# Efficient Check Node Processing for Min-Max NB-LDPC Decoding over Lower-Order Finite Fields

Xinmiao Zhang
The Ohio State University

***Abstract*—Low-density parity-check (LDPC) codes defined over non-binary (NB) finite field $GF(2^q)$ $(q > 1)$ achieve better error-correcting performance than binary LDPC codes when the codeword length is moderate. The decoder complexity increases very fast with the order of the field, $2^q$, although the error-correcting performance also improves. To reduce the complexity for practical applications, NB-LDPC codes over lower-order finite fields are of great interest. Previous designs have been focusing on decoders over either the smallest NB field $GF(4)$ or much larger fields, such as $GF(32)$ or higher. Prior optimization techniques are either not applicable or do not lead to efficient designs for codes over $GF(8)$ or other lower-order fields. In this paper, an efficient architecture is developed for the check node processing, which is the most complicated step in NB-LDPC decoding, for the Min-max algorithm over lower-order fields. By utilizing the properties of finite field elements, the max/min comparison results are shared and the number of comparators needed is reduced significantly. Compared to the best prior design, the proposed check node processing has 18.5% smaller area and shorter critical path for an example code over $GF(8)$.**

## I. Introduction

When the codeword length is moderate, non-binary low-density parity-check (NB-LDPC) codes over $GF(2^q)$ $(q > 1)$ achieve better error-correcting performance than their binary counterparts. However, the decoder complexity increases very fast with $q$. Hence the codes over lower-order fields, such as $GF(8)$, are preferred for practical systems. The simplest NB-LDPC decoding algorithm is the Min-max algorithm [1]. It replaces the complicated computations in the check node unit (CNU) by 'min' and 'max' comparisons and only leads to small performance loss compared to belief propagation.

For the first time, trellis representation of the CNU input vectors is introduced in [2]. CNU output messages are mapped to paths in the trellis. This mapping enables parallel processing and elimination of redundant computations. It was found in [3] that a path for output message computation can include more than one node from the same column in the trellis without bringing noticeable performance loss. The CNU complexity is further reduced in [4] by constructing a basis, from which all the messages in the corresponding output vector are generated using a combinational logic network. Redundancy among the computations of different output vectors is reduced by adding an extra column in the trellis to represent intrinsic messages that include the contributions of the nodes from every column [5]. Then the contributions from the nodes in column $n$ are excluded to derive the $n$-th output vector. Further improvements of this scheme have been proposed in [6]–[10]. Particularly, the $L$-trellis Min-max algorithm (L-TMMA) [8] computes the $L<q$ most reliable intrinsic messages and uses them to compute or approximate the output messages. The basic-set (BS-)TMMA [9] applies basis construction [4] to intrinsic message computation. The optimized (O-)TMMA [10] eliminates unnecessary sorting and simplifies the generation of the output vector from the most reliable intrinsic messages.

Despite the available simplification techniques, the complexity of NB CNUs is still much higher than that of binary CNUs. Besides, the storage requirement increases linearly with $2^q$. Hence, NB-LDPC codes over lower-order finite fields, such as $GF(8)$, are preferred for practical applications. The most efficient prior designs [8]–[10] utilize a small number of intrinsic messages as compressed messages. However, the required sorting and decompression lead to large area. As a result, these designs are not efficient when $2^q$ is smaller. Utilizing the property that $GF(4)$ only has three nonzero field elements, only one intrinsic message needs to be computed [11]. However, this design is not extendable to other fields.

In this paper, an efficient CNU architecture is proposed for NB-LDPC codes over lower-order finite fields. From the properties of finite fields, the pairs of messages to compare in the computation of different intrinsic messages share common entries. By analyzing the possible comparison results, the number of comparators needed is greatly reduced. Besides, an optimized procedure is provided to divide the comparison pairs into groups sharing common entries to minimize the number of comparators needed. The proposed CNU does not require expensive sorting for the intrinsic message computation and recovers the output vector by simple multiplexers. For an example code over $GF(8)$, the proposed CNU architecture requires at least 18.5% smaller area than prior designs utilizing compressed messages and has shorter critical path.

## II. Min-max NB-LDPC check node processing

An LDPC code is defined by a sparse parity check matrix $H$. A codeword, $c$, satisfies $cH^T = 0$. $H$ is also represented by a Tanner graph, in which the check and variable nodes are associated with the rows and columns of $H$, respectively. A nonzero entry in $H$ indicates that the corresponding check and variable nodes are connected by an edge. In the Min-max decoding algorithm for NB-LDPC codes over $GF(2^q)$, a message vector consists of $2^q$ log-likelihood ratios (LLRs)

This material is based upon work supported by the National Science Foundation under Award No. 2052641

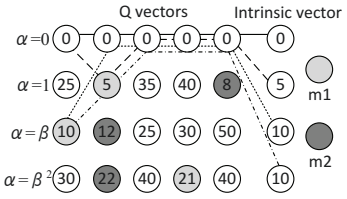


Fig. 1. Example trellis for codes over $GF(4)$

defined as $\log(P(\hat{\alpha})/P(\alpha))$, where $\alpha \in GF(2^q)$ and $\hat{\alpha}$ is the most likely field element. Such vectors are iteratively passed among the connected check and variable nodes to find a codeword. Denote the LLR vector from check (variable) node $m$ ($n$) to variable (check) node $n$ ($m$) by $R_{m,n}$ ($Q_{m,n}$). In each decoding iteration, the check node processing computes

$$R_{m,n}(\alpha) = \min_{(a_j)\in\mathcal{L}(m|a_n=\alpha)}(\max_{j\in S_v(m)\backslash n} Q_{m,j}(a_j)),$$

where $S_v(m)$ is the set of variable nodes connected to check node $m$ and $\mathcal{L}(m|a_n = \alpha)$ is called the configuration set. Each configuration is a sequence of finite field elements $(a_j)$ $(j \in S_v(m)\backslash n)$ such that $\sum_{j\in S_v(m)\backslash n} a_j = \alpha$ if the multiplications by the entries of $H$ are handled by separate units. The variable node processing adds the LLRs of the same finite field element from the connected check nodes.

For a code of row weight $d_c$, the $d_c$ $Q$ vectors sent to a CNU can be represented by a $d_c$-column trellis [2]. Fig. 1 shows an example trellis for a code over $GF(4)$ with $d_c = 5$. Utilizing the transformation $Q_{m,n}(\alpha) \Leftarrow Q_{m,n}(\alpha + \hat{\alpha})$ [3], $Q_{m,n}(0)$ becomes 0. $\mathcal{L}(m|a_n = \alpha)$ can be simplified to $\mathcal{L}_{(1,2)}(m|a_n = \alpha)$ that only considers the node with minimum LLR in each row of the trellis and includes up to 2 such nodes in each configuration [5]. The none-zero LLR nodes in a configuration are called deviation nodes. To reduce the redundancy among the computations of different $R$ vectors, an extra column is added to the trellis for the intrinsic messages [5]

$$I_m(\alpha) = \min_{(a_j)\in\mathcal{T}_{(1,2)}(m|\alpha)}(\max_{j\in S_v(m)} Q_{m,j}(a_j)). \quad (1)$$

The paths shown by the dashed lines in Fig. 1 are the configurations leading to $I_m(\alpha)$ and the values in the nodes are LLRs. $\mathcal{T}(m|\alpha)$ includes the contributions of the nodes in every column. Then the $R$ vector to variable node $n$ can be computed by excluding the contributions of the nodes in column $n$ from $I_m(\alpha)$ as

$$R_{m,n}(\alpha) = \begin{cases} I_m(\alpha), \text{ if } (n \neq col(\alpha'))\&(n \neq col(\alpha'')) \\ m2(\alpha), \text{ if } (n = col(\alpha') = col(\alpha'')) \\ m1(\alpha), \text{ otherwise} \end{cases} \quad (2)$$

In (2), $m1(\alpha)$ and $m2(\alpha)$ are the minimum and second minimum among all $Q_{m,j}(\alpha)$ for $0 \leq j < d_c$ and $col(\alpha)$ is the column index of the $m1(\alpha)$ node. The configuration leading to $I_m(\alpha)$ is represented as $(\alpha', \alpha'')$. In other words, $\alpha' + \alpha'' = \alpha$ and $\max(m1(\alpha'), m1(\alpha''))$ is not larger than the maximum LLR of any other configuration in $\mathcal{T}_{(1,2)}(m|\alpha)$. $\alpha' = \alpha''$ means that there is only one deviation node.

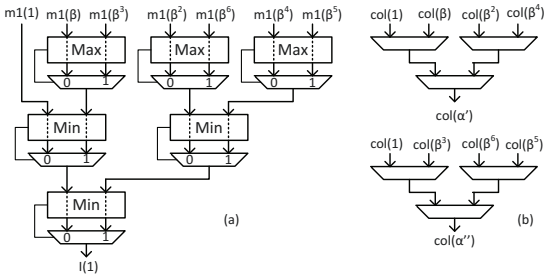


Fig. 2. Architectures for computing (a) $I(1)$; (b) indices of the corresponding deviation nodes for codes over $GF(8)$

To simplify the notations, the subscript '$m$' is dropped when no ambiguity occurs. To further reduce the complexity, the L-TMMA design in [8] sorts out the $L < 2^q$ smallest $m1(\alpha)$ in parallel and only computes the $L$ smallest $I(\alpha)$ as compressed messages. Approximations are used for the other un-computed $I(\alpha)$ and an $E$ vector is used to pre-select between $m1(\alpha)$, $m2(\alpha)$ and the approximated $I(\alpha)$. Then a large multiplexer network recovers the $R$ messages. In the BS-TMMA [9], a basis of $I$ is derived. Then every element in $I$ can be easily computed from the basis. However, the same expensive $L$-parallel sorter as in [8] is used and the calculation of the basis itself has high complexity. The O-TMMA [10] has a more efficient $L$-parallel sorter. Nevertheless, the derivation of the $I$ and $E$ vectors from the compressed messages is still complex.

## III. Min-max CNU for lower-order finite fields

When $q$ is large, compressing or using a basis of the intrinsic messages leads to significant CNU simplification. However, the required parallel message sorting or basis computation itself brings large overhead. Besides, complicated selection networks are needed to derive the CNU output vectors from the compressed messages or basis. As a result, prior designs targeting for codes over large finite fields are not efficient for codes over lower-order fields, such as $GF(8)$, which are more suitable for practical applications for complexity reason. A simplified CNU for codes over $GF(4)$ is developed in [11]. Since there are only 3 nonzero elements in $GF(4)$, a single instead of $4-1=3$ intrinsic messages needs to be computed. However, such simplification is not applicable to codes over other fields.

This section develops an efficient CNU architecture for codes over lower-order finite fields. Through utilizing the properties of finite field elements, the configurations for computing different intrinsic messages are divided into groups that share common entries. As a result, the max/min comparison results can be shared and the number of comparators needed to derive $I$ is substantially reduced. Once $I$ is available, simple selection according to (2) generates the CNU output vectors. The proposed scheme does not require complex compression or decompression selection and leads to CNU of much lower complexity for codes over smaller fields.

The proposed idea is explained by using $GF(8)$ as an example, while it is extendable to other fields. Assume that $GF(8)$ is constructed by using irreducible polynomial $x^3$+$x$+1, and

TABLE I
CONFIGURATION SETS FOR INTRINSIC MESSAGE COMPUTATION FOR NB-LDPC CODES OVER $GF(8)$

| | configuration set | | | |
|---|---|---|---|---|
| $I(1)$ | $(1, 0)$ | $(\beta, \beta^3)$ | $(\beta^2, \beta^6)$ | $(\beta^4, \beta^5)$ |
| $I(\beta)$ | $(\beta, 0)$ | $(1, \beta^3)$ | $(\beta^2, \beta^4)$ | $(\beta^5, \beta^6)$ |
| $I(\beta^2)$ | $(\beta^2, 0)$ | $(1, \beta^6)$ | $(\beta, \beta^4)$ | $(\beta^3, \beta^5)$ |
| $I(\beta^3)$ | $(\beta^3, 0)$ | $(1, \beta)$ | $(\beta^2, \beta^5)$ | $(\beta^4, \beta^6)$ |
| $I(\beta^4)$ | $(\beta^4, 0)$ | $(1, \beta^5)$ | $(\beta, \beta^2)$ | $(\beta^3, \beta^6)$ |
| $I(\beta^5)$ | $(\beta^5, 0)$ | $(1, \beta^4)$ | $(\beta, \beta^6)$ | $(\beta^2, \beta^3)$ |
| $I(\beta^6)$ | $(\beta^6, 0)$ | $(1, \beta^2)$ | $(\beta, \beta^5)$ | $(\beta^3, \beta^4)$ |

TABLE II
PARTIAL ORDER DECIDED FROM THE MIN-MAX COMPUTATION OF $(a, b), (c, d)$ AND THE MIN-MAX RESULT OF $(a, d), (b, c)$

| $f_0 f_1 f_2$ | order of $a, b, c, d$ | $\min(\max(a, d), \max(b, c))$ |
|---|---|---|
| 110 | $a < b < d$; $c < d$ | $\max(b, c)$ |
| 111 | $c < d < b$; $a < b$ | $\max(a, d)$ |
| 100 | $a < b < c$; $d < c$ | $\max(a, d)$ |
| 101 | $d < c < b$; $a < b$ | $\max(a, d)$ |
| 010 | $b < a < d$; $c < d$ | $\max(b, c)$ |
| 011 | $c < d < a$; $b < a$ | $\max(b, c)$ |
| 000 | $b < a < c$; $d < c$ | $\max(a, d)$ |
| 001 | $d < c < a$; $b < a$ | $\max(b, c)$ |

one of its root is $\beta$. Then 1=$\beta$+$\beta^3$=$\beta^2$+$\beta^6$=$\beta^4$+$\beta^5$. To simplify the notations, assume that more than one node in the same column of the trellis are allowed to be in a configuration. Such relaxation only brings negligible performance loss [2], [3], [9], [11]. In this case, $I(1)$=$\min(m1(1), \max(m1(\beta), m1(\beta^3))$, $\max(m1(\beta^2), m1(\beta^6))$, $\max(m1(\beta^4), m1(\beta^5)))$, and it can be implemented by the architecture in Fig. 2(a). It requires 6 max/min comparators and 6 multiplexers. Besides, using the comparison results, the column indices of the deviations nodes, $col(\alpha')$ and $col(\alpha'')$, can be derived as shown in Fig. 2(b). To compute another message in $I$, different pairs of $m1$ values according to the configurations listed in Table I are sent to the max comparators in Fig. 2 (a) and the inputs to the architecture in Fig. 2(b) are adjusted accordingly. In total, $q-1=7$ copies of such architectures are needed to compute $I$ in parallel.

The computations of different intrinsic messages do not have to be independent since their configurations share common entries. For example, from Table I, $(\beta, \beta^3)$ and $(\beta^2, \beta^6)$ are configurations for $I(1)$ computation. The same four elements in these configurations also appear in the two configurations, $(\beta, \beta^6)$ and $(\beta^2, \beta^3)$, for calculating $I(\beta^5)$ although they are in different combinations. Due to the shared entries, the number of comparisons needed to derive the min-max of the LLRs corresponding to these configurations can be reduced. Consider in general that the min-max of $(a, b)$, $(c, d)$ and $(a, d)$, $(b, c)$ need to be computed. The min-max of $(a, b)$, $(c, d)$ can be calculated using the three comparators on the left side of Fig. 3. Each max (min) comparator generates a flag that equals to '0' when the left input is larger (smaller) than the right input. Denote the three flags by $f_0, f_1, f_2$ as shown in Fig. 3. They tell partial orders of $a, b, c, d$ as listed in Table II. From the partial orders, it can be derived that the min-max of $(a, d)$, $(b, c)$ equals to $\max(a, d)$ or $\max(b, c)$ when $\bar{f}_2 f_1 + f_2 \bar{f}_0$='0' and '1', respectively. Hence, it can be

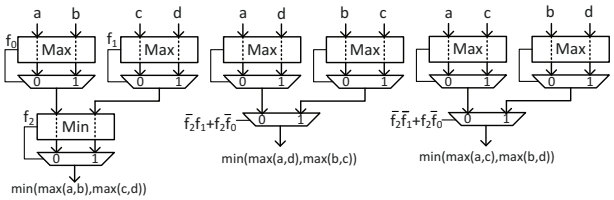


Fig. 3. Min-max computation architecture for configuration pairs with common entries.

calculated by the middle part of the architecture in Fig. 3, which saved one comparator compared to carrying out the two groups of min-max computations separately. Similarly, if the min-max of $(a, c), (b, d)$ needs to be calculated, the last min comparator can be also eliminated as shown in Fig. 3.

Now the question is how to divide the configurations into as many as possible pairs that share common entries. Basically, each configuration for $I(\alpha)$ computation consists of two field elements whose sum is $\alpha$. The configuration with single element $\alpha$ can be considered as the configuration $(0, \alpha)$. Pick two configurations $(\gamma, \gamma')$ and $(\delta, \delta')$ for calculating $I(\alpha)$. Note that $\gamma, \gamma', \delta, \delta \in GF(2^q)$ are distinct and $\alpha = \gamma + \gamma' = \delta + \delta'$. Accordingly, $\gamma + \delta = \gamma' + \delta'$ and $\gamma + \delta' = \delta + \gamma'$. Let $\alpha_1 = \gamma + \delta$ and $\alpha_2 = \gamma + \delta'$. Then the same four elements can be also found in two configurations for computing $I(\alpha_1)$ and $I(\alpha_2)$. Pick another element $\sigma \in GF(2^q) \backslash \gamma, \gamma', \delta, \delta'$. Let $\eta = \sigma + \alpha_1$, $\sigma' = \alpha + \sigma$, $\eta' = \alpha + \eta$. Then $(\sigma, \sigma')$ and $(\eta, \eta')$ are configurations for computing $I(\alpha)$, and $(\sigma, \eta)$ and $(\sigma', \eta')$ are configurations for $I(\alpha_1)$ calculation. Note that $\alpha = \alpha_1 + \alpha_2$. Hence $\sigma + \eta' = \sigma + \alpha + \eta = \alpha + \alpha_1 = \alpha_2$. Accordingly, $(\sigma, \eta')$ and $(\sigma', \eta)$ are configurations for $I(\alpha_2)$ calculation. As a result, another three pairs of configurations for computing $I(\alpha)$, $I(\alpha_1)$, and $I(\alpha_2)$ share the same elements. This process can be repeated to divide all the configurations for $I(\alpha)$, $I(\alpha_1)$, and $I(\alpha_2)$ calculation into $2^{q-2}$ groups of 3-pairs. Accordingly, $2 \times 2^{q-2}$ comparators can be saved in the corresponding min-max computations using the proposed scheme.

To divide the configurations for computing other intrinsic messages into pairs sharing common entries, first pick a configuration, say $(\xi, \xi')$ used in calculating $I(\xi + \xi') = I(\phi)$ $(\phi \neq \alpha, \alpha_1, \alpha_2)$. Then look for another configuration $(\mu, \mu')$ for $I(\phi)$ calculation such that neither $\phi_1 = \xi + \mu$ nor $\phi_2 = \xi + \mu'$ equal any field elements of the previously considered intrinsic messages, *i.e.*, $\alpha, \alpha_1, \alpha_2$. Then groups of 3-pair configurations with common elements can be found similarly for calculating $I(\phi), I(\phi_1), I(\phi_2)$. There are $2^q - 1$ intrinsic messages to compute. Repeating this process, if $(2^q - 1) | 3$, all the intrinsic messages can be divided into tuples of three and all the configurations for each tuple can be divided into groups of 3-pairs. The remainder of $(2^q - 1)/3$ can not be 2 but may also be 1. In this case, the intrinsic messages can be divided into $(2^q - 5)/3$ tuples of 3 and 2 tuples of 2. Similarly, the configurations can be divided into groups of 3-pair and 2-pair, respectively, sharing common entries. Following this procedure, the configurations that share common entries for intrinsic message computation over $GF(8)$ can be identified

TABLE III
COMPARISONS OF CNU ARCHITECTURES WITH 5-BIT LLRS FOR CODES WITH $d_c = 32$ OVER $GF(8)$

| | $m1\&m2$ sorter (# of XORs) | $I$ comp. (# of XORs) | | | Register $I$ etc. store | $R$ comp. (# of XORs) | Total (# of XORs) | Crit. path (# of gates) |
|---|---|---|---|---|---|---|---|---|
| | | $L$-min sorter | compress $I$/ basis comp. | $E\&I$ comp. | | | | |
| L-TMMA [8] | 595 | 723 | 57 | 969 | 107 | 121 | 2786 | 15 |
| BS-TMMA [9] | 595 | 723 | 302 | 716 | 74 | 116 | 2674 | 16 |
| O-TMMA [10] | 595 | 524 | 57 | 562 | 167 | 126 | 2365 | 15 |
| proposed | 595 | 760 | | | 140 | 152 | 1927 | 12 |

TABLE IV
NUMBER OF COMPARATORS AND MULTIPLEXERS NEEDED FOR COMPUTING THE INTRINSIC MESSAGE VECTOR

| | $GF(8)$ | | $GF(16)$ | |
|---|---|---|---|---|
| | comparator | multiplexer | comparator | multiplexer |
| original | 42 | 42 | 210 | 210 |
| proposed | 34 | 42 | 170 | 210 |

as color-coded in Table I.

## IV. HARDWARE COMPLEXITY COMPARISONS

This section first analyzes the reduction achieved by the proposed design on the number of comparators needed for computing the intrinsic messages. Then the complexity of the overall CNU using the proposed design is compared with prior work for an example NB-LDPC code over $GF(8)$.

As shown in Fig. 2(a), $2^q - 1$ min/max comparators and multiplexers are needed originally to compute each intrinsic message over $GF(2^q)$. From Fig. 3, for two and three configuration pairs with the same entries, one and two, respectively, comparators are saved using the proposed scheme. From the analysis in the previous section, if $(2^q - 1)|3$, then all the configurations can be divided into $2^{q-2} \times (2^q - 1)/3$ groups of 3-pairs sharing the same entries. In this case, $2 \times 2^{q-2} \times (2^q - 1)/3$ comparators are saved. Otherwise, all the configurations are divided into $2^{q-2} \times (2^q - 5)/3$ groups of 3-pairs and $2^{q-2} \times 2$ groups of 2-pairs. Accordingly, $2 \times 2^{q-2} \times (2^q - 5)/3 + 2^{q-2} \times 2$ comparators are saved. Table IV lists the numbers of comparators and multiplexers needed to compute the $I$ vector for codes over $GF(8)$ and $GF(16)$. The proposed scheme requires around 20% fewer comparators and this percentage remains about the same for different finite fields. When each LLR is represented by 5 bits, a comparator requires around twice the area of a multiplexer. Accordingly, the proposed design achieves around 13% complexity reduction on $I$ vector calculation.

For an example code over $GF(8)$ with $d_c = 32$, the overall CNU complexity using the proposed architecture with 5-bit LLRs is estimated from architectural level and compared with those of previous designs in Table III. From synthesis results using TSMC $65nm$ library, a 1-bit multiplexer and a register require around the same and three times, respectively, the area of an XOR gate. Also a 5-bit comparator requires the area of around 10 XOR gates. These assumptions are used in our estimation. Although synthesis tool is able to further combine logic gates, the relative complexities of different designs generated by architectural estimations is very close to those from synthesis reports as proved in our previous work [13].

Parallel check node processing can be achieved by processing either all message vectors sent to one check node or one vector sent to each check node simultaneously. The latter has shorter pipelining latency and is assumed in our design. Similarly, each CNU generates one $R_{m,n}$ vector at a time. The architecture for serial $m1$ and $m2$ sorting is available in [12]. The designs in [8], [10] have extra units used to avoid including more than one node from the same column of the trellis in a configuration. To compare with the proposed architecture shown in Fig. 3, the extra units are removed. Besides, the number of units in the architectures of [8]–[10] are adjusted to generate one output vector at a time. From Table III, it can be observed that the computation of the compressed intrinsic vector or its basis and the corresponding recovery of the intrinsic message vector from the compressed format or basis in [8]–[10] require large area and contribute to a significant portion of the CNU complexity when the code is over a lower-order finite field. Compared to prior designs, the proposed architecture achieves at least (1-1927/2365)=18.5% smaller area for the example code. The proposed design also has shorter critical path. Nevertheless, longer critical paths can be shortened by pipelining with one additional clock cycle in the latency.

The proposed design can be also adjusted to avoid including more than one node from the same column of the trellis in each configuration by adding equality testers and multiplexers to the architectures in Fig. 3. The number of comparators needed to compute each intrinsic message is linear to $2^q$ despite the proposed simplifications. Therefore, the proposed design becomes less efficient than the architectures utilizing compressed intrinsic messages or basis when $q$ is larger.

## V. CONCLUSIONS

This paper develops an efficient Min-max CNU architecture for NB-LDPC decoding over lower-order finite fields. Utilizing the properties of finite fields, it is discovered that the pairs of messages to compare share common entries. As a result, the total number of comparators needed for the min-max computation can be reduced. Procedures have also been provided to divide message pairs into groups that share common entries to the maximal extent. For codes over lower-order finite fields, the proposed design leads to significant complexity reduction compared to prior designs that first compute a compressed version of the intrinsic messages and then recover every message from the compressed information.

## REFERENCES


[1] V. Savin,"Min-Max decoding for non binary LDPC codes," *Proc. IEEE Intl. Symp. on Info. Theory*, pp. 960-964, Toronto, Canada, Jul. 2008.

[2] X. Zhang and F. Cai, "Reduced-complexity decoder architecture for non-binary LDPC codes," *IEEE Trans. on VLSI Syst.*, vol. 17, no. 7, pp. 1229-1238, Jul 2011.

[3] X. Chen and C. Wang, "High-throughput efficient non-binary LDPC decoder based on the simplified min-sum algorithm," *IEEE Trans. on Circuits and Syst.-I*, vol. 59, no. 11, pp. 2784-2794, Nov. 2012.

[4] F. Cai and X. Zhang, "Relaxed Min-max decoder architectures for nonbinary low-density parity-check codes," *IEEE Trans. on VLSI Syst.*. vol. 21, no. 11, pp. 1229-1238, Nov. 2013.

[5] E. Li, D. Declercq, and K. Gunnam, "Trellis-based extended min-sum algorithm for non-binary LDPC codes and its hardware structure," *IEEE Trans. on Commun.*, vol. 61, no. 7, pp. 2600-2611, Jul. 2013.

[6] J. O. Lacruz, F. Garcia-Herrero, J. Valls, and D. Declercq "One minimum only trellis decoder for non-binary low-density parity-check codes," *IEEE Trans. on Circuits and Syst.-I*, vol. 62, no. 1, pp. 177-184, Jan. 2015.

[7] J. O. Lacruz, F. Garcia-Herrero, M. J. Canet, and J. Valls, "High-performance NB-LDPC decoder with reduction on message exchange," *IEEE Trans. on VLSI Syst.*, vol. 24, no. 5, pp. 1950-1961, May 2016.

[8] J. O. Lacruz, F. Garcia-Herrero, M. J. Canet, and J. Valls, "Reduced-complexity nonbinary LDPC decoder for high-order Galois fields based on trellis Min-max algorithm," *IEEE Trans. on VLSI Syst.*, vol. 24, no. 8, pp. 2643-2653, Aug. 2016.

[9] H. P. Thi and H. Lee, "Basic-set trellis Min-max decoder architecture for nonbinary LDPC codes with high-order Galois fields," *IEEE Trans. on VLSI Syst.*, vol. 26, no. 3, pp. 496-507, Mar. 2018.

[10] J. Tian, S. Song, J. Lin, and Z. Wang, "Optimized trellis-based Min-max decoder for NB-LDPC codes," *IEEE Trans. on Circuits and Syst.-II*, vol. 67, no. 1, pp. 57-61, Jan. 2020.

[11] X. Zhang, "Low-complexity modified trellis-based Min-max non-binary LDPC decoders," *Journ. of Commun.*, vol. 10, no. 11, pp. 836-842, Nov. 2015.

[12] X. Zhang, *VLSI Architectures for Modern Error-Correcting Codes*, CRC press, 2015.

[13] Z. Xie and X. Zhang, "Fast nested key equation solvers for generalized integrated interleaved decoder," *IEEE Trans. on Circuits and Systems-I*, vol. 68, no. 1, pp. 483-495, Jan. 2021.