An Efficient VLSI Architecture for Nonbinary LDPC Decoders

Jun Lin, Jin Sha, *Member, IEEE*, Zhongfeng Wang, *Senior Member, IEEE*, and Li Li

***Abstract*—Low-density parity-check (LDPC) codes constructed over the Galois field GF(q), which are also called nonbinary LDPC codes, are an extension of binary LDPC codes with significantly better performance. Although various kinds of low-complexity quasi-optimal iterative decoding algorithms have been proposed, the VLSI implementation of nonbinary LDPC decoders has rarely been discussed due to their hardware unfriendly properties. In this brief, an efficient selective computation algorithm, which totally avoids the sorting process, is proposed for Min–Max decoding. In addition, an efficient VLSI architecture for a nonbinary Min–Max decoder is presented. The synthesis results are given to demonstrate the efficiency of the proposed techniques.**

***Index Terms*—Galois field, Min–Max decoding, nonbinary low-density parity-check (LDPC) codes, VLSI.**

I. Introduction

LOW-DENSITY parity-check (LDPC) codes were first introduced by Gallager [1], along with several message-passing decoding algorithms. It has been shown that binary LDPC codes, when decoded using the belief-propagation decoding (BP) algorithm, can approach the capacity of the additive white Gaussian noise channel [2]. However, binary LDPC codes start to show weakness when the code length is small or when a high-order modulation is applied in communication systems. It is shown in [3] that the performance of binary LDPC codes can significantly be enhanced by a direct extension to a higher order Galois field. For this class of LDPC codes, the nonzero entries in the parity-check matrix H are directly replaced by elements in a Galois field. In [4], it is also shown that nonbinary LDPC codes have superior performance in the presence of burst errors. Among various known structured or nonstructured codes, irregular nonbinary LDPC codes were proved to be the best performing LDPC codes [5].

A straightforward implementation of the BP algorithm to decode nonbinary LDPC codes was proposed in [3]. The computational complexity of the BP algorithm is dominated by $O(q^2)$ sum and product operations for each check-node processing, where q is the cardinality of the Galois field. The BP algorithm can also be implemented in the probability domain using m-dimensional two-point fast Fourier transforms (FFT) if the finite field is of characteristic two and order m [6]. Its complexity is dominated by $O(q \log_2 q)$ sum and product operations for each check-node processing. Although this improvement helps, the probability domain algorithm still needs complicated operations, including multiplications. In [4], the authors presented an algorithm that can remove the complicated multiplications in FFT-BP. However, this approach requires lots of exponential and logarithm operations, which are usually implemented with high-complexity lookup tables (LUTs). It would be impractical when the cardinality of the Galois field is large.

BP decoding can also be implemented in the logarithmic domain. A log-BP and its simplified version, which can be named max-log-BP, were presented in [7]. This log-BP decoding algorithm does not require message multiplications. In addition, it does not need normalization and has better numerical characteristics compared with algorithms implemented in the probability domain. The max-log-BP shows a degradation of 0.5 dB over BP decoding for nonbinary LDPC codes based on GF(8) [7]. The computational complexity of these two decoding schemes are both dominated by $O(q^2)$ sum and comparison operations. The authors in [8] proposed an extended min-sum (EMS) decoding algorithm, which is also processed in the log-likelihood ratio (LLR) domain. EMS decoding, whose complexity is dominated by $O(q \log_2 q)$ sum and comparison operations, uses the incoming messages concerning only a part of the Galois field during the check-node processing. Another log-domain decoding algorithm, which has similar computational complexity compared with EMS, was proposed in [9]. The authors showed that their algorithm outperforms EMS decoding in BER performance. Based on the method in [8], Min–Max decoding was proposed in [10]. This decoding algorithm does not need any sum operations during the check-node processing. The computational complexity can be further reduced by using the selecting algorithm proposed in [8].

Due to the inherent high complexity of nonbinary LDPC decoding algorithms, the research on the VLSI implementation of their decoders, which is crucial to real applications, is very limited. A field-programmable gate array implementation based on the decoding algorithm using FFT in the log domain was proposed in [11]. In [12], a kind of VLSI decoder architecture is proposed, which is based on the EMS decoding. In this brief, an efficient architecture for decoders using the Min–Max decoding algorithm is presented. A more hardware-friendly algorithm is developed to perform the selective computation part of Min–Max decoding.

Manuscript received April 25, 2009; revised August 12, 2009. Current version published January 15, 2010. This work was supported in part by the National 863 Program of China under Grant 2008AA01Z135 and in part by the National Nature Science Foundation of China under Grants 60876017 and 90307011. This paper was recommended by Associate Editor M. M. Mansour.

J. Lin, J. Sha, and L. Li are with the Institute of VLSI Design, Jiangsu Provincial Key Laboratory of Advanced Photonic and Electronic Materials, Physics Department, Nanjing University, Nanjing 210093, China (e-mail: njuphylinjun@yahoo.com; shajin@nju.edu.cn; lili@nju.edu.cn).

Z. Wang is with Broadcom Corporation, Irvine, CA 92617 USA (e-mail: zfwang@broadcom.com).

Digital Object Identifier 10.1109/TCSII.2009.2036542

The remainder of this brief is organized as follows. Section II introduces the Min–Max decoding algorithm for nonbinary LDPC codes. Section III discusses about the reformed Min–Max computation algorithm and its VLSI implementation. The overall architecture of the decoder, as well as the check- and variable-node processing units, will be illustrated in Section IV. Finally, the conclusions are drawn in Section V.

II. Min–Max Decoding for Nonbinary LDPC Codes

Let GF(q) $= \{0, 1, \ldots, q-1\}$ be the Galois field with q elements. Let H be the q-ary check matrix with M rows and N columns. The nonzero elements of H are selected from nonzero symbols of GF(q). Let $m(n)$ denotes a check node (variable node) of H. $M(n)$ is the set of neighboring check nodes connected to a variable node n. $N(m)$ is the set of neighboring variable nodes connected to a check node m. Assume a is a symbol in GF(q). Let $L_n(a)$ be the *a priori* information of the variable node n concerning the symbol a, and let $Q_n(a)$ be the *a posteriori* information of the same symbol. $R_{m,n}(a)$ and $Q_{m,n}(a)$ are used to denote the message from check nodes m to variable nodes n and the message from variable nodes n to check nodes m concerning symbols a, respectively. Let c_n be a code symbol of a codeword, and let s_n be the most likely symbol for c_n. The Min–Max decoding algorithm is given as follows:

Min–Max decoding
Initialization:
$L_n(a) = \ln(\Pr(c_n = s_n \mid channel) / \Pr(c_n = a \mid channel))$
$Q_{m,n}(a) = L_n(a)$
Iterations:
check-node processing
$$R_{m,n}(a) = \min_{(a_{n'})_{n' \in N(m)\setminus\{n\}} \in \mathbf{A}_{m,n}(a)} \left(\max_{n' \in N(m)\setminus\{n\}} Q_{m,n'}(a_{n'}) \right)$$
$$\mathbf{A}_{m,n}(a) := \{a_{n'} \mid h_{m,n}a + \sum_{n' \in N(m)\setminus\{n\}} h_{m,n'}a_{n'} = 0\}$$
variable-node processing
$$Q'_{m,n}(a) = L_n(a) + \sum_{m' \in M(n)\setminus\{m\}} R_{m',n}(a)$$
$Q'_{m,n} = \min_{a \in \mathrm{GF}(q)} Q'_{m,n}(a)$
$Q_{m,n}(a) = Q'_{m,n}(a) - Q'_{m,n}$
Tentatively decoding:
$$Q_n(a) = L_n(a) + \sum_{m \in M(n)} R_{m,n}(a)$$
$\tilde{c}_n = \arg\min(Q_n(a))$
$C = [\, \tilde{c}_0 \quad \tilde{c}_1 \quad \cdots \quad \tilde{c}_{N-1} \,]$.

For a better understanding of the preceding algorithm, it is worth mentioning that $\mathbf{A}_{m,n}(a)$ is a set of vectors that consist of Galois field symbols. Each vector has $N(m) - 1$ Galois field symbols, which satisfy the check equation specified in the Min–Max decoding algorithm. If C is checked to be a valid codeword or the maximum iteration number is reached, the decoding process will be terminated. Otherwise, another decoding iteration will be initiated. The check-node processing, which dominates the overall decoding complexity, can be implemented using a *forward–backward* method [10]. It is important to note that all these messages passed between check and variable nodes are always positive.

III. Efficient Implementation of the Min–Max Computation

The computational complexity of Min–Max decoding using the standard implementation is dominated by $O(q^2)$ soft-message comparisons. A selective implementation was proposed in [10] to reduce this complexity by reducing the number of symbols involved in the Min–Max computation, which is given as follows:

$$f(a) = \min_{\substack{h'a'+h''a''=ha \\ a',a'' \in \mathrm{GF}(q)}} \left(\max\left(f'(a'), f''(a'')\right) \right).$$

The most complex computation in this selective implementation is to find the $q+1$ smallest ones among $2q$ values. A sorting method has the complexity of $O(q \log_2 q)$. To avoid the sorting process, the authors in [10] proposed a method that divides the $2q$ values into small subsets. However, this algorithm transformation introduces lots of normalization operations that are computation hungry and, thus, is not suitable for VLSI implementation. In the following, a simplified selecting algorithm is proposed to make the algorithm more hardware friendly.

A. Proposed Selecting Algorithm

Suppose totally K bits are used to represent an LLR message and LLRV(j) is used to represent one of the $2q$ values. Let LLRV(j)[$K-1$] be the most significant bit (MSB) of message LLRV(j). $index$ and $location$ are two data arrays with the length of $2q$. Since the messages passed in the Min–Max decoder are all positive, the proposed selective algorithm will scan from the MSB of $2q$ values involved in the Min–Max computation. The location of the $q+1$ smallest value will be identified by the location of "1" in the $location$ array. The pseudocode describing the selecting algorithm is as follows:

```
selecting algorithm
for j = 2q − 1 to 0 begin
    index(j) = 0;
    location(j) = 0;
endfor
requestNum = q + 1;
tmpV = 0;
for i = K − 1 to 0 begin
    zeroNum = 0;
    for j = 2q − 1 to 0 begin
        if (LLRV(j)[i] == 0 & index(j) == 0)
            zeroNum + +;
    endfor
    if (zeroNum == requestNum) begin
        for j = 2q − 1 to 0 begin
            if (LLRV(j)[i] == 0 & index(j) == 0)
                location(j) = 1;
        endfor
        break and terminate;
    endif
    if (zeroNum > requestNum) begin
        for j = 2q − 1 to 0 begin
            if (LLRV(j)[i] == 1 & index(j) == 0)
```

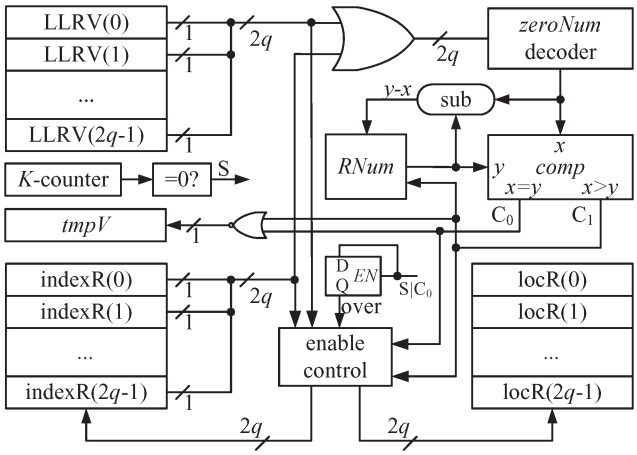

Fig. 1. VLSI implementation of the selecting algorithm.

$$index(j) = 1;$$
endfor
endif
if $(zeroNum < requestNum)$ begin
for $j = 2q - 1$ to 0 begin
if (LLRV$(j)[i] == 0$ & $index(j) == 0)$
begin
$index(j) = 1;$
$location(j) = 1;$
endif
endfor
$requestNum = requestNum - zeroNum;$
$tmpV = tmpV + 2^i;$
endif
endfor.

It will take K cycles at most for the proposed algorithm to find these $q+1$ smallest values among $2q$ values if there do exist such $q+1$ values. When the decoding process is converged (during the last decoding iterations), most of the $2q$ values are saturated. Under this condition, the number of finally selected values will be smaller than $q+1$, which can significantly reduce the decoding complexity. To accommodate this specific hardware situation, all the message values in LLRc, which are illustrated in Fig. 4, should be initialized to $tmpV$ to ensure correct Min–Max computation.

B. Implementation of the Proposed Selecting Algorithm

The VLSI implementation of the proposed selecting algorithm is shown in Fig. 1, where each message is stored in a K-bit register. The *comp* module has two outputs, which will be high when the corresponding condition listed in Fig. 1 is satisfied. $RNum$ is used to store the value of $requestNum$ specified in the proposed algorithm. The initial value of $RNum$ is $q+1$ and will be overwritten by a new value if C_1 is low. A K-bit register will be used to store the $tmpV$ value. The input will be shifted into the register from the least significant bit. A down counter, which is initialized to K, is used to facilitate the Min–Max computing.

The $zeroNum$ decoder computes the number of zero bits among the incoming $2q$ inputs. The design of the $zeroNum$ decoder could be complicated when the number of inputs is large. Suppose the number of inputs is I, there are totally 2^I different kinds of input combinations and $I+1$ possible output values (from 0 to I). Therefore, this decoder can be implemented with combinational logic using various kinds of logic simplification methods. This method is effective when the number of inputs is small. For big I, it will be more efficient to divide a big I into smaller groups, as shown in Fig. 2, where $I = I_0 + I_1 + \cdots + I_{p-1}$. These sub-$zeroNum$ decoders are constructed using the logic simplification method. For example, when I is equal to 32, these inputs can be divided into five groups with $I_j = 7$ $(0 \leqq j \leqq 3)$ and $I_4 = 4$. The gate count of a $zeroNum$ decoder with 32 inputs is about 240.

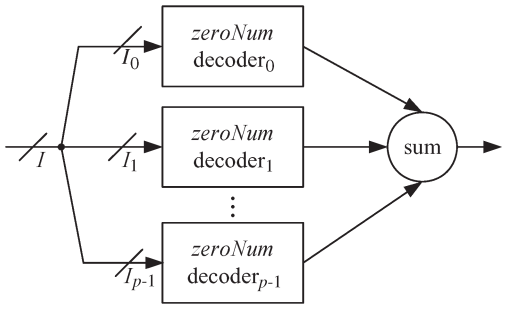

Fig. 2. Architecture of the proposed $zeroNum$ decoder.

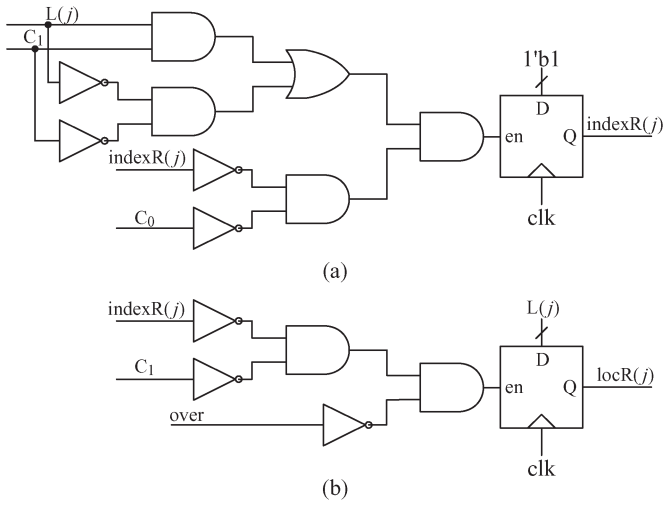

Fig. 3. (a) Control logic for indexR(j). (b) Control logic for locR(j).

Two single-bit register arrays (indexR and locR) are used in the proposed implementation of the selecting algorithm. The control logic for indexR(j) and locR(j) is shown in Fig. 3. L(j) is equal to LLRV$(j)[K-1-i]$ at the ith cycle after initialization. Synthesis results show that, with SMIC 0.18-μm CMOS process, the total gate count of the proposed implementation of the selecting computation is 1.55k for $q = 16$ under a 200-MHz clock speed. The complexity of the proposed selecting algorithm is only related to q and the bit width of the message.

C. VLSI Architecture for the Min–Max Computation

The VLSI architecture of the Min–Max computation [10] unit (MMCU) is shown in Fig. 4. The address generator module is designed to generate message pairs participating in the

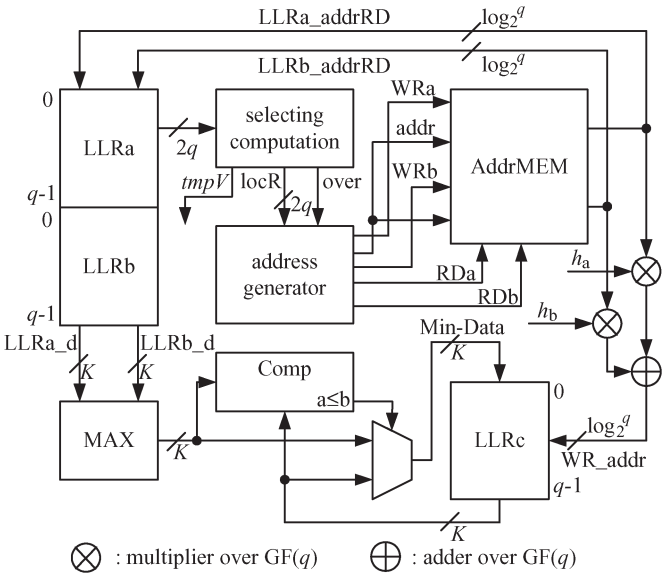

Fig. 4. VLSI architecture of the MMCU.

Min–Max computation from LLR values that have just been located. AddrMEM is a dual-in and dual-out memory used to store the addresses of selected LLR values. The length of this memory is $q+1$. These addresses are Galois field symbols related to corresponding LLR messages. At the positive edge of the "over" signal, each value of LLRc will be initialized to $tmpV$ if C_0 is high. Otherwise, all of them will be initialized to K-bits 1.

The operation of the address generator can be divided into two processes, namely, *address mapping* and *address output*. The address-mapping algorithm is shown as follows:

address mapping algorithm
addr $= 0$; $C_a = 0$; $C_b = 0$; WRa $= 0$; WRb $= q$;
for $i = 0$ to $q-1$ begin
  if (locR$(i) == 1$) begin
    AddrMEM(WRa) $=$ addr;
    $C_a = C_a + 1$; WRa $++$;
  endif
  if (locR$(q+i) == 1$) begin
    AddrMEM(WRb) $=$ addr;
    $C_b = C_b + 1$; WRb $--$;
  endif
  addr $++$;
endfor.

It is easy to see that the *mapping* process will take q cycles in total. C_a and C_b are used to record the number of selected LLR values in LLRa and LLRb, respectively. If either the first q bits or the last q bits of locR are all zero when the selecting algorithm is finished, the Min–Max computation process can be terminated at once. The *address output* process will be initiated once the *mapping* process is done. A pair of LLR value addresses will appear at the output ports of AddrMEM at each cycle. These addresses will be used to select proper LLR values, which will engage in the Min–Max computation, from LLRa and LLRb. LLRc is used to store the temporary Min–Max

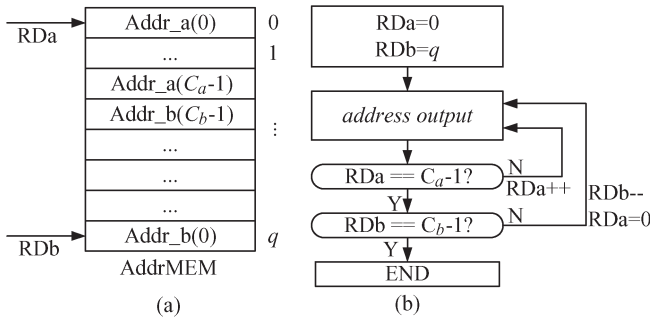

Fig. 5. (a) Storage pattern of address memory units. (b) Read control of address memory units.

computation results. One of the q LLR values from LLRc will be selected to join the Min–Max computing using the write address (WR_addr), which is generated using arithmetic over the Galois field, as shown in Fig. 4. Totally, $C_a \times C_b$ cycles are needed for the *address output* process. These LLR addresses are stored in AddrMEM with the pattern shown in Fig. 5(a). A simple finite-state machine is employed to control the value of RDa and RDb, which are shown in Fig. 5(b).

The total cycles needed to finish a Min–Max computation are $N_1 + q + C_a \times C_b$, where N_1 is not greater than K. Synthesis results show that, with SMIC 0.18-μm CMOS process, the total gate count of the proposed MMCU is 5.5k at 200 MHz for $q = 16$ and $K = 5$. The LLR message registers occupy about 50% of all the hardware resource.

IV. Min–Max Decoder Architecture

A. CNU and VNU Architectures

The VLSI architecture of a check-node unit (CNU) used in the Min–Max nonbinary LDPC decoder is just the same as the proposed MMCU, except that LLRc is stored in message memory such as on-chip synchronous dynamic random access memory. The CUN will read and write the message memory when performing the Min–Max computation. The variable-node unit (VNU) architecture is an extension of that used in binary LDPC decoders and can directly be implemented according to the Min–Max decoding algorithm.

B. Overall Architecture of the Min–Max LDPC Decoder

Due to the high complexity of the Min–Max decoding algorithm, it is unlikely that the decoder can be implemented in a high level of parallelism. A partially parallel architecture, which is shown in Fig. 6, is desirable for nonbinary Min–Max LDPC decoder design.

There are p $(p < M)$ CNUs and m $(m < N)$ VNUs in the proposed decoder architecture. The memory part of the proposed decoder mainly consists three parts, namely, check-to-variable message memory (CVM), variable-to-check message memory (VCM), and Galois field symbol memory, which is used to record all the nonzero elements of a nonbinary parity-check matrix. The check-node processing will be performed in a *forward–backward* fashion [10]. During the forward step, which includes $d_c - 2$ Min–Max computations, $d_c - 3$

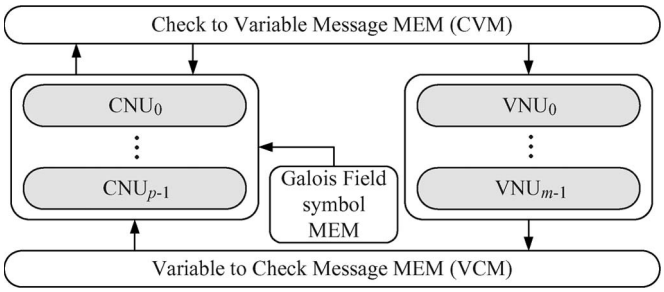

Fig. 6. Overall architecture of the Min–Max nonbinary LDPC decoder.

TABLE I
COMPARISONS BETWEEN DIFFERENT DECODING ALGORITHMS

	[11]	[12]	proposed
CNU Main Hardware Consumption	$q(\log_2 q+1)$ K-bits full adder; q e(x)-LUTs*; qK bits register;	$(3n_mK+5n_m\log_2 q)$ bits register; n_m K-bits 2-to-1 MUX; $n_m \log_2 q$-bits 2-to-1 MUX; n_m K-bits comparators; 2 K-bits full adder	$(3qK+q\log_2 q+4q)$ bits register; $29q$ NAND gates; 3 K-bits comparators; 1 K-bits 2-to-1 MUX; 1 K-bits full adder
Estimated Gate count (q=64 n_m=32 K=5)	26000	10600	10000
CNU Elementary Operations	$q(\log_2 q+1)$ addition;	$2n_m^2$ comparisons; $2n_m$ additions;	Worst case: $q^2/2$ comparisons; Best case: 0 comparisons
Message Memory (bit)	$2Nd_vqK$	$2n_m\times Nd_v(K+\log_2 q)$	$2Nd_vqK$

*The LUT is a ROM with $K2^K$ memory cell

temporary Min–Max computation messages and one updated check-to-variable message will be generated and stored in CVM. During the backward step, which includes $2(d_c-2)$ Min–Max computations, the remaining d_c-1 check-to-variable messages will be computed using messages from both CVM and VCM. The VNUs will compute updated variable-to-check messages once all the check nodes have been processed.

C. Comparison With Other Decoding Algorithms

It would be meaningful to compare the implementation of the Min–Max decoding algorithm with other approaches in [11] and [12], which are based on FFT-BP and EMS algorithms, respectively. The CNU complexity, which is dominated by the implementation of the elementary computation such as the Min–Max computing, is the bottleneck of the nonbinary LDPC decoder. Thus, in Table I, we list the main hardware component of the CNU for different algorithms, as well as the actual operations performed by hardware. The table is based on GF(q) and K-bit quantization assumption.

It is worth mentioning that the total gate count of the $zeroNum$ decoder and the control logic for indexR and locR is estimated to be $29q$. For the GF(64)-LDPC code used in [12], the performance of the EMS algorithm is very close to that of BP when n_m is equal to 32. The estimated gate count of these elementary implementations is also listed in Table I. It can be found that the FFT-BP algorithm, which usually needs more quantization bits than EMS and Min–Max, become hardware demanding when q is large. The implementation of the VNU based on FFT-BP also needs at least q LUTs for computing $\log(x)$-function, which will significantly cause hardware overhead. When near-BP performance is required ($n_m=32$ for $q=64$), the CNU elementary part of EMS and Min–Max approaches have similar hardware complexity. However, the proposed design needs fewer comparisons than that in [12], particularly when not so many iterations are needed for decoding a codeword. For decoders based on the EMS algorithm, messages received from the channel should be presorted before being fed into the CNU. This will increase the overall implementation complexity of the EMS algorithm. The proposed implementation achieves a good tradeoff between decoding performance, hardware cost, and power consumption.

V. CONCLUSION

This brief has presented an efficient algorithm that can be used in the implementation of the Min–Max nonbinary decoding algorithm. An original VLSI implementation of the MMCU has been proposed. Under SMIC 0.18-μm CMOS process, the total gate count of the proposed MMCU is 5.5k at 200 MHz with $q=16$ using five-bit quantization. The check- and variable-node processor unit architectures, as well as the overall decoder architecture, have been discussed in detail. To the best of our knowledge, this is the first VLSI implementation of the MMCU.

REFERENCES

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
[2] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.
[3] M. Davey and D. J. C. Mackay, "Low-density parity check codes over GF(q)," *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165–167, Jun. 1998.
[4] H. Song and J. R. Cruz, "Reduced-complexity decoding of Q-ary LDPC codes for magnetic recoding," *IEEE Trans. Magn.*, vol. 39, no. 2, pp. 1081–1087, Mar. 2003.
[5] D. J. C. Mackay, S. T. Wilsion, and M. Davey, "Comparison of construction of irregular Gallager codes," *IEEE Trans. Commun.*, vol. 47, no. 10, pp. 1449–1454, Oct. 1999.
[6] L. Barnault and D. Declercq, "Fast decoding algorithm for LDPC over GF(2^q)," in *Proc. IEEE Inf. Theory Workshop*, 2003, pp. 70–73.
[7] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of LDPC codes over GF(q)," in *Proc. IEEE Int. Conf. Commun.*, Paris, France, Jun. 2004, pp. 772–776.
[8] 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.
[9] C.-H. Liao, C.-Y. Wang, C.-H. Liu, and T.-D. Chiueh, "An $O(q\log q)$ log-domain decoder for non-binary LDPC over GF(q)," in *Proc. IEEE APCCAS*, Macao, China, Nov. 2008, pp. 1644–1647.
[10] V. Savin, "Min–Max decoding for non binary LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, ON, Canada, Jul. 2008, pp. 960–964.
[11] C. Spagnol, E. Popovici, and W. Marnane, "FPGA implementations of LDPC over GF(2^m) decoders," in *Proc. IEEE SiPS*, 2007, pp. 273–278.
[12] 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 ISCIT*, Sydney, Australia, Oct. 2007, pp. 1201–1206.