Flexible High Throughput QC-LDPC Decoder with Perfect Pipeline Conflicts Resolution and Efficient Hardware Utilization

Vladimir L. Petrović, *Graduate Student Member*, *IEEE*, Miloš M. Marković, Dragomir M. El Mezeni, Lazar V. Saranovac, *Member*, *IEEE*, and Andreja Radošević

***Abstract*—Modern communication standards, such as 5G new radio (5G NR), require a high speed decoder for highly irregular quasi-cyclic low density parity check (QC-LDPC) codes. A widely used approach in QC-LDPC decoders is a layered decoding schedule which processes the parity check matrix in parts, thus providing faster convergence. However, pipelined layered decoding architecture suffers from data hazards that reduce the throughput. This paper presents a novel architecture, which can facilitate any QC-LDPC decoding without stall cycles caused by pipeline hazards. The decoder conveniently incorporates both the layered and the flooding schedules in cases when hazards occur. The paper also presents the genetic algorithm based optimization of the decoding schedule for better signal-to-noise ratio (SNR) performance. The proposed architecture enables insertion of a large number of pipeline stages, thus providing high operating frequency. As a case study, the FPGA implementation for WiMAX, DVB-S2X, and 5G NR provided coded throughput of up to 1.77 Gbps, 4.32 Gbps, and 4.92 Gbps at 10 iterations, respectively. The results show a strong throughput increase of 30%–109% compared with the conventional layered decoder for 5G NR for the same SNR performance. The decoder provides highly efficient utilization of resources when compared with the state-of-the-art solutions.**

***Index Terms*—5G new radio, genetic algorithm optimization, high throughput, layered decoding, low density parity check (LDPC) codes, pipeline, quasi cyclic (QC) LDPC**

I. Introduction

Due to their excellent error correcting performance, low density parity check (LDPC) codes [1] are increasingly used in many applications, e.g. in storage devices [2] and in many wired [3] and wireless communication standards [4]–[8].

LDPC code is completely defined by its parity-check matrix (PCM), but can also be represented using the Tanner graph [9]. LDPC code is sparse, i.e. of low density, so both the encoding and the decoding processes can be of low computation complexity. The decoding process is usually based on the iterative message-passing algorithm [10], [11], which can provide very good performance in terms of achievable information rate, making LDPC codes able to closely approach the channel capacity [12].

Traditionally, LDPC codes, whose connections between variable and check nodes are generated randomly, provide the best achievable information rate. However, practical LDPC codes are designed to have some structural constraints in order to provide possibility for parallel processing of multiple nodes in both the encoding and the decoding processes [2]–[8]. Quasi-cyclic (QC) LDPC code [13] has a PCM that is composed of circularly shifted identity sub matrices. This code can be represented using the base graph matrix, and the width of the identity submatrix, frequently called the lifting size (Z) [14]. Base graph matrix contains nonnegative shift values at positions of identity sub matrices, which is convenient for the storage of the code parameters.

In the message-passing algorithm, nodes communicate using messages that are passed along the edges of the Tanner graph. The messages are associated with the probabilities that the corresponding bits are zero or one. Their values are iteratively updated in the graph nodes. In the so-called flooding schedule [15], all variable nodes simultaneously pass their messages to the corresponding check nodes and all check nodes simultaneously pass their messages to the variable nodes. In the layered schedule, the PCM is viewed as a set of horizontal [16] or vertical [17] layers where each layer represents a component code. In a single layered iteration, messages from variable to check nodes and vice versa are passed consecutively for each layer. This way, the probabilities are updated more frequently during a single iteration, thus speeding up the decoding process. This is particularly convenient for the QC-LDPC codes since their PCM is already naturally divided into layers. The row layered decoding (with the PCM divided in horizontal layers) is used more frequently due to more efficient memory utilization and lower computation complexity [18], [19].

The decoding computations can be done serially, but such configuration provides extremely small throughputs, although the required hardware is minimal. Fully parallel decoders are the fastest, but require extremely high amount of hardware resources, caused mainly by routing congestion, especially for long code words [20]. Consequently, the widely accepted approach is using the partially parallel architectures that allow design tradeoffs between the obtained throughput and hardware complexity [21].

High throughput partially parallel LDPC decoding can be achieved mainly in two ways: 1) by increasing the operating

Manuscript received on June 30, 2020; revised August 11, 2020; accepted August 16, 2020.

Vladimir L. Petrović, Dragomir M. El Mezeni and Lazar V. Saranovac are with the University of Belgrade – School of Electrical Engineering, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia (e-mail: petrovicv@etf.bg.ac.rs; elmezeni@etf.bg.ac.rs; laza@etf.bg.ac.rs).

Miloš M. Marković and Andreja Radošević are with the Tannera LLC, Braće Nedića 26, 11111 Belgrade, Serbia (e-mail: milos@tannera.io; andreja@tannera.io).

Digital Object Identifier: 10.1109/TCSI.2020.3018048
Final, published article available on IEEE Xplore: https://ieeexplore.ieee.org/document/9179021

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

clock frequency and 2) by increasing the number of parallel processing units [22]. The operating frequency is increased primarily by pipelining. Although superior in the speed of the convergence, pipelined layered decoding hardware suffers from data dependencies between successive layers, since pipeline stages induce delays in memory write operations. Consequently, additional stall cycles need to be inserted in order to provide pipeline conflict resolution.

Number of stall cycles can be reduced using the offline read/write scheduling based on the PCM reordering techniques [23]–[27]. In general, reordering techniques cannot eliminate all stall cycles, especially for less sparse base graph matrices. In order to increase the sparsity of the base graph matrix, reducing the size of the circulant sub matrices can be performed [24]. Reducing the lifting size reduces the parallelism, thus increasing latency and reducing the throughput of a decoder. However, the necessary hardware resources are reduced and used more efficiently since stall cycles are removed, thus it effectively increases the hardware usage efficiency (HUE) expressed as the throughput divided by used hardware resources. Multiple frame decoding has been presented in [28] and [29] for mitigation of data dependencies, but the latency of such approach is multiplied with the number of frames decoded at the same time [22]. Additionally, this method requires high additional memory cost [30]. In [31], the HUE is increased by shrinking the resources needed for storage of messages and by code specific based optimization of memory access schedule.

The method from [32] requires read operations of the a posteriori probability for both the calculation of variable-to-check messages and for the a posteriori probability update. In order to implement this idea, two separate memory blocks must constantly be used during the decoding which increases the memory cost, thus reducing the HUE. In [33], the a posteriori probability update is postponed whenever the pipeline conflict occurs. The conflicted check node contributions are stored in a separate register bank. They contribute to the a posteriori probabilities only when a new non-conflicted update happens. Such postponing of the a posteriori probability updates can reduce the layered schedule performance significantly if a base graph matrix is dense, i.e. if it has a large percent of nonnegative entries with regard to the total number of entries.

Decoding irregular codes whose PCM does not have the same number of ones in all rows brings additional challenges. Additional stall cycles are needed whenever the successive layers have different check node weights. However, irregular codes can achieve a higher information rate than regular codes for the same signal-to-noise ratio (SNR) [34], which is why they are used more frequently.

Throughput improvement can also be achieved using more parallel processing elements, which would provide processing more than Z nodes at the same time. Parallelism higher than the lifting size can reduce latency and increase throughput, but requires multiple simultaneous memory accesses. This indicates that code specific message memory mapping is necessary to avoid conflicts in the parallel memory accesses [33], [35]–[37]. When it is not possible to avoid all conflicts using the message memory mapping, stall cycles are added. As a result, the throughput cannot be multiplied with the same factor as the hardware utilization, which reduces the HUE.

If additional code structure constraints are allowed, it is possible to design a code which can be decoded using even higher parallelism level. One such code class, called cyclically-coupled QC-LDPC codes [30], has a PCM which is designed in such way that it is possible to instantiate multiple parallel sub decoders, thus providing high throughput. However, there is still no communication standard that supports these codes.

Another issue in highly parallel LDPC decoder implementations is the input data availability at the beginning of the decoding process. In most of the previous designs ([23], [25], [29], [33], [38]–[40]), the input data is stored in a separate memory buffer. The input data is rewritten to the decoding memory at the beginning of the decoding, which requires additional clock cycles. However, it would be better if the input buffer is organized in such a way that it can become the decoding memory for the newly loaded codeword, and that the decoding memory can become the input buffer memory for the next code word. This double buffering does not require additional clock cycles for rewriting the input data [27].

This paper focuses on a highly efficient solution that resolves most of the previously mentioned issues. This is achieved by the following main contributions:

1) Complete removal of stall cycles that come from change of check node weights by proper buffering inside check node processing units.

2) Complete resolution of pipeline conflicts that occur due to memory access hazards without postponing the a posteriori probability update. Whenever a conflict occurs, it is resolved by suitable switching to the flooding schedule. There is no waiting for the previous layer memory update. Such hybrid schedule decoding allows insertion of a large number of pipeline stages, thus leading to the high operating frequency of the decoder.

3) Hybrid schedule optimization, which reduces the number of pipeline conflicts, thus providing better SNR performance, which is almost the same as the SNR performance of the fully layered decoder. Even if the remaining performance loss is compensated with adding additional iterations, the throughput of the proposed decoder is significantly higher than the throughput of a fully layered decoder.

4) Stall cycles are removed in such a way that the proposed architecture can be applied to any QC-LDPC code or some irregular repeat accumulate (IRA) codes. As a case study, decoders for IEEE 802.16 (WiMAX), DVB-S2X and 5G NR standards are implemented.

The rest of the paper is organized as follows: Conventional layered LDPC decoder architecture and decoding algorithms are described in section II. Section III presents the architecture of the proposed hybrid schedule decoder and the algorithm for the offline SNR performance optimization. Results and the discussion are presented in section IV. Detailed SNR performance and throughput analysis is focused on most challenging irregular codes from 5G NR. The implementation and HUE results are given also for WiMAX and DVB-S2X for better comparison with previous works. In the end, the conclusion is given in section V.

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

II. LAYERED QC-LDPC DECODER

A. Message-passing decoding algorithms

This subsection briefly presents the soft iterative decoding algorithms most frequently used in binary LDPC decoders. Message-passing decoding known as Belief Propagation (BP) algorithm ([10], [13]) consists of the following steps: initialization, check node updates and variable node updates. In an additive white Gaussian noise (AWGN) channel, variable nodes v are initialized with the a priori log-likelihood ratios (LLRs) calculated from the channel outputs y_v and channel noise variance σ^2, as

$$LLR_v^{in} = \log\frac{P(b_v = 0 \mid y_v)}{P(b_v = 1 \mid y_v)} = \frac{2y_v}{\sigma^2}, \quad (1)$$

where $P(b_v = b \mid y_v)$ is the conditional probability that the b_v is equal to b, given that the channel output y_v is received. Right after the initialization, a priori LLRs are assigned to messages that variable nodes send to the check nodes along the edges of the Tanner graph – variable-to-check messages ($M_{v2c} = LLR_v^{in}$). Each check node calculates new messages that will be sent to the corresponding variable nodes using the following equation:

$$M_{c2v} = \Phi^{-1}\left(\sum_{v'\in V_c\setminus v}\Phi\left(\left|M_{v'2c}\right|\right)\right)\times\prod_{v'\in V_c\setminus v}\operatorname{sgn}\left(M_{v'2c}\right), \quad (2)$$

where V_c represents the set of all variable nodes connected to the check node c, and where $\Phi(x) = -\log(\tanh(x/2))$. Each variable node now updates associated LLR (also referred to as intrinsic LLR or the soft output for the variable node v) with its a posteriori value, as in (3), and new variable-to-check messages as in (4).

$$LLR_{v,apost}^{it} = LLR_{v,apost}^{it-1} + \sum_{c'\in C_v} M_{c'2v}^{it} \quad (3)$$

$$M_{v2c}^{it+1} = LLR_{v,apost}^{it-1} + \sum_{c'\in C_v\setminus c} M_{c'2v}^{it} \quad (4)$$

In (3) and (4), C_v represents the set of all check nodes connected to the variable node v and it is the index of the current iteration. Note that in the first iteration, old a posteriori LLR value is the a priori channel LLR, i.e. $LLR_{v,apost}^{0} = LLR_v^{in}$, and that $M_{c'2v}^{0} = 0$. Based on (3) and (4), variable-to-check messages can also be calculated as

$$M_{v2c}^{it+1} = LLR_{v,apost}^{it} - M_{c2v}^{it}. \quad (5)$$

Additionally, it is clear from (5) that LLRs and messages satisfy the following equation:

$$LLR_{v,apost}^{it} = M_{v2c}^{it+1} + M_{c2v}^{it}. \quad (6)$$

The described approach of updating variable and check nodes is frequently called a flooding decoding schedule.

As it can be observed in (2), the check node update requires a complex calculation which would be inefficient to implement in hardware. This is why it is often simplified as

$$M_{c2v} = \min_{v'\in V_c\setminus v}\left(\left|M_{v'2c}\right|\right)\times\prod_{v'\in V_c\setminus v}\operatorname{sgn}\left(M_{v'2c}\right), \quad (7)$$

which is called a min-sum approximation [41]. The min-sum approximation drastically reduces the complexity of the check node update calculation, but it yields significant loss in SNR performance of the decoding. This is happening because the magnitude of the check-to-variable message is usually overestimated. This effect can be compensated by correcting its magnitude by multiplication with a normalization factor α, or by subtracting an offset factor β like in (8) and (9). These approximations are known as a normalized min-sum and an offset min-sum approximation, respectively [42].

$$M_{c2v} = \alpha\min_{v'\in V_c\setminus v}\left(\left|M_{v'2c}\right|\right)\times\prod_{v'\in V_c\setminus v}\operatorname{sgn}\left(M_{v'2c}\right) \quad (8)$$

$$M_{c2v} = \max\left(\min_{v'\in V_c\setminus v}\left(\left|M_{v'2c}\right|\right)-\beta, 0\right)\times\prod_{v'\in V_c\setminus v}\operatorname{sgn}\left(M_{v'2c}\right) \quad (9)$$

For quick reference, the summary of notations used throughout the paper and in this section is given in Table 1.

TABLE I
SUMMARY OF FREQUENTLY USED NOTATIONS

Notation	Meaning
N	Codeword length
$\mathbf{H}$	Parity check matrix (PCM) of the LDPC code
Z	Lifting size of the QC-LDPC PCM
v	Variable node index
c	Check node index
V_c	Set of all variable nodes connected to the check node c
C_v	Set of all check nodes connected to the variable node v
y	Channel outputs
σ^2	Noise variance
LLR_v^{in}	Input a priori log-likelihood ratio for a variable node v
M_{v2c}	Message from variable node v to check node c
M_{c2v}	Message from check node c to variable node v
$LLR_{v,apost}$/LLR_v	A posteriori log-likelihood ratio for a variable node v
Superscript it or l	Indicates M_{v2c}, M_{c2v}, or $LLR_{v,apost}$ at iteration it or at layer l
it_{max}	Maximum number of iterations
N_l	Number of layers in the parity check matrix
$\mathbf{x}$	Vector of current codeword bits (defined by signs of a posteriori LLRs)
$\mathbf{s}$	Syndrome vector, calculated by $\mathbf{s} = \mathbf{x} \times \mathbf{H}^T$

B. Layered decoder architecture

As mentioned before, the PCM in the layered decoding schedule can be observed as a set of component codes' PCMs [16]. All component codes share the same variable nodes, meaning that multiple component codes contribute to the update of the same LLRs. In a single component code, i.e. layer, each variable node is connected to only one check node. However, one component code can contain multiple check nodes connected to separate variable nodes, as in QC-LDPC code PCM.

For each layer, variable-to-check messages are calculated based on the previously updated LLR values and check-to-variable messages from the previous iteration. The difference from the flooding schedule (equation (5)) is that LLRs are updated more frequently and that $LLR_{v,apost}^{it}$ value from (5) is replaced with the LLR calculated in one of the previous layers, as in

$$M_{v2c}^{lj} = LLR_v^{li} - M_{c2v}^{it-1}, \quad (10)$$

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

where *lj* is the index of the current layer, *li* is the index of the layer in which the previous update of the variable node *v* has happened, and *it* is the index of the current iteration. Thereby, the variable-to-check message already contains contributions from all check nodes from previous layers. Check-to-variable message is calculated using one of the check node update equations described in subsection II.A ((2), (7), (8) or (9)). The calculated check-to-variable message for the current layer is the check-to-variable message for the current iteration at the same time ($M_{c2v}^{lj} = M_{c2v}^{it}$). The variable-to-check message M_{v2c}^{lj} has been already updated indirectly through the LLR updates in previous layers and is never updated directly in a single layer, since the check node *c* is the only node connected to the variable node *v*. Hence, the LLR update is done using the already calculated variable-to-check message as

$$LLR_v^{lj} = M_{v2c}^{lj} + M_{c2v}^{it}. \tag{11}$$

The iterative process is shown algorithmically in Fig. 1.

Conventional layered decoder architecture is shown in Fig. 2. Variable-to-check messages are calculated in variable node units (VNU) for each layer from the a posteriori LLRs as in (10). These messages are buffered in the M_{v2c} FIFO for later use in the LLR update calculation. To facilitate proper connections between the variable and the check nodes, LLRs are cyclically shifted by the value equal to the corresponding shift of the identity submatrix in the PCM before the variable node calculation. Variable-to-check messages are passed to check node units (CNU) where new check-to-variable messages are calculated using (2), (7), (8) or (9). New check-to-variable messages are used for calculation of the new intrinsic LLRs, which are then cyclically shifted in the opposite direction. LLR memory always contains up-to-date LLRs.

If the arrangement of the LLRs in the LLR memory can be shuffled, it is worth mentioning that the usage of the reverse cyclic shifter is not necessary [39]. In that case, the shift values of the first cyclic shifter should be modified in order to obtain proper connections in the Tanner graph. Additionally, the decoded bits should be reversely shifted at the end of the decoding.

If the min-sum algorithm (or its variations) is used for the decoding, the check node unit calculates the minimum and the subminimum of the magnitudes of the received variable-to-check messages ($\left|M_{v2c}^{li}\right|$). Besides that, the sign product of signs of variable-to-check messages is calculated. When the reading of a single layer LLRs is finished, a minimum, a subminimum, a sign product and an index of the minimum $\left|M_{v2c}^{li}\right|$ is stored in the pipeline register. New check-to-variable message is calculated based on this data and M_{v2c}^{li}. The subminimum value is needed since the influence of the variable node to which the M_{c2v} is sent needs to be removed as in (7), (8) or (9).

If the check node weight (CNW) of two successive layers is different, stall cycles must be generated. If the CNW of the first layer is higher than the CNW of the second layer, the minimums calculation for the second layer needs to be paused

Inputs: parity check matrix **H,** input channel LLRs $\mathbf{LLR}^{\mathbf{in}}$
Initialization:
  for $v = 1 : N$
    $LLR_v = LLR_v^{in}$
    for $c \in C_v$
      $M_{c2v}^{0} = 0$
Decoding:
  while $it \le it_{max}$ and $\mathbf{s} \neq \mathbf{0}$
    for $l = 1 : N_l$
      for $c = 1 : Z$
        for $v \in V_c$
          load LLR_v
          $M_{v2c}^{l} = LLR_v - M_{c2v}^{l,it-1}$
        for $v \in V_c$
          calculate $M_{c2v}^{l,it} = f\left(\left\{M_{v'2c}^{l,it} \mid v' \in V_c \setminus v\right\}\right)$
          update $LLR_v = M_{v2c}^{l} + M_{c2v}^{l,it}$
          store LLR_v
    Soft decision: x_v = signbit(LLR_v), $\forall v \in \{0,1,\ldots, N-1\}$
    Calculate syndrome: $\mathbf{s} = \mathbf{x} \times \mathbf{H}^{\mathrm{T}}$
    $it = it + 1$
Outputs: decoded bits vector **x**

Fig. 1. Layered schedule decoding algorithm. *N* is the codeword length, it_{max} is the maximum number of iterations, N_l is the number of layers in the PCM, and *f*(.) is a check node update function defined by equations (2), (7), (8) or (9).

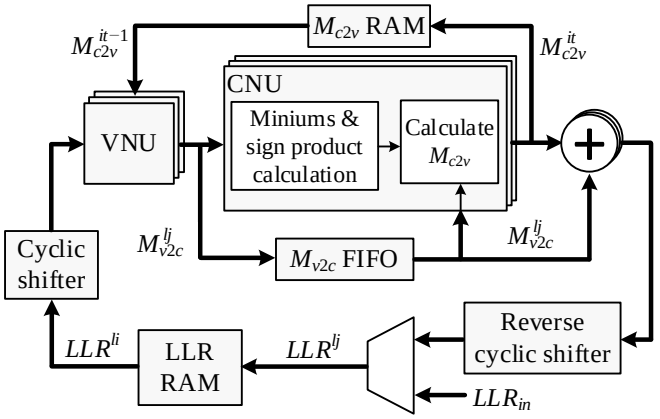

Fig. 2. Conventional layered decoder architecture for min-sum based decoding

since there are check-to-variable messages yet to be generated for the first layer. On the contrary, when all check-to-variable messages are generated, the CNU output needs to wait for minimums calculation block to be finished.

As mentioned before, in order to achieve high operating frequency, it is necessary to place pipeline registers at the data path. The added pipeline latency can cause memory access conflicts. If stall cycles are not added, LLR update can overwrite the contribution of the check nodes from the previous layer [24]. Detailed explanation of stall cycle generation will be given in subsection III.B.

If the number of the processing elements (VNUs and CNUs) is equal to the lifting size *Z*, the coded throughput of the layered decoder can be expressed as follows:

$$T = \frac{f_{CLK} N}{\left(n_{circ} + n_{stall}\right) it_{max} + n_{read}}, \tag{12}$$

where f_{CLK} is the operating frequency, *N* is the codeword length,

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

n_{circ} is the number of circularly shifted identity sub matrices in the PCM, n_{stall} is the number of inserted stall cycles because of the memory access pipeline conflicts or the CNW change, it_{max} is the maximum number of iterations, and n_{read} is the number of cycles needed for the preparation of all input LLRs if the new codeword LLRs are not stored in the LLR memory during the decoding of the previous codeword. The highest throughput is obviously obtained if n_{stall} and n_{read} are zero and if the decoding algorithm has fast convergence, i.e. if it_{max} is small. The highest hardware usage efficiency is obtained if the hardware overhead necessary for the above to be fulfilled is minimal.

III. Hybrid schedule QC-LDPC decoder

A. Hybrid decoding schedule

As it is shown in (11), the intrinsic LLR update in a single layered subiteration *lj* is done by the addition of the variable-to-check message and newly calculated check-to-variable message. The update can be further expanded as

$$\begin{aligned} LLR_v^{lj} &= M_{v2c}^{lj} + M_{c2v}^{it} = LLR_v^{li} - M_{c2v}^{it-1} + M_{c2v}^{it} \\ &= LLR_v^{li} + \Delta M_{c2v}, \end{aligned} \tag{13}$$

where ΔM_{c2v} is the contribution of the check node *c* to the LLR that corresponds to the variable node *v* [32], [38]. If the new LLR value is not written to the LLR memory at the moment when it is needed for calculation of another variable-to-check message $M_{v2c'}$, the contribution ΔM_{c2v} will be lost. For this reason, the layered decoder needs to wait until the LLR is updated.

If two check nodes, one from the layer *lj* and another from the layer *lk*, contribute to the intrinsic LLR at the same time, than the LLR value after both updates is:

$$LLR_v^{lk} = LLR_v^{li} + \Delta M_{c'2v} + \Delta M_{c''2v}, \tag{14}$$

where c' is the check node from the layer *lj* and c'' is the check node from the layer *lk*. This way of updating LLRs can be used to mitigate pipeline conflicts. Namely, if a memory access pipeline conflict occurs, like in a layered schedule, it is not necessary to wait for an LLR update. It is possible to read old LLR values (LLR^{li} in (14)) and add the check node contributions as in (14). However, in this case two layers use the same LLR value for variable-to-check message calculation. Hence, the first layer does not contribute to the LLRs used in the second one, which is characteristic for the flooding decoding schedule.

In [33], if a conflict occurs, the contribution $\Delta M_{c'2v}$ (called residue) was stored in a separate register file and later added to the LLR, which is updated by the second check node as

$$LLR_v^{lk} = \left(M_{v2c''}^{lk} + M_{c''2v}^{it}\right) + \Delta M_{c'2v}. \tag{15}$$

Therefore the update is postponed until another check node passes its message. This way, stall cycles are removed and the method is called residue-based layered decoding. However, if the base graph matrix is dense and if the number of pipeline stages is high, the memory access conflicts can occur frequently and postpone the LLR memory write operations more than once, which can significantly reduce benefits of the layered decoding schedule. Moreover, some LLR updates may never happen if they are postponed for each layer, which is an extreme case. Furthermore, if the number of pipeline stages is high, it is possible that many nodes would need postponed updates, which causes high increase of the necessary register file capacity. For these reasons, it would be important if the LLR write operation is not postponed, but done as soon as the check-to-variable message is ready. At the same time, the contribution of all check nodes must be kept.

In this paper, LLRs are updated as soon as possible with the preservation of all the check node contributions. As before, there is no wait for the LLR update if a memory access conflict occurs. The outdated LLRs are read in this case. When the conflicted LLRs from one of the previous layers are ready, they are written to the LLR memory as in the layered schedule, but they are also buffered and used later in the LLR update process of the current layer. The LLR update process in the current layer for conflicted LLRs is done using the contribution ΔM_{c2v} as

Inputs: parity check matrix **H,** input channel LLRs $\mathbf{LLR^{in}}$
Initialization:
  for $v = 1 : N$
    $LLR_v = LLR_v^{in}$
    for $c \in C_v$
      $M_{c2v}^{0} = 0$
Decoding:
  Thread 1: while $it \le it_{max}$ and $\mathbf{s} \neq \mathbf{0}$
    for $l = 1 : N_l$
      for $c = 1 : Z$
        for $v \in V_c$
          load LLR_v
          $M_{v2c}^{l} = LLR_v - M_{c2v}^{l,it-1}$
    $it = it + 1$
  Thread 2: while $it \le it_{max}$ and $\mathbf{s} \neq \mathbf{0}$
    for $l = 1 : N_l$
      for $c = 1 : Z$
        for $v \in V_c$
          calculate $M_{c2v}^{l,it} = f\left(\left\{M_{v'2c}^{l,it} \mid v' \in V_c \setminus v\right\}\right)$
          UPDATE: **if** *conflictFree* = ***true*** **then**
            $LLR_v = M_{v2c}^{l} + M_{c2v}^{l,it}$
          else
            load LLR_v
            $LLR_v = LLR_v + M_{c2v}^{l,it} - M_{c2v}^{l,it-1}$
          store LLR_v
    Soft decision: x_v = signbit(LLR_v), $\forall v \in \{0,1,\ldots, N-1\}$
    Calculate syndrome: $\mathbf{s} = \mathbf{x} \times \mathbf{H}^{\mathrm{T}}$
    $it = it + 1$
Outputs: decoded bits vector **x**

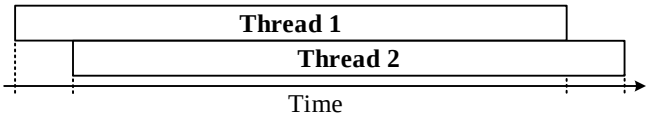

Fig. 3. Hybrid schedule decoding algorithm. The algorithm integrates two processes: calculation of variable-to-check messages in Thread 1 and calculation of new check-to-variable messages and LLR update in Thread 2. Both threads work in parallel. Thread 2 is delayed for simulation of the pipeline latency. The *conflictFree* variable is true if LLR_v used for calculation in Thread 1 was up-to-date.

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

$$LLR^{lk} = LLR^{lj} + \Delta M_{c"2v}. \quad (16)$$

This way, the LLR updates are as frequent as in the layered schedule. Nevertheless, the schedule is still not fully layered, since some LLRs are not always updated with the check node contributions of the previous layer before their usage in processing of one or a few next layers. Therefore, the decoding schedule in this paper is called a hybrid schedule. The algorithmic representation of the hybrid schedule is shown in Fig. 3.

B. Decoder architecture

The detailed architecture of the proposed LDPC decoder core is shown in Fig. 4. Almost all elements are modified compared with the conventional layered architecture.

Firstly, the LLR RAM supports double-buffering as in [27]. It is composed of two separate simple dual-port RAM blocks – one for the decoding of the current codeword (here referred as a decoding memory) and another block, which is used for the reading of the decoded LLRs of the previous codeword and for the writing of the LLRs of the next codeword (here referred as a buffer memory). The previous and the next codeword are in the separate memory spaces. The proposed decoding method supports soft outputs. However, if only hard outputs are needed, the decoded LLRs do not need to stay in the RAM block after the decoding. The better memory utilization is obtained if only signs of LLRs are stored in the separate RAM buffer. Described double buffering is necessary to avoid additional latency at the beginning of the decoding, as outlined in subsection II.B (it reduces the n_{read} parameter from (12) to 0).

Proposed decoder runs without any stall cycle. In conventional layered architecture, the CNU contains a register for the M_{v2c} sign product (*sgp*), minimum (*min*0), subminimum (*min*1) and the index of the minimum (*idx*0) – intermediate data. It is used for storage of the intermediate data from the previous layer while the new layer's minimums are calculated. At the same time, the stored intermediate data is used for calculation of new check-to-variable messages for the previous layer. The timing diagram of the CNU behavior for the example base graph matrix is shown in Fig. 5.a). The base graph matrix has 4 rows (layers in the PCM) and 8 columns. Each column of the base graph matrix represents a set of *Z* variable nodes. These sets are usually called variable node groups (VNGs). Each entry in the base graph matrix represents the PCM's identity submatrix shift value. The timing diagram shows that when the CNW of check nodes inside two consecutive layers is different, either minimums calculation or new check-to-variable messages generation must be stalled. In the worst case scenario the processing must be stalled for $d_{c,max} - d_{c,min}$ clock cycles, where $d_{c,max}$ is the maximum CNW and $d_{c,min}$ is the minimum CNW. In 5G NR, this number is 16, which is significantly high value ($d_{c,max}$ is 19 and the $d_{c,min}$ is 3). Each change of the CNW produces additional stall cycles.

In order to remove described stall cycles, the decoupling of the CNU's input and output must be done. Natural place for the decoupling is after the calculation of intermediate data. For this purpose, a decoupling FIFO buffer is inserted inside the check nodes. The decoupling FIFO buffer prevents overwriting of the intermediate data when layers with the different CNW are processed without interruption. The only requirement for maximal efficiency is that the first layer that is going to be processed should be the one with the maximal CNW. The necessary decoupling FIFO depth depends on the code irregularity. For Wi-Fi, WiMAX or DVB-S2X, the necessary depth is only 2. For more irregular codes the depth needs to be higher, but not higher than $\lceil d_{c,max} / d_{c,min} \rceil$.

Memory access pipeline conflicts are solved as follows. Whenever a reading of LLRs for the variable node group *vg* is needed, it is checked if that VNG is used previously for another layer calculation and is not yet updated. If so, the out-of-date LLRs are read from the LLR memory, but the LLR update for these LLRs will be done differently than in conventional layered architecture. VNUs calculate variable-to-check messages and pass them to the CNUs. However, if the outdated

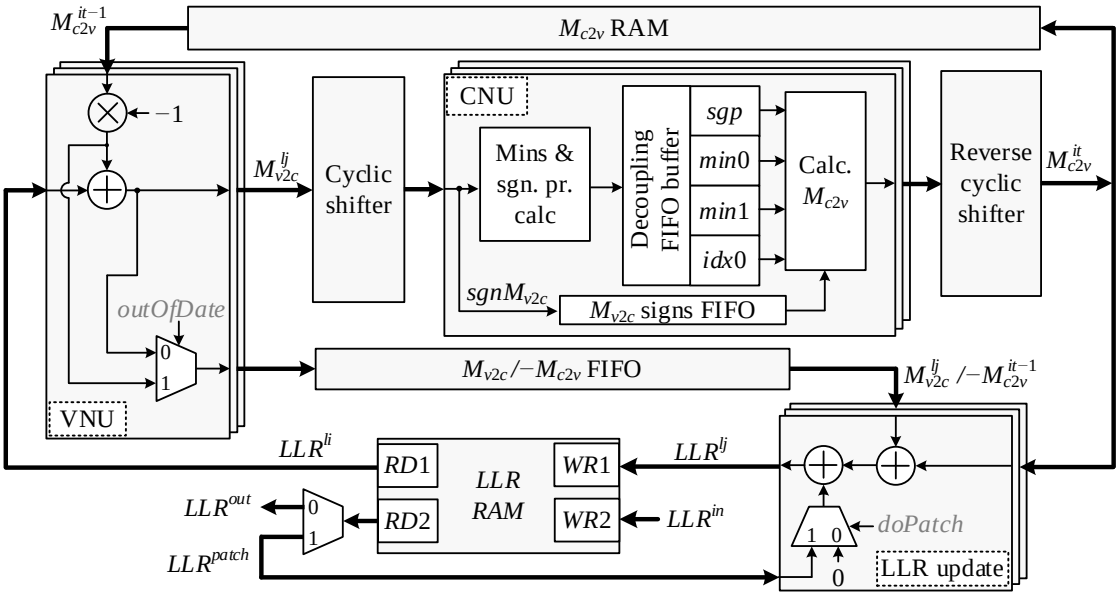

Fig. 4. Architecture of the proposed QC-LDPC decoder core for hybrid schedule decoding

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

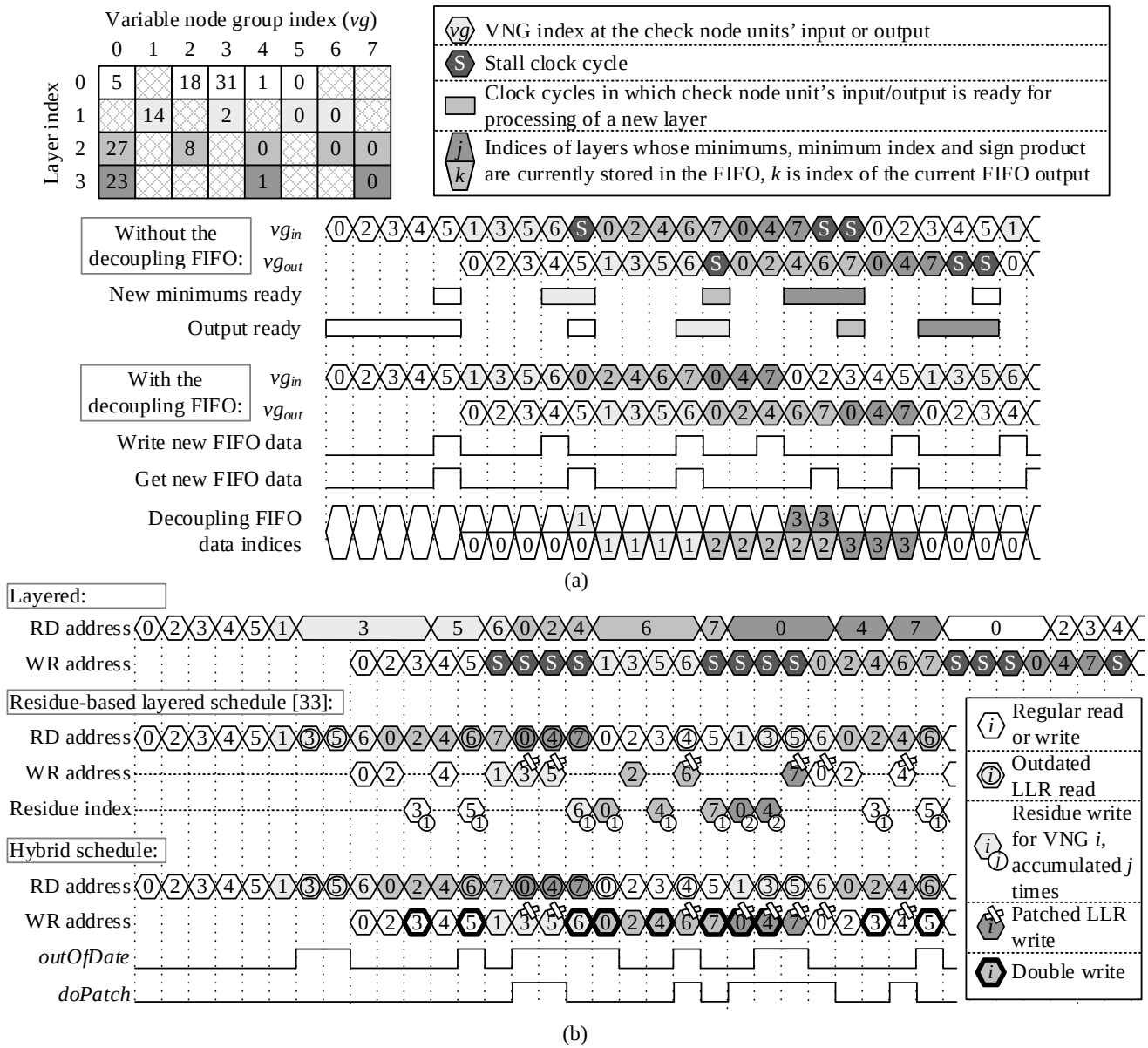

Fig. 5. Illustration of the stall cycles removal for an example irregular QC-LDPC code PCM. (a) Base graph matrix example and timing diagrams of check node unit's behavior without and with the decoupling FIFO buffer. If the decoupling FIFO buffer is not used, the processing must be stalled whenever new minimums are ready, but the check-to-variable messages for previous layer were not all calculated yet, and whenever there are still no calculated minimums for generation of new check-to-variable messages. If the decoupling FIFO is used, these stall cycles are removed. For better understanding, these diagrams do not include memory access pipeline conflicts. (b) Timing diagrams of LLR memory accesses in layered, residue-based layered [33] and hybrid schedule LDPC decoders for the same base graph matrix as in (a) and three pipeline stages. The layered decoder must not read LLRs that should be updated in the previous layer and must wait for their update. In residue-based and hybrid schedule decoding, LLRs are read even though they are outdated. However, in residue-based decoding, LLR updates are postponed in case of conflicts and check node contributions (called residues in [33]) are accumulated in a separate register bank. These contributions are added to LLRs when the first regular LLR write operation happens, here referred as patched LLR write. In the proposed hybrid schedule, when the time for the update of LLRs comes, they are written to both the decoding and the buffer memory (double write), since they are used later for patched LLR update. This way, LLR updates are not postponed and check node contributions are added as soon as they are ready, hence providing faster convergence.

LLRs were read (indicated by the *outOfDate* signal in Fig. 4), VNUs write negative old check-to-variable messages to the FIFO instead of the new variable-to-check messages. They are going to be used later for calculation of the contributions ΔM_{c2v}. Check-to-variable messages are calculated as usual.

The LLR update unit adds a new check-to-variable message and the data from the $M_{v2c}/-M_{c2v}$ FIFO. The result is either new intrinsic LLR or a contribution ΔM_{c2v}. In case that only the ΔM_{c2v} is calculated, it needs to be added to the LLR that is already in the memory – here referred as a patch LLR (LLR^{patch}). This is controlled by the *doPatch* signal in Fig. 4. The read port of the decoding memory is always busy, so the LLR should be read from the buffer memory. Therefore the buffer memory needs to be used sometimes during the decoding not only for the buffering of the previous and the next codeword LLRs.

Whenever the LLR write operation occurs, it is checked if LLRs, which are going to be written, are already read for one of the next layers calculation. If so, the LLRs are written to the buffer memory too. Further in the text, this will be called a

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

double write. Whenever the double write happens, the buffer memory's write port is not available for write operation of the next codeword LLRs. However, there are plenty of free cycles for new codeword LLRs write operation to happen. The similar situation is with the reading of the previous codeword LLRs.

In novel FPGA families, the RAM block size granularity is usually such that a lot of the memory space is not used at all. Even very long codewords leave considerable free space, which can be used for double writes. For example, the storage of the longest codeword in DVB-S2X would require $N/Z = 64800/360 = 180$ locations, which is a significantly smaller number than the minimal RAM block depth.

The hardware overhead for support of the hybrid schedule is small and provides removal of all stall cycles in the decoding.

The timing diagram of LLR memory accesses in conventional layered and proposed hybrid schedule architecture is shown in Fig. 5.b). Additionally, the timing diagram for residue-based schedule approach from [33] is shown too, as it is an approach that removes stall cycles too, but postpones LLR updates, as described in subsection III.A. The example base graph matrix is the same as in Fig. 5.a) and the number of pipeline stages is three. As shown, the layered architecture requires insertion of a large number of stall cycles, whereas in the residue-based and the proposed architecture there are no stall cycles at all. The proposed architecture provides LLR updates without postponing and hence gives faster convergence than the residue-based architecture.

It should be noted that cyclic shifters shift messages instead of LLRs, which provides resource savings, since message bit widths are usually smaller than LLR bit widths [33]. This is of crucial importance if flexible cyclic shifters should be designed, such as in WiFi, WiMAX or 5G NR, since their resource utilization is much higher than the resource utilization of the fixed lifting size cyclic shifter.

The input and output interface modules are not shown in Fig. 4. In a real-time system, input LLRs are usually streamed using a streaming interface in groups determined by an analog-to-digital converter (ADC), usually much smaller than Z LLRs (e.g. up to eight in Xilinx's RF-SoC platform [43]). That is why the input interface module is designed to pack input LLRs into groups of Z LLRs and write them to the buffer memory whenever the entire block of data is ready and when the buffer memory is available. If an input LLR that belongs to the next group of Z LLRs is received at the input, while waiting for the buffer memory availability, it is buffered inside the input module. The output interface module unpacks LLRs (or hard outputs) into the convenient bit width. Both input and output interfaces have the streaming handshake control and make the decoder easy to integrate in another system.

In many applications lifting sizes Z can significantly differ in runtime [4], [5], [8]. This sets a flexibility challenge in the design of the decoder. The main challenge in providing such runtime flexibility is in the design of cyclic shifters [44]. In this paper's implementation examples, three shifters were designed.

The cyclic shifter for DVB-S2X does not need any flexibility, since the lifting size is $Z = 360$. It is designed as a three-stage shifter, where each stage shifts data incrementally by multiples of 45, 8 and 1 respectively.

Flexible shifters for WiMAX and 5G NR are designed as two stage shifters as in [45], where the first stage is a pre-rotator and the second is a flexible QSN shifter [46]. The 5G NR lifting size can take values of the form $Z = a \cdot 2^j$, where $a \in \{2,3,5,7,9,11,13,15\}$ and $0 \le j \le 7$. Since j can take values as small as 0, the pre-rotator needs to have outputs for multiple rotation sizes. This is done as in [47].

C. SNR performance optimization

Hybrid decoding schedule can degrade the SNR performance when the switch from layered to the flooding schedule is frequent. This happens in cases when the base graph matrix is dense and when the number of pipeline stages is high. High operating frequency can be achieved only with a high number of pipeline stages, so the SNR performance loss is inevitable for QC-LDPC codes with a dense base graph matrix.

One way to avoid the performance loss is adding extra iterations. Additional iterations can drastically reduce throughput, but if aggressive pipelining provides a high increase in the operating frequency, there may be enough time margin to add extra iterations and still obtain significant throughput enhancement.

However, adding additional iterations is not necessary if the number of out-of-date LLR read operations is reduced using offline PCM reordering. This subsection describes a method for achieving optimal reordering based on the genetic algorithm, i.e. for enhancement of the hybrid schedule decoding.

Layers can be processed in any order, as well as the VNGs inside a single layer. The only constraint is that, in the end, the layer schedule should be rotated circularly in such a way that the first layer is the one with the maximal check node weight, as outlined in subsection III.B. The processing schedule can be represented as an array of N_l vectors. Each vector represents a single layer. These vectors' entries are VNG indices, i.e. addresses of corresponding LLRs inside LLR memory. The graphical representation of the original processing schedule for the same example of the base graph matrix from Fig. 5 is shown in Fig. 6.a). Since layers and VNGs inside layers can be processed in any order, it is possible to find a schedule that would give the minimal number of out-of-date LLR read operations and hence the best SNR performance of the decoder.

A random schedule can have any permutation of layer indices and any permutation of VNG indices inside any layer. Finding the optimal schedule belongs to the traveling salesman problem class, which is convenient for optimization using a genetic algorithm (GA) [48]. The genetic algorithm has been used for the layer reordering inside the PCM for a minimal number of stall cycles in the layered LDPC decoder [24]. However, in hybrid schedule decoding, only layer reordering cannot significantly reduce the number of outdated LLR updates, especially if a number of inserted pipeline stages is large. Consequently, the GA recombination and mutation processes are improved to include the ordering of VNG processing inside a single layer. The optimization procedure is as follows.

The cost in the optimization procedure is the number of outdated LLR read operations, or equivalently the number of

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

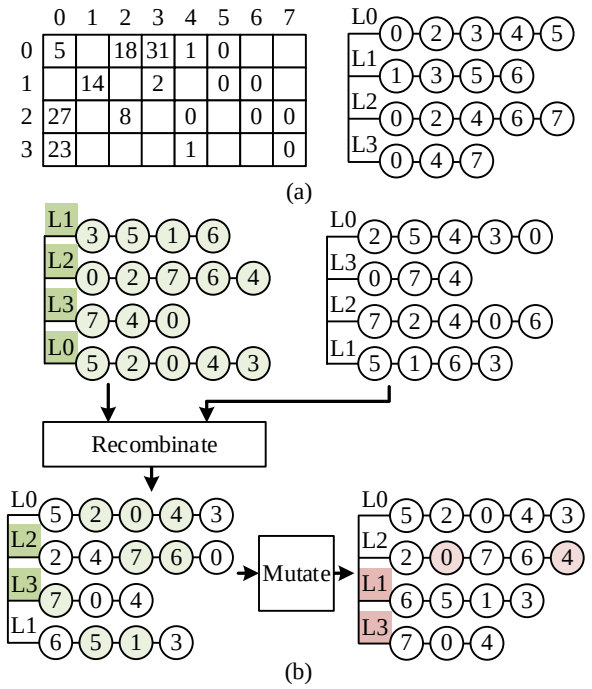

Fig. 6. (a) The base graph matrix example and its corresponding original processing schedule. (b) The example of the recombination and the mutation of schedules during the genetic algorithm optimization. Parent schedules are a permutation of both layer and VNG indices of the original schedule.

double writes. It is a function of the processing schedule and a number of inserted pipeline stages.

Initially, the population of random schedules is generated. Iteratively, in each generation, the population is changed after a recombination and a mutation. The recombination is done on a number of parent schedules with the best cost for both layer permutation and for each VNG permutation.

Recombination of two schedules is done by multiple two vector recombinations: 1) recombination of layer arrays and 2) recombinations of all vectors that represent a VNG schedule inside layers. The array of layers is seen as a vector of layer indices (e.g. (0, 1, 2, 3)).

Two vectors are recombined using the following procedure. A sub vector is cut from the first vector at random positions and placed to the same positions in the child vector. The remaining positions are filled with the entries from a second parent vector that are not already in the first vector while taking care to maintain the order of the entries from the second vector. Firstly, layer arrays are recombined. After that, each layer's VNG indices vector is matched with its corresponding VNG indices vector from another schedule. Every matched vector pair is recombined using the described procedure for vector recombination. The example for base graph matrix from Fig. 6.a) is shown in Fig. 6.b).

The mutation is done by changing places of random entries in the schedule.

IV. Results and Discussion

A. Schedule optimization results

Schedule optimization moves the decoder behavior towards fully layered. Fig. 7 shows the number of up-to-date LLR read operations for codes in 5G NR for different numbers of pipeline stages n_{PS}. All code rates for the base graph 1 were optimized for layered behavior as described in subsection III.C. The improvement in obtained number of up-to-date LLR read operations with respect to the original schedule is expressed in percents as

$$I = \frac{n_{up\text{-}to\text{-}date,optimized} - n_{up\text{-}to\text{-}date,original}}{n_{up\text{-}to\text{-}date,original}} \cdot 100\%, \quad (17)$$

where $n_{up\text{-}to\text{-}date}$ is the number of up-to-date LLR read operations in one decoder iteration. As presented in Fig. 7, the obtained improvement is between 25% and 120%, depending on the code rate and number of pipeline stages.

Fig. 8 shows the percentage of up-to-date LLR read operations depending on the number of pipeline stages for one high code rate (R = 22/27) and one low code rate (R = 22/68) from the 5G NR. Note that the results correspond to the PCM with any lifting size, since the optimization procedure is done based on the base graph matrix. In the 5G NR, low code rates' base graph matrices are sparser than high code rates' matrices. Both the Fig. 7 and Fig. 8 show that the obtained improvement is much higher at higher code rates. This is expected, since sparser matrices cause less pipeline conflicts. Moreover, it can be noticed from Fig. 8 that inserting pipeline stages induces

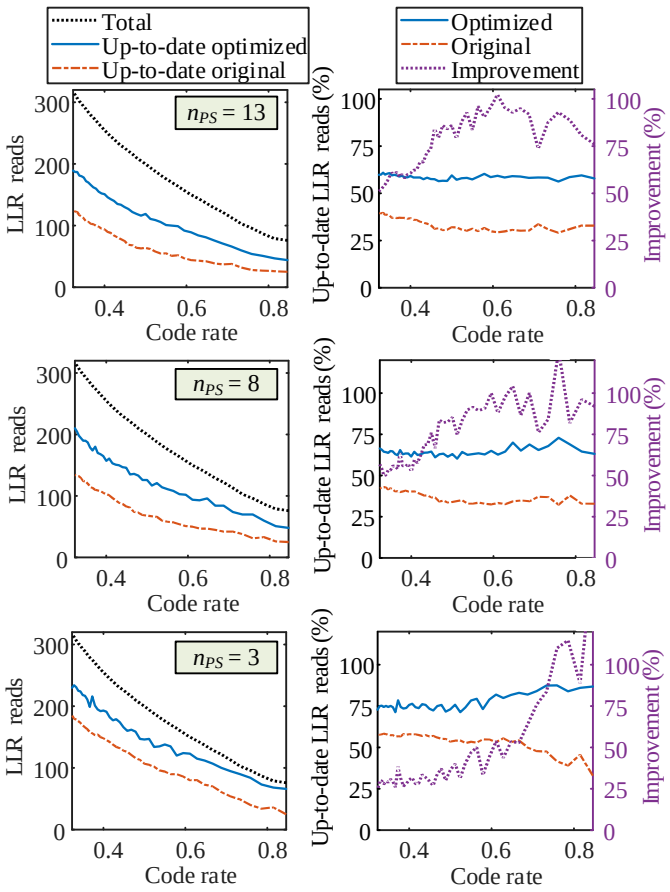

Fig. 7. Number of up-to-date LLR read operations in a single iteration in hybrid schedule for various numbers of pipeline stages n_{PS} and for all code rates from base graph 1 of 5G NR.

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

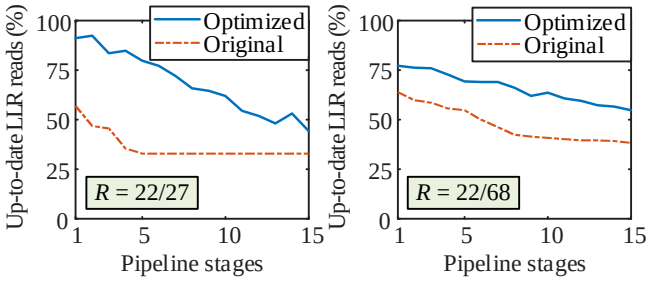

Fig. 8. Percentage of up-to-date LLR read operations as a function of number of pipeline stages for rate 22/27 and rate 22/68 codes from 5G NR.

more significant decrease of up-to-date LLR read operations for high code rates than for low code rates. Therefore, it is expected that the SNR performance for high code rates should be slightly more reduced than for low code rates, which will be discussed in the next subsection.

B. SNR performance results

In order to show the influence of the hybrid decoding schedule on the SNR performance, multiple Monte Carlo simulations were done. The decoder was implemented as a fixed-point offset min-sum decoder with LLRs quantized to 8 bits and messages quantized to 6 bits. All intermediate results were quantized to the minimum number of bits that prevented overflows, whereas only LLRs and messages were saturated after the calculation. The modulation, noise generation and demodulation were done in floating point precision, whereas input LLRs were rounded before decoding. The decoder supported layered, hybrid and flooding decoding schedules. For hybrid schedule, both the original and optimized schedules were simulated. Optimized schedule is here called an enhanced hybrid schedule, since it achieves better SNR performance than the original hybrid schedule. The implemented hybrid schedule simulation model is a bit-accurate model of the hardware implementation and provides the same SNR performance as the measurements at the physical hardware.

The simulation was performed for three different maximum iteration numbers (it_{max} = 10, it_{max} = 20, and it_{max} = 30). The number of pipeline stages was set to 13, since it provided high operating clock frequency in the hardware implementation of the hybrid schedule decoder, as will be seen in subsection IV.C.

Frame error rate (FER) curves for code (10368, 8448) and code (26112, 8448) from 5G NR (base graph 1 codes with code rate 22/27 and code rate 22/68 and lifting size Z = 384) in AWGN channel and for QPSK modulation are shown in Fig. 9. The enhanced hybrid schedule gives better results than the original hybrid schedule, especially at low code rate and small maximum iteration number. An SNR performance loss can be noticed for hybrid decoding schedule with respect to the layered schedule in all cases. However, for the enhanced hybrid schedule, the significant loss (above 0.2 dB) is seen only for high code rate at small maximum number of iterations. If a higher maximum iteration number is allowed or if the lower code rate is used, the loss becomes lower than 0.1 dB. This behavior is not unexpected, since the flooding schedule loss is much higher at a smaller number of iterations and the hybrid schedule is in between the flooding and the layered schedules.

As an additional analysis, Fig. 10 presents the average number of iterations necessary for successful decoding in enhanced hybrid schedule and layered decoders for several codes from 5G NR. It shows once again that higher code rates induce higher gap between performances of the two decoders.

As mentioned before, the remaining SNR performance loss in enhanced hybrid schedule decoding can be removed if additional iterations are added. Fig. 11.a) shows necessary numbers of additional iterations needed for the same or better SNR performance than the SNR performance of the layered decoder for the same codes as in Fig. 10. It is noticeable that more additional iterations should be added at high code rates than at low code rates, which agrees with the higher loss at high code rates if the same number of iterations is used.

Adding the additional iteration reduces throughput of the hybrid schedule decoder. However, the layered decoder needs more clock cycles for single decoding iteration, since stall cycles should be added in order to mitigate pipeline conflicts.

Since all pipeline conflicts are removed, the number of cycles necessary for a single iteration in a hybrid schedule decoder is equal to the number of entries in the base graph matrix. The number of stall cycles in a layered decoder depends on the PCM and number of pipeline stages. In order to make a fair comparison of layered and hybrid schedule decoders, the number of cycles for the layered decoder is calculated for the optimized decoding schedule. The layered schedule was optimized using the genetic algorithm optimization like in subsection III.C with the only difference in the cost, which in this case was the number of stall cycles. Furthermore, the pipeline conflicts due to the CNW change were considered to be solved using the decoupling FIFO buffer as described in subsection III.B. This way, the influence of factors that are not specifically connected to the decoding schedule is removed. Fig. 11.b) shows the calculated number of cycles per decoding iteration for both layered and hybrid schedule decoders for base graph 1 codes from 5G NR. It can be noticed that in layered decoding, a large number of stall cycles should be added, which can be explained with high density of 5G NR base graph matrices.

Finally, the throughput difference between the layered and the enhanced hybrid schedule decoder was calculated. The obtained throughput increase in enhanced hybrid schedule decoder is very significant and goes from 30.8% to 109.1%, for the same SNR performance as in the layered decoder. More detailed results for codes from 5G NR are shown in Fig. 11.c). For a layered decoder, the number of pipeline stages does not have to be as high as 13 if only one cyclic shifter is used. In [49], the number of pipeline stages was 9. However, even in the case when the layered decoder has 9 pipeline stages the equivalent throughput increase of the enhanced hybrid schedule decoder with 13 pipeline stages is between 16.5% and 75.2%.

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

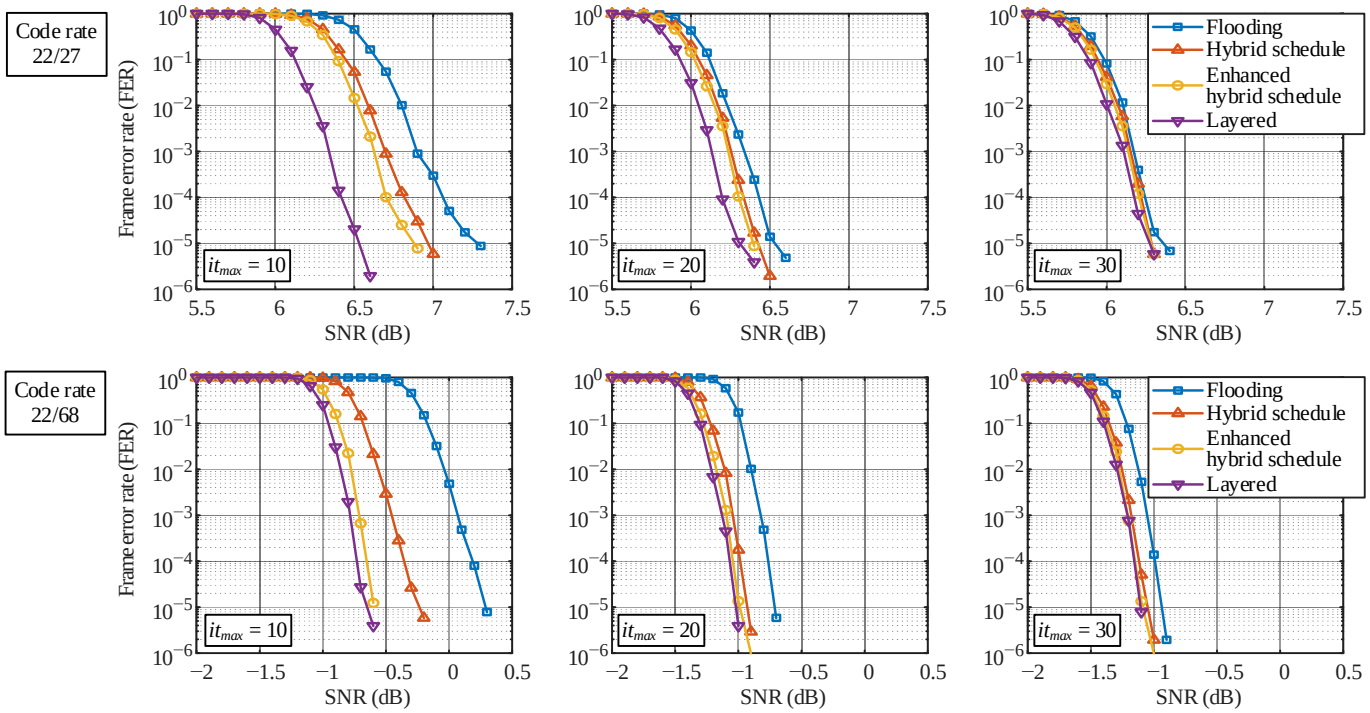

Fig. 9. SNR performance of various decoding schedules for rate 22/27 and rate 22/68 base graph 1 codes from 5G NR and lifting size Z = 384. Results are given for QPSK modulation and offset min-sum fixed point implementation with 8 bits for LLRs and 6 bits for messages. Simulated hybrid schedule decoders are with 13 pipeline stages.

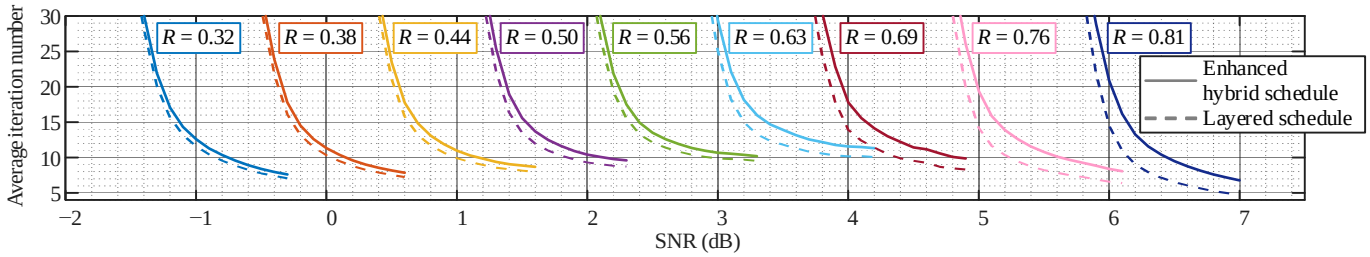

Fig. 10. Average iteration number necessary for decoding of various base graph 1 codes from 5G NR, with lifting size Z = 384. The codes are followng (left to right): (26112, 8448), (22272, 8448), (19200, 8448), (16896, 8448), (14976, 8448), (13440, 8448), (12288, 8448), (11136, 8448), and (10368, 8448).

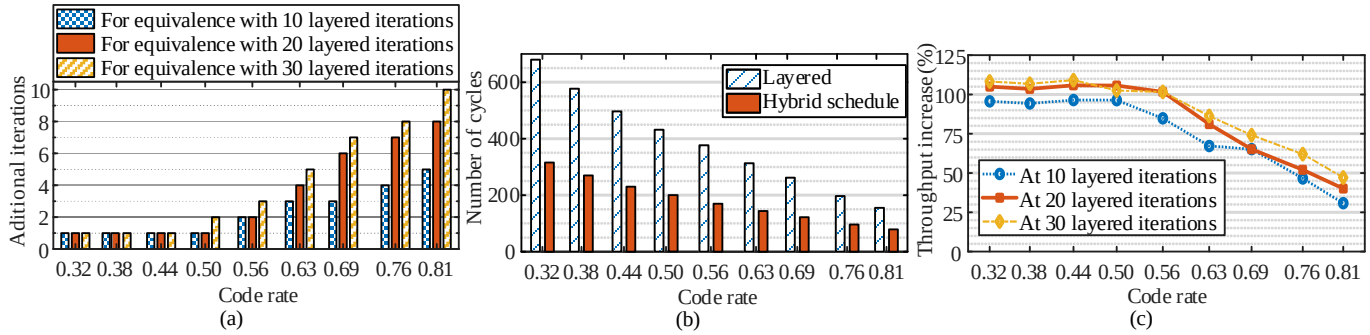

Fig. 11. Hybrid schedule decoder for 5G NR base graph 1 codes analysis. The number of pipeline stages was set to 13. (a) Number of additional hybrid schedule decoder iterations needed for the same SNR performance as of the layered decoder at maximum 10, 20, and 30 iterations, for FER = 10^{-5}. (b) Number of cycles per single iteration for layered and hybrid schedule decoders, including stall cycles. (c) Equivalent throughput increase of hybrid schedule decoder with respect to the layered decoder for the same SNR performance. Even though the hybrid schedule decoder needs more iterations for the same SNR performance, the throughput increase is significant because one single iteration in the hybrid schedule decoder lasts much shorter than in the layered decoder.

C. Implementation results

The decoder was implemented on the Xilinx ZCU111 development board with the Zynq UltraScale+ RF-SoC device (XCZU28DR).

Implementation results for WiMAX, DVB-S2X and 5G NR decoders are shown in Table 2. The number of parallel processing units for each decoder was set to the maximum lifting size required by the standard (96 for WiMAX, 360 for DVB-S2X, and 384 for 5G NR). All three versions support all codes from their respective standard. Additionally, decoders can be programmed to decode any QC-LDPC code with every

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

TABLE II
IMPLEMENTATION RESULTS FOR WIMAX, DVB-S2X, AND 5G NR LDPC DECODER AND COMPARISON WITH PRIOR WORK

	Standard	Codeword length [bit]	Device family	Quant	Resources utilization				f_{max} [MHz]	T_{norm} [Gbps]	HUE (T_{norm}/*Resources*)			
					Slices	LUTs	FFs	36k BRAMs			Mbps/ kSlice	Mbps/ kLUT	Mbps/ kFF	Mbps/ BRAM
[50]	WiMAX	2304	Zynq-7000	M4	3732	12250	3732	24	150.0	2.4*	649.5	197.9	301.5	101.0
[51]	WiMAX	2304	Virtex-5	M2	1137	3522	847	14.5	162.0	0.8	703.6	227.1	**944.5**	55.2
				M2	5583	18542	3992	80	126.0	3.2	573.2	172.6	801.6	40.0
[49]	DVB-S2X	64800	Stratix-V	L8-M6	-	63694	75372	-	250.0	10.3*	-	161.9	136.8	-
[33]	WiMAX	2304	Virtex-7	M4	12496	40700	35013	40.5	142.8	10.8	840	264.6	307.6	**265.9**
	DVB-S2X	64800		M4	59874	198810	112613	252.5	80.0	30.0	500	150.9	266.4	118.8
This work	WiMAX	2304	Virtex-7	L8-M6	7906	24228	23290	33.5	314.6	8.5	**1076.6**	**352.0**	366.1	254.6
	DVB-S2X	64800		L8-M6	23475	73136	68795	127.5	287.4	23.7	**1010.6**	**324.4**	**344.9**	**186.1**
	5G NR	26112		L8-M6	30824	100929	85431	136.5	261.0	20.9	679.1	207.4	245.0	153.4
	5G NR	10368		L8-M6						31.7	1029.0	314.2	371.3	232.4
	WiMAX	2304	Ultrascale+	L8-M6	4388	24043	23592	33.5	585.1	15.9	3614.3	659.6	672.2	473.4
	DVB-S2X	64800		L8-M6	13762	69274	69940	127.5	373.5	30.8	2240.3	445.1	440.8	241.8
	5G NR	26112		L8-M6	17665	96625	87751	136.5	404.8	32.5	1837.9	336.0	370.0	237.8
	5G NR	10368		L8-M6						49.1	2784.7	508.0	559.4	359.6

Quant: Quantization where L*X*-M*Y* means that *X* bits were used for LLRs and *Y* bits were used for messages; f_{max}: Maximum operating frequency; T_{norm}: Normalized throughput to one iteration; HUE: Hardware Usage Efficiency; -: not available; *: calculated with the formula from the paper.

lifting size supported by the cyclic shifter at runtime.

Number of pipeline stages in WiMAX and 5G NR decoders was set to 13, whereas in DVB-S2X it was set to 11, due to the smaller cyclic shifter complexity. The obtained clock frequencies were 585.1 MHz, 373.5 MHz and 404.8 MHz for WiMAX, DVB-S2X and 5G NR decoder respectively. For maximum lifting sizes, the obtained coded throughput at 10 iterations is between 1.59 Gbps and 1.77 Gbps for WiMAX, between 3.08 Gbps and 4.32 Gbps for DVB-S2X, and between 3.25 Gbps and 4.92 Gbps for 5G NR.

The implementation results are compared with the relevant previous work in Table 2: WiMAX implementations from [33], [50], and [51], and DVB-S2X implementations from [33] and [49]. The throughput results are given for WiMAX rate 3/4 code, DVB-S2X rate 140/180 code and 5G NR rate 22/68 code and rate 22/27 code. The results for WiMAX implementation from [50] and DVB-S2X implementation from [49] are calculated based on the formula given in the paper. All decoders in the referenced literature were designed for different scenarios, hence the maximum number of iterations varied. Therefore, the throughput normalized to one iteration was placed in the Table 2 for uniformity.

In previous publications, the Hardware Usage Efficiency (HUE) was usually expressed as the obtained throughput divided with the number of used FPGA slices. In Table 2, the HUE is also calculated for all three other most significant resources: look-up tables (LUTs), flip-flops (FFs) and RAM blocks (BRAMs). All FPGA families from Table 2 have LUTs that can implement any 6-input logic function. All FPGA families have RAM blocks of 36 kb, except Stratix-V, used in [49], whose blocks are of 20 kb. Nevertheless, the RAM block utilization is not presented in [49]. The FPGA family used for the implementation of the proposed decoder is one of today's most modern FPGA families. In order to make a better and fair comparison with the previous designs, the proposed decoder is also implemented on the older Virtex-7 FPGA (XC7VX690T), as recommended in [22]. These results were compared with the prior work in Table 2 and they show that the decoder presented in this paper provided the highest HUE for almost all metrics.

The quantization in previous works has been done in various ways. The column Quant in Table 2 shows the number of bits used for LLRs (if used at all) and for messages. Even with the highest precision compared with the previous works, the obtained HUE in this paper is significantly higher than in the most previous solutions.

It is important to notice that all previous implementations supported codes with small differences in check node weights between the layers. The proposed solution provides easy implementation of the decoder for highly irregular codes with large differences in CNW, such as codes from 5G NR.

V. CONCLUSION

In this paper, a novel architecture for QC-LDPC decoding is presented. The proposed architecture is efficient for decoding highly irregular QC-LDPC codes without any stall cycles caused by memory access pipeline conflicts or check node weight change, even when the number of pipeline stages is large. The architecture performs the hybrid decoding schedule,

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

which works as the layered schedule, but switches to flooding schedule only in cases when a pipeline conflict would occur in the conventional layered architecture. The hybrid schedule is then optimized for better SNR performance using the genetic algorithm based offline PCM reordering. Such enhanced hybrid schedule leaves almost negligible loss in SNR performance compared with the fully layered schedule for most test scenarios, which can be eliminated by adding a few additional iterations. Even with the addition of extra iterations, the stall cycles removal provides such high gain in throughput that, for the same SNR performance, the obtained throughput increase is between 30.8% and 109.1% for 5G NR codes, in case when 13 pipeline stages are used. Aggressive pipelining provided very high clock frequencies for decoders implemented for WiMAX, DVB-S2X, and 5G NR, which further induced high throughput and high hardware usage efficiency. Implemented decoders provided multi-gigabit throughputs and showed significant HUE improvement when compared with the state-of-the-art works.

ACKNOWLEDGMENT

Authors are thankful to Srđan Brkić, Đorđe Sarač, and Predrag Ivaniš from the University of Belgrade – School of Electrical Engineering, Belgrade, Serbia, for valuable comments and discussions.

REFERENCES

[1] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[2] J. Kim and W. Sung, "Rate-0.96 LDPC decoding VLSI for soft-decision error correction of NAND flash memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 5, pp. 1004–1015, May 2014.

[3] *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 Standard 802.3an-2006, 2006.

[4] *Standard for Information Technology—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard 802.11-2016, 2016.

[5] *Standard for Local and Metropolitan Area Networks—Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, IEEE Standard 802.16-2004, 2004.

[6] *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 2: DVB-S2 Extensions (DVB-S2X)*, ETSI EN 302 307-2 V.1.1.1 (2014-10), 2014.

[7] *DOCSIS 3.1: Data-Over-Cable Service Interface Specifications DOCSIS 3.1, Physical Layer Specification*, CM-SP-PHYv3.1-I11-170510, 2017.

[8] *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; NR; Multiplexing and channel coding (Release 16)*, 3GPP TS 38.212 V16.1.0 (2020-03), 2020.

[9] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[10] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[11] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, pp. 599–618, 2001.

[12] D.J.C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[13] R. G. Gallager, "Analysis of Number of Independent Decoding Iterations," in *Low-Density Parity-Check Codes*, Cambridge, MA, USA, MIT Press, 1963, Appendix *C*, pp. 81–88.

[14] T. J. Richardson and S. Kudekar, "Design of low-density parity check codes for 5G new radio," *IEEE Commun. Mag.*, vol. 56, no. 3, pp. 28–34, Mar. 2018.

[15] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 219–230, Feb. 1998.

[16] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. IEEE Work. Signal Process. Syst.*, Austin, TX, USA, Oct. 2004.

[17] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, Feb. 2005.

[18] M. K. Roberts and R. Jayabalan, "An area efficient and high throughput multi-rate quasi-cyclic LDPC decoder for IEEE 802.11n applications," *Microelectron. J.*, vol. 45, no. 11, pp. 1489–1498, Nov. 2014.

[19] S. Ajaz, T. T. B. Nguyen, and H. Lee, "An Area-Efficient Half-Row Pipelined Layered LDPC Decoder Architecture," *J. Semicond. Technol. Sci.*, vol. 17, no. 16, pp. 845–853, Dec. 2017.

[20] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, pp. 404–412, Mar. 2002.

[21] Z. Wang and Z. Cui, "Low-complexity high-speed decoder design for quasi-cyclic LDPC codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, pp. 104–114, 2007.

[22] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A survey of FPGA-based LDPC decoders," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1098–1122, 2nd Quart. 2016.

[23] M. Rovini, G. Gentile, F. Rossi, and L. Fanucci, "A scalable decoder architecture for IEEE 802.11n LDPC codes," in *Proc. IEEE Global Telecommun. Conf.*, Nov. 2007, pp. 3270–3274.

[24] C. Marchand, J. B. Dore, L. Conde-Canencia, and E. Boutillon, "Conflict resolution for pipelined layered LDPC decoders," in *Proc. IEEE Work. Signal Process. Syst.*, Tampere, Finland, Oct. 2009, pp. 220–225.

[25] X. Zhao, Z. Chen, X. Peng, D. Zhou, and S. Goto, "DVB-T2 LDPC decoder with perfect conflict resolution," *Inf. Media Technol.*, vol. 7, no. 2, pp. 584-592, Feb. 2012.

[26] Z. Wu, D. Liu, and Y. Zhang, "Matrix reordering techniques for memory conflict reduction for pipelined QC-LDPC decoder," in *Proc. IEEE/CIC Int. Conf. Commun. China*, Shanghai, China, Oct. 2014, pp. 354–359.

[27] Z. Wu and K. Su, "Updating conflict solution for pipelined layered LDPC decoder," in *Proc. IEEE Int. Conf. Signal Process. Commun. Comput.*, Ningbo, China, Sep. 2015.

[28] C.-W. Sham, X. Chen, W. M. Tam, Y. Zhao, and F. C. M. Lau, "A layered QC-LDPC decoder architecture for high speed communication system," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, Kaohsiung, Taiwan, Dec. 2012, pp. 475–478.

[29] S. Kumawat, R. Shrestha, N. Daga, and R. Paily, "High-throughput LDPC-decoder architecture using efficient comparison techniques and dynamic multi-frame processing schedule," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 5, pp. 1421–1430, May 2015.

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

[31] H.-C. Lee, M.-R. Li, J.-K. Hu, P.-C. Chou, Y.-L. Ueng, "Optimization techniques for the efficient implementation of high-rate layered QC-LDPC decoders," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 64, no. 2, pp. 457–470, Feb. 2017.

[32] Implementation and performance of LDPC decoder, document R1-1700111, 3GPP, Ericsson, Spokane, USA, Jan. 2017.

[33] 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 Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 4, pp. 1643–1656, Apr. 2019.

[34] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[35] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures," *IEEE Trans. Inf. Theory*, vol. 50, no. 9, pp. 2002–2009, Sep. 2004.

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

[36] E. Amador, R. Pacalet, and V. Rezard, "Optimum LDPC decoder: A memory architecture problem," in *Proc. 46th ACM/IEEE Des. Autom. Conf.*, San Francisco, CA, USA, Jul. 2009, pp. 891–896.
[37] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory system optimization for FPGA-based implementation of quasi-cyclic LDPC codes decoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 1, pp. 98–111, Jan. 2011.
[38] S. Muller, M. Schreger, M. Kabutz, M. Alles, F. Kienle, and N. Wehn, "A novel LDPC decoder for DVB-S2 IP," in *Proc. Design, Autom. Test Eur.*, Nice, France, Apr. 2009, pp. 1308–1313.
[39] J. Jin and C. Tsui, "An energy efficient layered decoding architecture for LDPC decoder," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 8, pp. 1185–1195, Aug. 2010.
[40] T. T. Nguyen-Ly, V. Savin, X. Popon, and D. Declercq, "High throughput FPGA implementation for regular non-surjective finite alphabet iterative decoders," in *Proc. IEEE Int. Conf. Commun. Work.*, Paris, France, May 2017, pp. 961–966.
[41] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.
[42] J. Chen and M. P. C. Fossorier, "Density evolution for two improved BP based decoding algorithms of LDPC codes," *IEEE Commun. Lett.*, vol. 6, no. 5, pp. 208–210, May 2002.
[43] Zynq UltraScale+ RFSoC RF Data Converter v2.3 - LogiCORE IP Product Guide, Xilinx, San Jose, CA, USA, Jun. 2020.
[44] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A flexible FPGA-based quasi-cyclic LDPC decoder," *IEEE Access*, vol. 5, pp. 20965–20984, Mar. 2017.
[45] H.-J. Kang and B.-D. Yang, "Low-complexity, high-speed multi-size cyclic-shifter for quasi-cyclic LDPC decoder," *Electron. Lett.*, vol. 54, no. 7, pp. 452-454, Apr. 2018.
[46] X. Chen, S. Lin, and V. Akella, "QSN-A simple circular shift network for reconfigurable quasi-cyclic LDPC decoders," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 10, pp. 1549–7747, Oct. 2010.
[47] Y. Jung, Y. Jung, S. Lee, and J. Kim, "Low-complexity multi-way and reconfigurable cyclic shift network of QC-LDPC decoder for Wi-Fi/WIMAX applications," *IEEE Trans. Consum. Electron.*, vol. 59, no. 3, pp. 467–475, Aug. 2013.
[48] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artif. Intell. Rev.*, vol. 13, no. 2, pp. 129–170, 1999.
[49] C. Marchand and E. Boutillon, "LDPC decoder architecture for DVB-S2 and DVB-S2X standards," ," in *Proc. IEEE Work. Signal Process. Syst.*, Hangzhou, China, Dec. 2015.
[50] S. Yeşil and M. Arslan, "Dual port ram based layered decoding for multi rate quasi-cyclic LDPC codes," in *Proc. 12th Int. Conf. Signal Process.*, Hangzhou, China, Oct. 2014, pp. 1524–1530.
[51] V. A. Chandrasetty and S. M. Aziz, "Resource efficient LDPC decoders for multimedia communication," Integration, vol. 48, pp. 213–220, Jan. 2015.

Vladimir L. Petrović (S'19) received the Dipl. Ing. and M.S. degrees in electrical engineering from the University of Belgrade, Serbia, in 2014 and 2015, respectively.

He is currently a Teaching and Research Assistant at the University of Belgrade - School of Electrical Engineering, Serbia, and a Ph.D. candidate at the same school. His research interests include VLSI design, communication systems architectures, digital signal processing, and hardware implementations of signal processing algorithms.

Mr. Petrović was a recepient of the Best young researcher's paper award at IcETRAN 2016, Serbia.

Miloš M. Marković received the Dipl. Ing. and M.S. degrees in electrical engineering from the University of Belgrade, Serbia, in 2018 and 2019, respectively.

In 2018 he joined Tannera LLC, Belgrade, Serbia, where he works as a hardware engineer. His research interests include VLSI design, communication systems architectures, and embedded systems.

Dragomir M. El Mezeni was born in Belgrade, Serbia in 1985. He received the B.S. degree in electrical engineering from the University of Belgrade, Serbia, in 2008, M.S. degree in electronics in 2010 and Ph.D. degree in electronics in 2018 at the same University.

He is currently Assistant Professor of electrical engineering at the University of Belgrade, Serbia. His research interests include digital image/video processing, computational photography and very large scale integration architectures for digital signal processing.

Lazar V. Saranovac (M'94) was born in Sremska Mitrovica, Serbia in 1961. He received the B.S. degree in electrical engineering from the University of Belgrade, Serbia, in 1987, M.S. degree in electronics in 1993 and Ph.D. degree from the same University in 2001.

He is currently Full Professor of electrical engineering at the University of Belgrade, Serbia. His research interests include embedded systems, digital signal processing and design of digital systems.

Andreja Radošević earned a Ph.D. degree from Department of Electrical and Computer Engineering at UC San Diego, USA, in 2012. His academic research interests included information and coding theory, adaptive modulation, channel estimation and equalization, and spectrally efficient MIMO-OFDMA transmission schemes, mainly in the context of underwater acoustic communications.

In 2012, he joined Qualcomm, where he was one of the key contributors to development of baseband modems supporting 4G-LTE, WCDMA and TD-SCDMA standards. His role included algorithm designs for time and frequency synchronization, turbo decoding, power control, and interference cancellation. In 2018, he founded Tannera LLC, Belgrade, Serbia, as an effort to join the wireless and Internet-of-Things industry frontier by delivering disruptive and state-of-the-art system solutions.

© 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.