BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 59, No. 2, 2011
DOI: 10.2478/v10175-011-0019-9

Pipeline processing in low-density parity-check codes hardware decoder

W. SUŁEK*

Institute of Electronics, Silesian University of Technology, 16 Akademicka St., 44-100 Gliwice, Poland

Abstract. Low-Density Parity-Check (LDPC) codes are one of the best known error correcting coding methods. This article concerns the hardware iterative decoder for a subclass of LDPC codes that are implementation oriented, known also as Architecture Aware LDPC. The decoder has been implemented in a form of synthesizable VHDL description. To achieve high clock frequency of the decoder hardware implementation – and in consequence high data-throughput, a large number of pipeline registers has been used in the processing chain. However, the registers increase the processing path delay, since the number of clock cycles required for data propagating is increased. Thus in general the idle cycles must be introduced between decoding subiterations. In this paper we study the conditions for necessity of idle cycles and provide a method for calculation the exact number of required idle cycles on the basis of parity check matrix of the code. Then we propose a parity check matrix optimization method to minimize the total number of required idle cycles and hence, maximize the decoder throughput. The proposed matrix optimization by sorting rows and columns does not change the code properties. Results, presented in the paper, show that the decoder throughput can be significantly increased with the proposed optimization method.

Key words: channel coding, LDPC codes, iterative decoding, decoder implementation, pipelined processing.

1. Introduction

Modern communication demands in transmission throughput motivate continuous progress of error correcting coding systems. Low-Density Parity-Check (LDPC) codes are one of the best known coding methods that allow achieving very low bit error rates at code rates approaching Shannon's channel capacity limit. (The second known method is turbo coding). Thus LDPC codes have recently attracted intense research interest. Their main advantage over turbo-codes is highly parallel decoding scheme.

LDPC codes were first introduced by Gallager in 1962 [1], but soon forgotten. Their implementation complexity was exceeding capabilities of the accessible technology, so they were not considered for practical applications. The codes were rediscovered in the late 90's [2] and since then they have been under interests of many researchers. Despite the constant progress in electronics technology, the hardware design for LDPC coding systems is still not straightforward. The main challenges include: 1) to define low complexity, high throughput decoder architectures, 2) to make the architecture versatile, i.e. capable of decoding large family of codes.

The fully parallel LDPC iterative decoding architecture can achieve high decoding throughput, but it suffers from large hardware complexity caused by a large set of processing units and complex interconnections. A practical solution for area efficient decoders is to use the partially parallel architecture in which a processing step is performed in a several time slots using some number of processing units working in parallel. It has been recognized that the partially parallel decoder architectures can be accomplished well for some subclass of codes, with structured parity check matrix, known as Architecture-Aware LDPC (AA-LDPC, [3]), VLSI-Oriented [4] or Structured LDPC [5].

A programmable partially parallel decoder has been implemented in the form of synthesizable VHDL description. The decoder is capable for decoding any code that has the parity check matrix in the Architecture-Aware form. The target hardware platform for the implemented decoder is FPGA; Xilinx VirtexII devices were used for decoder verification. In this paper we briefly present the decoder structure. Then we focus on pipeline processing optimization that has been proposed to speed up the decoding process.

The pipeline registers has been used to achieve high clock frequency of the decoder hardware implementation and in consequence high data-throughput. However, the registers increase the processing path delay as the number of clock cycles required for propagating data is increased. Thus the main problem connected with pipeline processing is that the new step of data processing must not be started before the previous one is completed if the data updated in the previous step is required for the new one. Hence in general the idle clock cycles must be introduced. However we show in this article that the number of required idle cycles is somehow dependent on the parity check matrix structure. Moreover we present the parity check matrix optimization algorithm that can minimize the required idle cycles number. As a result of the idle time reduction, the decoder throughput is significantly increased, which is shown in the experimental results.

The paper is organized as follows. First, we present a basic concepts connected with LDPC codes and the Architecture Aware subclass of codes. Particularly decoding algorithm will

*e-mail: wsulek@polsl.pl

be described. Then the decoder structure will be presented with emphasis on pipeline processing elements. In the following sections we study the conditions for necessity of idle cycles and provide a method for calculation the exact number of required idle cycles on the basis of parity check matrix structure. Then we propose a parity check matrix optimization method to minimize the total number of idle cycles and finally present results obtained with proposed optimization method for several LDPC codes.

2. LDPC codes basics

LDPC codes are linear block codes [6] with sparse parity-check matrix. The parity-check matrix $\mathbf{H}_{M\times N}$ of a code $\mathcal{C}$ represents the relation between N bits of the codeword and M parity-check equations. Vector $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ is a correct codeword ($\mathbf{x} \in \mathcal{C}$) iff the parity check condition $\mathbf{Hx}^{\mathrm{T}} = \mathbf{0}$ is satisfied (in $GF(2)$ field).

In the encoder an information vector $\mathbf{u} = \{u_1, u_2, \ldots, u_K\}$ of length $K = N - M$ is transformed into a proper codeword by combining it with M parity bits. The coderate $R = K/N$ characterizes the amount of redundancy in the codeword. In the decoder, where information about bit values is distorted, the most probable codeword is determined on the basis of received vector $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$. The inputs to the decoder algorithm are in the form of received bits in the case of hard-decision decoding or in the form of a priori probabilities of bit values $P(x_n = 0|y_n), P(x_n = 1|y_n)$ (or some functions of probabilities) in the case of soft-decision decoding. The latter case allows obtaining significant better error correcting performance.

Based on the parity check matrix, a bipartite graph $\mathcal{G}$ (Tanner graph) is defined with bit nodes corresponding to bits and check nodes corresponding to parity-check equations (Fig. 1). Formally: $\mathcal{G} = (\mathcal{V}_c \cup \mathcal{V}_b, \mathcal{E})$, where $\mathcal{V}_c = \{c_1, c_2, \ldots, c_M\}$ is the set of the check nodes, $\mathcal{V}_b = \{b_1, b_2, \ldots, b_N\}$ is the set of bit nodes and $\mathcal{E} \subseteq \mathcal{V}_b \times \mathcal{V}_c$ is the set of edges. An edge $e_i = (b_n, c_m)$ belongs to $\mathcal{E}$ if and only if $h_{mn} \neq 0$. A Tanner graph is (d_c, d_b)-regular if all its check nodes have degree d_c and all its bit nodes have degree d_b. Otherwise, the graph is irregular and the corresponding code is called irregular LDPC.

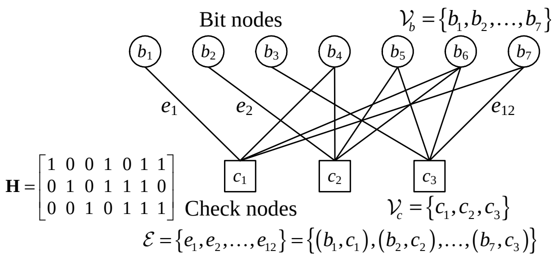

Fig. 1. Parity check matrix and Tanner graph

The Tanner graph visualizes iterative message passing algorithms used for decoding. The algorithms are performed by exchanging messages (beliefs) between bit nodes and check nodes through the edges in both directions. Each node of the graph represents computation of updated beliefs. In the case of LLR-BP algorithm (Log-Likelihood Ratio Belief Propagation), messages are log-likelihood ratios of beliefs, hence operations are sums (bit nodes) and sums of nonlinear functions of messages (check nodes). LLR-BP and its modifications are considered the most frequently [3, 7] for hardware implementations. Inputs to the decoding algorithm are bit values altogether with measures of its reliability based on channel observations (received channel soft values), which are in the form of log-likelihood ratios; they are called intrinsic channel reliability values and will be denoted as δ_n for nth bit:

$$\delta_n = \log\left[\frac{P(x_n = 0|y_n)}{P(x_n = 1|y_n)}\right]. \tag{1}$$

The basic two-phase message-passing (TPMP) algorithm is described in reach literature, e.g. [2, 6, 7].

2.1. TDMP decoding algorithm. Here we focus on Turbo-Decoding Message-Passing (TDMP) algorithm [3, 8] that has been used for the presented hardware decoder. It is based on a modification in message passing scheme of the classic TPMP algorithm, where $\mathcal{C}$ is considered as the concatenation of some number of codes. It means that the set of rows of $\mathbf{H}$ is virtually partitioned into a number of subsets. Each subset constitutes a code, which will be called a subcode. A word is a correct codeword if it belongs to all constituent subcodes.

Let $\mathcal{C}^d$ denote the dth subcode with parity check matrix $\mathbf{H}^d$, $d = 1, \ldots, D$. The code is then $\mathcal{C} = \mathcal{C}^1 \cap \ldots \cap \mathcal{C}^D$ and its parity check matrix is $\mathbf{H}_{M\times N} = \left[\mathbf{H}^1; \mathbf{H}^2; \ldots; \mathbf{H}^D\right]^{\mathrm{T}}$. Furthermore, the check node set $\mathcal{V}_c$ of the Tanner graph of the code is partitioned as $\mathcal{V}_c = \mathcal{V}_c^1 \cup \ldots \cup \mathcal{V}_c^D$. It is assumed that the rows in each submatrix $\mathbf{H}^d$ do not overlap (each bit node is incident to at most 1 check node in the subset $\mathcal{V}_c^d$). Following [2], we denote the set of indexes of check nodes adjacent to bit node b_n by $\mathcal{M}(n)$ and – similarly – the set of indexes of bit nodes adjacent to check node c_m by $\mathcal{N}(m)$. Furthermore, $\mathcal{N}(m)\backslash n$ represents the set $\mathcal{N}(m)$ excluding n.

The TDMP algorithm determines a codeword iteratively in D subiterations, with one subiteration per constituent code $\mathcal{C}^d$. Following [9], we denote Λ_n^d the LLR reliability value for bit nth assuming that the codeword belongs to subcode $\mathcal{C}^d$ and λ_n^d the reliability values calculated in the previous iteration. The sum of Λ_n-messages for all subcodes in addition to the channel values δ_n is the posterior reliability value, denoted by Γ_n and updated at each subiteration. Thus Γ_n represents „all the information" about the value of the bit x_n in the current stage of decoding process. The Γ_n value calculated at previous subiteration is denoted by γ_n. The TDMP algorithm is summarized as follows.

1) **Initialization**
Initialize posterior γ values to the intrinsic channel reliability LLRs and the messages λ for all subcodes to zeros.

$$\gamma_n := \delta_n = \log\left[\frac{P(x_n = 0|y_n)}{P(x_n = 1|y_n)}\right], \tag{2}$$

$$\lambda_n^d := 0, \qquad n = 1, \ldots, N \qquad d = 1, \ldots, D \tag{3}$$

2) **Iteration**
Carry out D decoding subiterations corresponding to subcodes C^d, $d = 1, \ldots, D$. For each subiteration compute messages Λ_n^d as well as posterior reliability values Γ_n. At subiteration d:
– For each m such that $c_m \in \mathcal{V}_c^d$, for each $n \in \mathcal{N}(m)$, compute

$$Q_n^d := \gamma_n - \lambda_n^d, \tag{4}$$

$$\Lambda_n^d := \left(\prod_{n'} \mathrm{sgn}\left(Q_{n'}^d\right)\right) \times \psi^{-1}\left(\sum_{n'} \psi\left(|Q_{n'}^d|\right)\right), \tag{5}$$

$$\Gamma_n := Q_n^d + \Lambda_n^d = \gamma_n - \lambda_n^d + \Lambda_n^d, \tag{6}$$

where $n' \in \mathcal{N}(m) \backslash n$, where c_m is the check node in the subset $\mathcal{V}_c^d$ incident to b_n (as we assumed, there is only one such check node) and $\psi(x)$ is a nonlinear function defined as: $\psi(x) = \psi^{-1}(x) = -\ln(\tanh(x/2))$ that can be calculated making use of some known approximations [7, 12].
– Store the Λ and Γ-values as λ and γ-values to be used as inputs in the next subiteration:

$$\gamma_n := \Gamma_n, \qquad \lambda_n^d := \Lambda_n^d, \qquad n = 1, \ldots, N \tag{7}$$

3) **Stop Criterion**
After all subiterations make hard decisions $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ such that

$$x_n := \begin{cases} 1, & \Gamma_n \leq 0 \\ 0, & \Gamma_n > 0 \end{cases} \tag{8}$$

If the parity check equation $\mathbf{Hx}^T = \mathbf{0}$ is satisfied or a maximum number of iterations $i_{\max}$ is reached, halt the algorithm with $\mathbf{x}$ as output. Otherwise, go to step 2.

The decoder receives soft channel values δ and generates reliability values of the decoded bits Γ. These values are updated at each subiteration. Furthermore, for each constituent subiteration, extrinsic reliability values Λ are computed assuming that the codeword belongs to the subcode, according to (4)–(6). Intrinsic λ-messages pertaining to the code under consideration are subtracted from γ (Eq. (4)) to eliminate correlation between newly generated messages and the previously generated. Thus, the Q_n^d represents LLR reliability value for bit nth based on messages from all subcodes except subcode under consideration. These Q values are used for updating extrinsic messages Λ corresponding to the subcode being decoded.

The main advantages of TDMP scheme over standard TPMP are that it exhibits a faster convergence behavior (about 20–50% fewer decoding iterations) as well as it allows a memory savings due to eliminating multiple bit-to-check messages [3].

2.2. AA-LDPC codes. As is well known, efficient partially-parallel decoder implementation is possible for parity-check matrices with certain constraints on their form [3, 10, 11]. Firstly, as we assumed, the rows of $\mathbf{H}$ are partitioned such that in each submatrix $\mathbf{H}^d$ all columns contain at most single non-zero element. Secondly, in order to suitably organize message (Γ) memory, the set of bit nodes of the graph is partitioned into L subsets $\mathcal{V}_b = \mathcal{V}_b^1 \cup \ldots \cup \mathcal{V}_b^L$ of P nodes, such that considering subcode $\mathcal{C}^d$, each bit node from any subset $\mathcal{V}_b^j$ is adjacent to exactly one check node from subset $\mathcal{V}_c^d$ or each bit node from this subset is not adjacent to any check node from subset $\mathcal{V}_c^d$. Then single memory word may consist of P Γ-values (corresponding to bits from single subset $\mathcal{V}_b^j$) that are delivered to P computing modules in a single clock period by configurable interleaver that shuffles the fragments of the memory word according to the local graph structure of the code.

With such code-graph organization, a parity check matrix is similar to the one shown in Eq. (9):

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}_{1,1} & \mathbf{P}_{1,2} & \cdots & \mathbf{P}_{1,L} \\ \mathbf{P}_{2,1} & \mathbf{P}_{2,2} & \cdots & \mathbf{P}_{2,L} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{D,1} & \mathbf{P}_{D,2} & \cdots & \mathbf{P}_{D,L} \end{bmatrix} \tag{9}$$

Matrix $\mathbf{H}$ is composed of $D \times L$ submatrices, where each submatrix $\mathbf{P}_{d,l}$ of size $P \times P$ is either an all-zero matrix or a permutation matrix obtained by permuting columns of an identity matrix [5]. Placement of nonzero submatrices in $\mathbf{H}$ is specified by so-called seed matrix $\mathbf{W}$. It is a $D \times L$ matrix with elements $w_{d,l} = 0$ if $\mathbf{P}_{d,l}$ is the all-zero submatrix and $w_{d,l} = 1$ if $\mathbf{P}_{d,l}$ is the permutation submatrix. Codes with parity check matrix arranged in this manner are known as Architecture Aware subclass of LDPC codes (AA-LDPC).

3. Decoder architecture

A configurable decoder, based on TDMP scheme has been implemented in the form of synthesizable VHDL model. This model can be adjusted for decoding any regular or irregular code that has the matrix $\mathbf{H}$ in the presented Architecture-Aware form. The decoder structure has been described in detail in other papers [5, 13]. Here we focus on pipeline processing that has been used to speed up the decoding process.

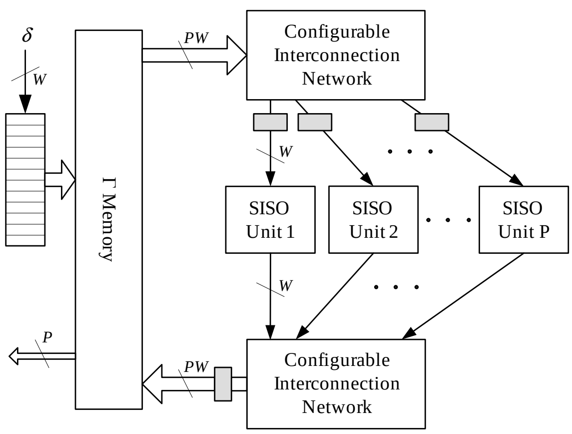

Fig. 2. Structure of the decoder

The simplified block diagram of the decoder is presented in Fig. 2. Reliability values of the decoded bits Γ are stored in Γ-memory, which is initialized with soft channel values δ with wordlength W. Typical wordlength is in range $W = 5 \ldots 8$ and its selection defines the tradeoff between the decoder performance and resources required for the implementation. Computations corresponding to (4)–(6) are performed in so-called Soft-Input-Soft-Output (SISO) modules. Each of the modules is responsible for single check node $c_m \in \mathcal{V}_c^d$ messages calculation during subiteration d. Since $||\mathcal{V}_c^d|| = P$, it is convenient to use P SISO modules operating in parallel. (Generally: not greater than P SISO modules; we use P modules for the highest parallelism.) One full iteration consists of D subiterations ($d = 1, \ldots, D$), thus it is performed in D time slots.

In a single time slot, messages $\Gamma_n, n \in \mathcal{N}(m)$ are fed to the SISO unit in a serial manner through Configurable Interconnection Network, updated in the SISO, and then serially fed back to the Γ-memory. A memory word consists of P Γ-values, hence wordlength for Γ-memory equals PW and memory depth is L. Configurable Interconnection Network ensures proper messages propagation, according to the structure of the permutation submatrices $\mathbf{P}_{d,l}$ of the constituent subcode d (see Eq. (9)). The messages Λ are stored in Λ-memory, which is partitioned and included in SISO units as small memory buffers. When stop criterion is met, current word decoding is halted and data is outputted in words of length P (the MSBs of the P Γ-messages in a single memory word are equivalent to the decoded bits).

Time slot duration, i.e. the number of clock cycles for executing single subiteration, depends on check-nodes degree as well as the number of pipeline registers in the processing chain. The pipeline registers – denoted as grey boxes in the figures – are essential components to achieve high clock frequency of the decoder hardware implementation and in consequence high data-throughput. However, the registers increase the processing path delay as the number of clock cycles required for propagating data is increased. Here the main problem connected with pipeline processing arises. The new subiteration must not be started before the previous one is completed if the data (Γ messages) updated in the previous subiteration are required for the new one. Hence in general the idle clock cycles must be introduced to await for the completion of the previous subiteration. The number of required idle cycles depends on the processing chain delay, thus the increase in clock frequency due to the pipelining is counterfeited by the increase in number of idle clock cycles.

In the classic literature concerning TDMP decoding implementation [3, 8, 9] the problem mentioned above is not treated at all. In the next sections we first study the conditions for necessity of idle cycles and provide a method for calculation the exact number of required idle cycles on the basis of parity check matrix structure. Then we propose the formula for the calculation of the total number of idle cycles per iteration and finally propose a parity check matrix optimization method to minimize the total number of idle cycles. In the end we present results obtained with proposed optimization method for several cases.

4. Number of idle cycles calculation

The basic block diagram of the SISO module is presented in Fig. 3. The main component of SISO is CNU (Check Node Unit), which is responsible for Λ messages calculation as in (5). Since the CNU operates in a double recursion scheme (see e.g. [12]), the output messages are in reversed order. Blocks denoted as Subtr* and Add* are subtractor and adder respectively with clipping elements that constrain the results of addition (subtraction) to W bits.

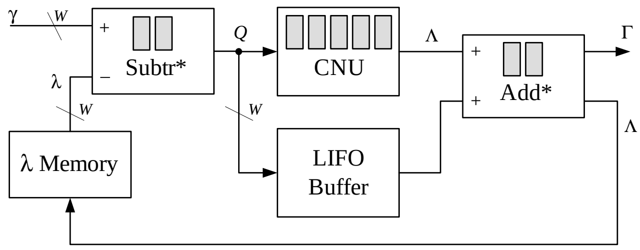

Fig. 3. SISO computing module

To achieve high clock frequency, five pipeline registers (grey boxes in Fig. 3) has been used in the CNU as well as two registers in blocks Add* and Subtr*. The placement of the registers has been designed experimentally by observing the synthesis results for VirtexII FPGA and trying to exploit the maximum achievable clock speed.

Let T_P be the total pipeline delay of the decoder defined by the number of clock cycles from the last Γ-memory read to the first Γ-memory write. The T_P delay is equal to the sum of delays due to interconnection network T_{NET} and due to SISO module T_{SISO}:

$$T_P := 2T_{NET} + T_{SISO} \tag{10}$$

In the case of the implemented decoder we have: $T_{NET} = 1$ (Fig. 2), $T_{SISO} = 9$ (Fig. 3), thus $T_P = 11$.

Let T_{idle}^d be the number of idle cycles required before subiteration d is started (i.e. the first message is fetched). Obviously T_{idle}^d depends on T_P, but also – as will be shown – it is dependent on the existence of nonzero elements in the same columns of dth, $d-1$, $d-2$ and $d-3$ rows of the seed matrix. It is illustrated by a following example.

Figure 4 presents a part of some parity check matrix, where the numerated boxes represent permutation submatrices and grey boxes – all-zero submatrices. At the bottom of the figure we present a sequence of data transmitted from / to the Γ-memory, where the numbers indicate memory cells that are read / written, which are consistent with the location of the nonzero submatrices in the parity check matrix.

Before the 2nd subiteration can be started (blue marks in Fig. 4), the decoder has to await a proper time for memory cell 9 update from the 1st subiteration (black marks). Precisely: the pause T_{idle}^2 has to ensure that memory cell 9 read (2nd subiteration) is at least one cycle after memory cell 9 write (1st subiteration). (It is assumed that "write first" configuration of two-port memory is used for implementation.) Thus the idle cycles are needed in the case of one (or more) common messages are processed in the consecutive subiterations, which is a result of "overlapping" ones in consecutive rows of seed matrix (here: ones in column 9th, "black" and "blue" rows). In the example presented in Fig. 4, $T^2_{\text{idle}} = T_P - 2$.

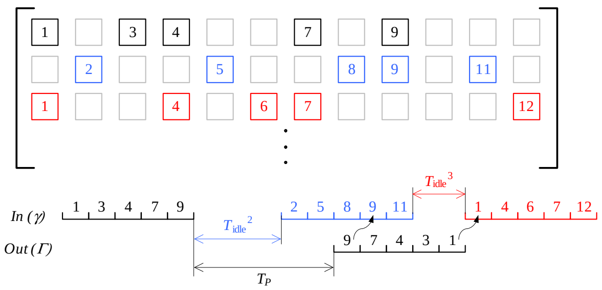

Fig. 4. Data sequence on the Γ-memory ports

Furthermore, the need for idle cycles is still possible even if two consecutive rows of seed matrix do not contain ones in a common column. Such a case is presented in Fig. 4 for the 3rd subiteration (red marks). The pause T^3_{idle} has to ensure memory cell 1 read (3rd iteration) is at least one cycle after this memory cell update (1st subiteration). The common memory cell 1 usage is a result of "overlapping" ones in every second rows of seed matrix (here: "black" and "red" rows).

Here we propose a formula for exact calculation of required idle cycles T^d_{idle}, on the basis of seed matrix structure. Let $X_{(d_2,d_1)}$ be an auxiliary variable defined as:

- if rows d_2th and d_1th of the seed matrix do not contain ones in a common column, then $X_{(d_2,d_1)} = \infty$
- if rows d_2th and d_1th of the seed matrix contain a one in a common column l, then $X_{(d_2,d_1)}$ is a difference between the number of ones in d_2th row in columns with indexes lower than l and the number of ones in d_1th row in columns with indexes greater than l, diminished by 1.
- if rows d_2th and d_1th of the seed matrix contain ones in more than one column, then the rule stated above applies for the column with the lowest index l.

Formally it can be stated as:

$$X_{(d_2,d_1)} = \begin{cases} \infty \iff \\ \qquad \forall_{l\in 1\ldots L} \quad w_{d_2,l} = 0 \vee w_{d_1,l} = 0 \\ \sum\limits_{i=1}^{l-1} w_{d_2,i} - \sum\limits_{i=l+1}^{L} w_{d_1,i} - 1 \iff \\ \qquad \exists_l : w_{d_2,l} = 1 \wedge w_{d_1,l} = 1 \end{cases} \tag{11}$$

With reference to the above example, for the case presented in Fig. 4, e.g. $X_{(2,1)} = 3 - 0 - 1 = 2$ and $X_{(3,2)} = \infty$.

We can observe that (in the case $X_{(d_2,d_1)} \neq \infty$) the value of $X_{(d_2,d_1)}$ represents the difference between the pipeline delay T_P and the number of required idle cycles. Let T^d_1 be the number of idle cycles before subiteration d is started with considering only awaiting for the completion of the previous $(d-1)$ subiteration. It can easily be shown that:

$$T^d_1 = \max\left[0,\ (T_P - X_{(d,\,d-1)})\right], \qquad 2 \le d \le D. \tag{12}$$

For the above example, $T^2_1 = T_P - 2$ because $X_{(2,1)} = 2$ and $T^3_1 = 0$ because $X_{(3,2)} = \infty$. So $X_{(d,d-1)} = \infty$ simply means that idle cycles are not required before subiteration d.

If awaiting for the completion of penultimate $(d-2)$ subiteration is considered ("red" row in Fig. 4), the number of idle cycles, denoted as T^d_2, equals:

$$T^d_2 = \max\left[0,\ (T_P - X_{(d,\,d-2)} - \deg(c_{d-1}) - T^{d-1}_{\text{idle}})\right], \qquad 3 \le d \le D, \tag{13}$$

where $\deg(c_{d-1})$ is degree of check node c_{d-1} in the seed graph, which is equal to the number of write cycles to the memory at subiteration $d-1$. The T^d_2 is (similarly to T^d_1) a difference between T_P and $X_{(d,\,d-2)}$, but diminished by the number of cycles exploited for subiteration $d-1$: the write to the memory cycles $\deg(c_{d-1})$ and the idle cycles T^{d-1}_{idle}.

Finally, sometimes the need for idle cycles is due to awaiting for completion of $d-3$ subiteration. In this case the number of idle cycles equals:

$$\begin{aligned} T^d_3 = \max\big[0,\ (T_P - X_{(d,\,d-3)} - \deg(c_{d-1}) - \\ - T^{d-1}_{\text{idle}} - \deg(c_{d-2}) - T^{d-2}_{\text{idle}})\big], \\ 4 \le d \le D. \end{aligned} \tag{14}$$

To determine the desired value of T^d_{idle}, the case among the mentioned above that gives the highest number of idle cycles has to be considered. Thus:

$$\begin{aligned} T^2_{\text{idle}} &= T^2_1, \\ T^3_{\text{idle}} &= \max\left[T^3_1,\ T^3_2\right], \\ T^d_{\text{idle}} &= \max\left[T^d_1,\ T^d_2,\ T^d_3\right], \qquad 4 \le d \le D. \end{aligned} \tag{15}$$

Equation (15) along with (12)–(14) show how to calculate the number of required idle cycles for the consecutive subiterations. For the particular decoder, calculations can be made according to (12)–(15) with increasing $d = 2, \ldots, D$ starting with $d = 2$.

5. Optimization of the parity check matrix

In order to minimize the total idle time of the decoder and hence substantially increase the decoder throughput, a heuristic optimization algorithm has been developed. The algorithm takes advantage of the following linear code properties:

- shuffling the rows of the parity check matrix does not change the code defined by the matrix at all,
- shuffling the columns of the parity check matrix does not change error performance of the code – it results only in adequate shuffling of the bit sequence in the codeword.

Thus the algorithm consist of sorting the seed matrix columns and rows in a way to minimize the total number of idle cycles in a full iteration, defined as:

$$T_{\text{idle}} = \sum_{d=1}^{D} T^d_{\text{idle}}. \tag{16}$$

The proposed algorithm is presented below as algorithm 1. In the first step, columns of $\mathbf{W}$ are sorted according to their growing weights (number of ones). It is motivated by the fact that more idle cycles are needed, when ones are located in a common columns with lower indexes (Fig. 4). Shifting columns with large weight to the right side of seed matrix decreases probability of this unfavorable situation.

Next, the algorithm searches for a matrix $\mathbf{W}'$ consisting of all rows of $\mathbf{W}$ arranged in a way that minimizes T_{idle} determined according to the Eq. (16). The first row of $\mathbf{W}'$ is selected randomly and every other (dth) is chosen among rows that ensure obtaining the lowest possible number of idle cycles T_{idle}^d determined as in (12)–(15). A random choice is made if more than one row satisfies this condition.

The described procedure is repeated a number of times and among obtained seed matrices, the one with the lowest T_{idle} is selected as a final result. As experiments have shown, 1000 repetitions is a sufficient number for obtaining satisfactory results (for practical number of rows in the seed matrix – up to a few hundreds). In most cases increasing this number above 1000 does not improve results significantly.

A large number of experiments have been performed in order to verify performance of the proposed optimization algorithm. Results of optimization for several seed matrices are shown in Tables 1 and 2. We present seed matrices with different sizes and coderates R, regular and irregular constructions. All seed matrices were constructed with Progressive Edge Growth algorithm [14]. The seed matrix $\mathbf{W}_{10\times 20}$ before and after optimization is shown in Fig. 5, where black squares indicate placement of nonzero elements. In this case all neighboring overlapping rows were eliminated by rows reordering procedure, thus for every d, $T_1^d = 0$, but still there exist a number of rows with nonzero T_2^d and T_3^d. Thus the total T_{idle} is pretty large, which is the case for all matrices with small sizes, due to their relatively dense placement of nonzero elements.

Table 1
Results: decoder idle cycles before and after optimization

Seed matrix	T_{idle}			
	$\mathbf{W}$	$\mathbf{W}'$ k_m=100	$\mathbf{W}'$ k_m=1000	$\mathbf{W}'$ k_m=5000
$\mathbf{W}_{10\times 20}$ ($R = 0.5$), reg.	122	71	67	67
$\mathbf{W}_{55\times 110}$ ($R = 0.5$), reg.	217	0	0	0
$\mathbf{W}_{32\times 64}$ ($R = 0.5$), irreg.	311	10	2	2
$\mathbf{W}_{64\times 128}$ ($R = 0.5$), irreg.	813	0	0	0
$\mathbf{W}_{30\times 90}$ ($R = 0.66$), irreg.	318	17	16	16
$\mathbf{W}_{16\times 64}$ ($R = 0.75$), reg.	253	86	80	78
$\mathbf{W}_{25\times 100}$ ($R = 0.75$), irreg.	305	39	35	34

Table 2
Results: decoder throughput before and after optimization

Seed matrix	Throughput	
	$\mathbf{W}$	$\mathbf{W}'$ k_m=1000
$\mathbf{W}_{10\times 20}$ ($R = 0.5$), reg.	$P \cdot 0.797$ [Mb/s]	$P \cdot 1.14$ [Mb/s]
$\mathbf{W}_{55\times 110}$ ($R = 0.5$), reg.	$P \cdot 1.46$ [Mb/s]	$P \cdot 2.41$ [Mb/s]
$\mathbf{W}_{32\times 64}$ ($R = 0.5$), irreg.	$P \cdot 0.888$ [Mb/s]	$P \cdot 2.18$ [Mb/s]
$\mathbf{W}_{64\times 128}$ ($R = 0.5$), irreg.	$P \cdot 0.727$ [Mb/s]	$P \cdot 1.99$ [Mb/s]
$\mathbf{W}_{30\times 90}$ ($R = 0.66$), irreg.	$P \cdot 1.42$ [Mb/s]	$P \cdot 2.79$ [Mb/s]
$\mathbf{W}_{16\times 64}$ ($R = 0.75$), reg.	$P \cdot 1.56$ [Mb/s]	$P \cdot 2.56$ [Mb/s]
$\mathbf{W}_{25\times 100}$ ($R = 0.75$), irreg.	$P \cdot 1.77$ [Mb/s]	$P \cdot 3.19$ [Mb/s]

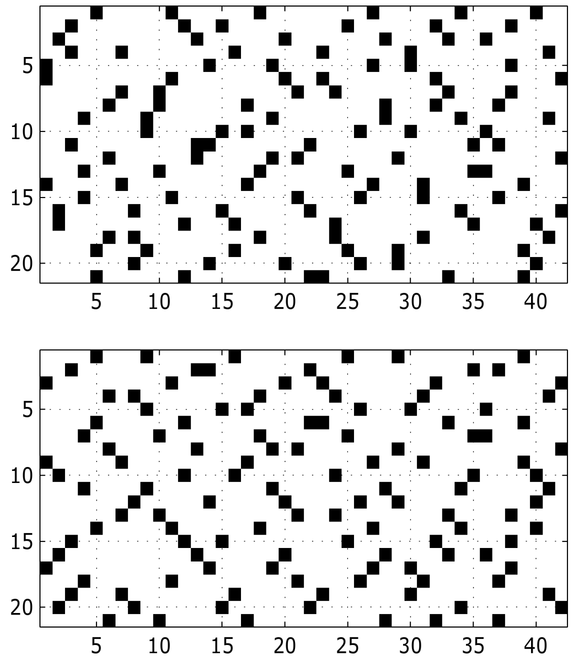

Fig. 5. Matrix $\mathbf{W}_{10\times 20}$ before and after optimization

Table 1 presents total idle times for original matrix $\mathbf{W}$ as well as matrices obtained with the algorithm 1. We included results for different number of repetitions ($k_m = 100, 1000, 5000$) of the main loop in the algorithm. As was mentioned, more than 1000 repetitions didn't improve the results in most cases and in the other few cases the improvement was really insignificant.

Algorithm 1: Seed matrix optimization in order to minimize the total idle time of the decoder T_{idle}

Input: Seed matrix $\mathbf{W}_{D\times L}$, pipeline delay T_P
Output: Optimized seed matrix $\mathbf{W}'_{D\times L}$

Sort columns of $\mathbf{W}$ according to their weights: on the left side of $\mathbf{W}$ columns with the lowest weight.
$T_{\text{opt}} := \infty$
for *k=1,...,1000* **do**
  $\mathbf{W}' := \mathbf{0}$
  Select randomly a row of $\mathbf{W}$ and fix it as the first row of $\mathbf{W}'$.
  for $d = 2, \ldots, D$ **do**
    Among rows of $\mathbf{W}$ not included in $\mathbf{W}'$ yet, select a row that fixed as a dth row of $\mathbf{W}'$ gives the lowest number of idle cycles T_{idle}^d (15). If more than one row satisfies this condition, then make a random selection among them.
    Fix the selected row as a dth row of the matrix $\mathbf{W}'$.
  Determine the total number of idle cycles T_{idle} for matrix $\mathbf{W}'$, according to the eq. (16).
  if $T_{\text{idle}} < T_{\text{opt}}$ **then**
    $\mathbf{W}_{\text{opt}} := \mathbf{W}'$
    $T_{\text{opt}} := T_{\text{idle}}$
$\mathbf{W}' := \mathbf{W}_{\text{opt}}$

It can be seen that the number of required idle cycles after seed matrix optimization is very low, compared to this number before optimization. In many cases $T_{\text{idle}} = 0$ can be obtained, especially for codes with lower rates ($R = 0.5$ or less). For codes with higher rates R, some greater than 0 idle time is usually necessary, because due to lower number of rows in $\mathbf{H}$ and their higher weights, it is harder to select "non-overlapping" rows and thus to bring the constituent T^d_{idle} numbers to zero. Moreover we can observe that for matrices with larger sizes, the results obtained are better (for example: compare $\mathbf{W}_{16\times64}$ and $\mathbf{W}_{25\times100}$), which is related to the greater sparsity.

As a result of the idle time reduction, the decoder throughput is significantly increased, which is shown in Table 2. The presented throughput values are dependent on the number of SISO units P used for implementation (P is usually equal to the permutation submatrix size). The throughput has been determined making use of synthesis results for Xilinx XC2V3000 FPGA device, where we achieved $f_{\text{clk}} = 145\,\text{MHz}$ for a normalized Min-Sum messages calculation algorithm [15] used in SISO implementation. Equation (17) shows the relationship between actual throughput TH (the number of information bits decoded per second) and the parameters of the code and decoder. The sum $\sum\limits_d \sum\limits_l h_{d,l}$ is equal to the number of working clock cycles, so the denominator in Eq. (17) is the total number of cycles for single word decoding. For the results presented in Table 2 we assumed the iteration number $i_{\max} = 10$, which is a common assumption for throughput calculation [9]. The numerator in Eq. (17) is the number of information bits in a single word $M = P(L - D)$.

$$TH = f_{\text{clk}} \frac{P \cdot (L-D)}{\left(\sum\limits_{d=1}^{D} \sum\limits_{l=1}^{L} h_{d,l} + T_{\text{idle}}\right) \cdot i_{\max}} \tag{17}$$

The typical number of SISO units is in range $P = 10\ldots100$. For example, a decoder with $P = 32$ SISO units, wordlength $W = 6$, occupies about $1/3$ of the available resources of the mentioned XC2V3000 device. For a rate $R = 0.5$ code the throughput is then more than 60 Mb/s and it is about twice as much as for the decoder without proposed parity check matrix optimization. Thus the presented throughput increase is the main virtue of the work presented in this paper.

6. Conclusions

In the first part of this article we briefly described LDPC decoder architecture based on TDMP decoding scheme. The main drawback of the straight pipelined implementation is the necessity for idle cycles that reduce the throughput. However, as was shown, by means of proper parity check matrix columns and rows reordering, the number of required idle cycles can be significantly reduced. The heuristic algorithm for such optimization of the parity check matrix has been developed. The goal of the algorithm is minimization of the total idle time of the decoder. We performed a large number of experiments, for different seed matrices. The achieved reduction of the idle time is always significant, in many cases even to zero. The resulting decoder throughput increase is also meaningful. Hence the full advantage of the speedup due to pipelined processing can then be taken.

REFERENCES

[1] R.G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, 1963.
[2] D.J.C. MacKay, "Good error-correcting codes based on very sparse matrices", *IEEE Trans. Inf. Theory* 45 (2), 399–431 (1999).
[3] M.M. Mansour and N.R. Shanbhag, "High throughput LDPC decoders", *IEEE Trans. VLSI Syst.* 11 (6), 976–996 (2003).
[4] H. Zhong and T. Zhang, "Block-LDPC: A practical LDPC coding system design approach", *IEEE Trans. Circuits Syst.* 1, 52 (4), 766–775 (2005).
[5] W. Sułek, "Seed graph expansion for construction of structured LDPC codes", *IEEE Int. Symposium on Wireless Communication Systems (ISWCS)* 1, 216–220 (2009).
[6] S. Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd Edition, Prentice-Hall Inc., New Jersey, 2004.
[7] J. Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes", *IEEE Trans. on Communications* 53 (8), 1288–1299 (2005).
[8] M.M. Mansour, "A turbo-decoding message-passing algorithm for sparse parity-check matrix codes", *IEEE Trans. Signal Process.* 54 (11), 4376–4392 (2006).
[9] M.M. Mansour and N.R. Shanbhag, "A 640-Mb/s 2048-Bit programmable LDPC decoder chip", *IEEE J. Solid-State Circuits* 41 (3), 684–698 (2006).
[10] L. Yang, H. Liu, and C.J.R. Shi, "Code construction and FPGA implementation of a low-error-floor multi-rate low-density parity-check code decoder", *IEEE Trans. Circuits Syst.* 1, 53 (4), 892–903 (2006).
[11] M. Rovini, N.E. L'Insalata, F. Rossi, and L. Fanucci, "VLSI design of a high-throughput multi-rate decoder for structured LDPC codes", *DSD 2005 Euromicro Conf. Digital System Design* 1, 202–209 (2005).
[12] X.-Y. Hu, E. Eleftheriou, D.M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes", *IEEE Globecom* 1, 1036–1036E (2001).
[13] W. Sułek and D. Kania, "Code construction algorithm for architecture aware LDPC codes with low-error-floor", *Proc. IEEE Region 8 Int. Conf. on Computational Technologies in Electrical and Electronics Engineering – SIBIRCON 2008* 1, 1–6 (2008).
[14] X.-Y. Hu, E. Eleftheriou, and D.M. Arnold, "Regular and irregular progressive edge-growth tanner graphs", *IEEE Trans. Inf. Theory* 51 (1), 386–398 (2005).
[15] J. Chen and M.P.C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes", *IEEE Trans. on Communications* 50 (3), 406–414 (2002).