Message Quantization Scheme for Nonbinary LDPC Decoder FPGA Implementation

Wojciech Sulek
Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, Gliwice, Poland
Email: Wojciech.Sulek@polsl.pl

Abstract —The non-binary Low Density Parity Check (LDPC) codes over Galois Fields GF($q = 2^p$) have evolved from the binary LDPC codes that are today an industry standard for channel coding. The performance of short block length codes is significantly higher for non-binary LDPC, at the cost of increased decoding complexity. The efficient decoder hardware implementation is still a challenging task. Most of the recently proposed hardware realizations are ASIC-oriented as they employ multiplierless computation units. This article concerns a different decoder design approach that is specifically intended for an FPGA implementation. The reformulated mixed-domain FFT-BP decoding algorithm is applied that does not exclude the multiplication units. This allows mapping a part of the algorithm to the multiplier cores embedded in an FPGA. In this article we concentrate on the important issue of the proper selection of the numeric precision employed. We present the partially parallel extension of the mixed-domain decoder that operates for the structured codes. Then we carefully analyze the finite precision effects on the decoding performance. By simulation and synthesis results we show that it is advantageous to segment the decoder dataflow into 3 parts with different precision. The provided results also facilitate the precision selection for maximum performance or for some performance-complexity tradeoff.

Index Terms—Error correcting codes, LDPC codes, Non-binary codes, Iterative decoder, Hardware decoder

I. Introduction

Binary LDPC codes, after their rediscovery in the late 90's [1], have attracted great research attention due to their excellent error-correcting performance and highly parallel iterative decoding scheme. They have become one of the industry standards for error correction coding in wireless communication. However when the codeword length has to be small to moderate or when higher order modulation is applied [2], the non-binary LDPC codes defined over the Galois Field GF(q) [3] can outperform their binary counterparts.

The major drawback of codes over higher order fields is their decoding algorithm complexity. Both binary and non-binary LDPC codes are decoded using Belief Propagation (BP) on their factor graphs, however the decoding complexity of GF(q) codes scales exponentially with the number of bits in the symbol.

Manuscript received June 25, 2014; revised January 30, 2015.
doi:10.12720/jcm.10.1.86-92

It is known that the complexity of BP can be reduced with its dual form in the frequency domain [4]. Another algorithm reformulations use logarithm domain with log-density [5] or log-density-ratio representation of messages, which require fewer quantization levels due to lower sensitivity to quantization [6]. Moreover in [7] the mixed domain algorithm has been proposed, where the FFT operation is performed in the real domain and the VNs and CNs operations in the log domain, so that only additions and subtractions are needed. Some algorithms with reduced complexity have also been proposed, such as Min-Max algorithm [8] and the extended min-sum (EMS) algorithm [9]. However implementation of these algorithms is inevitably connected with some decoding performance degradation.

An FPGA implementation of the mixed domain decoder serial realization is also presented in [7]. In [10] the min-max decoder is proved to be more efficiently implementable than the mixed-domain decoder [7] as well as the EMS decoder [11]. Other recent state of the art hardware decoder designs are also based mostly on the Min-Max or the EMS algorithms, or some variations of them [12]–[16]. The so-called layered decoding is applied in some implementations [17].

The common characteristic of the mentioned hardware implementations is that all of them adopt variations of the BP decoding algorithm with exclusion of the multiplication operations. Such an approach makes the design ASIC (Application Specific Integrated Circuit) oriented. However in the case of FPGA (Field Programmable Gate Array) implementation, a large number of hardware multipliers is available for designer disposal in a typical modern FPGA chip. It is then often desirable to make use of the multiplier cores.

The primary motivation for this work is to design a decoder that is: 1) easily scalable for an FPGA devices of different sizes; 2) capable of achieving the highest possible throughput for a single FPGA by means of utilization most of the available resources. The first goal is achieved by means of partially parallel decoder architecture with configurable number of processing units. In order to achieve the second goal, not only the basic Logic Elements (Slices), but also the hardware Multiplier Blocks (DSP48s in the case of Xilinx devices) are utilized. This is not possible with any multiplierless implementation similar to the state of the art works mentioned above. Therefore in [18] we have presented the decoding algorithm reformulation and the FPGA decoder serial implementation that efficiently utilizes all the FPGA resource types, including hardware Multiplier Blocks. The decoder operates over mixed (real / logarithm) message domains, which enables employing the Multiplier Blocks.

In this article some additional details of the designed decoder operation are discussed. Firstly, we present the partially parallel extension of the decoder architecture that operates for the structured GF(q)-LDPC codes. The presented architecture enables the mentioned throughput scalability, which is confirmed by the provided synthesis results. Secondly, we discuss the messages quantization issues and provide experimental results that aid selection of the fixed-point messages format and precision. The message quantization scheme is very important issue, because the wordlength of the fixed-point representation defines the performance-area tradeoff of the decoder. We will show that in the decoder employing the mixed domain algorithm, the wordlength should be chosen separately for the decoding domains. Moreover by simulation and synthesis results we show that it is advantageous to segment the decoder dataflow into 3 parts with different precision.

The paper is organized as follows. In the next section the structured LDPC codes over GF(q) are defined, then in section III the reformulated decoding algorithm that has been proposed is recalled. The partially parallel decoder FPGA implementation is presented in section IV, which is followed by the discussion of the messages quantization issues in section V. Finally synthesis results and conclusions are presented in section VI and VII respectively.

II. The Structured GF(q) LDPC Codes

LDPC codes are a class of linear block codes defined over the Galois field GF(q) with restriction to fields of the size being power of two (q=2^p). In the case of the well known binary codes the field size is 2 (thus p=1), whereas for the non-binary codes p>1.

The (N,K) LDPC code with a source vector length K and a code vector length N is defined by a low density parity check matrix $\mathbf{H}_{M\times N}$ with GF(q) entries h_{mn}, where M=N-K is the number of parity checks.

An efficient partially-parallel decoder implementation is possible for parity-check matrices with a special constraint on their form. The main building blocks of partially-parallel decoder are message memories and a number of computation units. In order to organize memory accesses without contentions, the parity check matrix should be in a structured form, partitioned into a square submatrices.

The structured GF(q) LDPC code is defined by the parity check matrix $\mathbf{H}$ being a composite of a square submatrices:

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

where each submatrix $\mathbf{P}_{d,l}$ of size $P\times P$ is either an all-zero matrix or a matrix with exactly one nonzero element (coefficient) in every row and every column. In other words it is a permutation of an identity matrix multiplied by a coefficient from GF(q).

The submatrix size P dictates the number of parallel computation units in the partially parallel decoder architecture that will be presented. Therefore P is directly related with the achievable decoder throughput. Remark that the structured code definition (1) gives us a more flexible submatrix size P election than for the QC-LDPC codes subclasses as defined in [19] that are employed in most of the other state of the art NB-LDPC decoder implementations [13], [14], [20].

III. Nonbinary Codes Decoding

A row vector $\mathbf{c}$ (over GF(q)) of length N is a valid codeword if it satisfies the parity check equation:

$$\mathbf{Hc}^T=\mathbf{0}_{M\times 1}, \tag{2}$$

where the operations are performed in the Galois field arithmetic. Equation (2) can be partitioned into M checks associated with M rows of $\mathbf{H}$.

The goal of the decoder is to find the most probable originally transmitted vector $\mathbf{c}$ that satisfies (2), taking into account the received channel values $\mathbf{y}$=[y_1,y_2,…,y_N]. In the soft decision decoding system, the values initializing the decoder are likelihoods:

$$f_n^a=P(c_n=a\,|\,y_n),\quad n=1,\ldots,N,\ \ a\in\mathrm{GF}(q) \tag{3}$$

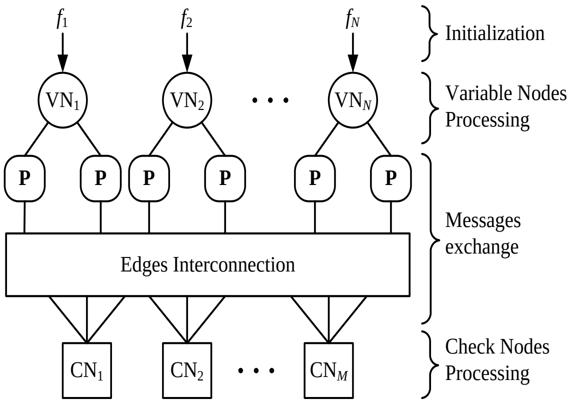

Fig. 1. Graph representation of the decoding algorithm.

The LDPC codes are decoded with iterative algorithms, so-called Belief Propagation algorithm or some modification of it. Considering the iterative decoding algorithm, a convenient representation of the parity check matrix is the Tanner graph (Fig. 1).

The variable nodes (VNs) and the check nodes (CNs) represent *N* decoded code vector elements and *M* checks respectively. The edges in the graph are associated with positions of the non-zero entries in **H**. The values of the non-zero entries can be represented in the graph by the edge labels or some additional nodes (permutation nodes) denoted as **P** in Fig. 1. The decoding algorithms are based on iterative recalculation and exchanging messages (beliefs) between graph edges.

It is well known that the complexity of the classic Belief Propagation (BP) decoding algorithm [3] over higher order GF fields can be scaled down if its dual form in the frequency domain (FFT-BP) is used [4]. Another algorithm reformulations use logarithm domain with log-density [5] or log-density-ratio representation of messages, which require fewer quantization levels due to lower sensitivity to quantization [6].

The logarithm domain implementation for FFT-BP algorithm can be proposed as well [5]. In this method it is possible to replace product operations with additions for both VNs and CNs processing, however the additions and subtractions computed in the real domain FFT do not have straightforward equivalent in the logarithm domain. Therefore in [7] the mixed domain algorithm has been proposed, where the FFT operation is performed in the real domain and the VNs and CNs operations in the log domain, so that only additions and subtractions are needed.

In this article we discuss a decoder implementation based on similar to the proposed in [7] mixed domain algorithm formulation with the main difference being that check nodes (CNs) operate in the real domain. Such an approach is intended specifically for FPGA decoder implementation and the motivation is as follows:

- CNs operating in the real domain require multiplication operations, which fits well into the contemporary FPGA devices containing a large amount of hardware multipliers that would remain unused otherwise.
- Since the CNs processing include FFT and IFFT, which is calculated in the real domain anyway, our proposition requires only two domain changes per iteration (on the input and output of the CNs) instead of four domain changes in [7].
- Using logarithm domain on the VNs side allows memory savings, because messages stored in memories require substantially fewer bits due to the mentioned lower sensitivity to quantization effects.

Here we review the employed mixed real-logarithm domain permutation FFT-BP algorithm formulation [18]. The algorithm is initialized with logarithms of likelihoods $F_n^a = \log(f_n^a)$. The real (probability) domain message vectors propagated from VN$_n$ to CN$_m$ are denoted as $\mathbf{q}_{mn} = [q_{mn}^0, q_{mn}^1, \ldots, q_{mn}^{2^p}]$ and the real domain message vectors from CN$_m$ to VN$_n$ are denoted as $\mathbf{r}_{mn} = [r_{mn}^0, r_{mn}^1, \ldots, r_{mn}^{2^p}]$. The log-domain messages are denoted with uppercase, Q_{mn}^a and R_{mn}^a respectively. The message vector permutations [7] are expressed by matrix-vector multiplication with permutation matrices $\mathbf{P}_{h_{mn}}$ and $\mathbf{P}_{h_{mn}}^{-1}$. The $N(m)$ denotes the set of indexes of variable nodes adjacent to check node CN$_m$ and the $M(n)$ denotes the set of indexes of check nodes adjacent to variable node VN$_n$. Notation $X \setminus x$ represents a set X excluding x. The algorithm can be summarized as follows:

1) **Initialization**.
For $n = 1, \ldots, N;\ m \in M(n);\ a \in \mathrm{GF}(q)$:
$$Q_{mn}^a := F_n^a \tag{4}$$

2) **Check Nodes processing**.
For $m = 1, \ldots, M;\ n \in N(m);\ a \in \mathrm{GF}(q)$:
$$q_{mn}^a := \exp\left(Q_{mn}^a\right) \tag{5}$$
gather: $\mathbf{q}_{mn} = [q_{mn}^0, q_{mn}^1, \ldots, q_{mn}^{2^p}]$, then:
$$\begin{aligned} \mathbf{u}_{mn} &:= \alpha_{mn} FFT\left(\mathbf{P}_{h_{mn}} \mathbf{q}_{mn}\right) \\ \mathbf{r}_{mn} &:= \mathbf{P}_{h_{mn}}^{-1} IFFT\left(\cdot \prod_{j \in N(m) \setminus n} \mathbf{u}_{mj}\right) \end{aligned} \tag{6}$$
where α_{mn} is the normalization factor; $\cdot\prod$ is the term by term product of vector elements. CNs output:
$$R_{mn}^a := \log\left(r_{mn}^a\right) \tag{7}$$

3) **Variable Nodes processing**.
For $n = 1, \ldots, N;\ m \in M(n);\ a \in \mathrm{GF}(q)$:
$$Q_{mn}^a := F_n^a + \sum_{j \in M(n) \setminus m} R_{jn}^a \tag{8}$$

4) **Tentative decoding**. For $n = 1, \ldots, N$:
$$\hat{c}_n := \arg\max_a \left(F_n^a + \sum_{j \in M(n)} R_{jn}^a\right) \tag{9}$$

If $\mathbf{H}\hat{\mathbf{c}}^T = \mathbf{0}$ then halt the decoding with **c** as output, otherwise go to the check nodes processing (step 2).

IV. Hardware Implementation of the Partially Parallel Decoder

The overall architecture of the designed hardware decoder operating under the mixed domain algorithm is presented in Fig. 2. The partially-parallel architecture provides throughput-complexity tradeoff capabilities by means of configurable number *P* of parallel Variable Node Units (VNU) and Check Node Units (CNU). This number of units corresponds to the submatrix size of **H** in

(1). In the extreme case of P=1 the architecture is serial (see Fig. 3).

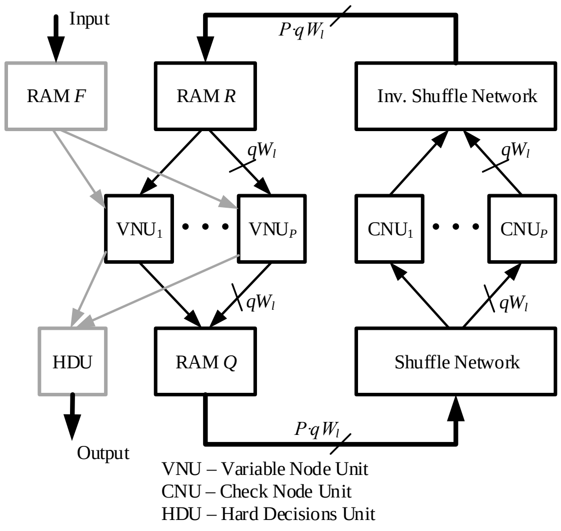

Fig. 2. Partially parallel decoder architecture.

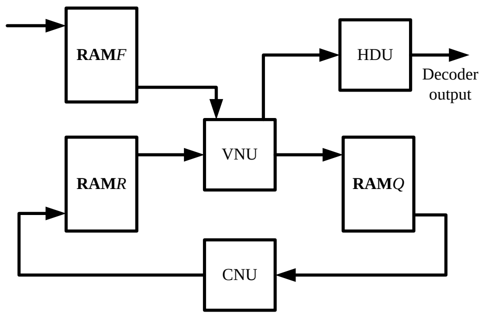

Fig. 3. Serial decoder architecture.

The log-domain messages are exchanged between computation units: every CNU accepts the vector of q values Q_{mn}^{a} (for every $a \in \mathrm{GF}(q)$) of woldlength W_l and produces the vector of q values R_{mn}^{a}. Outputs of P parallel CNUs are combined to a single memory word of length PqW_l stored in RAMR. The shuffle networks perform configurable cycle shifts according to the structure of submatrices of **H**. Every VNU accepts vectors from RAMR memory and the input data F_n^a from RAMF memory and produces Q_{mn}^{a} values that are saved to RAMQ memory. Moreover the VNU delivers data to the Hard Decision Unit (HDU), which makes tentative decisions according to (9).

A. Variable Node Unit

The VNU is composed of q parallel subunits (see Fig. 4), where every subunit realizes (8) for a single value a. For the nth variable node, VNU calculates messages for every $m \in M(n)$. At first the sum of F_n^a and all incoming values R_{mn}^{a} is calculated in the accumulator. This sum is outputted to the hard decision unit (HDU). Then the input values (delayed by the shift register) are subtracted from the sum to form the exclusive sums over $j \in M(n) \setminus m$ in (8). The shift register is dynamic, thus it enables variability of VN degrees.

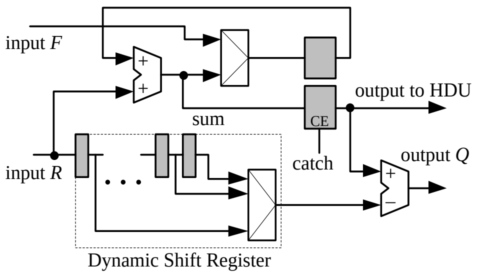

Fig. 4. Subunit of the Variable Node Unit (VNU).

B. Check Node Unit

Architecture of the check node unit (CNU) operating according to (5)-(7) is shown in Fig. 5. The data propagated through the CNU block are message vectors of size q with elements represented by a fixed point numbers with diversified precision. The finite precision issues will be discussed in the next section.

The permutation blocks realize a message vector cyclic shifts defined by $\mathbf{P}_{h_{mn}}$ matrices (and its inverse respectively) for every nonzero h_{mn} entry of **H**. The permutation blocks are composed of multiplexers allowing to process those cyclic shifts. To efficiently implement these blocks block we have adopted the structure of a Banyan switch that is known to be an efficient interconnection structure for a configurable cyclic shifter [21]. The domain transformations are performed with q Look Up Tables (LUT) realizing exp(.) and log(.) nonlinear functions for every message vector element separately. Each of the domain transformation LUTs as well as each of the permutation blocks contains a single layer of pipeline registers that are applied in order to achieve a relatively high clock frequency. The FFT and IFFT blocks are implemented with networks of adders and subtractors [18] with embedded pipeline registers as well.

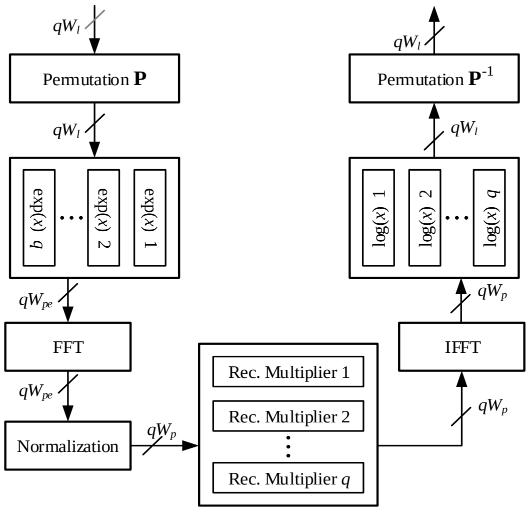

Fig. 5. Check Node Unit (CNU).

Since the core operation in (6) is the term by term product of vectors, it is realized in q blocks, separately for every message vector element. The operation of a multiplier block associated with an element a of the message vector for the mth check node can be expressed:

$$v_{mn}^{a} := \prod_{j \in N(m)\setminus n} u_{mj}^{a} \tag{10}$$

where u_{mj}^{a} is an element of $\mathbf{u}_{mj}$ in (6) and v_{mn}^{a} is an element of $\mathbf{v}_{mn} = \cdot\prod \mathbf{u}_{mj}$ in (6). For the mth check node processing, the multiplier block calculates v_{mn}^{a} values for every $n \in N(m)$. It is therefore convenient to realize this calculations by a well known forward-backward recursion scheme. In the designed decoder, the recursive multiplier subblocks operate under this scheme. The multiplication operations are realized making use of the hardware resources available in the FPGA devices, e.g. DSP48E1 Slices in the Virtex6 family. More details about the decoder implementation can be found in [18].

V. Recommended Message Quantization Scheme

The data propagated through the CNU block is in the form of vectors of $q=2^p$ messages. A very important issue is the selection of the numeric precision, because the wordlength of the fixed-point representation defines the performance-area tradeoff of the decoder. When the decoder employs mixed domain algorithm, the wordlength should be chosen separately for the decoding domains. Due to the lower sensitivity of log-domain messages to the quantization effects [6], it is possible to set the wordlength in the logarithm domain side (W_l) significantly lower than in the probability domain side, without performance degradation. Moreover, we recommend additional partition of the probability domain side into normalized section with wordlength W_p and non-normalized section with extended wordlength W_{pe} (see Fig. 5). It is motivated by the fact that the messages before normalization possess higher dynamic range, thereby their representation requires more bits.

Since the values of messages in the probability domain are in the range [0,1], the suitable representation is a standard unsigned fixed-point notation constrained to the fractional part. The situation is different for the logarithm domain case, because logarithms of likelihoods are always negative. Moreover the lower limit is equal to the logarithm of the smallest representable probability, i.e. $\log(2^{-W_{pe}})$. Therefore the logarithm domain messages are limited to $[\log(2^{-W_{pe}}),0]$, uniformly quantized and represented by unsigned numbers with precision W_l bits (the always negative sign is omitted).

Here we provide some experimental results that facilitate the message wordlength selection and confirm that the proposed wordlength differentiation (W_l, W_p, W_{pe}) is reasonable. The simulation and synthesis results presented in Fig. 6- Fig. 7 have been obtained with FPGA implementation of the decoder in the Xilinx Virtex6 device. The BIAWGN channel model has been employed for simulations and two rate-1/2 codes have been used: a (600,300) code over GF(16) and a (400,200) code over GF(64). The parity check matrices were generated making use of the classic PEG algorithm [22] with nonzero entries selected row by row with a method similar to the proposed in [23]. The synthesis results concern the serial implementation, P=1.

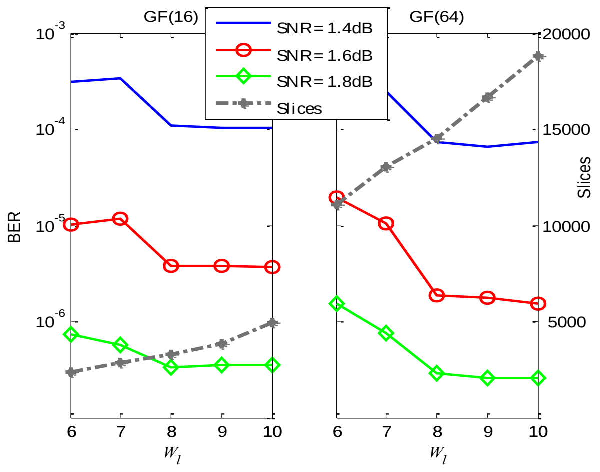

Fig. 6. Decoder performance (BER) and area (Virtex6 Slices) with respect to the wordlength W_l, for W_p = 16, W_{pe} = 20.

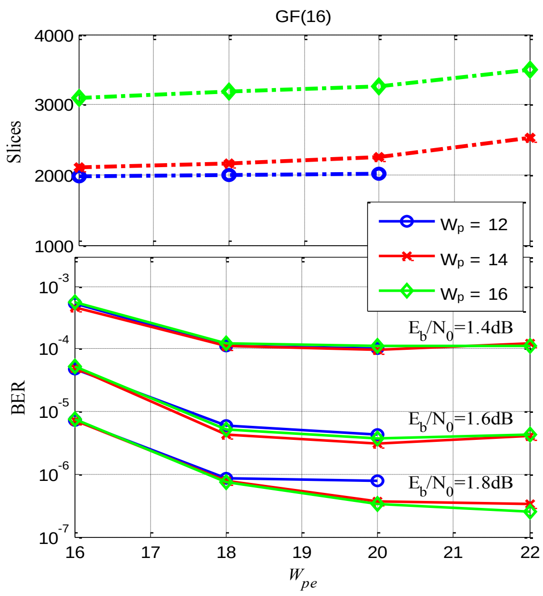

Fig. 7. Decoder performance (BER) and area (Virtex6 Slices) with respect to the wordlength W_{pe}.

Let us comment the presented results. According to the results in Fig. 6, increasing the W_l over 8bits is pointless because of the Bit Error Rate (BER) saturation. Therefore W_l=8 should be chosen for an optimum performance. Lower wordlength could also be useful for area reduction at the cost of performance reduction; for instance W_l=6 entails the loss of about 0.1dB for GF(16) and about 0.2dB for GF(64).

Results presented in Fig. 7 facilitate W_p and W_{pe} selection. First remark that for W_p=14 (red curve) and

W_p=16 (green curve) there is little to no difference in performance, regardless of the W_{pe} value. Meanwhile increasing W_{pe} up to 20 results in a significant BER improvement. This observation justifies our recommendation for the probability domain segmentation. Without W_p–W_{pe} distinction, for the optimum performance W_{pe}=W_{pe}=20 should be chosen. The FPGA area would then be enormous. The proposed segmentation allows fixing W_{pe}=20 while reducing W_p down to 14 without performance degradation. Significant FPGA resources can then be saved, which is confirmed by the synthesis results provided also in Fig. 7. For example reducing the W_p from 16 down to 14 enables saving about 30% of Slices. Remark that the W_{pe} reduction (with fixed W_p) would not enable such a large savings. Concluding we claim that for optimized performance, relatively large W_{pe} and reduced W_p should be set, for instance we recommend W_{pe}=20 and W_p=14 for near optimal performance with reduced area. Meanwhile lower wordlengths could be used for the FPGA area reduction at the cost of performance reduction.

VI. Synthesis and Simulation Results

The decoder has been implemented and verified with Xilinx Virtex-4 as well as Virtex-6 devices. Synthesis results for a code with block length 2160 bits over GF (8), similar to the code used in [7], are presented in Table I. We have also included the synthesis results taken from [7]. The Table I includes the numbers of Slices utilized, the numbers of Block RAMs and the numbers of DSP48E1 Slices (multipliers) along with the throughput assessment. The serial implementation (P=1) is used in this case. The wordlength selected according to the results presented in the previous section guarantees nearly full-precision performance.

Table I: FPGA Synthesis Results for the Designed Decoder over GF(8) in Comparison with the Results Presented in [7]

	[3]	This work
Field order	GF(8)	GF(8)
Code	(720,360)	(720,360)
Decoding Algorithm	FFT-BP	FFT-BP
Number of iterations	unknown	15
Precision	8 bits	W_l=8, W_p=14
Synthesis Target	Virtex-2P	Virtex-4
Slices	4660	1920
BRAMs	16	25
Multipliers (DSP48s)	0	16
Max. clock frequency	99.7 MHz	177 MHz
Throughput	1.09 Mbps	2.02 Mbps

The Virtex-2P family from the prior implementation [7] is no longer supported by the recent software. However the Slices embedded in the Virtex4 devices hold structure very similar to the Virtex-2P Slices, comprising two 4-input LUTs. Block RAMs (BRAMs) and Multipliers have also the same parameters. Therefore the utilization results can be compared directly with [7]. The serial implementation of the designed decoder utilizes significantly less Slices, at the cost of 16 Multiplier Cores. The utilization is equally distributed among Slices, BRAMs and DSPs, thus employing efficiently the FPGA fabric in the case of semi parallel implementation.

Fig. 8 and Fig. 9 show the Word Error Rate (WER) performance for codes with block length 2400 bits, binary and nonbinary over GF(2^2), GF(2^4) and GF(2^6). The parity check matrices for the codes have been constructed making use of the PEG algorithm [22], structured with an algorithm [24], and the nonzero entries selection method similar the proposed in [23]. QPSK modulation (Fig. 8) as well as QAM-64 modulation (Fig. 9) with AWGN channel model have been used for the simulations. The presented results confirm that the higher order Galois Fields allow achieving higher performance with the same code block length.

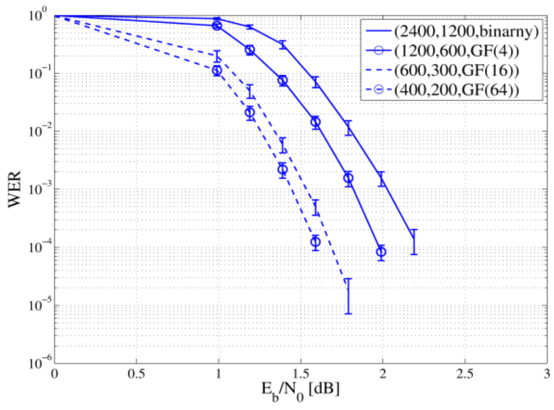

Fig. 8. Nonbinary codes decoding simulation results for 64-QAM modulation over AWGN channel and different field orders.

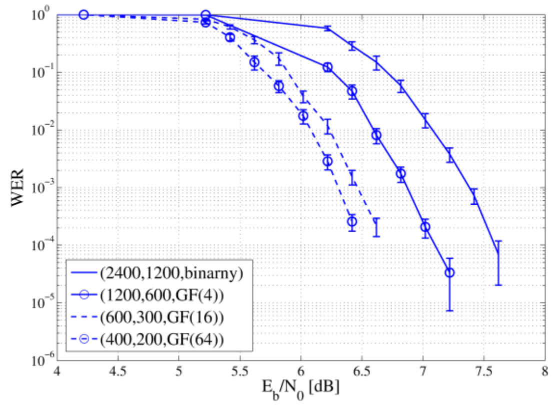

Fig. 9. Nonbinary codes decoding simulation results for QPSK modulation over AWGN channel and different field orders.

VII. Conclusions

The efficient partially parallel GF(q)-LDPC decoder implementation devoted for FPGA devices is proposed. This construction is an extension of the serial architecture. The partially parallel implementation can achieve an increased throughput and allows throughput-complexity tradeoff. Efficiency of the presented realization is based on the balanced utilization of all the types of FPGA resources, particularly making use of the multiplier blocks.

The important issue of the messages quantization method and precision has been discussed in this paper. We have shown that the message wordlength of the designed decoder should be chosen separately for the logarithm domain (W_l) and probability domain (W_p) parts of the decoder dataflow. Moreover yet other precision should be selected for non-normalized messages (W_{pe}). We provided experimental results that facilitate the message wordlength selection and confirm that the proposed wordlength differentiation is reasonable. The recommendations for the precision selection have also been given for the three parts of the decoder dataflow. Specifically the decoder with precision W_l=8, W_{pe} =20 and W_p =14 achieves near optimal performance, while the FPGA resources are significantly reduced in comparison with the solution without the proposed precision differentiation.

REFERENCES

[1] D. J. C. MacKay, “Good error-correcting codes based on very sparse matrices,” *IEEE Trans. Inf. Theory*, vol. 45, pp. 399–431, March 1999.

[2] J. Huang, S. Zhou, and P. Willett, “Nonbinary LDPC coding for multicarrier underwater acoustic communication,” *IEEE J. Sel. Areas Commun.*, vol. 26, pp. 1684–1696, December 2008.

[3] M. C. Davey and D. MacKay, “Low-density parity check codes over GF(q),” *IEEE Commun. Lett.*, vol. 2, pp. 165– 167, June 1998.

[4] L. Barnault and D. Declercq, “Fast decoding algorithm for LDPC over GF(2^q),” in *Proc. (IEEE) Information Theory Workshop*, Paris, France, 2003.

[5] H. X. Song and J. R. Cruz, “Reduced-complexity decoding of q-ary LDPC codes for magnetic recording,” *IEEE Trans. Magn.*, vol. 39, pp. 1081–1087, March 2003.

[6] H. Wymeersch, H. Steendam, and M. Moeneclaey, “Computational complexity and quantization effects of decod- ing algorithms for non-binary LDPC codes,” in *Proc. (IEEE) International Conference on Acoustics, Speech, and Signal Processing*, Montreal, Canada, 2004.

[7] C. Spagnol, E. M. Popovici, and W. P. Marnane, “Hardware implementation of GF(2^m) LDPC decoders,” *IEEE Trans. Circuits Syst. I*, vol. 56, pp. 2609–2620, December 2009.

[8] V. Savin, “Min-Max decoding for non binary LDPC codes,” in *Proc. (IEEE) International Symposium on Information Theory*, Toronto, Canada, 2008, pp. 960–964.

[9] D. Declercq and M. Fossorier, “Decoding algorithms for nonbinary LDPC codes over GF(q),” *IEEE Trans. Commun.*, vol. 55, 633–643, April 2007.

[10] X. Zhang and F. Cai, “Efficient partial-parallel decoder architecture for quasi-cyclic non- binary LDPC codes,” *IEEE Trans. Circuits Syst. I*, vol. 58, pp. 402–414, February 2011.

[11] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, “Architecture of a low-complexity non-binary LDPC decoder for high order fields,” in *Proc. (IEEE) International Symposium on Communications and Information Technologies*, Sydney, Australia, 2007, pp. 1201–1206.

[12] T. Lehnigk-Emden and N. Wehn, “Complexity evaluation of non-binary galois field LDPC code decoders,” in *Proc. (IEEE) 6th International Symposium on Turbo Codes & Iterative Information Processing*, Brest, France, 2010.

[13] X. H. Chen and C. L. Wang, “High-throughput efficient non-binary LDPC decoder based on the simplified min-sum algorithm,” *IEEE Trans. Circuits Syst. I*, vol. 59, pp. 2784–2794, November 2012.

[14] E. Boutillon, L. Conde-Canecia, and A. A. Ghouwayel, “Design of a GF(64)-LDPC decoder based on the EMS algorithm,” *IEEE Trans. Circuits Syst. I*, vol. 60, pp. 2644–2656, October 2013.

[15] E. Li, D. Declercq, and K. Gunnam, “Trellis-based extended min-sum algorithm for non-binary LDPC codes and its hardware structure,” *IEEE Trans. Commun.*, vol. 61, pp. 2600–2611, July 2013.

[16] F. Cai and X. Zhang, “Relaxed min-max decoder architectures for nonbinary low-density parity-check codes,” *IEEE Trans. VLSI Syst.*, vol. 21, pp. 2010–2023, November 2013.

[17] Y. L. Ueng, C. Y. Leong, C. J. Yang, C. C. Cheng, K. H. Liao, and S. W. Chen, “An efficient layered decoding architecture FOR Nonbinary QC-LDPC codes,” *IEEE Trans. Circuits Syst. I*, vol. 59, pp. 385–398, February 2012.

[18] W. Sulek, M. Kucharczyk, and G. Dziwoki, “GF(q) LDPC decoder design for FPGA implementation,” in *Proc. (IEEE) 10th Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, USA, 2013, pp. 445–450.

[19] B. Zhou, J. Kang, S. Song, S. Lin, K. Abdel-Ghaffar, and M. N. Xu, “Construction of non-binary quasi-cyclic LDPC codes by arrays and array dispersions,” *IEEE Trans. Commun.*, vol. 57, pp. 1652–1662, June 2009.

[20] J. Lin, J. Sha, Z. F. Wang, and L. Li, “Efficient decoder design for nonbinary quasicyclic LDPC codes,” *IEEE Trans. Circuits Syst. I*, vol. 57, pp. 1071–1082, May 2010.

[21] C. L. Wu and T. Y. Feng, “On a class of multistage interconnection networks,” *IEEE Trans. Comput.*, vol. C-29, pp. 694–702, 1980.

[22] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, “Regular and irregular progressive edge-growth tanner graphs,” *IEEE Trans. Inf. Theory*, vol. 51, pp. 386–398, January 2005.

[23] C. Poulliat, M. Fossorier, and D. Declercq, “Design of regular (2,dc)-LDPC codes over GF(q) using their binary images,” *IEEE Trans. Commun.*, vol. 56, pp. 1626–1635, October 2008.

[24] M. E. O’Sullivan, “Algebraic construction of sparse matrices with large girth,” *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 718–727, February 2006.

Wojciech Sulek was born in Tarnowskie Gory, Poland, in 1978. He received the Ph.D. in the discipline of Electronics from Silesian University of Technology, Gliwice, Poland, in 2009. He is an assistant professor in the Institute of Electronics at this University. His Ph.D. thesis regarded Architecture Aware LDPC codes design and hardware decoder implementation. Modern coding theory and coding systems hardware design are under his main research interests up till today.