Chapter 5

Design Trade-Offs for FPGA Implementation of LDPC Decoders

Alexandru Amaricai and Oana Boncalo

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/66085

Abstract

Low density parity check (LDPC) decoders represent important throughput bottlenecks, as well as major cost and power-consuming components in today's digital circuits for wireless communication and storage. They present a wide range of architectural choices, with different throughput, cost, and error correction capability trade-offs. In this book chapter, we will present an overview of the main design options in the architecture and implementation of these circuits on field programmable gate array (FPGA) devices. We will present the mapping of the main units within the LDPC decoders on the specific embedded components of FPGA device. We will review architectural trade-offs for both flooded and layered scheduling strategies in their FPGA implementation.

Keywords: forward error correctionLDPC decoder, FPGA, digital circuits

1. Introduction

Low density parity check (LDPC) codes are a class of capacity approaching codes which provide increased error correction capability for both binary symmetric channel (BSC) and binary-input additive white Gaussian noise (BIAWGN) channel models [1]. Therefore, LDPC codes are used in a wide range of standards for both wireless communication [2]—WiFi, WIMAX, DVB-S2, etc—as well as for FLASH-based storage systems [3].

Decoding of LDPC codes is performed in an iterative manner, using message passing algorithms [4, 5]. These algorithms rely on simple computations—additions and comparisons on a small number of bits—which are performed on dedicated computational nodes. Although the node level computational complexity is low, LDPC codes implemented in communication and storage standards employ thousands or tens of thousands of such computational nodes, which leave a wide range of design options and trade-offs for the implementation of decoding

architectures [2]. These trade-offs take into account the throughput, error correction capability, cost of the hardware implementation, and power consumption.

In this chapter, we will present the most important architectural options for both flooded and layered LDPC decoders implemented on field programmable gate array (FPGA) devices. The implementation of LDPC decoders on such devices is motivated by the increased flexibility of FPGAs, which make them suitable to implement highly versatile solutions in both wireless communications—such as software defined radios—and storage systems—such as software defined storage—as well high level of parallelism degree for fixed-point computations, which ensure the possibility of obtaining high throughputs for the decoders.

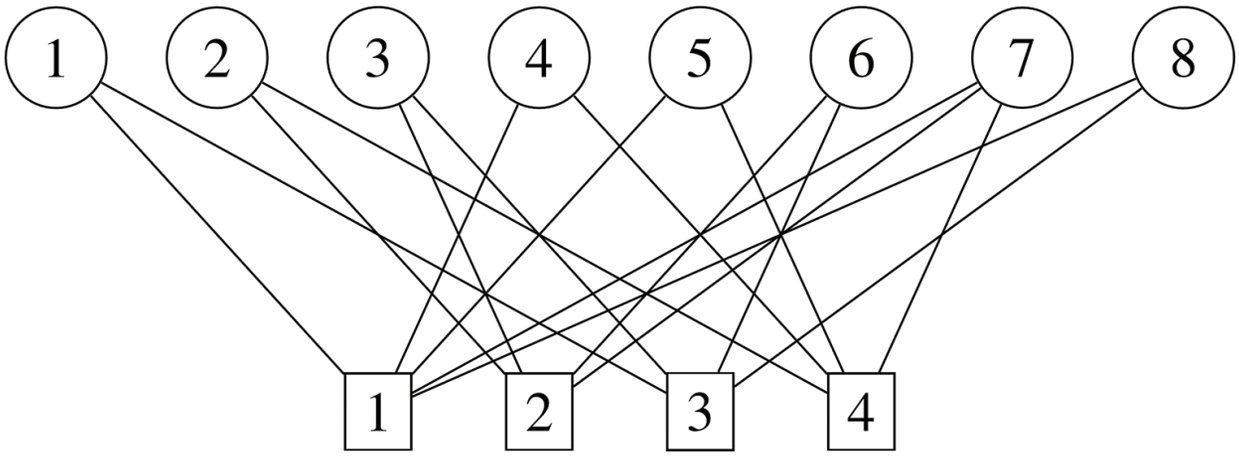
This book chapter is organized as follows: Section 2 presents the algorithms for the LDPC decoding, as well as the strategies for it; Section 3 summarizes the main features and building blocks of modern FPGA devices; Section 4 presents the implementation and design trade-offs for FPGA-based flooded LDPC decoding architectures; layered architectures are detailed in Section 5; last section is dedicated to the concluding remarks.

2. Theoretical background of LDPC decoding

LDPC codes are a class of linear algebraic codes, defined by a sparse parity check matrix H [1]. The LDPC code can also be represented by a bipartite graph, called the Tanner graph [6]. This graph contains two types of nodes: variable or bit nodes—corresponding to the columns in the H matrix and the codeword bits—and check nodes—corresponding to the rows in the H matrix and the parity check equations. A check node is connected to a variable node if the corresponding value in the parity check matrix is nonzero. **Figure 1** depicts a simple parity check matrix and its associated Tanner graph. LDPC decoding is performed in an iterative manner, consisting of the message exchange between the check and variable nodes along the edges of the Tanner graphs in several rounds or iterations. This type of decoding is called message passing (MP) decoding [4]. LDPC codes defined in communication or storage standards use parity check matrices consisting of thousands of columns, such as the 2304 columns for WiMAX, 64800 columns for DVB-S2, or 1944 columns for WiFi. The number of nonzero entries on each column represents the variable node degree—d_v —, while the number of the nonzero elements on each row in the H matrix represents the check node degree—d_c. An LDPC code is said to be regular if all the rows/columns in the parity check matrix contain an equal number of nonzero entries; otherwise, the LDPC code is irregular.

In order to enable efficient hardware implementations, quasi-cyclic LDPC (QC-LDPC) codes are used in most of the standards [7]. These subclasses of LDPC codes present highly structured parity check matrices, defined by blocks of circulant matrices. A QC-LDPC code is defined by a base matrix B, consisting of -1 elements and nonnegative elements. The parity check matrix H is obtained from the matrix B in the following way: -1 elements are expanded by $z * z$ all 0 matrix, while nonnegative elements within the matrix B are expanded by the $z * z$ identity matrix permutated with the nonnegative element value. The coefficient z is known as the expansion factor for the QC-LDPC code. **Figure 2** depicts the B matrix for the WiMAX

http://dx.doi.org/10.5772/66085



$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 1. Parity check matrix and its associated Tanner graph.

-1	94	73	-1	-1	-1	-1	-1	55	83	-1	-1	7	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	27	-1	-1	-1	22	79	9	-1	-1	-1	12	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	24	22	81	-1	33	-1	-1	-1	0	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1
61	-1	47	-1	-1	-1	-1	-1	65	25	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1
-1	-1	39	-1	-1	-1	84	-1	-1	41	72	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1
-1	-1	-1	-1	46	40	-1	82	-1	-1	-1	79	0	-1	-1	-1	-1	0	0	-1	-1	-1	-1
-1	-1	95	53	-1	-1	-1	-1	-1	14	18	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1
-1	11	73	-1	-1	-1	2	-1	-1	47	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1
12	-1	-1	-1	83	24	-1	43	-1	-1	-1	51	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1
-1	-1	-1	-1	-1	94	-1	59	-1	-1	70	72	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1
-1	-1	7	65	-1	-1	-1	-1	39	49	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
43	-1	-1	-1	-1	66	-1	41	-1	-1	-1	26	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	0

Figure 2. Base matrix for WiMAX rate ½ LDPC code [2].

LDPC code, rate ½, with 2304 columns and 1152 rows, and an expansion factor of 96. A horizontal layer of H matrix is defined as the set of z consecutive rows which correspond to one row within the base matrix. Composite layers, consisting of integer multiples of z rows within the parity check matrix, may be also used.

MP LDPC decoding may be performed using different scheduling strategies. These strategies indicate the order in which the check node and variable node computations are performed during the decoding iterations [13]. Two types of strategies may be employed: flooded and layered. Flooded decoding represents the conventional approach for decoding: each iteration consists of the update of messages at the check nodes, which subsequently pass their output messages to the variable nodes, which, in turn, update their corresponding messages [5]. Using this strategy, both the variable nodes and the check nodes are updated once per iteration. The

layered scheduling consists of splitting the parity check matrix in horizontal layers; these layers are processed in a serial manner, while the check node updates within the same layer are processed in a similar manner with the flooded scheduling [8]. The variable node updates are performed after each layer processing. Therefore, in layered scheduling, the updates per iteration at variable node level are equal to the number of layers. Layered scheduling has two major advantages with respect to flooded: (i) faster convergence and (ii) reduced memory requirement [8]. The flooded approach has the advantage of increase resilience to faults in the hardware architectures [9], as well as the possibility for very high throughputs due to the high level of parallelism at the decoder level.

LDPC decoding can be performed by different types of algorithms, with different error correction capabilities. These can be split into two major classes [13]:

1. Hard-decision algorithms: These algorithms rely on 1-bit messages exchanged between the processing units. Such algorithms include bit-flipping, gradient descent and probabilistic gradient descent bit-flipping, Gallagher-A and Gallagher-B. The advantage of these algorithms is represented by the low requirements in terms of resource usage and power consumption. Their main drawback is represented by their low error correction capability with respect to soft-decision algorithms, for both BSC and BIAWGN channel models.

2. Soft-decision algorithms: These algorithms use messages quantized on several bits (usually between 3 and 7), which are exchanged between the variable nodes and check nodes. The hardware implementations for soft-decision algorithms are significantly more costly with respect to the hard-decision versions. However, using soft decoding, LDPC codes are able to have the capacity approaching error correction capabilities which make them suitable candidates for a wide range of communication standards.

In this chapter, we will discuss the implementation aspects related to the soft-decision-based LDPC decoders. The most important class of soft-decision LDPC decoding is represented by the min-sum (MS) algorithm [13] and its variants: offset MS (OMS) [10], normalized MS (NMS) [10], self-correcting MS (SCMS) [11], and finite alphabet iterative decoding (FAID) [12]. In these algorithms, the following messages are used [13]:

1. Input log-likelihood-ratio (LLR): These messages represent the input from the communication channel. For BSC channel model—used in storage systems—the input LLR is on 1 bit, while for BIAWGN channel model—used in wireless communication—the input LLR is quantized on several bits. The input LLR is denoted as γ and is quantized on $quant(\gamma)$ bits.

2. Variable node messages: These messages are the outputs of the check node units and serve as inputs for the variable node units. These messages are denoted as α and are quantized on $quant(\alpha)$ bits.

3. Check node messages: These messages represent the output of the variable nodes and are the inputs for the check nodes. These messages are denoted as β and are quantized on $quant(\beta)$ bits.

4. A posteriori LLR (AP-LLR): These messages represent the output of each decoding iteration/layer. The output of the decoder is given by the sign of the AP-LLR. It is denoted as $\tilde{\gamma}$.

http://dx.doi.org/10.5772/66085

Flooded MS decoding of LDPC codes consists of several iterations, where each variable node message—and check node message—is updated once. Each iteration consists of the following steps [5, 13]:

1. Variable node update

$$\alpha_{i,j} = \gamma_i + \sum_{k\epsilon\{C(i)\backslash j\}} \beta_{i,k}, \forall j \epsilon C(i) \tag{1}$$

$$\tilde{\gamma}_i = \gamma_i + \sum_{k\epsilon C(j)} \beta_{i,k}, \forall j \epsilon C(i) \tag{2}$$

2. Check node update

$$sign(\beta_{l,j}) = \prod_{k\epsilon\{V(l)\backslash j\}} sign(\alpha_{l,k}), \forall j \epsilon V(l) \tag{3}$$

$$|\beta_{l,j}| = \min(\alpha_{l,k}), \forall j \epsilon V(l), k\epsilon\{V(l)\backslash j\} \tag{4}$$

$C(i)$ denotes all the check node messages connected to the variable node i, while $V(l)$ denotes all the variable node messages connected to the check node l. The number of variable nodes is equal to number of columns in the parity check matrix, while the number of check nodes is equal to the number of rows in the H matrix.

Layered decoding is performed layer by layer, each layer consisting of the following steps [8, 13]:

1. Variable node update

$$\alpha_{i,j} = \tilde{\gamma}_i - \beta_{i,j}, \forall j \epsilon V(i) \tag{5}$$

2. Check node update

$$sign(\beta_{i,j}) = \prod_{k\epsilon\{V(i)\backslash j\}} sign(\alpha_{i,k}), \forall j \epsilon V(i) \tag{6}$$

$$|\beta_{i,j}| = \min(\alpha_{i,k}), \forall j \epsilon V(i), k\epsilon\{V(i)\backslash j\} \tag{7}$$

3. AP-LLR update

$$\tilde{\gamma}_i = \alpha_{i,j} - \beta_{i,j}, \forall j \epsilon V(i) \tag{8}$$

Both for flooded and layered scheduling, decoding is stopped either when a codeword is found—all the parity check equations are satisfied—or when the maximum number of iterations is reached.

The MS decoding, in both layered and flooded strategies, comprises of simple arithmetic operations, performed on small operands (3–8 bits). The variations of the MS algorithms target decoding performance improvement. OMS and NMS are based on the fact that the minimum computation at the check node level represents an overestimation of the check node message [10]. Therefore, both approaches try to reduce the value of the check node message computed by the check node unit.

The OMS approach uses a -1 addition from the absolute value of the $\beta_{i,j}$ in order to reduce its value. The check node computation in the OMS algorithm becomes:

$$sign(\beta_{i,j}) = \prod_{k\epsilon\{V(i)\backslash j\}} sign(\alpha_{i,k}), \forall j \in V(i) \tag{9}$$

$$|\check{\beta}_{i,j}| = \min(\alpha_{i,k}), \forall j \in V(i), k\epsilon\{V(i)\backslash j\} \tag{10}$$

$$|\beta_{i,j}| = |\check{\beta}_{i,j}| - 1 \tag{11}$$

The NMS approach uses scaling of the absolute value of the $\beta_{i,j}$ in order to reduce its value, by a normalization factor λ (usually with the values of 0.75 or 0.875) multiplication. The check node computation in the NMS algorithm becomes:

$$sign(\beta_{i,j}) = \prod_{k\epsilon\{V(i)\backslash j\}} sign(\alpha_{i,k}), \forall j \in V(i) \tag{12}$$

$$|\check{\beta}_{i,j}| = \min(\alpha_{i,k}), \forall j \in V(i), k\epsilon\{V(i)\backslash j\} \tag{13}$$

$$|\beta_{i,j}| = \lambda * |\check{\beta}_{i,j}| \tag{14}$$

SCMS represents an approach which aims at improving the error correction capability by erasing the variable node messages which change their sign after an iteration [11]. The erasure process cannot be performed in two consecutive iterations. The modification of the variable node update for a layered scheduling for the SCMS algorithm is:

$$\alpha_{i,j}^{new} = \tilde{\gamma}_i - \beta_{i,j}, \forall j \in V(i) \tag{15}$$

$$e_{i,j}^{new} = (!e_{i,j}^{old}) \& (sign(\alpha_{i,j}^{new}) \oplus sign(\alpha_{i,j}^{old})) \tag{16}$$

$$\alpha_{i,j} = \begin{cases} \alpha_{i,j}^{new}, e_{i,j}^{new} = 0 \\ 0, e_{i,j}^{new} = 1 \end{cases} \tag{17}$$

FAID decoding aims at improving the error floor region of the LDPC decoding. It changes the variable node operations, by implementing nonlinear dedicated function for the variable node message update, based on the channel information and the check node messages [12]. For a flooded scheduling, the variable node processing becomes:

http://dx.doi.org/10.5772/66085

$$\alpha_{i,j} = FAID(\gamma_i, \beta_{i,k}), \forall j \epsilon C(i), \forall k \epsilon \{C(i) \backslash j\} \tag{18}$$

$$\tilde{\gamma}_i = \gamma_i + \sum_{k \epsilon C(j)} \beta_{i,k}, \forall j \epsilon C(i) \tag{19}$$

The implementation of the *FAID* function is done using dedicated look-up tables (LUT). The complexity of these tables is dependent on the check node message quantization and the variable node degree d_v.

3. Architectural components of FPGA devices

FPGAs are digital devices with a programmable structure. This programmable structure provides FPGAs with very high flexibility, which makes them the ideal candidates for prototyping, as well as products with very low time-to-market constraints or applications which require high degree of flexibility. Furthermore, FPGAs have a built-in structure which allows a high degree of parallelization for applications that rely on fixed-point computations.

The main digital building blocks of modern FPGA devices are the configurable logic block (CLB), the embedded memory block RAM (BRAM), and the DSP block. DSP blocks implement 18 bit or wider multiplication, multiply-accumulate or multiply-add fused, and addition operations [14]. Because they are optimized for operand sized of 18 bit or more, and mainly for multiplication-based operations, they are of little use for the implementation of LDPC decoders.

CLBs are the main logic resource, which implement both sequential and combinational logic elements [15]. Usually, CLBs are composed of several slices, each of the slice being composed of a look-up table (LUT) and a D flip-flop, plus additional dedicated logic, such as logic and dedicated wire for ripple carry addition. The combinational logic is implemented using LUT, with modern FPGAs having six-input LUTs. Therefore, in a LUT and flip-flop pair, six-input combinational functions have the same cost as one or two input combinational functions. For specific families, the LUT can also be used as a memory circuit such as the distributed RAM in Xilinx FPGAs. The D flip-flop is used as the basic sequential logic. Because the combinational logic is paired with the D flip-flop in the same structural unit, pipelining can be easily and without significant resource consumption implemented in modern FPGA devices.

Another important feature of modern FPGAs is represented by the built-in memory blocks [16]. For large memories, FPGAs include the block RAM, which is block of 9 or 18 kbits. They have configurable width (9, 18, 36, or 72 bit), with the depth of the BRAM being determined by the width (for an 18 kbit BRAM and 72 bit word, the depth is 512 words). The number of BRAMs for a design is highly dependent on the width and the depth of required memory. For example, a memory which requires 96 bit words, and only 64 words, will consume 2 BRAM blocks, although the number of memory bits is significantly less with respect to the number of memory bits in a BRAM. Another important issue of the BRAM block is the number of read/

write ports: it is optimized for 1 read and 1 write port. The maximum number of memory ports for a BRAM is 2 read and 2 writes, but with limitations in the size of the word. For memories with few bits, and/or memories with a high number of ports, the distributed RAM implemented in CLBs is used.

From an LDPC decoder perspective, the FPGA implementation will make use of the CLBs for the implementation of the processing nodes and the routing network, and memories, either BRAM or distributed RAM.

4. Flooded LDPC decoders

The straightforward LDPC decoder architecture is represented by the hardware implementation of the corresponding Tanner graph. This type of architecture is known as the fully parallel decoder [17]. It consists of:

1. Processing nodes: A fully parallel decoder contains a number of variable node units equal to the number of columns in the parity check matrix and a number of check node units equal to the number of rows in the H matrix.
2. Routing network: The routing network is represented by wires which connect the variable node units with the check node units, according to the parity check matrix.

Although this kind of architecture is straightforward, the main problem arises due to the routing network. For LDPC codes that have thousands of rows and columns in the parity check matrix, the routing network involves tens of thousands of connections between the variable node units and check node units. Furthermore, the H matrix presents an irregular structure, which makes the interconnections component highly irregular. This will further contribute to the increase in cost, as well as reduction in the maximum operating frequency—due to the routing delay across the routing components of the FPGA. Another disadvantage of fully parallel LDPC decoder is the low flexibility: the decoder is specific to a LDPC code, and a slight modification in the code leads to the entire decoder redesign. Furthermore, these types of architecture cannot easily accommodate features such as multi-rate decoder, which is desired due to the fact that each communication and storage standard uses multiple LDPC codes with different rates. The main advantage of this architecture is represented by its high throughput, due to low number of clock cycles required for an iteration [17].

In order to reduce the complexity of these decoders, one approach relies on the reduction of the wires between the check node unit and variable node units. One such solution relies on the bit-serial decoder: the check node messages and the variable node messages are sent bit by bit to their corresponding processing unit [18]. Thus, the connection between a variable node unit and a check node unit consists of only two wires, instead of a $quant(\alpha)$ bit and a$quant(\beta)$ bit wires. This decoder trades throughput for reduced cost. Other solution relies on reduced quantization for the messages [19, 20]. The reduced quantization leads to a reduced number of wires between the processing units and thus to a reduction in the interconnection network. These solutions trade the error correction capability for reduced cost.

http://dx.doi.org/10.5772/66085

The other approach to reduce the complexity and the cost of the flooded LDPC decoder relies on the serialization of the check node and variable node operations at different levels. Thus, partially parallel flooded architectures are employed [21–28]. These partially parallel decoders exploit the regular structure of the QC-LDPC codes in order to obtain regular, low complexity architectures. Because serialization is employed at different levels, messages have to be stored in dedicated memory units. Stored messages have to be routed from the memory blocks to the processing units according to the LDPC matrix. In order to provide a flexible way for message routing, barrel shifters are employed. The read/write addresses for the memories, as well as the shift amounts employed in routing, are generated from a dedicated control unit. The main components for a partial parallel flooded decoder are as follows:

1. Processing nodes: The number of variable node units and check node units is dependent on the different parallelism degrees at different level. Furthermore, the number of inputs and outputs for such units can also vary, depending on how many messages can be processed each clock cycle.

2. Routing network: The routing is implemented using barrel shifters. The number and size of the barrel shifters may vary with message quantization, circulant size of the base matrix, different level parallelization degrees, etc. High-frequency pipelined barrel shifters may be implemented without additional cost in modern FPGA devices due to the LUT and D flip-flop pair which compose the basic component of the CLB.

3. Memory blocks: Memory blocks are used to store both the input LLRs and the check node and variable node messages. Usually, high degrees of parallelism—increased throughput—require wide memory words and multi-port memories. In many implementations, the multi-port memories are replaced by independent memory banks, which can be easily mapped on the FPGA BRAM blocks.

4. Control unit: The control unit is used to generate the shift amounts, the read/write memory addresses, as well as the control signals for the processing units. The shift amounts and the memory addresses are code dependent; this kind of information is usually stored in dedicated ROM memories.

For a quasi-cyclic LDPC decoder, two types of partial parallel flooded architectures have been proposed:

1. Parallel circulant, serial row/column processing: In this type of architecture, a number of z rows/columns are processed in parallel, while the rows and columns of the base matrix are processed sequentially [21–24]. This decoder is depicted in **Figure 3**. This kind of architecture requires z variable node units and z check node units. The memory words will consist of z messages. An important design parameter is represented by the parallelism degree at the processing node level—the number of processed messages per clock cycle. For the variable node unit, the maximum parallelism degree is d_v, while for the check node unit is d_c. Increasing parallelism at the processing node level will greatly influence the FPGA resource consumption of the decoder. This is due to the increased number of barrel shifters, which will lead to an increase in the conventional slice-based resource consumption, as well as for the increase in the number of memory ports, or the number of memory banks.

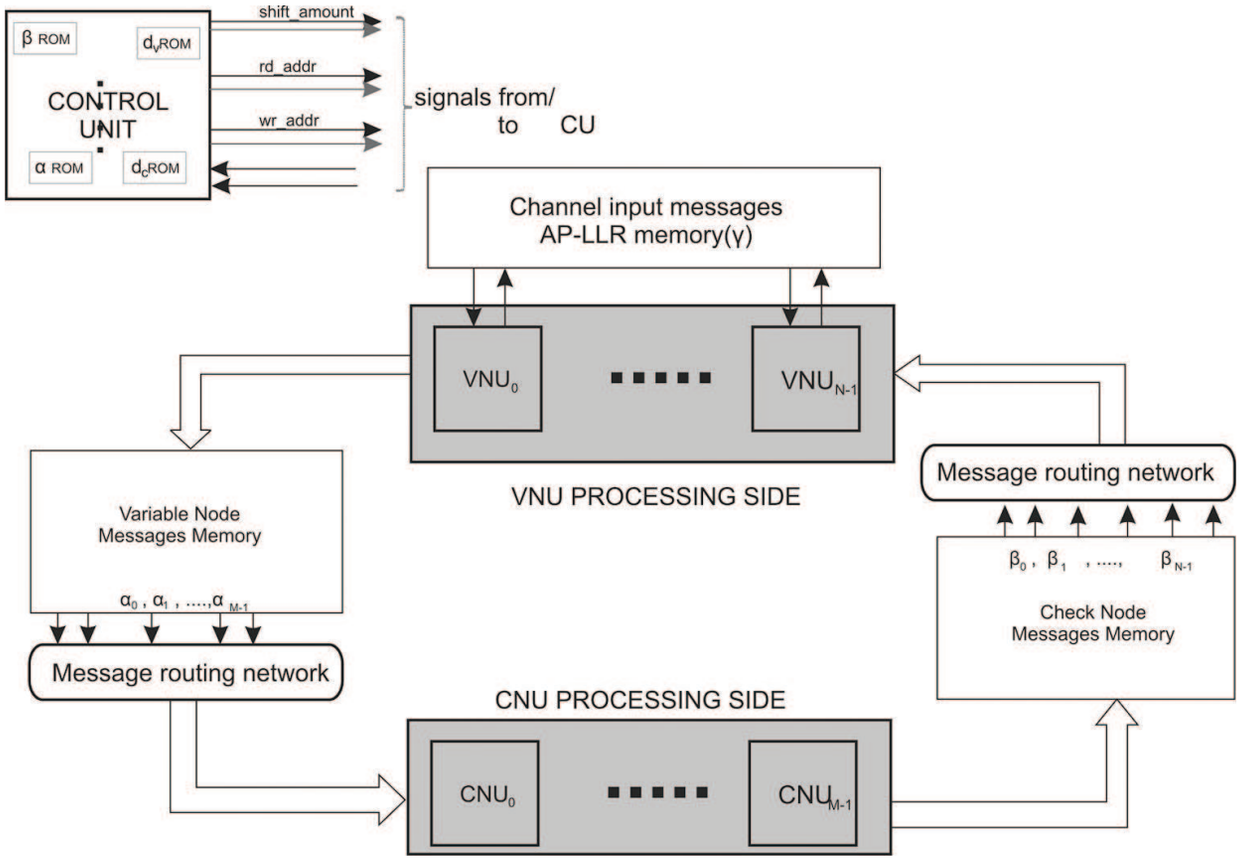


Figure 3. Parallel circulant, serial row/column processing flooded architecture.

Increasing the number of memory ports will lead to the implementation of the message memories with distributed RAM, while the increase in the memory banks will lead to an increase in the number of BRAM blocks.

2. Serial circulant, parallel row/column processing: In this kind of architecture, the rows/columns of the base matrix are processed in parallel, while the elements corresponding to a vertical/horizontal layer are processed sequentially [24–28]. This type of architecture is depicted in **Figure 4**. The number of check node units is equal to the number of rows in the B matrix, while the number of variable node units is equal to the number of columns in the base matrix. The number of columns in the base matrix gives also the number of input LLR message memories, while the variable and check node messages are stored in a $d_v nr\_col(B)$ memory blocks. Each memory has a depth equal to the circulant size and a width equal to the message quantization. This type of memory organization is suitable for FPGA devices, as each memory block maps to a BRAM block. This kind of decoder does not use dedicated routing circuits, as the routing of the messages between the memory blocks and the processing units is done via the offset address within each memory block. The processing units are fully parallel, as the read/write operations are done from d_v or d_c memory blocks. In order to increase the throughput, vectorization technique is proposed [25, 26]. This technique relies on packing multiple messages within a single memory word, which to be processed in parallel. Increasing the vectorization degree will lead to alignment problems,

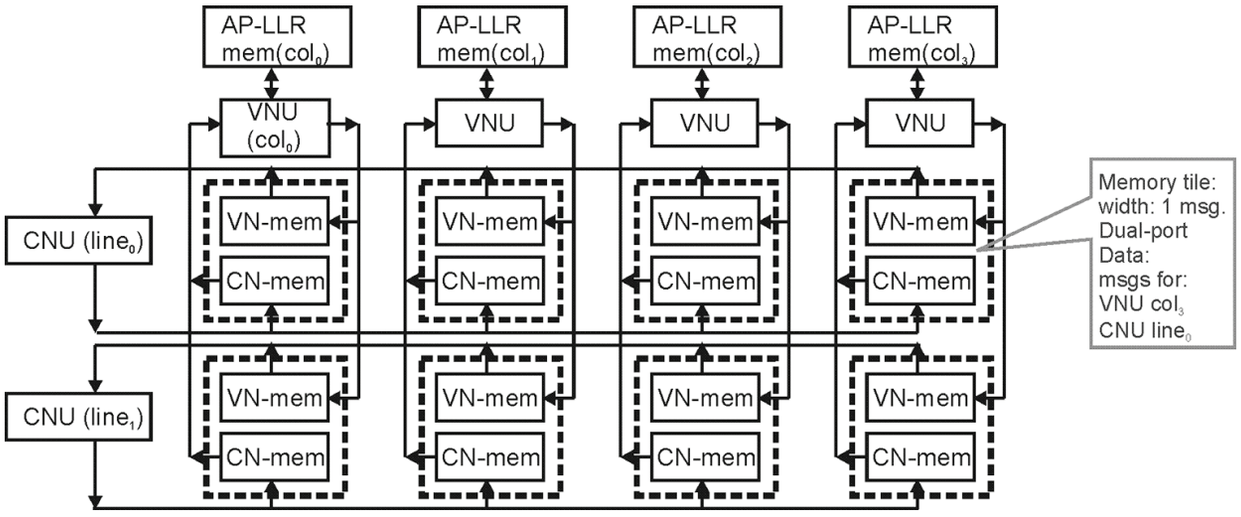


Figure 4. Serial circulant, parallel row/columns processing flooded architecture.

which lead to increased additional logic, as well as the number of stall clock cycles. Therefore, the maximum number of packed messages used with vectorization has been limited to four.

Partial parallel flooded FPGA architectures have two drawbacks:

1. Idle times for processing units: A major disadvantage of flooded decoder is represented by significant idle times for both variable node and check node units, during the variable node processing, the check node units, and vice-versa. Therefore, during one decoding iteration, only half of the decoder is utilized. Two strategies are employed:
   a. Processing two different codewords in parallel [22, 23]—while variable nodes compute the variable node messages for one codeword, the check nodes compute the check node messages for a second codeword; this solution implies small changes in the control unit, a double memory for the input LLR messages, and the hard-decision bits, with the advantage of a double throughput.
   b. Using waiting time minimization algorithms [25, 26]—using these algorithms, the order in which the rows/columns within the base matrix or within the parity check matrix are processed can be determined, without having data hazards and memory conflicts when performing the variable node and check node updates; therefore, almost simultaneous variable node and check node processing can be achieved; a second optimization obtained by employing these types of algorithms is represented by reduced memory usage; because data hazards and memory conflicts are avoided, the check node messages and variable node messages can be stored in the same memory locations.
2. Low usage of BRAM memories: In parallel circulant, serial row/column processing architectures, the memory word for the variable node messages is $zquant(\alpha)$, while the number of memory words is $d_v nr\_col(B)$. For LDPC code with circulant size of 96, 24 columns in the base matrix, $d_v = 3$, and message quantization of 4 bits, the word size is 384 bits, while the number of words in the memory is 72. For BRAM blocks consisting of 72 bits memory

words and 512 words, this kind of configuration results in the usage of 6 BRAM block, with only 72/512 utilization for each BRAM. For the second type of flooded architectures, for the same LDPC code, for each memory block required to store the variable node messages, the memory word is of 4 bits, while the number of words is 96. Also, in this case, it can be observed that the BRAM has poor usage. Several approaches have been proposed to address this issue. One is to use multiple codewords. The solution in [23] targets the increase in the memory words within the BRAM. The codewords are processed in serial. This solution achieves increase in the BRAM utilization for the same logic usage and throughput. The solution in [28] targets increase in the memory word size stored in the BRAM and addresses serial circulant, parallel row/column processing architectures. In the same memory word are stored messages from multiple codewords. The number of processing units is increased in order to process in parallel the codewords. This solution results in an increase of CLB logic usage, as well as throughput increase. Also for the serial circulant, parallel row/column processing architectures in [26] are presented folding, which aims at storing in the same BRAM messages associated with different columns/rows within the base matrix.

It can be observed that FPGA implementations of flooded architectures present a wide range of architectural variations, with different parallelism degrees at different levels, which aim at different throughput/cost/error correction capability trade-offs. The fully parallel solution presents increased throughput, but high cost due to routing, as well as low flexibility. Partial parallel solutions use memories for message storage. For these architectures, BRAM-based memory units are targeted in the FPGA implementation. However, employing BRAM blocks leads to several challenges related especially to the low usage of these.

5. Layered LDPC decoders

Layered architectures have been proposed first in [8], with the main goal of reducing the required memory bits. In the case of a layered decoder, two types of messages require memory storage: the AP-LLR messages and the check node messages. A typical layered LDPC decoder [29–33], depicted in **Figure 5**, contains the following components:

1. Processing units: The processing in the layered scheduling consists of the computation of the variable node messages, computation of the check node messages, and the AP-LLR update. The variable node message is computed from the AP-LLR and the check node message. The check node message is computed in the same way as in flooded scheduling, while the AP-LLR is updated from the new values of the variable node and check node messages. Because messages do not require routing between processing nodes—as in flooded—and just routing between memories and processing units, a combined unit—variable-check unit—is employed for processing. The number of processing units in the typical layered decoder is equal to the number of rows which constitute one layer, which is usually given by the circulant size. A combined unit contains an adder to perform the variable message computation, a FIFO buffer, used for routing the updated variable node

http://dx.doi.org/10.5772/66085

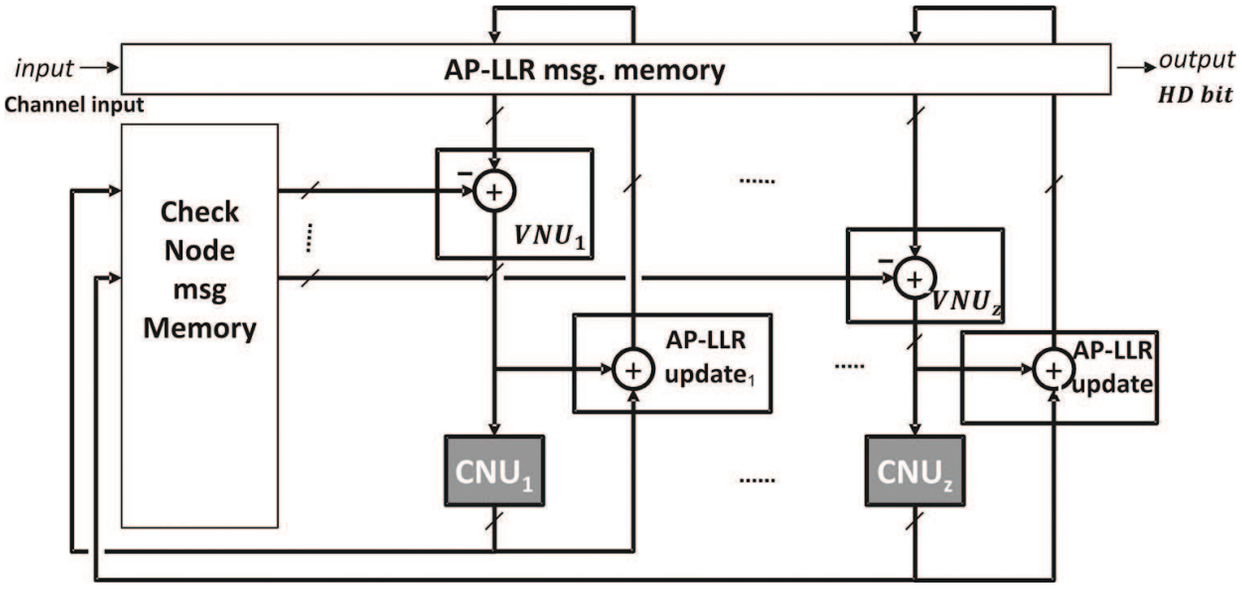


Figure 5. Layered decoding architecture.

message to the AP-LLR update, a comparator for updating the check node message, and the addition unit for the AP-LLR update [29–32]. Specific FPGA optimization can be implemented within the combined processing unit, which includes the use of the 6-input LUT within the CLB for comparator implementation—the comparator is implemented as ROM memories [30]—as well as the usage of the dedicated shift register chains for the implementation of the FIFOs. The processing unit has as inputs d_c AP-LLR messages and d_c check node messages, and outputs d_c updated AP-LLR messages and d_c updated check node messages. An important parameter for the entire decoding architecture is represented by the parallelism degree at the variable-check unit level, which represents the number of AP-LLR messages processed each clock cycle (maximum parallelism degree is equal to d_c). A higher degree of parallelism requires more simultaneous AP-LLRs read/write, as well as routing, which leads to increased number of memory ports or memory banks, and barrel shifters for routing [33].

2. Memory blocks: Layered decoders require the storage of two types of messages: AP-LLRs and check node messages. The AP-LLRs are messages which are routed between different processing units between layer processing. The check node messages are specific to each processing unit: these do not require routing from a processing unit to another between different layers. Therefore, the AP-LLR memory is a shared, global memory, while the check node message memories are local to each processing unit. Regarding the AP-LLR memory, the memory word for each bank is of $quant(\tilde{\gamma})$, while the maximum depth of this memory is equal to the number of columns in the base matrix. Regarding BRAM implementation of the AP-LLR memory, a drawback is represented by the low usage of the embedded block memory. Regarding the check node messages, two variants for their storage are used: (i) uncompressed form, when the β messages in their conventional two's complement format, and (ii) compressed form [34]. The compressed check node message is based on the fact that d_c−1β messages within a row corresponding to a row in the parity

check matrix have the same absolute value, equal to the minimum of the α messages connected to the corresponding check node unit, while the d_c-th check node message absolute value is equal to the second minimum. Therefore, a compressed check node message can be used, consisting of the signs, first minimum, second minimum, and the index of the first minimum. Regarding the FPGA implementation, the compressed form is suitable for shift register-based implementation in conventional CLB logic, while the uncompressed form is suitable for BRAM implementation. However, in BRAM-based implementation of the check node message, memory in compressed form is proposed for layered decoder with serial processing at processing node level. Routing from the BRAM blocks containing the check node messages to the processing units is achieved using large shift registers.

3. Routing network: Routing network is implemented using barrel shifters. The number of barrel shifters is dependent on the degree of parallelism in the processing unit. For each AP-LLR input of the processing unit, a pair of barrel shifters—one for routing read messages and one for routing the update message required for write—is required.

4. Control unit: The control unit is responsible for the generation of read/write addresses for the two memories, the shift amounts for the barrel shifter, as well as the control signals corresponding to the processing units. As in the case of the flooded decoders, ROM type of memories is used to embed the LDPC code information, from which are computed the memory addresses, as well as the shift amounts.

A major issue in the layer architecture is represented by the data hazards. Depending on the LDPC code, read-after-write (RAW) data hazards may affect the AP-LLR update: the updated value of the AP-LLR has not been written into the memory, before it is read for a new layer processing [35]. The problem of data hazards is aggravated by the usage of pipeline stages, both in the barrel shifters and in the processing units.

6. Conclusions

This book chapter presents an overview of the main design trade-offs in the implementation of LDPC decoders on FPGA devices. We detail how the main architectural choices for both flooded and layered scheduling strategies map on the built-in resources of modern FPGA devices. The main conclusions which can be drawn from this survey are as follows:

1. The degree of parallelism at processing node level has a major influence in the resource consumption of the LDPC decoder: it gives the number of barrel shifters used for routing, as well as the number of memory ports or memory banks used for message storage.

2. Routing represents an important factor in the cost/performance of the LDPC decoder; high-performance pipelined barrel shifter-based routing can be advantageously implemented in modern FPGA devices using conventional CLB resources.

3. Memories for message storage in partial parallel flooded LDPC decoder or layered decoders can be implemented using embedded BRAM blocks; the main problem is represented by the low usage of the memory bits within the BRAM.

The implementation of LDPC decoders on FPGA devices has a wide range of architectural and design parameters, which present different throughput/cost/error correction capability trade-offs. Furthermore, many FPGA-specific optimizations may be applied in the LDPC decoder design, such as the message memory mapping or optimization in the processing units.

Regarding the future use of the LDPC codes and decoder architectures, throughput and flexibility will represent highly important features. Regarding throughput, future wireless communication will require tens or hundreds of Gbps, which will impose new architectural challenges. Furthermore, the use of software-defined radios and software-defined flash will require highly flexible architectures, which can adapt code rate, quantization, as well as other features.

Acknowledgements

This work has been supported by bilateral UEFISCDI-ANR project DIAMOND.

Author details

Alexandru Amaricai* and Oana Boncalo

*Address all correspondence to: alexandru.amaricai@cs.upt.ro

Computer and Information Technology Department, University Politehnica Timisoara, Timisoara, Romania

References

[1] R. G. Gallagher. Low Density Parity Check Codes. MIT Press; 1963

[2] P. Hailes, L. Xu, R. Maunder, B. M. al-Hashimi, L. Hanzo. A survey of FPGA-based LDPC decoders. IEEE Communications Surveys and Tutorials. 2016;**18**(2):1098–1125. doi:10.1109/COMST.2015.2510381

[3] K. Zhao, W. Zhao, H. Sun, T. Zhang, X. Zhang, N. Zheng. LDPC-in-SSD: making advanced error correction codes work effectively in solid state drives. In: Proceedings of the 11th USENIX conference on File and Storage Technologies FAST'13; USENIX Association, Berkeley; 2013. pp. 243–256.

[4] T. J. Richardson, R. L. Urbanke. The capacity of low-density parity. IEEE Transactions on Information Theory. 2001;**47**(2):599–618. doi:10.1109/18.910577

[5] F. R. Kschischang, B. J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. IEEE Journal on Selected Areas in Communications. 1998;**16**(2):219–230. doi:10.1109/49.661110

[6] R. Tanner. A recursive approach to low complexity codes. IEEE Transactions on Information Theory. 1981;**27**(5):533–547. doi:10.1109/TIT.1981.1056404

[7] M. P. C. Fossorier. Quasicyclic low-density parity-check codes from circulant permutation matrices. IEEE Transaction on Information Theory. 2004;**50**(4):1788–1793. doi:10.1109/TIT.2004.831841

[8] D. E. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. In: IEEE Workshop on Signal Processing Systems; 13–15 October; IEEE; Austin, Texas, USA, 2004. pp. 107–112. doi:10.1109/SIPS.2004.1363033

[9] C. L. KameniNgassa, V. Savin, D. Declercq. Analysis of min-sum based decoders implemented on noisy hardware. In: Asilomar Conference on Signals, Systems and Computers; Pacific Groove, California, USA, 2013. pp. 866–870. doi:10.1109/ACSSC.2013.6810411

[10] J. Chen, M. P. C. Fossorier. Near optimum universal belief propagation based decoding of low-density parity check codes. IEEE Transaction on Communication. 2002;**50**(3):406–414. doi:10.1109/26.990903

[11] V. Savin. Self-corrected min-sum decoding of LDPC codes. In: IEEE International Symposium on Information Theory; IEEE; Toronto, Canada, 2008. pp. 146–150. doi:10.1109/ISIT.2008.4594965

[12] S. K. Planjery, D. Declercq, L. Danjean, B. Vasic. Finite alphabet iterative decoders—Part I: decoding beyond belief propagation on the binary symmetric channel. IEEE Transactions on Communications. 2013;**61**(10):4033–4045. doi:10.1109/TCOMM.2013.090513.120443

[13] V. Savin. LDPC decoders. In: D. Declerq, M.P.C. Fossorier, E. Biglie, editors. Channel Coding: Theory, Algorithms, and Applications; Elsevier; 2015. doi:10.1016/B978-0-12-396499-1.00004-2

[14] Xilinx. 7 Series DSP48E1 Slice User Guide—UG479. 2014.

[15] Xilinx. 7 Series FPGA Configurable Logic Block User Guide—UG474. 2014.

[16] Xilinx. 7 Series Memory Resources User Guide—UG473. 2014.

[17] V.Torres, A. Perez-Pascual, T. Sansaloni, J. Valls. Fully-parallel LUT-based (2048, 1723) LDPC code decoder for FPGA. In: 19th IEEE International Conference on Electronics, Circuits and Systems (ICECS); IEEE; 2012. p. 408–411. doi:10.1109/ICECS.2012.6463663

[18] A. Darabiha, A. C. Carusone, F. R. Kschischang. A bit-serial approximate min-sum LDPC decoder and FPGA implementation. In: 2006 IEEE International Symposium on Circuits and Systems; 21–24 May; IEEE; Island of Kos, Greece, 2006. doi:10.1109/ISCAS.2006.1692544

[19] V. A. Chandrasetty, S. M. Aziz. An area efficient LDPC decoder using a reduced complexity min-sum algorithm. Integration, The VLSI Journal. 2012;**45**(2):141–148. doi:10.1016/j.vlsi.2011.08.002

[20] A. Balatsoukas-Stimming, A. Dollas. FPGA-based design and implementation of a multi-GBPS LDPC decoder. In: 22nd International Conference on Field Programmable Logic and Applications (FPL); IEEE; Oslo, Norway, 2012. doi:10.1109/FPL.2012.6339191

[21] C. Beuschel, H.-J.Pfleiderer. FPGA implementation of a flexible decoder for long LDPC codes. In: 2008 International Conference on Field Programmable Logic and Applications; IEEE; Heidelberg, Germany, 2008. pp. 185–190. doi:10.1109/FPL.2008.4629929

[22] A. Blad, O. Gustafsson. FPGA implementation of rate-compatible QC-LDPC code decoder. In: 20th European Conference on Circuit Theory and Design (ECCTD); IEEE; Linkoping, Sweden, 2011. p. 777–780. doi:10.1109/ECCTD.2011.6043844

[23] A. Amaricai, O. Boncalo, I. Mot. Memory efficient FPGA implementation for flooded LDPC decoder. In: 23rd Telecommunications Forum Telfor (TELFOR); Belgrade, Serbia, 2015. pp. 500–503. doi:10.1109/TELFOR.2015.7377516

[24] Z. Wang, Z. Cui. A memory efficient partially parallel decoder architecture for quasi-cyclic LDPC codes. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2007;**15**(4):483–488. doi:10.1109/TED.2007.895247

[25] Y. Chen, K. Parhi. Overlapped message passing for quasi-cyclic low-density parity check codes. IEEE Transactions on Circuits and systems—I: Regular Papers. 2004;**51**(6):1106–1113. doi:10.1109/TCSI.2004.826194

[26] X. Chen, J. Kang, S. Lin, V. Akella. Memory system optimization for FPGAbased implementation of quasi-cyclic LDPC codes decoders. IEEE Transactions on Circuits and Systems I: Regular Papers. 2011;**58**(1):98–111. doi:10.1109/TCSI.2010.2055250

[27] X. Chen, Q. Huang, S. Lin, V. Akella. FPGA-based low-complexity high-throughput tri-mode decoder for quasi-cyclic LDPC codes. In: 47th Annual Allerton Conference on Communication, Control, and Computing; IEEE; Monticello, Illinois, USA, 2009. pp. 600–606. doi:10.1109/ALLERTON.2009.5394917

[28] S. Nimara, O. Boncalo, A. Amaricai, M. Popa. FPGA architecture of multi-codeword LDPC decoder with efficient BRAM utilization. In: IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS); IEEE; Kosice, Slovakia, 2016. doi:10.1109/DDECS.2016.7482452

[29] S. Mhaske, H. Kee, T. Ly, A. Aziz, P. Spasojevic. High-Throughput FPGA-based QC-LDPC Decoder Architecture. In: IEEE 82nd Vehicular Technology Conference (VTC Fall); IEEE; Boston, Massachusetts, USA, 2015. doi:10.1109/VTCFall.2015.7390967

[30] O. Boncalo, A. Amaricai, A. Hera, V. Savin. Cost-efficient FPGA layered LDPC decoder with serial AP-LLR processing. In: 24th International Conference on Field Programmable Logic and Applications (FPL); IEEE; Munich, Germany, 2014. doi:10.1109/FPL.2014.6927474

[31] S. Kim, G. E. Sobelman, H. Lee. A reduced-complexity architecture for LDPC layered decoding schemes. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2011;**19**(6):1099–1103. doi:10.1109/TVLSI.2010.2043965

[32] K. Zhang, X. Huang, Z. Wang. High-throughput layered decoder implementation for quasi-cyclic LDPC codes. IEEE Journal on Selected Areas in Communications. 2009;**27** (6):985–994. doi:10.1109/JSAC.2009.090816

[33] O. Boncalo, P. Mihancea, A. Amaricai. Template-based QC-LDPC decoder architecture generation. In: 10th International Conference on Information, Communications and Signal Processing (ICICS); Singapore, 2015. doi:10.1109/ICICS.2015.7459838

[34] O. Boncalo, A. Amaricai, P. Mihancea, V. Savin. Memory trade-offs in layered self-corrected min-sum LDPC decoders. Analog Integrated Circuits and Signal Processing. 2016;**87**(2):169–180. doi:10.1007/s10470-015-0639-3

[35] Z. Wu, K. Su. Updating conflict solution for pipelined layered LDPC decoder. In: IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC); IEEE; Xi'an, China, 2016. doi:10.1109/ICSPCC.2015.7338879