Efficient Implementation of a Threshold Modified Min-Sum Algorithm for LDPC Decoders

Yanfang Liu, Wei Tang, *Member, IEEE*, and David G. M. Mitchell, *Senior Member, IEEE*

***Abstract*—In this brief, we present a hardware efficient implementation of a threshold modified min-sum algorithm (MSA) to improve the performance of a low density parity-check (LDPC) decoder. The proposed architecture introduces a novel lookup table based threshold attenuation technique, called threshold attenuated MSA (TAMSA). The proposed TAMSA implementation is shown to improve bit error rate (BER) performance compared to the conventional AMSA and MSA. Furthermore, a layered version of the TAMSA implementation is investigated to reduce hardware cost. Utilizing circuit optimization techniques, including a parallel computing structure, the proposed layered TAMSA field-programmable gate array (FPGA) implementation results show that the modified architecture requires no extra circuit power or circuit area compared to conventional AMSA, and only 0.07% extra leaf cells compared to conventional MSA.**

***Index Terms*—LDPC codes, min-sum algorithm, LDPC decoder, attenuation, FPGA.**

I. Introduction

As a class of linear block codes, low-density parity-check (LDPC) codes were originally proposed by Gallager in the 1960's [1], but were not considered practical for a long time due to prohibitive hardware requirements [2]. MacKay rediscovered LDPC codes in the 1990's [3] and showed that LDPC codes are capable of approaching channel capacity with low-complexity iterative message passing (MP) decoding. Since then, significant effort has been made to develop hardware efficient decoders, e.g., [4], [5], [6], and LDPC codes have been included in many communication standards, such as IEEE 802.6, IEEE 802.20, IEEE 802.3, DVB-RS2, and CMMB. The min-sum algorithm (MSA) is widely used for hardware implementation of LDPC decoders since no estimation of the channel signal-to-noise ratio (SNR) is needed over the additive white Gaussian noise (AWGN) channel, as well as its low complexity and robustness against quantization error [7], [8], [9]. However, the MSA incurs a degradation in performance when compared to the (more complex) sum product algorithm (SPA) due to approximations involved in the message computation [2]. Empirically, MSA is observed to have little degradation in performance compared to the SPA for short code lengths, but for long code lengths the degradation can vary from several tenths of a decibel (dB) to one dB [10].

This material is based upon work supported by the National Science Foundation under Grant Nos. ECCS-1710920, OIA-1757207, ECCS-2015573, and ECCS-1652944.

Yanfang Liu, Wei Tang, and David G. M. Mitchell are with the Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM, USA. (Email: {viviliu,wtang,dgmm}@nmsu.edu)

Correspondence should be addressed to: Yanfang Liu, 1125 Frenger Mall, Las Cruces, New Mexico 88003 USA. Email: viviliu@nmsu.edu

To improve the decoding performance, two modifications of MSA, called attenuated MSA (AMSA) and offset MSA (OMSA) were proposed in [11], [12] to reduce the approximation error. Both variants have been shown to achieve better bit error rate (BER) performance at low to moderate SNRs when compared to the conventional MSA. Moreover, to save hardware resources, a layered version of MSA, AMSA, and OMSA were employed and shown to have faster convergence speed, reducing iterations and decoder power consumption, as well as circuit area [13], [14], [15], [16].

To further improve the performance of quantized LDPC decoders, threshold AMSA (TAMSA) and threshold OMSA (TOMSA) were proposed in [17]. The TAMSA (respectively, TOMSA) selectively attenuates (offsets) the outgoing log-likelihood ratio (LLR) message used to update a variable node during MP decoding of an LDPC code if this value has a magnitude below some threshold τ, while allowing an LLR to reach the maximum quantizer level if the magnitude is greater than τ. Given that most of the decoding failures in the high SNR regime occur due to problematic graphical objects that are randomly distributed in the Tanner graph of LDPC codes, the authors of [17] showed that the new algorithms are less prone to decoding failures and can significantly improve the performance compared to AMSA and OMSA.

In this paper, we present a novel implementation of the TAMSA algorithm using look-up tables (LUTs) for message quantization and attenuation, and also investigate a layered TAMSA algorithm to reduce hardware cost. To demonstrate our approach, we implement the $(155, 64)$ Tanner code [18], which is attractive for hardware implementation due to its quasi-cyclic (QC) structure. Simulation results of MSA, AMSA, TAMSA, and layered TAMSA show that the layered TAMSA decoder gains approximately 0.4dB at a bit error rate (BER) equal to 10^{-9} over the MSA and AMSA, with a 0.1dB performance gain compared to TAMSA. We then consider a hardware implementation of these algorithms where, according to the QC structure, we consider a full-parallel architecture to speed up the decoding process. It is shown that, as a result of the LUT-based approach, the performance gain achieved by the proposed layered TAMSA implementation is achieved with no extra hardware cost when compared to AMSA by comparing the LUT, leaf cell, power, and area values from the synthesis results, and only 0.07% extra leaf cells compared to conventional MSA.

II. Min-Sum Algorithms

In the following, we briefly describe the MSA and its modifications. MP decoding [2] of LDPC codes operates by

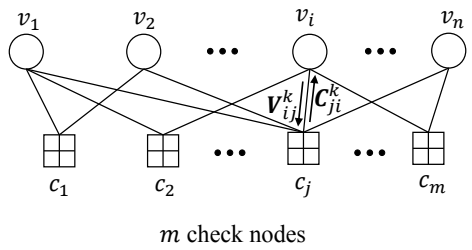

Fig. 1. Tanner graph of an LDPC code.

iteratively exchanging messages in the Tanner graph of an LDPC code between variable nodes (white circles) and check nodes (plus boxes), see Fig. 1. At the k^{th} iteration, let $\mathbf{V}_{ij}^k$ denote the LLR value passed from variable node v_i to check node c_j and let $\mathbf{C}_{ji}^k$ denote the LLR value passed from check node c_j to variable node v_i. The set of check nodes in the graph connected to v_i are represented by $N(v_i)$ and the set of variable nodes connected to c_j are represented by $N(c_j)$. Assume that codeword $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ is transmitted on an AWGN channel under binary phase shift keyed (BPSK) modulation, where each zero is mapped to $+1$ and each one is mapped to -1. Let p_0 represent the probability that a 0 is received from the channel and let p_1 represent the probability that a 1 is received from the channel. Let $r_i = \ln(\frac{p_0}{p_1})$ denote the LLR values received from channel for bit i [2]. The MSA algorithm is initialized in iteration 0 by passing the received value r_i from each variable node v_i to the check nodes in $N(v_i)$ as

$$\mathbf{V}_{ij}^0 = r_i. \tag{1}$$

Following initialization, the outgoing message $\mathbf{C}_{ji}^k$ from check node c_j to variable node v_i at iteration k is given by

$$\mathbf{C}_{ji}^k = \left(\prod_{i' \in N(c_j)\backslash i} \mathbf{sign}(\mathbf{V}_{i'j}^k) \right) \min_{i' \in N(c_j)\backslash i} |\mathbf{V}_{i'j}^k|, \tag{2}$$

where $N(c_j)\backslash i$ denotes the set of all variable nodes connected to check node j except v_i. For iteration $k > 0$, the outgoing message $\mathbf{V}_{ij}^k$ from variable node v_i to check node c_j is given by

$$\mathbf{V}_{ij}^k = r_i + \sum_{j' \in N(v_i)\backslash j} \mathbf{C}_{j'i}^k, \tag{3}$$

where $N(v_i)\backslash j$ denotes the set of all check nodes connected to variable node i except c_j. After all check nodes and all variable nodes are updated, the hard decision estimate is computed

$$\hat{u}_i^k = \begin{cases} 0, & r_i + \sum_{j' \in N(v_i)} \mathbf{C}_{j'i}^k > 0, \\ 1, & r_i + \sum_{j' \in N(v_i)} \mathbf{C}_{j'i}^k < 0. \end{cases} \tag{4}$$

If the hard decision $\hat{\mathbf{u}}$ is a codeword, decoding stops, otherwise the decoder starts the next iteration until some pre-specified amount of decoder iterations I_{max} are reached.

To reduce the BER performance loss of MSA when compared to SPA, the attenuated MSA (AMSA) was proposed in [11]. AMSA operates as MSA, but where (2) is replaced by

$$\mathbf{C}_{ji}^k = \alpha \left(\prod_{i' \in N(c_j)\backslash i} \mathbf{sign}(\mathbf{V}_{i'j}^k) \right) \min_{i' \in N(c_j)\backslash i} |\mathbf{V}_{i'j}^k|, \tag{5}$$

where $\alpha > 0$ is a constant. AMSA reduces the negative effect of overestimating the LLR magnitudes in MSA and improves performance in the low SNR region; however, neither of them necessarily achieves good performance in the high SNR region. Threshold AMSA (TAMSA) was proposed in [17] to improve performance in high SNR region compared to AMSA and MSA. The new algorithm is based on the assumption that small problematic graphical objects, called trapping sets, are the major cause of the performance loss in high SNR. TAMSA operates as MSA, but where (2) is replaced by

$$\mathbf{C}_{ji}^k = \begin{cases} \left(\prod_{i' \in N(c_j)\backslash i} \mathbf{sign}(\mathbf{V}_{i'j}^k) \right) \min_{i' \in N(c_j)\backslash i} |\mathbf{V}_{i'j}^k|, \\ \qquad \text{if } \min_{i' \in N(c_j)\backslash i} |\mathbf{V}_{i'j}^k| \geq \tau, \\ \alpha \left(\prod_{i' \in N(c_j)\backslash i} \mathbf{sign}(\mathbf{V}_{i'j}^k) \right) \min_{i' \in N(c_j)\backslash i} |\mathbf{V}_{i'j}^k|, \\ \qquad \text{otherwise.} \end{cases} \tag{6}$$

TAMSA *locally* reduces the magnitudes of the check node LLRs by adding a simple threshold test compared to AMSA (5), which improves the performance with a negligible complexity increase.

In this paper, we also consider a layered version of TAMSA, with modified update rules. The algorithm is initialized by (1), then the outgoing message $\mathbf{V}_{ij}^k$ at iteration $k > 0$ is replaced by

$$\mathbf{V}_{ij}^k = \mathbf{V}_i^{k-1} - \mathbf{C}_{ji}^{k-1}, \tag{7}$$

where $\mathbf{C}_{ji}^0 = 0$, and the outgoing message $\mathbf{C}_{ji}^k$ for some subset of check nodes is computed following (6). The choice of subsets will very depending on the code and desired parallelization (See Section IV-A). Message $\mathbf{V}_{ij}^k$ is updated again for the variable nodes connected to the selected subset of check nodes as

$$\mathbf{V}_{ij}^k = \mathbf{V}_{ij}^k + \mathbf{C}_{ji}^k. \tag{8}$$

The decoder repeats (6) and (8) until all check nodes and variable nodes are updated. Finally, the hard decision estimate is replaced by

$$\hat{u}_i^k = \begin{cases} 0, & \mathbf{V}_i^k > 0, \\ 1, & \mathbf{V}_i^k < 0. \end{cases} \tag{9}$$

If the hard decision $\hat{\mathbf{u}}$ is a codeword, decoding stops, otherwise the decoder starts the next iteration from (7) until some pre-specified amount of decoder iterations I_{max} are reached.

III. Finite Precision Representation of LLRs

Practical hardware implementation of LDPC decoders requires a finite precision representation of LLRs. The effects of clipping and quantization on the MSA were investigated in [12]. Moreover, a quantized density evolution (DE) algorithm was used to find the optimal attenuation parameter α from (5) for quantized AMSA in [11]. In the remainder of this section, we describe our method of representing LLRs with finite precision in the hardware implementation of these algorithms and present simulation results comparing MSA, AMSA, and TAMSA (both regular and layered), based on this strategy.

In [17], the authors used a 5-bit quantizer for LLR values. In this paper, we propose a 4-bit LUT to map the magnitude of LLRs (and the LLRs after attenuation), and one extra bit for the sign of the LLRs. Table I shows the LUT used to convert received floating-point LLRs to quantized LLRs, where the LLRs are represented as a range. This mapping is done once in order to quantize r_i as a 4-bit string with 1 bit sign. After this, all operations in (2)-(9) are performed with $(4+1)$-bit strings. Attenuation (multiplication by α in (5) or (6)) is not computed in real-time, rather it is computed in advance for each range of LLRs, for a given α, with a resulting LUT for the new mapping. The LUT for attenuation of the mapping in Table I with $\alpha = 0.8$ is shown in Table II. Threshold attenuation can be achieved by modifying Table II. For example, for $\tau = 1.425$ and $\alpha = 0.8$, quantized LLRs smaller than 1010 will be attenuated according to (6). In this case, the TAMSA LUT will be the same as Table II for LLRs 0000 to 1001, but LLRs 1010 to 1111 will not be attenuated.[1]

TABLE I
FLOATING-POINT LLRS TO 4-BIT STRINGS

Received LLR	Map	Received LLR	Map
$[0, 0.075)$	0000	$[1.125, 1.275)$	1000
$[0.075, 0.225)$	0001	$[1.275, 1.425)$	1001
$[0.225, 0.375)$	0010	$[1.425, 1.575)$	1010
$[0.375, 0.525)$	0011	$[1.575, 1.725)$	1011
$[0.525, 0.675)$	0100	$[1.725, 1.875)$	1100
$[0.675, 0.825)$	0101	$[1.875, 2.025)$	1101
$[0.825, 0.975)$	0110	$[2.025, 2.175)$	1110
$[0.975, 1.125)$	0111	$[2.175, \infty)$	1111

TABLE II
ATTENUATED 4-BIT STRINGS FOR $\alpha = 0.8$

LLR	Attenuated LLR	LLR	Attenuated LLR
0000	0000	1000	0110
0001	0001	1001	0111
0010	0010	1010	1000
0011	0010	1011	1001
0100	0011	1100	1010
0101	0100	1101	1010
0110	0101	1110	1011
0111	0110	1111	1100

Fig. 2 shows simulation results for quantized MSA, AMSA, TAMSA, and layered TAMSA with an attenuation factor $\alpha = 0.8$ for all attenuated algorithms and a threshold $\tau = 1.425$ for threshold algorithms (using the LUTs as described above). All algorithms were allowed a maximum of 100 iterations. We observe that both TAMSA and layered TAMSA result in significantly improved performance over the AMSA and MSA; with the best performance resulting from layered TAMSA, which offers close to 0.4dB gain at a BER equal to 10^{-9} over AMSA and MSA. AMSA and TAMSA (the two dashed curves) have exactly the same performance for 1dB-4dB. TAMSA, by design, will attenuate almost all of the time at low E_b/N_0 (for an appropriately chosen threshold τ) since there is a high probability of at least one weak incoming LLR to a check node, resulting in approximately equal BER performance to AMSA. On the other hand, at high E_b/N_0, TAMSA selectively attenuates based upon message reliability and yields significantly better BER performance than both AMSA and MSA.

[1]Note that the TAMSA algorithm makes use of quantizer mapping 1111, unlike AMSA, which helps decoder convergence.

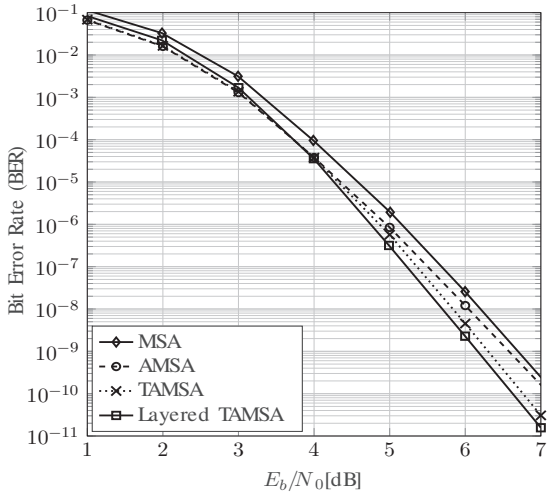

Fig. 2. Simulation results comparing the decoding performance of quantized MSA, AMSA, TAMSA, and layered TAMSA for a $(155, 64)$ QC Tanner LDPC code.

Another important metric related to decoder power consumption is the average number of iterations (ANI) performed for each algorithm. The results are summarized in Table III. We observe that both AMSA and TAMSA provide a significant reduction in the average number of iterations when compared to MSA at low SNR, with similar numbers elsewhere.

TABLE III
AVERAGE NUMBER OF ITERATIONS RECORDED FOR THE $(155, 64)$ QC TANNER CODE

E_b/N_0	MSA [17]	AMSA [17]	TAMSA [17]
1dB	68.95	59.28	59.24
2dB	30.4	23.13	22.9
3dB	7.82	6.28	6.2
4dB	3.06	2.95	2.87
5dB	1.97	1.98	1.98
6dB	1.44	1.46	1.46
7dB	1.09	1.10	1.10
8dB	0.85	0.86	0.86

IV. SYSTEM DESIGN CONSIDERATIONS

In this work, we implement layered TAMSA with a $(155, 64)$ QC Tanner code as an example. In this section, we introduce this particular LDPC code and strategies to implement the corresponding decoder in hardware.

A. LDPC codes

The parity-check matrix of the $(155, 64)$ QC Tanner code is given by

$$\mathbf{H} = \begin{bmatrix} I_1 & I_2 & I_4 & I_8 & I_{16} \\ I_5 & I_{10} & I_{20} & I_9 & I_{18} \\ I_{25} & I_{19} & I_7 & I_{14} & I_{28} \end{bmatrix}, \quad (10)$$

where I_x is a 31×31 identity matrix with rows shifted cyclically to the left by x positions. According to this specific QC structure, we use a full-parallel architecture to implement layered MSA, layered AMSA, and layered TAMSA to speed

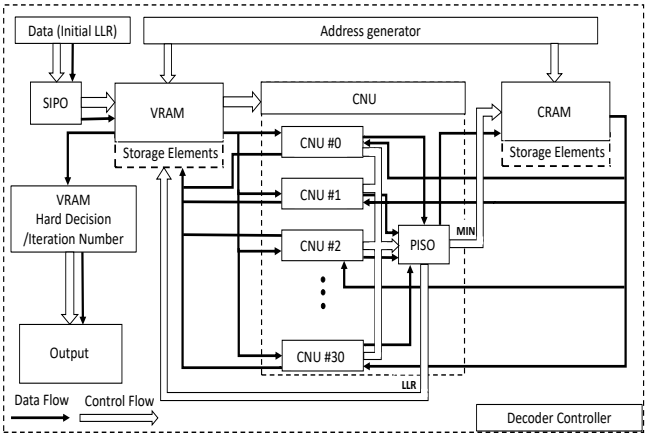

Fig. 3. System Diagram.

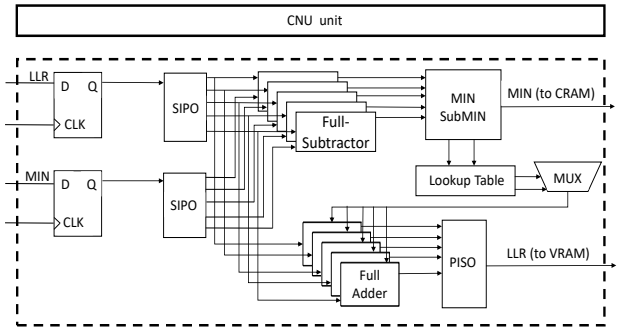

Fig. 4. CNU Architecture.

up the decoding process. Specifically, we use 31 check node unit (CNU) modules in the LDPC decoder. At each iteration, message $\mathbf{V}_{ij}^k$ is computed by (7). We then compute $\mathbf{C}_{ji}^k$ using (2), (5), or (6), where appropriate for the first 31 rows in parallel ($j = 1, 2, \ldots, 31$), then update all connected variable node LLRs using $\mathbf{V}_{ij}^k$ using (8). This is repeated for the next 31 rows ($j = 32, 33, \ldots, 62$), and then the final 31 rows ($j = 63, 64, \ldots, 93$) in the parity-check matrix (10). After these three batches of parallel computation, one iteration is completed, the iteration number increases by 1, and the sign of the LLRs is calculated for the hard decision according to (9). The decoder stops either if the hard decisions give a valid codeword or the iteration number achieves a preset maximum iteration number I_{max}.

B. System Design

The *Decoder* system consists of several important building blocks, as shown in Fig. 3, where the black arrows represent data flow and the white arrows represent control flow. The input serial data $\mathbf{V}_{ij}^0$ is first converted into parallel data by the *SIPO* (serial-in parallel-out). The data is then stored in the *VRAM*. The *VRAM* also stores the temporary variable node LLRs $\mathbf{V}_{ij}^k$ during the decoding process. The *Decoder Controller* controls the decoding process, and the values of check nodes and variable nodes are updated according to the status of the *Decoder Controller* and the equations (5)-(8). First, according to the parity check matrix of the $(155, 64)$ QC Tanner code in (10), the *Decoder Controller* asks the *Address Generator* to generate several addresses to access the data $\mathbf{V}_{ij}^0$ stored in the *VRAM*. Then this data is sent to the *CNU*, where the minimum values and sub-minimum (second minimum) values are calculated according to (2), (5), or (6), the *Decoder Controller* then asks *Address Generator* to generate addresses to store the minimum and sub-minimum values to *CRAM*. Meanwhile, $\mathbf{V}_{ij}^k$ is computed according to (8) and stored back into the *VRAM*. After a decoding iteration is complete, the *Decoder Controller* asks the *VRAM Hard Decision* to make a hard decision according to the the sign of LLRs and decide whether it is a valid codeword (which means the decoding is successful). If it is successful, then the final data is sent to the output of the *Decoder*. If not, the *Decoder Controller* compares the number of current iterations with a predetermined maximum iteration number I_{max}. If the number of iterations is smaller than I_{max}, the decoder starts the next iteration of decoding, computing $\mathbf{V}_{ij}^k$ using (7) and updating the *VRAM*, otherwise, the *Decoder Controller* finishes the decoding process and outputs the result from the *VRAM*.

To implement the *CNU*, we consider the full-parallel structure described in Section III. For the *CNU* unit, the LLRs and minimal values from previous iteration are sent to the unit serially, where two *SIPO* units are applied. There are 5 Full-Subtractor modules used to implement (7), and 5 Full-Adder modules used to implement (8). The sign and the magnitude values to be sent to each variable node are calculated separately. First, the signs of all variable nodes connected to this check node are multiplied together to form $\prod_{i \in N(c_j)} \mathbf{sign}(\mathbf{V}_{ij})$. The sign of the outgoing message to each variable node is computed by multiplying $\prod_{i \in N(c_j)} \mathbf{sign}(\mathbf{V}_{ij})$ with the sign of the corresponding variable node. Second, the minimum value and the sub-minimum value among the incoming messages from $v_i \in N(c_j)$ are determined. In [19], the authors proposed bit-serial circuits to determine the minimum and sub-minimum values. Here, we transform them to a parallel design to speed up the *CNU* module. The architecture used to determine the minimum is shown in Fig. 5. There are five 4-bit inputs corresponding to the five incoming quantized LLRs (Data i bit 1, Data i bit 2, ..., Data i bit 4, for $i = 0, 1, \ldots, 4$) and one 4-bit output (min bit 1, min bit 2, ..., min bit 4). The circuits to determine the sub-minimum value is similar, except that it has four 4-bit data inputs because the previously found minimum value is not used. Let $\mathbf{M}_{1,j}$ represent the minimum value and let $\mathbf{M}_{2,j}$ represent the sub-minimum value input to c_j. Finally, we compare each value of the variable node with $\mathbf{M}_{1,j}$ to determine the minimum value: if the variable node message v_i^k equals $\mathbf{M}_{1,j}$, we assign $\mathbf{M}_{2,j}$ as the minimum value in (2), (5) and (6), otherwise, $\mathbf{M}_{1,j}$ is used for minimum value. Using this method, we avoid multiple calculations to update each check node. Layered AMSA and TAMSA require an additional LUT for attenuation; however,

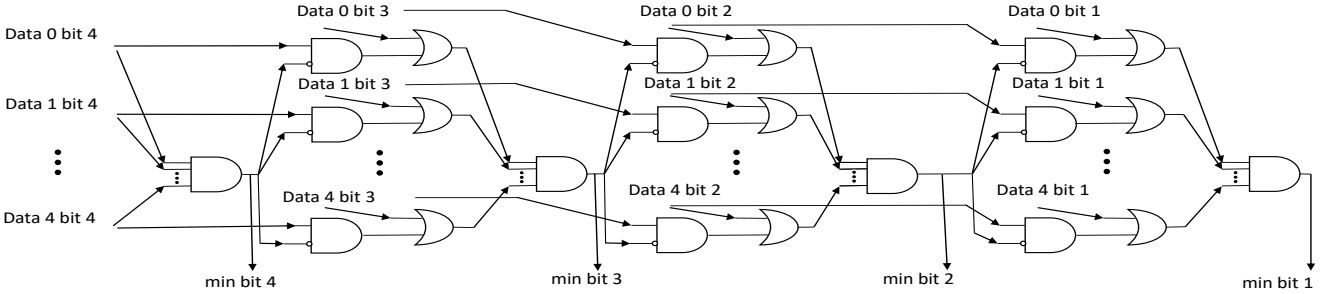

Fig. 5. Circuit architecture for minimum computation.

the hardware costs are same for each case. Note that, although we have focused our discussion on the implementation on the $(155, 64)$ QC Tanner code, the above architecture suitably generalizes for other QC LDPC codes.

Regarding low-power hardware design considerations, we do not consider a pipeline structure for our particular design given that we use a full-parallel architecture and since the layered decoder requires a complete computation of the first 31 rows in the parity-check matrix to continue the computation of the following 31 rows. However, for different LDPC codes, a pipeline structure is a popular choice to speed up the decoding process. The comparison of hardware resources used in layered MSA, layered AMSA, and layered TAMSA are summarized in Table IV. The FPGA device used is a Xilinx Kintex 7 FPGA and the Vivado design suite software was used for the design and simulation environment. The power consumption and area are determined using a Cadence SOC Encounter tool based on a 0.18um CMOS process. The highest frequency that the design can achieve is 700 MHz. The power and area of layered MSA, AMSA, and TAMSA are the same, when the clock is 500 MHz (CLK1) and 100 MHz (CLK2), respectively. The data comparison shows that layered TAMSA requires no extra hardware resources compared to layered AMSA, and both attenuated algorithms require only 0.07% extra leaf cells compared to conventional layered MSA.

TABLE IV
COMPARISON OF HARDWARE RESOURCES

	Layered MSA	Layered AMSA	Layered TAMSA
LUT	14.9k	14.9k	14.9k
FF	10.4k	10.4k	10.4k
BRAM	13.50	13.50	13.50
Leaf cells	2830	2832	2832
Power (CLK1)	38480011.06	38480011.06	38480011.06
Area (CLK1)	72314.61	72314.61	72314.61
Power (CLK2)	9854972.14	9854972.14	9854972.14
Area (CLK2)	71167.19	71167.19	71167.19

V. CONCLUSION

In this paper, we presented a hardware efficient implementation of the threshold attenuated min-sum algorithm (TAMSA) to improve the performance of an LDPC decoder. The TAMSA algorithm was shown to improve the BER performance compared to the conventional AMSA with no extra hardware cost. Furthermore, a layered TAMSA architecture was proposed to reduce the hardware cost. Using circuit optimization techniques, including a full-parallel computing structure, the FPGA implementation results of layered TAMSA show that the modified architecture adds no extra circuit power or circuit area compared to conventional AMSA, and only 0.07% extra leaf cells compared to conventional MSA.

REFERENCES

[1] R. G. Gallager, *Low Density Parity Check Codes*. MIT Press, 1963.

[2] S. Lin and D. J. Costello, *Error control coding, Second Ed*. Prentice hall, 2001.

[3] D. J. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[4] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 54, no. 6, pp. 542–546, 2007.

[5] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A survey of FPGA-based LDPC decoders," *IEEE Commun. Surveys & Tutorials*, vol. 18, no. 2, pp. 1098–1122, 2015.

[6] T. T. Nguyen-Ly, V. Savin, K. Le, D. Declercq, F. Ghaffari, and O. Boncalo, "Analysis and design of cost-effective, high-throughput LDPC decoders," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 26, no. 3, pp. 508–521, 2017.

[7] J. H. Lee and M. H. Sunwoo, "Low-complexity first-two-minimum-values generator for bit-serial LDPC decoding," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 63, no. 5, pp. 483–487, 2015.

[8] K. Cushon, C. Leroux, S. Hemati, S. Mannor, and W. J. Gross, "A min-sum iterative decoder based on pulsewidth message encoding," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 57, no. 11, pp. 893–897, 2010.

[9] K. Gunnam, J. M. C. Perez, and F. Garcia-Herrero, "Algorithms and VLSI architectures for low-density parity-check codes: part 1-low-complexity iterative decoding," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 4, pp. 57–63, 2016.

[10] J. Chen, R. M. Tanner, C. Jones, and Y. Li, "Improved min-sum decoding algorithms for irregular LDPC codes," in *IEEE Int. Symp. Inf. Theory.*, 2005, pp. 449–453.

[11] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, 2005.

[12] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes," *IEEE Trans. Commun.*, vol. 53, no. 4, pp. 549–554, 2005.

[13] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *IEEE Workshop on Signal Processing Syst.*, 2004, pp. 107–112.

[14] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," *IEEE Journal on Selected Areas in Commun.*, vol. 27, no. 6, pp. 985–994, 2009.

[15] Y. Sun and J. R. Cavallaro, "VLSI architecture for layered decoding of QC-LDPC codes with high circulant weight," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 21, no. 10, pp. 1960–1964, 2012.

[16] K. Gunnam, G. Choi, W. Wang, and M. Yeary, "Multi-rate layered decoder architecture for block LDPC codes of the IEEE 802.11 n wireless standard," in *IEEE Int. Symp. Circuits Syst.*, 2007, pp. 1645–1648.

[17] H. Hatami, D. G. Mitchell, D. J. Costello, and T. Fuja, "A Threshold-Based Min-Sum Algorithm to Lower the Error Floors of Quantized LDPC Decoders," in *IEEE Trans. Commun.*, vol. 68, no. 4, pp. 2005-2015, 2020.

[18] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, 2004.

[19] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2006, pp. 149–152.