the binary representation of (y_0, y_1) in a memory cell which holds $2q$ bits and, therefore, is extremely simple.

So far, the operation shown in Fig. 1 is limited to processing two inputs. LUTs implementing complete degree d_c check node operations which process $d_c - 1$ inputs to generate extrinsic information can be designed by simply concatenating two-input LUTs, as it is illustrated in Fig. 2 for a degree $d_c = 6$ check node. In the shown example, the decomposed node processes $d_c - 1 = 5$ inputs using several two-input LUTs. The message y_5 is excluded at the input on the left, such that the output t corresponds to the message for the target edge which delivered y_5. Several different possible two-input decompositions exist. Anyway, we focus on degree $d_c = 6$ check node operations with the shown decomposition for a reason described in Sect. 4.2. The principle of designing mutual information maximizing LUTs with the Information Bottleneck method can also be applied for the variable node operations. Also for variable nodes with arbitrary degrees d_v, two-input decompositions can be found, such that only two-input LUTs which input and output unsigned integers are required for their implementation. In this work, we use the two-input decomposition described in [5], [6], [9] for the variable nodes.

The described method requires knowledge of joint distributions $p(x, y)$ because these distributions serve as inputs of the applied Information Bottleneck algorithms which design the LUTs. The fundamental idea for determination of the required joint distributions is application of discretized density evolution to track the input distributions for the design of each two-input LUT in each decoding iteration. The complete density evolution based decoder construction technique is described in detail in [9] and left out here for brevity, as the focus lies on implementation aspects. However, we want to stress that following [9], the LUTs are constructed only once, for a fixed design-E_b/N_0. The choice of this design-E_b/N_0 is important and is described in detail in [9]. As a result, the proposed technique yields a set of *static* LUTs which are applied in the resulting decoder implementation. Indexing and lookup operations are the only required signal processing steps in the resulting decoder and no on the fly generation of the applied LUTs is required.

Please note that due to the evolution of the joint input distributions $p(x, y)$ for the Information Bottleneck method, different two-input LUTs for each node type in each decoding iteration are required. Therefore, in Fig. 2, we have added subscripts $m \in \{0, 1, 2\}$ identifying the particular two-input LUT and also superscripts (i) for the decoder iterations to the two-input LUTs $f_m^{(i)}(u, v)$. Some of the shown LUTs are identical due to identically distributed inputs. Therefore, they have the same indices m and (i). As an example, consider $f_0^{(i)}(y_0, y_1)$ and $f_0^{(i)}(y_2, y_3)$. Both tables process pairs of incoming integer messages (y_k, y_l) which are delivered to the node by the variable nodes. In contrast, LUT $f_1^{(i)}(t_0, t_1)$ processes two intermediate results (t_0, t_1) which follow a different joint distribution than (y_k, y_l). Thus, the applied

table is different. It is practical to store all distinct LUTs for a particular node type in a long vector **LUT**. Please note that a vector **LUT** has to be stored for the check nodes and a different one is required for the variable nodes. Letting M denote the number of *distinct* two-input LUTs required to implement the variable or the check node operation in a particular iteration, the length of this vector for the respective node type is $i_{\max} M \cdot 2^{2q}$, where $i_{\max}$ denotes the maximum allowed number of decoder iterations. The memory amount required to store the **LUT** vectors for exemplary LDPC codes will be investigated in Sect. 5. With vector **LUT** which holds the two-input LUTs of a node type for all iterations concatenatedly, the m-th two-input LUT in the i-th decoding iteration then can calculate its output w under inputs (u, v) as

$$w = \mathbf{LUT}[\mathrm{ptr}(i, m) + \mathrm{idx}(u, v)]. \tag{5}$$

Function $\mathrm{ptr}(i, m)$ points to the beginning of the subvector which characterizes the two-input LUT $f_m^{(i)}(u, v)$ in iteration i. It is calculated as

$$\mathrm{ptr}(i, m) = iM \cdot 2^{2q} + m \cdot 2^{2q}. \tag{6}$$

So far, design of receiver subsystems with the proposed Information Bottleneck design approach has only been described for applications in LDPC decoding in this paper. However, the method can also be applied to other receiver sided signal processing such as channel estimation and detection [10], [13]. The applied principle is identical. Information on a relevant quantity X is extracted from an observed random varible Y by a system which is the design target. For this purpose, one determines the joint probability distribution of this relevant quantity and the observation $p(x, y)$ and feeds it to an Information Bottleneck algorithm. The choice of the relevant quantity is made by the system designer. The algorithm then delivers the desired relevant information preserving mapping $p(t|y)$ describing the respective receiver component as a LUT. The output cardinality of the Information Bottleneck algorithm determines the bit width required to store the system output. Adjusting this parameter allows to easily trade preservation of relevant information for a more compact representation of the system output and thus a smaller bit width in the hardware. In [10], [13] we have presented complete signal processing chains performing channel estimation, detection and decoding by processing only unsigned integers with the proposed method. Their performance is very close to double precision receivers.

4. DSP Implementation Aspects of Information Bottleneck LDPC Decoders

In this section we discuss several important implementation aspects of Information Bottleneck LDPC decoders. Our focus lies on DSP implementations, where the decoder is developed in software with the $C++$ programming language. Therefore, our results are especially interesting for software defined radio (SDR) applications [14]. Some of the discussed implementation aspects focus on $(d_v, d_c) = (3, 6)$-regular codes because we investigate two such codes in Sect. 5. However, the design principle of Information Bottleneck LDPC decoders is extendable also to other node degrees and also for irregular LDPC codes [8], [15].

4.1 Lookup Table Implementation

In the following we present two distinct options for the implementation of LUTs on a DSP platform. We compare the performance of both approaches in the practical evaluation in the following section.

4.1.1 Lookup Table Reduction

Information Bottleneck LDPC decoders process q bit unsigned integer messages. Preliminary results [6], [9] show that $q = 4$ bit messages are sufficient to provide near optimum performance for quantized output additive white Gaussian noise (AWGN) channels, so we will focus on $q = 4$ from now on. Most DSPs natively support data types consisting of several bytes (8 bits). From a programming perspective, it is completely natural to store each **LUT** vector in an array. An array is a connected memory region consisting of multiple cells that all hold data of the same data type. Access to an array is done using indexing. The smallest standard data type in $C++$ which is suitable to store a $q = 4$ bit message is a *uint8* integer, where *uint* abbreviates *unsigned integer* and the appended 8 denotes the wordlength of 8 bits. Reserving a *uint8* memory cell for each LUT entry, however, wastes the upper half word of the cell because when storing a number from $\{0, 1, \ldots, 2^4 - 1\}$ in this cell, the upper halfword is always 0000 in the binary representation. A very simple idea to reduce the memory footprint of the decoder is to use the upper halfword and the lower halfword to store two consecutive LUT entries. This of course halves the size of the LUT arrays in comparison to the described straightforward allocation of a *uint8* array. Accessing an entry in the LUT, however, requires some extra operations. In Eq. (5), the vector **LUT** is accessed at address $\mathrm{addr} = \mathrm{ptr}(i, m) + \mathrm{idx}(u, v)$. If only one table entry is stored in the array holding **LUT**, accessing the array at index addr automatically delivers the desired table entry. If, in contrast, two table entries are held concatenatedly in the memory cell, addr needs to be divided by 2 according to integer division rules first. Array access at $\lfloor \mathrm{addr}/2 \rfloor$ then fetches 8 bits, that is, *two consecutive* table entries in one byte from the memory. Thus, depending on whether addr is odd or even, the upper or the lower halfword needs to be extracted by an additional bitshift operation. Many DSPs offer very fast *intrinsic* operations for the required bit field extraction. Finally, we note that the principle of storing multiple $q = 4$ bit integers in longer standard data types is not limited to LUT implementation, but can also be applied for the exchanged messages. Consider a degree $d_v = 3$ variable node which has to process a channel message and $d_v = 3$ incoming messages from its connected

check nodes for its bit decision. With $q = 4$ bit messages, a single *uint16* integer occupying only two bytes is enough to store these four messages.

4.1.2 Lookup Table Expansion

In the degree $d_c = 6$ check node operation shown in Fig. 2, intermediate results t_m are processed as inputs of following LUTs. For example, $f_1^{(i)}(t_0, t_1)$ processes t_0 and t_1 both obtained from table $f_0^{(i)}(u, v)$. After t_0 and t_1 have been read from the memory, the next step is address calculation using t_0 and t_1 to determine the result of $f_1^{(i)}(t_0, t_1)$. Expanding the address of the next accessed array entry according to Eq. (5) for the following table with subscript $m = 1$ gives an interesting insight. This address is

$$\text{addr} = \underbrace{iM \cdot 2^{2q} + (0+1)2^{2q} + t_0 \cdot 2^q}_{\text{Integer } t_0' \text{ only depends on } i,\, m=1,\, t_0} + t_1. \quad (7)$$

The braced part in Eq. (7) only depends on the current iteration index i, the index of the respective two-input table $m = 1$ and the first input message t_0. Therefore, instead of storing t_0 as it is in vector **LUT**, one could also store the complete braced part of (7) at the address of t_0. We denote the braced part in Eq. (7) t_0'. If t_0' is read from the LUT memory, the next calculated address can simply be obtained by a single integer addition as addr $= t_0' + t_1$. This greatly reduces complexity in comparison to storing t_0 in **LUT** and evaluating (5). Please note that t_0 and t_1 are obtained using the same LUT with index $m = 0$. Therefore, also a modified t_1' instead of t_1 is read if $f_0^{(i)}(u, v)$ is implemented this way, but $t_1 \neq t_1'$ is required to determine the next address according to Eq. (7). However, it is simple to reobtain t_1 from t_1' as

$$t_1 = (\underbrace{(iM \cdot 2^{2q} + (0+1)2^{2q} + t_1 \cdot 2^q}_{t_1' \text{ read from the LUT}})/2^q) \bmod 2^q. \quad (8)$$

Almost all concatenated LUTs stored in **LUT** can be modified using the described technique. Only the last concatenated table in a two-input decomposition which has index $m = M-1$ provides exchanged messages $t \in \{0, 1, \ldots, 2^q - 1\}$. Therefore, these tables stay unchanged for all iterations i. The t_m' of course are no longer 4 bit integers because they include offsets for the iteration and the table index m. Therefore, a larger base data type is required to store the **LUT** vectors in arrays. For the parameters used in this paper, the smallest integer data type large enough to handle all t_m' that appear was a *uint16*. In comparison to the LUT reduction technique described in Sect. 4.1.1, this larger base data type increases the size of the LUT arrays in the memory by a factor of four. However, as all messages output by a node are still $q = 4$ bit integers, it does not affect the bit width of the messages.

4.2 Reusing Intermediate Results

Each variable and each check node has to generate outgoing messages for all its connected edges during the iterative decoding process. So far, we have discussed message generation using two-input LUTs for one particular target edge, as it is depicted in Fig. 2. Taking into account message generation for all involved edges, many operations can be saved by reusing intermediate results t_m (or t_m' respectively, if LUT expansion is applied). Figure 3 illustrates a possibility to calculate all $d_c = 6$ outgoing messages of a check node using only twelve two-input operations in $M = 3$ distinct two-input LUTs. If no intermediate results were reused, message generation for all d_c outgoing edges would require to process $d_c - 1$ messages for each edge and result in evaluation of $d_c(d_c - 2)$ two-input operations which is 24 in this example. The decomposition in Fig. 3 includes the one already depicted in Fig. 2 as it is highlighted using dashed edges. Finding optimum two-input decompositions with a small number of distinct required LUTs and additionally a maximum reuse potential of intermediate results is a very complex problem. The presented decomposition is the best we found for a degree $d_c = 6$ check node operation.

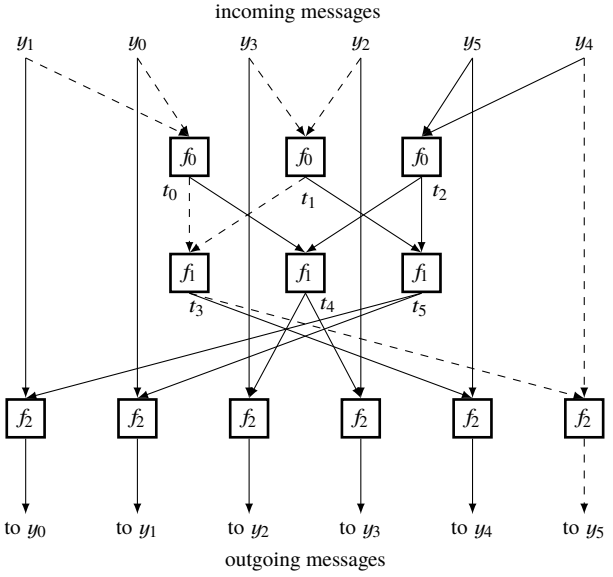

Fig. 3 Degree $d_c = 6$ check node decomposition which utilizes reuse of intermediate results to reduce the number of operations.

We note that the shown decomposition is not limited to Information Bottleneck LDPC decoders, but can also be applied in all state-of-the-art LDPC decoding algorithms. There, the LUTs shown in Fig. 3 correspond to the box-plus operation (3) or the min-sum approximation. Since these decoders also profit from performing fewer operations, we apply the same decomposition there.

5. Practical Results

We have implemented a fixed point precision belief propagation decoder which evaluates Eq. (3) using a small LUT for $f_c(x)$ at the check nodes, as it has been described in Sect. 2.2, and also a min-sum LDPC decoder on a Texas Instruments TMS320C6474 fixed point DSP. In order to not

disadvantage the conventional decoders, we have optimized the fixed point number format used in these decoders and found that 8 bit messages with 4 bits assigned to the fractional part offered best performance and throughput on this device. Higher precision, e.g. using 16 bit fixed point arithmetic, did not improve the decoder performance, but slowed down the decoding process significantly. Moreover, we have implemented Information Bottleneck decoders with either LUT expansion or LUT reduction on the same device and measured the bit error rates over E_b/N_0 and the net decoding throughputs of the different decoders. Finally, we performed an analysis of the memory footprints of the decoders. We have chosen two $(3, 6)$-regular LDPC codes from [16] for performance evaluation. The identifiers of the codes are a) *Margulis.2640.1320.3* and b) *8000.4000.3.483*. The first number in the identifier is the codeword length. All LDPC decoders performed a parity-check in each iteration and stopped decoding once all parity-checks were satisfied or after $i_{\max} = 50$ iterations.

5.1 Memory Demand

The memory demands of the distinct decoders are depicted in Table 1. At the top we list the memory amounts needed to store the Tanner graph structures for code a) and code b). These memory amounts are identical for all decoders. In the center part of Table 1, the memory required for LUTs in the distinct decoders is provided. The min-sum decoder does not use a LUT at all. For the fixed point belief propagation decoder, we found that in the applied 8 bit fixed point format a LUT with only 22 entries was sufficient to achieve the best performance of this decoder, therefore resulting in a LUT size of only 22 bytes. The Information Bottleneck decoders naturally need more space to store LUTs for the two-input operations in each iteration. The Information Bottleneck decoder with LUT expansion requires the largest amount of 153.6 kilobytes for the LUTs. If LUT memory is rare, LUT reduction from Sect. 4.1 allows to significantly reduce this memory amount by a factor of four to 38.4 kilobytes. Handling LUTs of the listed sizes was not a problem on the target DSP. We emphasize that LDPC codes with higher node degrees will require more memory for the LUTs in the Information Bottleneck decoders. However, this memory demand scales linearly with the number M of distinct two-input LUTs applied in the respective node operations and hence can be expected to stay manageable also for higher node degrees. The bottom of Table 1 finally compares the memory amount needed to store the exchanged messages. As it is observable, the Information Bottleneck decoders here can profit from smaller $q = 4$ bit representation of the exchanged messages. In summary, Table 1 allows for the conclusion that the memory demand required to implement the Information Bottleneck LDPC decoders on a DSP is a little larger than the one for conventional decoders, but still moderate and available on typical DSP platforms.

Table 1 Comparison of memory demand of distinct decoders.

Decoder	code	Memory for graph structure
all	code a)	15.84 kilobytes
all	code b)	48.0 kilobytes

Decoder	Memory for LUTs
min-sum decoder	0 kilobyte
belief propagation decoder	0.022 kilobyte
Information Bottleneck with LUT reduction	38.4 kilobytes
Information Bottleneck with LUT expansion	153.6 kilobytes

Code	Decoder	Memory for messages
a)	min-sum decoder	10.56 kilobytes
a)	belief propagation decoder	10.56 kilobytes
a)	Information Bottleneck	5.28 kilobytes
b)	min-sum decoder	32.0 kilobytes
b)	belief propagation decoder	32.0 kilobytes
b)	Information Bottleneck	16.0 kilobytes

notation: 1.0 kilobyte = 1000 bytes

5.2 Bit Error Rate Performance

Figure 4 shows the bit error rate performances of all applied decoder types over E_b/N_0 for a transmission over a quantized output AWGN channel with binary phase shift keying (BPSK) modulation. The solid curves refer to code a). The ones for the longer code b) with better error correction capability use dashed lines. For all decoders the channel output was quantized using a 4 bit quantizer which was designed as explained in [9]. For the conventional decoders, LLRs were calculated at the output of the quantizer in fixed point precision. The Information Bottleneck decoder just processed the plain 4 bit quantization indices from the quantizer. The Information Bottleneck decoder clearly outperforms the min-sum decoder by about 0.4 dB in the waterfall region of both codes. At the same time, it comes very close to the performance of the belief propagation decoder. The loss with respect to this decoder is typically smaller than 0.1 dB over E_b/N_0 in the waterfall region of both codes. Therefore, one can summarize that the Information Bottleneck decoder which uses $q = 4$ bit to represent the exchanged messages

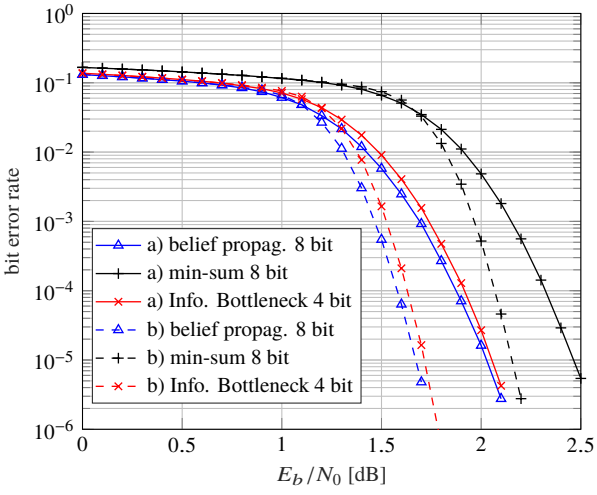

Fig. 4 Bit error rate for LDPC code a) (solid curves) and LDPC code b) (dashed curves) for several decoders implemented on the DSP.

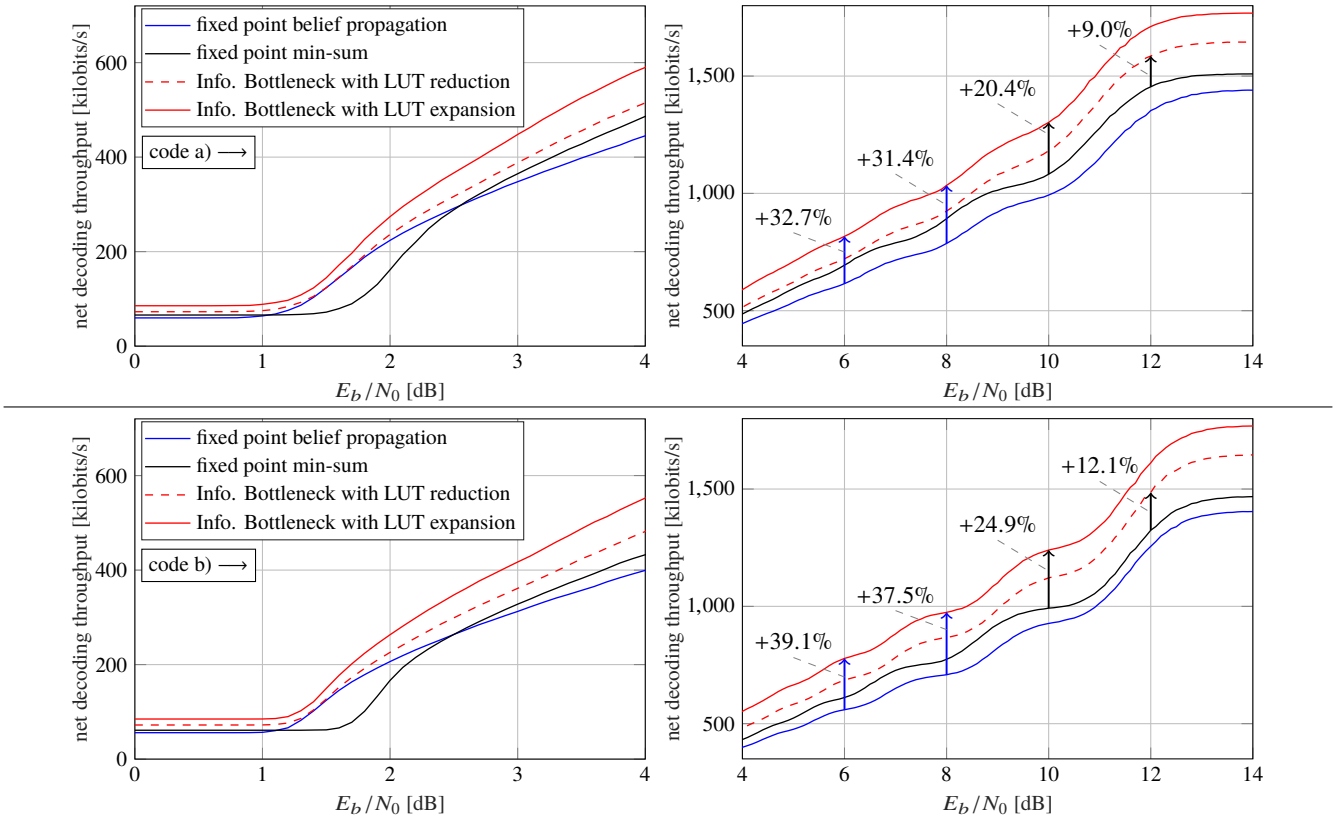

Fig. 5 Net decoding throughputs of distinct LDPC decoders on the TMS320C6474 DSP for code a) (top) and code b) (bottom). Information Bottleneck decoders use $q = 4$ bit messages. The conventional decoders use 8 bit fixed point arithmetic with 4 bits allocated to the fractional part.

offers much better bit error rate performance than the fixed point min-sum decoder with 8 bit messages. Moreover, it comes very close to a fixed point belief propagation decoder with 8 bit messages.

5.3 Net Decoding Throughput

We have measured the average number of decoded information bits per second for $E_b/N_0 \in [0, 14]$ dB to determine the net decoding throughputs of the distinct decoders. The DSP was clocked with a main frequency of 1 GHz in this experiment. Figure 5 shows the obtained results. The upper part corresponds to code a) and the lower one corresponds to code b). We have split up the E_b/N_0 axis from $E_b/N_0 = 0$ dB to $E_b/N_0 = 4$ dB (left) and from $E_b/N_0 = 4$ dB to $E_b/N_0 = 14$ dB (right) for both codes. This was done to enlarge the curves in the low E_b/N_0 regime where they show some interesting behaviour. As it is evident, the Information Bottleneck decoder with LUT expansion offers the highest net decoding throughput of all implemented decoders. This is true for the complete range of E_b/N_0 and both codes and is the most important result of this work. The Information Bottleneck decoder with LUT reduction offers the second highest decoding throughput. Interestingly, in a small part of the low E_b/N_0 regime from approximately 1.2 dB to 1.6 dB shown in the left subplots in Fig. 5, the fixed point belief propagation decoder catches up a little to the Information Bottleneck decoders. We found this to be caused by a slightly smaller number of iterations performed by the conventional decoder on average. This can be explained by the slightly better error correction performance of the fixed point belief propagation decoder in this E_b/N_0 range (cf. Fig. 4). Anyway, following the curves further to the right into the right subplots reveals that the net decoding throughput of the Information Bottleneck decoders significantly exceeds the throughputs of the fixed point belief propagation decoder and the min-sum decoder on the DSP. This is especially important when recalling that the bit error rate performance of the Information Bottleneck decoder is much better than the one of the min-sum decoder and can compete with the performance of the fixed point belief propagation decoder. To quantify the throughput gains of the proposed Information Bottleneck decoders, we have added them in percent values to the curves in the right part of Fig. 5. Arrows between the curves mark the respective gains between the distinct decoders for exemplary E_b/N_0.

6. Conclusion

We have presented and investigated an Information Bottleneck approach to design receiver components with low quantization resolution. The focus of this investigation lay

on the practical development and evaluation of novel LDPC decoders in a DSP implementation. The proposed method yields decoders which completely work on unsigned integers and implement all signal processing using LUTs. The resulting decoders perform better than the min-sum decoder and offer even higher decoding throughput. They moreover outperform a fixed point belief propagation decoder in terms of net decoding throughput while having almost the same bit error correction capability. The Information Bottleneck design method can be extended to other receiver sided signal processing problems which require near optimum performance with low quantization bit width. Therefore, it should be paid attention to it in design of future communication receivers with low quantization resolution for 5G.

References

[1] J. Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier, and X.Y. Hu, "Reduced-complexity decoding of LDPC codes," IEEE Trans. Commun., vol.53, no.8, pp.1288–1299, Aug. 2005.
[2] Z. Zhang, L. Dolecek, M. Wainwright, V. Anantharam, and B. Nikolic, "Quantization effects in low-density parity-check decoders," Proc. IEEE Intern. Conf. on Commun., pp.6231–6237, June 2007.
[3] N. Tishby, F.C. Pereira, and W. Bialek, "The information bottleneck method," Proc. 37th Allerton Conf. on Commun. and Comp., pp.368–377, Monticello, USA, Sept. 1999.
[4] N. Slonim, R. Somervile, N. Tishby, and O. Lahav, "Objective classification of galaxy spectra using the information bottleneck method," Monthly Notices Roy. Astron. Soc., vol.323, no.2, pp.270–284, May 2001.
[5] B.M. Kurkoski, K. Yamaguchi, and K. Kobayashi, "Noise thresholds for discrete LDPC decoding mappings," Proc. IEEE Global Telecommun. Conf., pp.1–5, New Orleans, USA, Dec. 2008.
[6] F.J. Cuadros Romero and B.M. Kurkoski, "LDPC decoding mappings that maximize mutual information," IEEE J. Sel. Areas Commun., vol.34, no.9, pp.2391–2401, Sept. 2016.
[7] 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," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.26, no.2, pp.329–340, Feb. 2018.
[8] M. Meidlinger and G. Matz, "On irregular LDPC codes with quantized message passing decoding," Proc. IEEE 18th Intern. Workshop on Sig. Proc. Advan. in Wireless Commun., pp.1–5, Sapporo, Japan, July 2017.
[9] J. Lewandowsky and G. Bauch, "Information-optimum LDPC decoders based on the information bottleneck method," IEEE Access, vol.6, pp.4054–4071, Jan. 2018.
[10] J. Lewandowsky, M. Stark, and G. Bauch, "Information bottleneck graphs for receiver design," Proc. IEEE Intern. Symp. on Inf. Theory, pp.2888–2892, Barcelona, Spain, July 2016.
[11] R.E. Blahut, "Computation of channel capacity and rate-distortion functions," IEEE Trans. Inf. Theory, vol.18, no.4, pp.460–473, July 1972.
[12] S. Hassanpour, D. Wübben, A. Dekorsy, and B.M. Kurkoski, "On the relation between the asymptotic performance of different algorithms for information bottleneck framework," Proc. IEEE Intern. Conf. on Commun., pp.1–6, Paris, France, May 2017.
[13] G. Bauch, J. Lewandowsky, M. Stark, and P. Oppermann, "Information-optimum discrete signal processing for detection and decoding," Proc. IEEE 87th Vehicular Techn. Conf., Porto, Portugal, pp.1–6, July 2018.
[14] T. Ulversoy, "Software defined radio: challenges and opportunities," IEEE Commun. Surveys Tuts., vol.12, no.4, pp.531–555, May 2010.
[15] M. Stark, J. Lewandowsky, and G. Bauch, "Information-optimum LDPC decoders with message alignment for irregular codes," Proc. IEEE Global Commun. Conf., Dec. 2018.
[16] D.J.C. MacKay, "Encyclopedia of sparse graph codes," http://www.inference.phy.cam.ac.uk/mackay/codes/data.html, accessed Nov. 14th 2017.

Jan Lewandowsky received the M.Sc. degree in electrical engineering from the Universität der Bundeswehr Munich, Germany in 2011. He is currently a researcher with the Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE) in Wachtberg, Germany. He also absolves a Ph.D. program at the Hamburg University of Technology, Hamburg, Germany. His research interests focus on practical applications of information theory.

Gerhard Bauch received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering from Munich University of Technology in 1995 and 2001, respectively and the master in economics degree from Fernuniversität Hagen in 2001. In 1996, he was with the German Aerospace Center (DLR), Germany. From 1996 to 2001, he was a member of scientific staff with Munich University of Technology. In 1998 and 1999 he was with AT & T Laboratory Research, USA. In 2002, he joined DOCOMO Euro-Labs, Munich, Germany. In 2007, he was also appointed Research Fellow with DOCOMO Euro-Labs. From 2003 until 2008, he was an Adjunct Professor with Munich University of Technology. He was a Full Professor with Universität der Bundeswehr Munich from 2009 to 2012. Since 2012, he has been the head of the Institute of Communications with Hamburg University of Technology.

Matthias Tschauner received the Dipl.-Ing. degree in electrical engineering from the RWTH Aachen University in 2011. Since 2011, he is a researcher with the Communication Systems department at the Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE). His research interests include software defined radio technology, wireless communication systems and waveform design. Matters of particular interest are channel coding and modulation schemes.

Peter Oppermann received the B.Sc. degree in general engineering science and the M.Sc. degree in computer science and engineering from the Hamburg University of Technology. After graduation he joined the Fraunhofer Institute for Medical Image Computing (MEVIS), where his current research focus is the development of clinical decision support systems for hospitals.