High-Throughput Architecture and Implementation of Regular (2, d_c) Nonbinary LDPC Decoders

Yaoyu Tao, Youn Sung Park, Zhengya Zhang
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor

Abstract—**Nonbinary LDPC codes have shown superior performance, but decoding nonbinary codes is complex, incurring a long latency and a much degraded throughput. We propose a low-latency variable processing node by a skimming algorithm, together with a low-latency extended min-sum check processing node by prefetching and relaxing redundancy control. The processing nodes are jointly designed for an optimal pipeline schedule. This low-latency, high-throughput architecture is applied to a class of high-performance (2, d_c)-regular nonbinary LDPC codes constructed based on their binary images. A conflict-free memory is proposed to resolve data hazards caused by the non-structured nature of these codes. A complete (2, 4)-regular, (960, 480) GF(64) nonbinary LDPC decoder is demonstrated on a Xilinx Virtex-5 FPGA. The decoder delivers an excellent error-correcting performance at a 9.76 Mb/s coded throughput, representing a significant improvement of state-of-the-art extended min-sum decoder implementations.**

I. Introduction

Nonbinary low-density parity-check (NB-LDPC) codes designed in high order Galois fields (GF) have shown great potential even in a moderate block length [1], [2]. Compared to binary LDPC codes and Turbo codes, NB-LDPC codes perform closer to the Shannon limit with good error floors.

NB-LDPC codes are decoded by the belief propagation (BP) algorithm [1], [2]. Unlike binary LDPC decoding, each message exchanged between processing nodes in a factor graph is an array of likelihood ratios corresponding to each possible GF element. In particular, check processing node (CN) is implemented as a forward-backward recursion that demands high memory and a large number of gates, and the complexity of which grows as a quadratic function of the GF size q and a linear function of the check node degree d_c. Simplifications have been proposed, such as the truncated extended min-sum (EMS) algorithm [3], [4] and the min-max algorithm [5]. Variable processing node (VN) complexity grows as a linear function of q and the variable node degree d_v. VN complexity is important as a large number of VNs are needed for the popular layered decoding scheme [6].

The decoder complexity is to a large extent determined by code designs. The error-correcting performance of moderate block length NB-LDPC codes can be improved by increasing q, [1], [2], but it significantly increases the decoder complexity. Good and low-complexity NB-LDPC code constructions have also been proposed, such as quasi-cyclic codes [7], [8] and regular (2, d_c) codes [9]. The quasi-cyclic codes enabled an efficient partial parallel layered min-max decoder that achieved a 9.3 Mb/s throughput [10], [11]. A class of regular (2, d_c) NB-LDPC codes based on the algebraic properties of their binary images offers excellent error-correcting performance and a low decoder complexity [4], [12]. The sparsity of this class of codes simplifies the decoder architecture, but the codes' lack of systematic structure introduces data dependencies, complicating high-throughput decoder designs.

We base our work on the truncated EMS algorithm [4], which reduces the number of GF elements to process in each decoding step from q to n_m, $n_m < q$, while still ensures an excellent functional performance. We adapt the bubble check algorithm [13] in a low-latency, prefetching elementary CN (ECN) that relaxes redundancy control with a small loss in functional performance at low error rate. A new skimming algorithm is proposed to reduce the latency of VN to match CN for the perfect interleaving in an efficient layered decoding architecture. To achieve the highest possible pipeline efficiency, we propose a conflict-free memory to resolve data dependencies caused by the non-structured codes. A complete (2, 4)-regular, (960, 480) GF(64) LDPC decoder is prototyped on a Xilinx Virtex-5 FPGA. The decoder achieves a 9.76 Mb/s throughput and good error-correcting performance.

II. Background

An NB-LDPC code is defined by a sparse parity-check matrix H whose components belong to a Galois field GF(q) [1], [2]. The H matrix of a regular (d_v, d_c) NB-LDPC code has constant column weight d_v and constant row weight d_c. The H matrix can also be represented by a factor graph, where each column is mapped to a variable node, and each row to a check node. An edge connects variable node v_j and check node c_i if $\mathrm{H}(i, j) \neq 0$. $N(i)$ denotes the set of variable nodes adjacent to check node c_i and $M(j)$ denotes the set of check nodes adjacent to variable node v_j. A parity check is specified by each row of the H matrix as $\sum_j \mathrm{H}(i, j)v_j = 0$, where i is the row index and $j \in N(i)$.

A class of (2, d_c)-regular codes constructed based on their binary images have shown excellent performance even with a low d_c [9], which also permits a low-complexity decoder implementation. We select a (2, 4)-regular (960, 480) code over GF(64) for our investigation. The performance of this code was shown to be better than the Davey-MacKay method [9]. The advantage is more pronounced at low error rate, which is important for high-throughput applications that require a low error floor.

A. Decoding Algorithm

An NB-LDPC code is decoded by an iterative message passing on a factor graph. A number of efficient algorithms have been proposed with different error-correcting performance and implementation complexity [3]-[5], [14]. The EMS algorithm offers a good tradeoff [3]: it achieves a performance close to the original BP algorithm [1], [2] and its complexity is relatively low. The truncated EMS algorithm achieves an even lower complexity and demonstrates great potential for practical adoption [4].

For completeness, we briefly introduce the five steps of the truncated EMS algorithm [4]: (1) variable nodes are initialized with prior log-likelihood ratios (LLR): one LLR is associated with one of the q GF elements. The largest n_m ($n_m < q$) LLRs

This research is supported by NSF under Grant CCF-1054270.

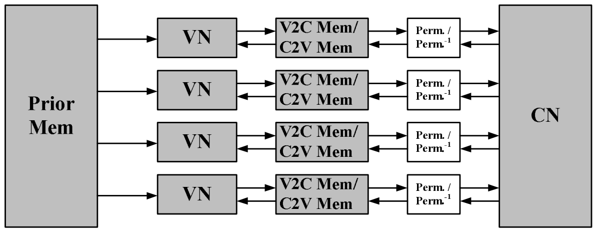

Fig. 1. Layered architecture of a (2, 4)-regular NB-LDPC code.

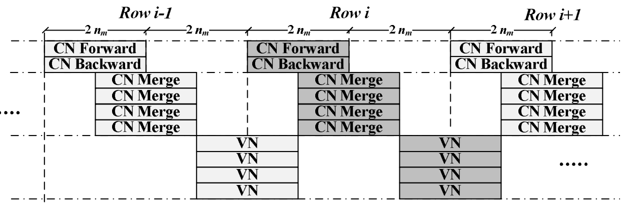

Fig. 2. Scheduling of the (2, 4) layered decoder architecture.

along with the corresponding GF element (or index) are stored in vector L and β_L respectively in descending order; (2) variable-to-check (v-c) messages are permuted based on the H matrix and sent to the check nodes. In the first iteration, the priors are used as the v-c messages; (3) for each adjacent variable node v_j, check node c_i computes the check-to-variable (c-v) message $\{V_{ij}[k]\}$, $k \in \{0, \ldots, n_m - 1\}$, that the parity-check equation is satisfied if $v_j = \beta_{Vij}[k]$ (where $\beta_{Vij}[k]$ is a vector containing GF elements or indices). The computation is done using a forward-backward recursion. Note that only the n_m highest probabilities are computed and stored; (4) the c-v messages are inverse permuted before being sent to the variable nodes; (5) each variable node v_j is updated with messages from the adjacent check nodes. A v-c message $\{U_{ji}[k]\}$, $k \in \{0, \ldots, n_m - 1\}$, is computed for each adjacent check node c_i, based on the prior L and all received c-v messages except from check node c_i. The procedure repeats itself from step (2).

B. *Decoder Architecture and Challenges*

A high-level block diagram of a row-layered NB-LDPC decoder architecture is shown in Fig. 1 for the selected (2, 4)-regular code. The architecture consists of 4 VNs and 1 CN. The processing schedule for this architecture is shown in Fig. 2. CN reads v-c messages and performs forward-backward recursion on a 4-stage trellis in three steps: (1) 1 forward step, (2) 1 backward step, and (3) 4 merging steps. Forward and backward steps can be overlapped, and 4 merging steps can be overlapped too. Each step in the recursion is done by an elementary CN (ECN). ECN carries out the max-log computation and sorts the results. The sorter length is n_m and latency of each ECN operation is at least $2n_m$ clock cycles. The bubble check algorithm [13] reduces the sorter length to approximately $\sqrt{n_m}$.

After the CN operation is complete and c-v messages have been written to memory, VN operation is started. A v-c message is calculated by adding the c-v message and the prior. Note that the addition requires the matching of GF indices, where a content-addressable memory is required. The results also need to be sorted, with a latency of at least $2n_m$ cycles. Altogether the processing of one row of the H matrix takes $4n_m$ cycles. With 80 rows in the selected code and $n_m = 16$, one decoding iteration takes at least 5,120 cycles.

The decoding latency and throughput can be further degraded due to the structure of the (2, d_c)-regular codes. Pipeline stalls need to be inserted to resolve data dependencies. The challenges with the decoder architecture call for new processing node designs and efficient memory access schemes.

III. Design for Low Latency and High Throughput

We introduce improvements in both CN and VN operations in order to enable a pipeline with lower latency and higher throughput.

A. *Low-Latency ECN and Improved Pipeline Schedule*

ECN is the elementary building block of CN, and it is used for forward, backward, or merging operation. ECN takes two LLR vectors U and I along with the corresponding GF indices β_U and β_I to produce V and β_V using the max-log algorithm [4]: $V[i] = \max_{S(\beta V[i])} (U[j] + I[p])$, $i, j, p \in \{0, \ldots, n_m - 1\}$ and $S(\beta_V[i])$ is the set of all combinations of $\beta_V[i]$, $\beta_U[j]$, and $\beta_I[p]$ that satisfy $\beta_V[i] + \beta_U[j] + \beta_I[p] = 0$ over GF(q).

The proposed low-latency ECN is based on the bubble check algorithm [13]. We improve upon the original algorithm by pre-fetching and relaxing redundancy control (i.e., allowing duplicate GF entries), which together shorten the latency of an ECN operation from at least $2n_m$ to $n_m + L_{S\text{-}ECN} + 2$ clock cycles, where $L_{S\text{-}ECN}$ is the sorter length used in ECN and $L_{S\text{-}ECN} < n_m$ [13]. Simulation shows that relaxing redundancy control introduces functional performance loss at high error rate, but the loss becomes negligible at low error rate that is of more practical interest. The proposed ECN is described below, assuming both U and I are sorted in descending order:

```
(1) Initialization: insert $U[j] + I[0]$, where $j = 0, \ldots, L_{S\text{-}ECN} - 1$, sequentially to the sorter in descending order.
(2) Set $j_{curr} = 0$ and $p_{curr} = 1$.
(3) Fetch $U[j_{curr}]$ and $I[p_{curr}]$. Compute $S_{in} = U[j_{curr}] + I[p_{curr}]$. Insert $S_{in}$. If the sorter is full, output the maximum value in the sorter $S_{max}$ before inserting $S_{in}$.
(4) Find the next pair of indices $j_{next}$ and $p_{next}$. (Define a direction flag $R$, with $R = 1$ initially. The second largest value in the sorter is denoted $S_{max,2}$, which is the sum of $U[j_{max,2}]$ and $I[p_{max,2}]$.)
    (a) If ($S_{max,2} < S_{in}$), then $j = j_{curr}$ and $p = p_{curr}$,
        else $j = j_{max,2}$ and $p = p_{max,2}$
    (b) if ($j = 0$), then $R = 1$,
        else if ($p = 0$ and $j \geq L_{S\text{-}ECN} - 1$), $R = 0$.
    (c) Set $j_{next} = j + !R$ and $p_{next} = p + R$ (! denotes inversion).
(5) Set $j_{curr} = j_{next}$ and $p_{curr} = p_{next}$.
(6) Go back to (3) until all $n_m$ values have been output from the sorter.
```

The main difference between the proposed algorithm and the original bubble check algorithm is the use of $S_{max,2}$ and S_{in} to decide one cycle ahead the next inputs to the sorter. The prefetching shortens the latency because both sorting and reading can execute concurrently without waiting. The algorithm stops after n_m outputs, and redundancy (duplicate GF entries) in the output is permitted. The high-level architecture of the ECN is shown in Fig. 3.

The length of the sorter $L_{S\text{-}ECN}$ is determined by n_m. For the case of $n_m = 16$, the maximum number of pending candidates will be 6 and therefore $L_{S\text{-}ECN} = 6$ is the best choice [13]. The latency of the first ECN output is $L_{S\text{-}ECN} + 2 = 8$ cycles, which enables early starts on subsequent ECN or VN operations. In total, one complete ECN operation takes 24 cycles to produce 16 outputs.

The proposed ECN allows the pipeline latency of CN to be shortened and throughput improved. Fig. 4 shows the updated pipeline schedule, where the 4 parallel merge steps start when the first output from the forward and backward steps are ready. With a short 8-cycle latency, the merge steps are effectively overlapped with the forward and backward steps. Furthermore, using two sets of memories in CN RAM for alternating rows, the forward and backward step of the next row can start right after the current row is complete, hence making full utilization of the hardware. The improved pipeline schedule enables a complete CN processing in every 24 cycles.

B. Low-Latency VN

In order to fully take advantage of the latency reduction of the CN and the updated pipeline schedule, the VN operation also needs to be improved to avoid becoming the bottleneck. The four outputs produced by the CN need to be processed in parallel by VNs within 24 cycles, or else it will stop the pipeline.

A VN takes two LLR vectors V (c-v message) and L (prior likelihoods) along with their GF indices β_V and β_L to produce a new U (v-c message) and its GF index β_U [5]. Since $n_m < q$, the VN operation needs to scan both the V vector and L vector for matching entries with a latency of at least $2n_m$.

We propose a simplified VN algorithm to achieve a low-latency by skimming the prior messages, i.e., allocate only $L_{S\text{-}VN}$ cycles to scan the L vector, where $L_{S\text{-}VN}$ is the sorter length used in VN and $L_{S\text{-}VN} < n_m$. Due to limited space, we will simply introduce the algorithm and defer the full discussion to a follow-up publication. The algorithm is divided into two stages, assuming both L and V are sorted in descending order:

(1) Scan the top $L_{S\text{-}VN}$ entries in β_L sequentially: search $\beta_L[l]$, $l \in \{0, \ldots, L_{S\text{-}VN} - 1\}$, in β_V for matching entries. Compute $S_{\text{in}} = L[l] + V[i]$, if $\beta_L[l] = \beta_V[i]$; or $S_{\text{in}} = L[l] + Y_V$, if no matching entry is found, where Y_V is a compensation constant based on $V[n_m - 1]$. Insert S_{in} to the sorter.

(2) Scan the n_m entries in V sequentially. Compute $S_{\text{in}} = V[i] + Y_L$, where Y_L is a compensation constant based on $L[n_m - 1]$. Insert S_{in} to the sorter. The maximum entry in the sorter is output every time a new S_{in} is inserted to the sorter.

The architecture of the proposed VN is shown in Fig. 5. A content-addressable memory is used to search matching entries. If we choose $L_{S\text{-}VN} = 6$, a complete VN operation takes 24 cycles to produce 16 outputs for the perfect interleaving with CN as shown in Fig. 4. Note that since $L_{S\text{-}VN} < n_m$, some low order entries in vector L will be missed, which degrades the functional performance by 0.65 dB at FER of 10^{-5} as shown in Fig. 6. To compensate the performance loss, we can increase $L_{S\text{-}VN}$ and use two sets of VNs to accommodate a higher processing latency without stopping the pipeline. In this way, while the first set is working on one row, the second set starts to process the second row immediately when the inputs are available. Alternatively, a short pipeline stall can be inserted. Simulation results in Fig. 6 show that a small increase to $L_{S\text{-}VN} = 10$ eliminates the performance loss.

IV. Memory Conflict Resolution

Unlike quasi-cyclic codes, the (2, d_c)-regular NB-LDPC code lacks a systematic structure. When multiple VNs operate in parallel, both intra- and inter-iteration memory access conflicts arise, causing pipeline stalls. In particular, CN reads 4 v-c messages at the same time, requiring them to be stored in 4 sets of memory. CN then passes the c-v messages to 4 VNs that will write their v-c messages to a centralized memory to be read by later CN processing. However, as illustrated in Fig. 7, writing to the v-c memory can be problematic because conflicts can occur when two or more VNs write to the same memory set. The worst-case conflict occurs when all 4 VNs are attempting to write to one set. Our solution is to subdivide each set into smaller subsets determined by the code structure. The conflicting v-c messages would be written to different subsets within the same set. Note that this solution would require a separate look-up table in each set to select the correct subset for each read and write access, but the size of the v-c memory will remain the same.

Read-after-write (RAW) hazards can also occur due to data dependencies across consecutive iterations. For instance, if the VN that writes back to the first row shown in Fig. 7 does not finish in time (e.g., $v_0 \rightarrow c_0$), the next iteration would read the old values. We resolve this inter-iteration data dependency by shuffling the rows in the H matrix such that the last row does not produce any output that is needed by the first row. The conflict-free scheme ensures a stall-free pipeline for a high throughput.

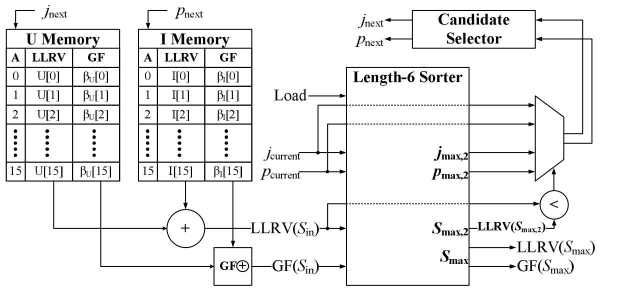

Fig. 3 ECN architecture.

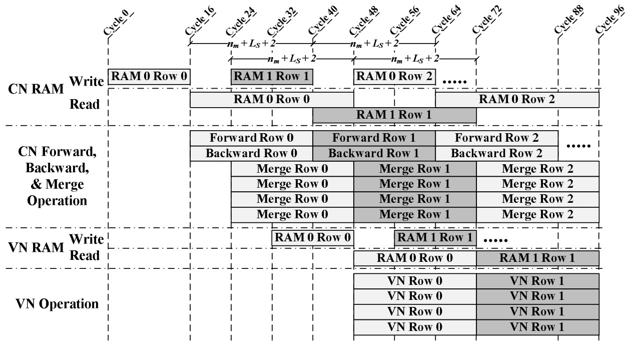

Fig. 4. Overlapped pipeline schedule.

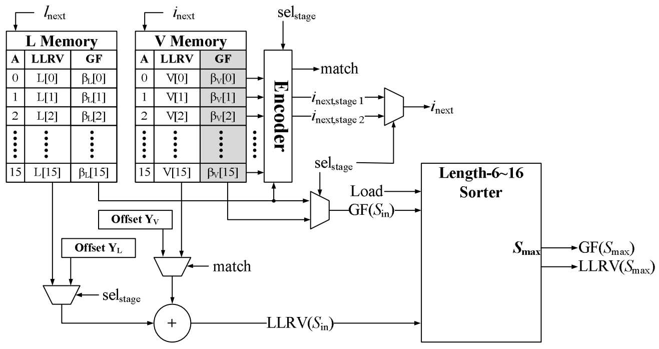

Fig. 5. VN architecture.

TABLE I FPGA MAPPING RESULTS
(BASED ON XILINX VIRTEX-5 XC5VLX155T)

Resource	1× parallel (single-set VN, $L_{S\text{-}VN} = 6$)	1× parallel (dual-set VN, $L_{S\text{-}VN} = 16$)	4× parallel (single-set VN, $L_{S\text{-}VN} = 6$)
Slice Registers	12,444 (13%)	16,533 (17%)	49,847 (54%)
Slice LUTs	15,099 (15%)	19,505 (20%)	64,423 (64%)
Occupied Slices	5,954 (24%)	8,128 (33%)	23,839 (97%)
BRAMs	49 (23%)	82 (39%)	197 (93%)

V. FPGA PROTOYPING

The proposed decoder architecture has been prototyped for a (2, 4)-regular (960, 480) NB-LDPC code over GF(64) on a Xilinx Virtex-5 FPGA. Table I shows the hardware utilization of the two approaches proposed in Section III: one is based on $L_{S\text{-}VN} = 6$ using one set of 4 VNs, and the other based on $L_{S\text{-}VN} = 16$ using 8 VNs grouped in two sets. The latter costs more hardware resources and but produces better functional performance. An 8-bit fixed-point quantization has been used in all the implementations.

The FPGA decoders operate at a clock frequency of 100 MHz for a coded throughput of 2.44 Mb/s with 10 decoding iterations. We were able to map a 4× parallel decoder based on the proposed architecture on this FPGA device for a throughput of 9.76 Mb/s. The result represents a significant improvement over the state-of-the-art EMS decoder implementations [4], [12], [15]. By layered decoding, the convergence speed is improved and simulations show that the average number of iterations is reduced from 2.39 to 1.77 at FER = 10^{-5}.

VI. CONCLUSION

We present a high-throughput, low-latency decoder architecture and an FPGA prototype for the (2, d_c)-regular NB-LDPC codes. New VN and CN designs based on skimming, prefetching and relaxing redundancy control are proposed to reduce latency and enable an efficient pipeline schedule. A conflict-free memory resolves data hazards to avoid pipeline stalls altogether. These new techniques have been applied in a 9.76 Mb/s NB-LDPC decoder design on a Xilinx Virtex-5 FPGA. Results show good decoding performance down to low error rate levels.

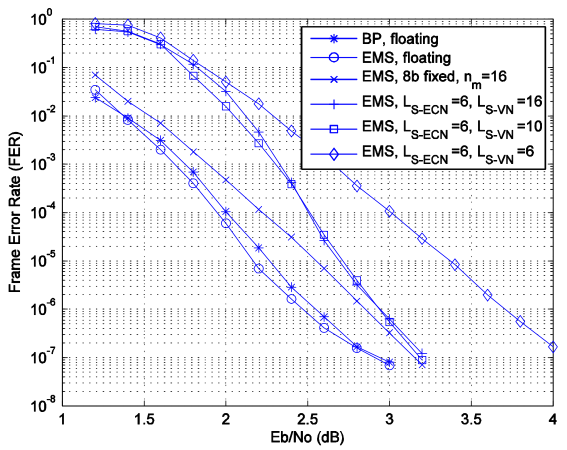

Fig. 6. Performances of a (2, 4)-regular, (960, 480) NB-LDPC over GF(64).

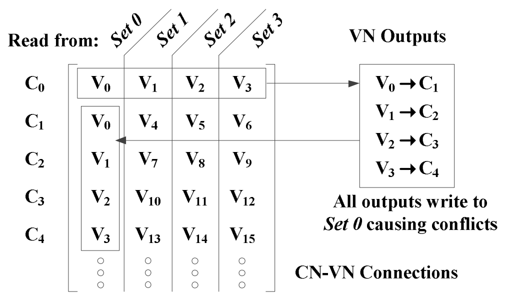

Fig. 7. Example of worst case intra-iteration memory conflicts.

REFERENCES

[1] M. C. Davey and D. Mackay, “Low-density parity check codes over GF(q),” *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165-167, Jun. 1998.

[2] M. C. Davey, “Error-correction using low-density parity-check codes,” Ph.D. dissertation, Univ. Cambridge, Cambridge, UK, 1999.

[3] D. Declercq and M. Fossorier, “Decoding algorithms for nonbinary LDPC codes over GF(q),” *IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633-643, Apr. 2007.

[4] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, “Low-complexity decoding for non-binary LDPC codes in high order fields,” *IEEE Trans. Commun.*, vol. 58, no. 5, pp.1365-1375, May 2010.

[5] V. Savin, “Min-max decoding for non binary LDPC codes,” in *IEEE Int. Symp. Information Theory*, Toronto, Canada, Jul. 2008, pp. 960-964.

[6] D. E. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes,” in *IEEE Workshop Signal Process. Syst.*, Austin, TX, Oct. 2004, pp. 107-112.

[7] L. Zeng, L. Lan, Y. Y. Tai, S. Song, S. Lin, and K. Abdel-Ghaffar, “Constructions of nonbinary quasi-cyclic LDPC codes: a finite field approach,” *IEEE Trans. Commun.*, vol. 56, no. 4, pp. 545-554, Apr. 2008.

[8] B. Zhou, J. Kang, S. Song, S. Lin, K. Abdel-Ghaffar, and M. Xu, “Construction of non-binary quasi-cyclic LDPC codes by arrays and array dispersions,” *IEEE Trans. Commun.,* vol. 57, no. 6, pp. 1652-1662, Jun. 2009.

[9] C. Poulliat, M. Fossorier, and D. Declercq, “Design of regular (2, d_c) – LDPC codes over GF(q) using their binary images,” *IEEE Trans. Commun.,* vol. 56, no. 10, pp. 1626-1635, Oct. 2008.

[10] X. Zhang and F. Cai, “Efficient partial-parallel decoder architecuture for quasi-cyclic nonbinary LDPC Codes,” *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 58, no. 2, pp. 402-414, Feb. 2011.

[11] X. Zhang and F. Cai, “Reduced-complexity decoder architecure for non-binary LDPC codes,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 19, no. 7, pp.1229-1238, Jul. 2011.

[12] A. Voicila, F. Verdier, D. Declercq, M. Fossorier, and P. Urard, “Architecture of a low-complexity non-binary LDPC decoder for high order fields,” in *IEEE Int. Symp. Communications and Information Technology*, Sydney, Australia, Oct. 2007, pp.1201-1206.

[13] E. Boutillon and L. Conde-Canencia, “Bubble check: a simplified algorithm for elementary check node processing in extended min-sum non-binary LDPC decoders,” *IEE Electron. Lett.*, vol. 46, no. 9, pp. 633-634, Aug. 2010.

[14] C. Spagnol, E. M. Popovici, and W. P. Marnane, “Hardware implementation of GF(2^m) LDPC decoders,” *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 56, no. 12, pp. 2609-2620, Dec. 2009.

[15] J. Lin, J. Sha, Z. Wang, and L. Li, “Efficient decoder design for nonbinary quasicyclic LDPC codes,” *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 57, no. 5, pp. 1071-1082, May 2010.