PAPER

An Efficient LDPC Decoder Architecture with a High-Performance Decoding Algorithm

Jui-Hui HUNG[†a)], *Nonmember and* Sau-Gee CHEN[†], *Member*

SUMMARY In this work, a high performance LDPC decoder architecture is presented. It is a partially-parallel architecture for low-complexity consideration. In order to eliminate the idling time and hardware complexity in conventional partially-parallel decoders, the decoding process, decoder architecture and memory structure are optimized. Particularly, the parity-check matrix is optimally partitioned into four unequal sub-matrices that lead to high efficiency in hardware sharing. As a result, it can handle two different codewords simultaneously with 100% hardware utilization. Furthermore, for minimizing the performance loss due to round-off errors in fixed-point implementations, the well-known modified min-sum decoding algorithm is enhanced by our recently proposed high-performance CMVP decoding algorithm. Overall, the proposed decoder has high throughput, low complexity, and good BER performances. In the circuit implementation example of the (576,288) parity check matrix for IEEE 802.16e standard, the decoder achieves a data rate of 5.5 Gbps assuming 10 decoding iterations and 7 quantization bits, with a small area of 653K gates, based on UMC 90 nm process technology.

key words: *channel coding, LDPC, decoder, algorithm, hardware*

1. Introduction

Low-density parity check (LDPC) code was introduced in 1962 [1], which can achieve performance close to Shannon bound. Hence, LDPC code has been adopted by many state-of-the-arts communication systems, such as the DVB-S2, DMB-TH and IEEE 802.16e systems. It is a binary linear block code whose parity-check matrix is sparse with much fewer 1's than a common matrix. A sparse parity-check matrix facilitates simple decoding algorithms and low-complexity decoder designs.

The $m \times n$ parity-check matrix H of a LDPC code is often represented by a bipartite graph, called Tanner graph [2], which is composed of n bit nodes v_j, j=1 to n, and m check nodes c_i, i=1 to m. Those bit nodes and check nodes are connected by edges defined by nonzero entries of the check matrix. Tanner graph shows a clear picture of all the information exchange links in a decoding process as depicted in Fig. 1 an example of 4 check nodes and 8 bit nodes. The number of 1's in each column of H determines the number of edges for each bit node connected to check nodes, and the number of 1's in each row of H determines the connections from each check node to bit nodes. In a Tanner graph, a check node and a bit node need to execute their own check

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

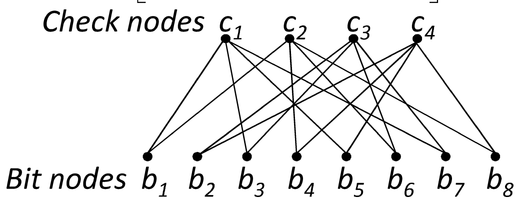

Check nodes c_1 c_2 c_3 c_4
Bit nodes b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8

Fig. 1 Tanner graph of a parity check matrix.

node equation (CNE) and bit node equation (BNE).

Although LDPC codes are efficient, huge computation is required particularly for the decoding processes. As a result, serial LDPC decoder architectures are not suitable for practical applications. To meet the decoding throughput requirement, fully-parallel LDPC decoders [3]–[5] were proposed. In those architectures, each CNE and BNE is realized by its own check node unit (CNU) and bit node unit (BNU), respectively. However, the incurred large area and long on-chip wire delays are the major drawbacks. To reduce these problems, partially-parallel architectures [6]–[8] were introduced, which contain some parallel CNUs and BNUs shared by all the CNEs and BNEs, respectively. The area reduction is dependent on the degree of parallelism. The partially-parallel architecture in [8] can concurrently decode two codewords at a time, which can get almost double throughput rate improvement over the single-codeword designs [6], [7], at the cost of some additional controller circuits, memory and a large number of multiplexers.

In this work, an efficient decoding schedule and its associated architecture are proposed which can concurrently decode two codewords more efficiently than the design of [8]. The decoding schedule is optimized so that it greatly reduces the number of multiplexers required in the design of [8]. Besides, in order to reduce the required number of CNUs and BNUs, generally current partially-parallel architectures need to reorder the columns and rows of $\boldsymbol{H}$ [7]–[10]. Nonetheless, for the decoding correctness, one needs to reorder the decoding results back to the original order. Thus, it will introduce additional area and time penalty. Here, without reordering $\boldsymbol{H}$, this works simply divide $\boldsymbol{H}$ into four optimally-partitioned unequal-size sub-matrices that lead to low-complexity and efficient architectures with shared BNU

Manuscript received March 4, 2010.
Manuscript revised June 17, 2010.

[†]The authors are with the Institute of Electronics, National Chiao Tung University, 1001 University Road, Hsinchu, Taiwan 300, ROC.
a) E-mail: drpaholisi.ee96g@g2.nctu.edu.tw

DOI: 10.1587/transcom.E93.B.2980

and CNU. In all, the proposed LDPC decoder achieves both high time and area performances.

In LDPC decoding, the sum-product algorithm (SPA) [11] provides excellent decoding performances. Nevertheless, it has very high computational complexity that impedes its practical applications. Some decoding algorithms are proposed to reduce the SPA computational complexity, such as the min-sum algorithm (MSA) [12] and the modified MSA [13]. In addition, for the consideration of low-power and low-complexity designs, fixed-point implementations instead of floating-point implementations are adopted. However, fixed-point realizations always cause quantization errors and rounding errors. Therefore, decoding performances will be significantly degraded. To remedy the problem, recently we proposed an improved MSA, called CMVP algorithm [14]. It achieves better performances than the mentioned decoding algorithms, under the condition of the same fixed-point precision. Besides, the hardware overhead of the CMVP algorithm over conventional modified MSA is small. In the proposed decoder architecture, we will combine this decoding algorithm for high-performance fixed-point decoding results.

This paper is organized as follows. In Sect. 2, the widely used SPA and MSA decoding algorithms are introduced. Section 3 details the design specifications and architecture of the proposed LDPC decoder, as well as the concept of CMVP algorithm. The specifications are defined after extensive simulations for determining the fixed-point word lengths, subject to the best tradeoff of cost and performance. Section 4 presents the hardware synthesis and chip implementation results. Finally, Sect. 5 is the conclusion.

2. Decoding Algorithms for LDPC Codes

Generally, LDPC decoding is based on the mentioned high-performance SPA [11], whose message is in the form of log-likelihood ratio (LLR): $L(x) = \log(P\{x=0\}/P\{x=1\})$. In this algorithm, the LLR message $L_{bi\to cj}$ sent by bit node b_i to check node c_j is defined by the following BNE:

$$L_{bi\to cj} = channel(b_i) + \sum_{c_k \in C\backslash c_j} L_{ck\to bi} \tag{1}$$

where $C\backslash c_j$ is a set containing all the check nodes connected to bit node b_i, excluding check node c_j; $channel(b_i)$ is the received LLR channel value of bit node b_i. In the equation, $L_{cj\to bi}$ is the LLR message passed from check node c_j to bit node b_i as defined by the following CNE:

$$L_{cj\to bi} = \prod_{b_k \in B\backslash b_i} sign(L_{bk\to cj})\phi\left(\sum_{b_k \in B\backslash b_i} \phi(|L_{bk\to cj}|)\right) \tag{2}$$

where $B\backslash b_i$ is a set containing all the bit nodes connected to check node c_j excluding bit node b_i, and $\phi(x) = \ln((e^x+1)/(e^x-1))$. It is hard to implement Eq. (2) which involves logarithm and exponential functions. The MSA [12] as shown below is a popular and low-complexity approximation to Eq. (2),

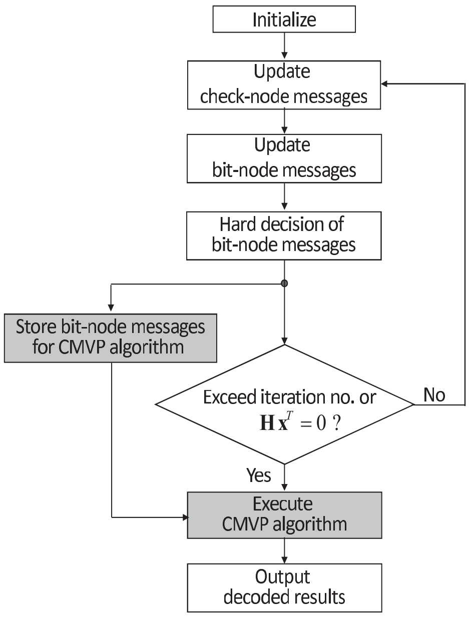

Initialize
Update check-node messages
Update bit-node messages
Hard decision of bit-node messages
Store bit-node messages for CMVP algorithm
Exceed iteration no. or $\mathbf{Hx}^T = 0$?
No
Yes
Execute CMVP algorithm
Output decoded results

Fig. 2 The flow chart of iterative decoding for LDPC codes.

$$L_{cj\to bi} \approx \left(\prod_{b_k \in B\backslash b_i} sign(L_{bk\to cj})\right) \min_{b_k \in B\backslash b_i} \left(|L_{bk\to cj}|\right). \tag{3}$$

Since the value of Eq. (3) is larger than the value of Eq. (2), the modified MSA [13] multiplies Eq. (3) by a normalization factor β which is smaller than one for a better approximation to Eq. (2). After the execution of all the CNEs and BNEs, the bit-node message of bit node b_i can be updated by

$$L_{bi} = channel(b_i) + \sum_{c_k \in C} L_{ck\to bj} = L_{bi\to cj} + L_{cj\to bi} \tag{4}$$

and the decoded value of bit node b_i is obtained from the hard-decision value of L_{bi}.

Figure 2 shows the conventional decoding process (composed of the white functional blocks) of a LDPC code, and the additional proposed decoding process (composed of the shaded functional blocks). Conventional LDPC decoding process involves back-and-fourth message update iterations between check nodes and bit nodes. The decoded bits are output when the decoded information bits satisfy the zero syndrome vector constraint, i.e., $\boldsymbol{xH}^T = \vec{0}$, or when the iteration number exceeds a maximum number, where x denotes the decoded codeword in row vector form. The introduced extra process further enhances the decoding correctness by saving some bit-node messages of Eq. (4) for later use by the additional CMVP unit, as will be introduced in the next section.

3. The Proposed LDPC Decoder Architecture

For the demonstration of the proposed architecture, we consider a LDPC code example of code rate 1/2 and a 288-by-576 parity-check matrix $\boldsymbol{H}$, as defined in the IEEE 802.16e standard. Before introducing the proposed LDPC decoder architecture, some hardware specifications are decided as follows. Figure 3 shows the bit error rate (BER) simulation results vs. SNR, in various iteration numbers, using BPSK modulation and the modified MSA with the normalization factor β=0.75 in AWGN channel. As shown, 10 decoding iterations is a good choice in terms of cost and performance for this design example.

Since practical implementation is in fixed-point operations, we need to determine adequate word length (in bits). Let $[t{:}f]$ denote the fixed-point quantization format, where t and f represent the total word length and the fraction length, respectively. Based on the simulation results of 10 iterations, the [7:5] format is chosen in the proposed design, which achieves good decoding performances with short word length and accordingly low hardware complexity.

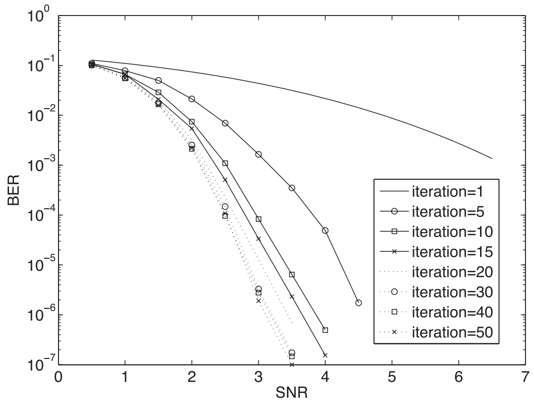


Fig. 3 Decoding performances of the floating-point modified MSA in various iteration numbers.

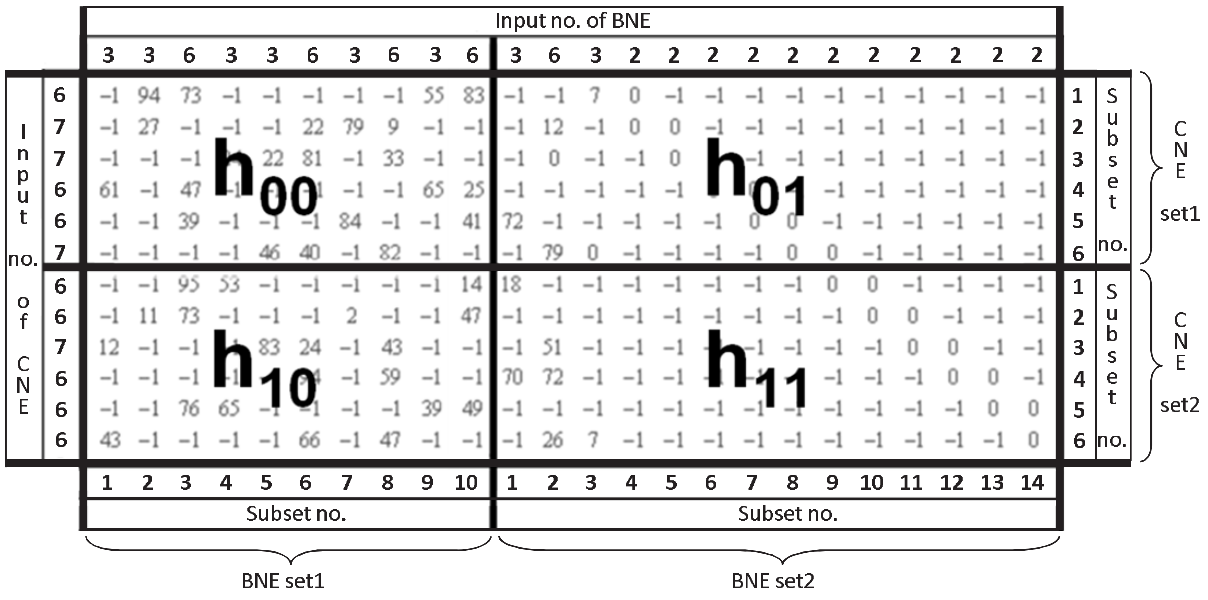
3.1 The Partition of $\boldsymbol{H}$

According to Tanner graph, the row and column numbers of H correspond to the BNE and CNE numbers, respectively. For the design of a partially-parallel LDPC architecture, $\boldsymbol{H}$ matrix can be partitioned into $M \times M$ sub-matrices. Consequently, there will have M CNE sets and M BNE sets. In the partially-parallel structure design, all the M CNE sets and M BNE sets will share the same CNU set and BNU set for their operations, respectively. According to our analysis, M=2 is a better choice over other M values. It leads to simple control hardware design and facilitates concurrent decoding operations of two independent codewords with 100% hardware utilization efficiency, based on a new scheduling scheme, as will be detailed later. Therefore, the parity-check matrix $\boldsymbol{H}$ is divided into four unequal-size sub-matrices (i.e., $\boldsymbol{h}_{00}$, $\boldsymbol{h}_{01}$, $\boldsymbol{h}_{10}$ and $\boldsymbol{h}_{11}$), and the syndrome vector $\boldsymbol{s}$ of codeword vector $\boldsymbol{x}$ can be rewritten as:

$$\boldsymbol{s} = \boldsymbol{x}\boldsymbol{H}^T = [\ \boldsymbol{x}_0 \quad \boldsymbol{x}_1\ ]\begin{bmatrix} \boldsymbol{h}_{00} & \boldsymbol{h}_{01} \\ \boldsymbol{h}_{10} & \boldsymbol{h}_{11} \end{bmatrix}^T \tag{5}$$

where $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are sub-vectors of $\boldsymbol{x}$.

Specifically, the optimized $\boldsymbol{h}_{00}$ and $\boldsymbol{h}_{10}$ are both 144×240 matrices, while both $\boldsymbol{h}_{01}$ and $\boldsymbol{h}_{11}$ are 144×336 matrices. Correspondingly, sub-vectors $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are 240×1 and 336×1 vectors, respectively. The reason why $\boldsymbol{H}$ is unequally partitioned is that it facilitates balanced CNU and BNU realizations in the proposed decoder. As a result, the reordering of the rows and columns of $\boldsymbol{H}$ [9] is unnecessary in the proposed design. Hence, higher hardware utilization rate, shorter delay time and smaller hardware area can be achieved, compared with the design of [7]–[10]. Detailed explanation on the optimized matrix partition will be given



		Input no. of BNE																								
		3	3	6	3	3	6	3	6	3	6	3	6	3	2	2	2	2	2	2	2	2	2	2	2	
Input no. of CNE	6	−1	94	73	−1	−1	−1	−1	−1	55	83	−1	−1	7	0	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	1
	7	−1	27	−1	−1	−1	22	79	9	−1	−1	−1	12	−1	0	0	−1	−1	−1	−1	−1	−1	−1	−1	−1	2
	7	−1	−1	−1	[illegible]	22	81	−1	33	−1	−1	−1	0	−1	−1	0	[illegible]	−1	−1	−1	−1	−1	−1	−1	−1	3
	6	61	−1	47	−1	[illegible]	−1	−1	−1	65	25	−1	−1	−1	−1	−1	[illegible]	0	[illegible]	−1	−1	−1	−1	−1	−1	4
	6	−1	−1	39	−1	−1	−1	84	−1	−1	41	72	−1	−1	−1	−1	−1	0	0	−1	−1	−1	−1	−1	−1	5
	7	−1	−1	−1	−1	46	40	−1	82	−1	−1	−1	79	0	−1	−1	−1	−1	0	0	−1	−1	−1	−1	−1	6
	6	−1	−1	95	53	−1	−1	−1	−1	−1	14	18	−1	−1	−1	−1	−1	−1	−1	0	0	−1	−1	−1	−1	1
	6	−1	11	73	−1	−1	−1	2	−1	−1	47	−1	−1	−1	−1	−1	−1	−1	−1	−1	0	0	−1	−1	−1	2
	7	12	−1	−1	−1	83	24	−1	43	−1	−1	−1	51	−1	−1	−1	[illegible]	−1	−1	−1	−1	0	0	−1	−1	3
	6	−1	−1	−1	−1	[illegible]	94	−1	59	−1	−1	70	72	−1	−1	−1	−1	[illegible]	[illegible]	−1	−1	−1	0	0	−1	4
	6	−1	−1	76	65	−1	−1	−1	−1	39	49	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	0	0	5
	6	43	−1	−1	−1	−1	66	−1	47	−1	−1	−1	26	7	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	0	6
		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
		Subset no.										Subset no.														

Fig. 4 The optimized $\boldsymbol{H}$ partition of the proposed LDPC decoder, for the (576,288) code in the 802.16e standard.

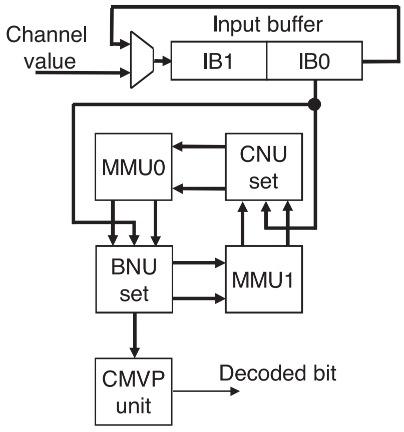


Fig. 5 The proposed LDPC decoder architecture.

in the latter subsections on CNU and BNU designs.

Figure 4 shows the detailed pattern of the example 288 × 576 parity-check matrix $\boldsymbol{H}$, where the sub-matrices of $\boldsymbol{H}$ are separated by the dark solid lines. For convenience, each BNE and CNE set is further divided into many subsets as marked in the second bottom row and the second rightmost column, respectively. An entry in the matrix represents a 24-by-24 sub-matrix; those entries with -1 value correspond to zero matrices, while a non-negative entry represents a specific cyclically-shifted identity matrix defined by the entry value [15]. Note that the entries with zero values represent the 24-by-24 unit matrix. A specific number in the second top row indicates the input number to each of its 24 corresponding BNEs defined by those 24 × 24 sub-matrices just beneath the number. For example, since the column corresponding to subset number 1 of BNE set 1 has three entry values other than -1, there are three inputs to each of the 24 BNEs of this subset. Similarly, a specific number in the second leftmost column represents the number of inputs to all of its corresponding 24 CNEs in the CNE subset.

3.2 Decoding Steps of the Proposed Architecture

As shown in Fig. 2, in the iterative decoding process, before updating the bit node messages, one needs to update the check node messages first, and vice versa. It means that CNU and BNU will be respectively idling when BNU and CNU are operating, in the case of decoding a single codeword at a time. This results in low hardware utilization efficiency. Hence, to achieve 100% utilization rate of both CNU and BNU sets, the proposed partially parallel architecture (as shown in Fig. 5) processes two codewords at a time by interleaving and reordering data sequences in the decoding procedures. Besides, as mentioned before, some bit-node messages computed by BNUs are stored in CMVP unit. This architecture contains an input buffer, the mentioned CNU and BNU sets, two message memory units (i.e., MMU0 and MMU1) and a CMVP unit for enhancing the decoding accuracy. Note that the input buffer is divided into IB0 and IB1 buffers; and each can hold 576 channel values of a codeword.

Before introducing the proposed two-codeword decoding process and decoder structure in detail, the basic operation steps of decoding a single codeword $\boldsymbol{x}$ are explained first as below, followed by the process of decoding two code words at a time.

3.2.1 Basic Decoding Steps for a Single Codeword

Let the message value set of a codeword $\boldsymbol{x}$ corresponding to the sub-matrix $\boldsymbol{h}_{mn}$ be denoted as L_{mn} which dynamically includes all the message values of either $L_{cj\to bi}$ or $L_{bi\to cj}$ associated with sub-matrix $\boldsymbol{h}_{mn}$, m, n=0 or 1.

Operation (1) Update check node messages for CNE set 1: Update check node messages for CNE set 1: CNU set updates the message value set pair of $\{L_{00}, L_{01}\}$ according to Eq. (3), and temporarily stores the updated values in MMU0.

Operation (2) Update check node messages for CNE set 2: Similar to Operation (1), CNU set updates the message value set pair of $\{L_{10}, L_{11}\}$, and temporarily stores the updated values in MMU0.

Operation (3) Reorder check node messages: The message value sets L_{00}, L_{01}, L_{10} and L_{11} in MMU0 are reordered and paired as $\{L_{00}, L_{10}\}$ and $\{L_{11}, L_{01}\}$ for BNE set 1 and BNE set 2 operations, respectively.

Operation (4) Load check node messages for BNE set 1: MMU0 feeds the value set pair $\{L_{00}, L_{10}\}$ to BNU set inputs, and at the same time IB0 sends the first 240 channel values of $\boldsymbol{x}$ (i.e., $\boldsymbol{x}_0$) parallelly to the inputs of BNU set for BNE set 1 operations described by Eq. (1) and Eq. (4) (which will be performed in Operation (5)).

Operation (5) Update bit node messages for BNE set 1: BNU set updates the message value sets L_{00} and L_{10} (according to Eq. (1)) and temporarily stores the updated values in MMU1. Besides, BNU set also computes and sends the corresponding bit-node messages (described by Eq. (4)) to MMU1, and stores the necessary information for CMVP unit. Then, IB circularly shifts right by 240 positions within one clock cycle. That means $\boldsymbol{x}_0$ is now in the leftmost part of IB1, while $\boldsymbol{x}_1$ is in the rightmost part of IB0.

Operation (6) Load check node messages for BNE set 2: Similar to Operation (4), MMU0 feeds the value set pair $\{L_{11}, L_{01}\}$ to BNU set inputs, and IB0 concurrently provides 336 $\boldsymbol{x}_1$ channel values to the inputs of BNU set for BNE set 2 operations (which will be performed in Operation (7)).

Operation (7) Update bit node messages for BNE set 2: Similar to Operation (5), BNU set updates the message value sets L_{11} and L_{01}, and temporarily stores the updated values in MMU1. Besides, BNU set also sends the corresponding bit-node messages to CMVP unit for further error correction of decoded bits. After that, IB circularly shifts right by 336 positions within one clock cycle. At this moment, the entire $\boldsymbol{x}$ channel values are now in IB1.

Operation (8) Reorder bit node messages: The message

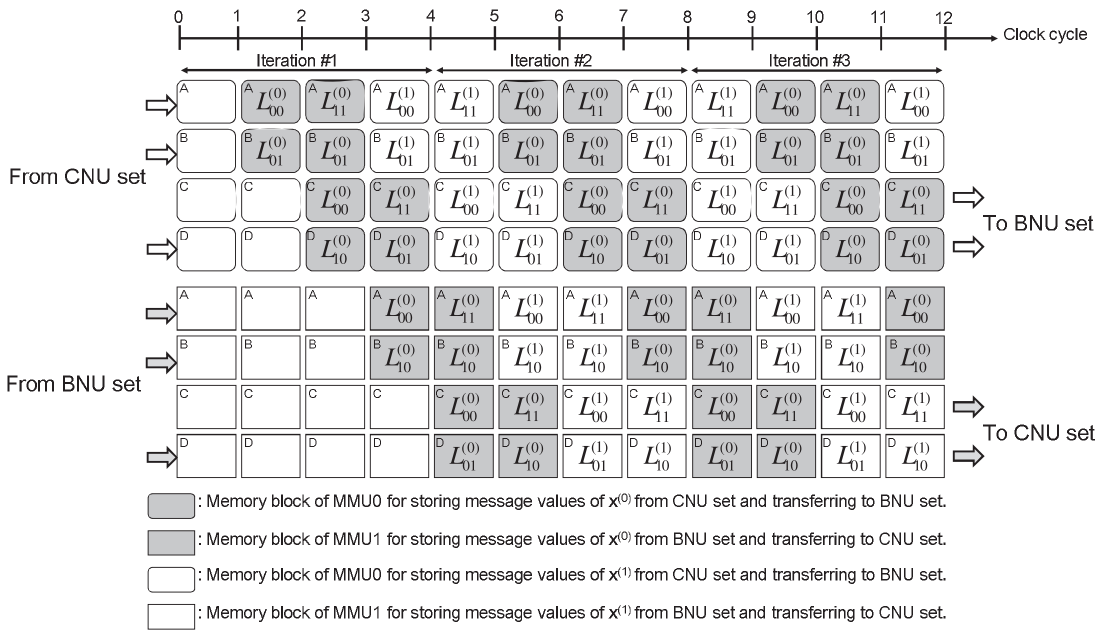

Clock cycle
Iteration #1
Iteration #2
Iteration #3
From CNU set
To BNU set
From BNU set
To CNU set
: Memory block of MMU0 for storing message values of x(0) from CNU set and transferring to BNU set.
: Memory block of MMU1 for storing message values of x(0) from BNU set and transferring to CNU set.
: Memory block of MMU0 for storing message values of x(1) from CNU set and transferring to BNU set.
: Memory block of MMU1 for storing message values of x(1) from BNU set and transferring to CNU set.

Fig. 6 The timing diagram of the message passing process between BNU/CNU and MMU0/MMU1.

value sets L_{00}, L_{01}, L_{10} and L_{11} in MMU1 are reordered and paired as $\{L_{00}, L_{01}\}$ and $\{L_{11}, L_{10}\}$.

Operation (9) Load bit node messages for CNE set 1: MMU1 feeds the value set pairs $\{L_{00}, L_{01}\}$ to the input of CNU sets.

Operation (10) Load bit node messages for CNE set 2: Similar to Operation (9), MMU1 feeds the value set pairs $\{L_{11}, L_{10}\}$ to the input of CNU sets.

Operation (11) Final enhancement of the decoded bits: Based on the hard-decision values of the bit-node messages stored for CMVP unit, CMVP unit performs further correction operations of the decoded bits, and output the decoded codeword.

3.2.2 Decoding Steps for Two Codewords at a Time

Basically, the two-codeword processing can be regarded as two independent single-codeword processings operated in parallel and interleaved fashion. Besides, to reduce control complexity, these two decoding processes will be terminated only when the iteration numbers reach a set value.

Consider two independent codewords' channel values $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$. Further, denote Operation (i_0) and Operation (i_1) as Operation (i) of $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$ in the single-codeword operation, respectively. Then the complete concurrent LDPC decoding steps (in terms of clock cycles) of $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$ on the proposed decoder architecture can be described below, assuming that initially the channel values (in LLR form) of two codeword $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$ has been already shifted into IB0 and IB1, respectively:

Step 1) Load channel values of $\boldsymbol{x}^{(0)}$: IB0 feed all 576 $\boldsymbol{x}^{(0)}$ channel values to CNU set. After that, IB0 and IB1 swap their contents (i.e., $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(0)}$ are in IB0 and IB1, respectively) within one clock cycle.

Step 2) Update check node messages of $\boldsymbol{x}^{(0)}$ and load channel values of $\boldsymbol{x}^{(1)}$: Execute Operation (1_0) and feed all 576 $\boldsymbol{x}^{(1)}$ channel values instantly from IB0 to CNU set simultaneously. Then, IB0 and IB1 exchange their contents (i.e., $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$ are in IB0 and IB1, respectively).

Steps i=3 to 11) Parallel iteration steps for two codewords: Execute Operation ($(i-1)_0$) and Operation ($(i-2)_1$) simultaneously.

Step 12) Check termination condition and load bit node messages of $\boldsymbol{x}^{(1)}$: If the maximum iteration number is not reached, execute Operation (1_0) and Operation (10_1) simultaneously and then go to Step 3.

Step 13) Final enhancement of the decoded bits $\boldsymbol{x}^{(0)}$ and load bit node messages of $\boldsymbol{x}^{(1)}$: Execute Operation (11_0) and Operation (10_1).

Step 14) Final enhancement of the decoded bits $\boldsymbol{x}^{(1)}$: Execute Operation (11_1)

When the decoding process for these two codewords is completed, the decoder can immediately begin a new decoding process for the next pair of codewords by repeating the whole decoding process.

Figure 6 shows the timing snapshots of the data flow between CNU/BNU and the memory units MMU0/MMU1 when decoding two codewords at a time, where $L_{mn}^{(0)}$ and $L_{mn}^{(1)}$ are the message values of codewords $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$, respectively, corresponding to sub-matrix $\boldsymbol{h}_{mn}$. In the figure, the rounded-corner rectangles in the upper part (i.e., MMU0) and sharp-corner rectangles in the lower part (i.e., MMU1) represent the message value set L_{mn} passed from check nodes to bit nodes and bit nodes to check nodes, respectively. Detailed MMU structure and mechanism will be given in the next section. This timing diagram shows only

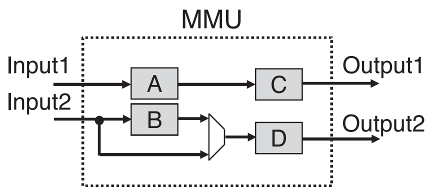


Fig. 7 The proposed MMU0 and MMU1 architecture.

three decoding iterations for simplicity.

As shown, there is no idling time slot in BNU set, CNU set, MMU0 and MMU1 in the iterative decoding process, owing to the proposed interleaving and reordering schemes.

3.3 Function Units of the Proposed Architecture

3.3.1 Input Buffer (IB)

As shown in Fig. 5, the input buffer (IB) is a specialized shift register divided into two equal parts IB0 and IB1; and each part has 576 words for storing the channel values of a codeword. Furthermore, without loss of generality, it is assumed that each input channel value word is in [7:5] fixed-point format. For hardware simplicity, the communication between IB and CNU/BNU is through IB0. As such, significant amount of MUXs and wire interconnections can be saved compared to the design of [8]. The whole BNE set is divided into two BNE sets which have 240 and 336 inputs respectively. Accordingly, the shift register can do circular right-shift operations by one word position (for initially loading channel values), 240 word positions (for Operation (5)), 336 word positions (for Operation (7)) or 576 word positions (for IB0 and IB1 data swapping in Step 1)), within one clock cycle. Moreover, all the data and partial data of IB0 can be selectively connected to CNU set (for loading the input data) and BNU set (for updating the bit node messages) at the same time.

3.3.2 MMU

As shown in Figs. 6 and 7, both MMU0 and MMU1 have four memory sub-blocks as marked with letters *A*, *B*, *C* and *D*, where sub-blocks *A*, *B* and *D* capture the outputs from CNU set for MMU0, or from BNU set for MMU1; sub-blocks *C* and *D* of MMU0 and MMU1 deliver their stored message values to BNU and CNU, respectively. Detailed MMU structure is shown in Fig. 7. Specifically, the input data pairs to MMU, i.e., $\{L_{00}, L_{01}\}$ and $\{L_{11}, L_{10}\}$ are for MMU0 while $\{L_{00}, L_{10}\}$ and $\{L_{11}, L_{01}\}$ are for MMU1, are stored in sub-blocks $\{A, B\}$ and $\{A, D\}$, respectively. Besides, MMU feeds the reordered data to its succeeding function unit from the sub-blocks $\{C, D\}$ as shown in the figure. As depicted, the proposed MMU architecture only needs one multiplexer and four memory sub-blocks instead of three multiplexer and five memory sub-blocks in [8]. Due to these merits, the proposed design has a lower hardware complexity than that of [8], while it is also free of any idling

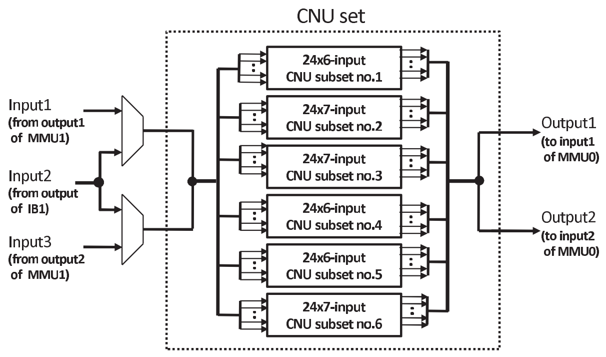

CNU set
Input1
(from output1 of MMU1)
Input2
(from output of IB1)
Input3
(from output2 of MMU1)
24x6-input CNU subset no.1
24x7-input CNU subset no.2
24x7-input CNU subset no.3
24x6-input CNU subset no.4
24x6-input CNU subset no.5
24x7-input CNU subset no.6
Output1
(to input1 of MMU0)
Output2
(to input2 of MMU0)

Fig. 8 The block diagram of the proposed complete CNU architecture.

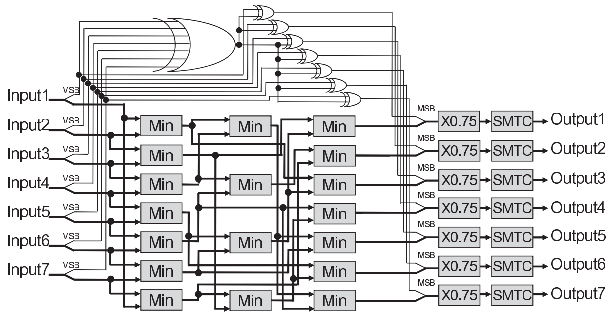


Fig. 9 The seven-input CNU architecture in the proposed LDPC decoder.

time slots in the iteration process.

3.3.3 CNU Set

Since the numbers of inputs to all the CNEs are uniformly either 6 or 7, the total 12 CNE subsets are equally partitioned into CNE set 1 and set 2, each contains 6 CNE subsets, as shown in Fig. 4. Hence, both CNE set 1 and set 2 contain 24×6 CNEs. Therefore, the decoder only requires a set of 24×6 parallel CNUs shared by the two CNE sets and each CNU in the CNU set is responsible for the execution of a CNE (i.e., Eq. (3)) in CNE set 1and another CNE in CNE set 2. As such, each CNU should be able to handle the maximum input numbers of those two CNEs. According to CNE input numbers in Fig. 4, the input number of each CNU is either six or seven. As shown in Fig. 8, the complete CNU set contains three 24×6-input and three 24×7-input CNU subsets. That means each 24×6-input CNU subset and 24×7-input CNU subset comprises 24 six-input CNUs and 24 seven-input CNUs, respectively. Notice that since all the CNUs inside the 2nd and 6th CNU subsets have seven inputs, when they are executing the six-input CNEs of the 2nd and 6th CNE subsets in CNE set 2, those unused 7th inputs are fed with the maximum information value so that those inputs will not affect the outputs.

The seven-input CNU architecture, shown in Fig. 9, has been proposed in our preliminary result [16]. In this figure, thick and thin lines represent the multi-bit and single-bit buses, respectively. The six-input CNU has a similar structure to the seven-input CNU. Besides, for low-complexity

Table 1 Mapping between BNU subsets and BNE subsets.

BNU set		BNE set1		BNE set2	
Subset no.	**Subset input no.**	Subset no.	Subset input no.	Subset no.	Subset input no.
1	**3**	1	3	X	X
2	**3**	2	3	1	3
3	**6**	3	6	2	6
4	**3**	4	3	X	X
5	**3**	5	3	3	2
6	**6**	6	6	4/5/6	2/2/2
7	**3**	7	3	7	2
8	**6**	8	6	8/9/10	2/2/2
9	**3**	9	3	11	2
10	**6**	10	6	12/13/14	2/2/2

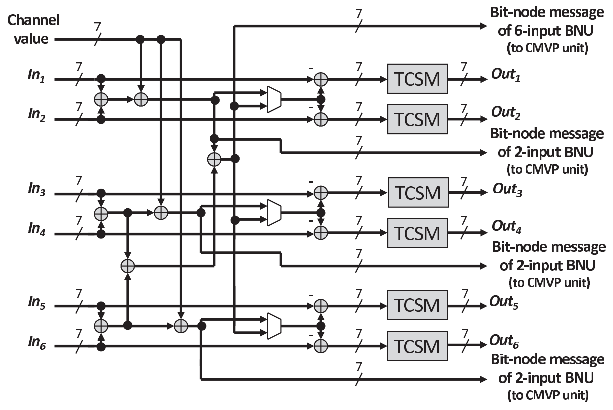
and high-performance consideration, all the CNUs are designed to execute the modified min-sum algorithm [13] with a normalization factor of 0.75 which is equal to $2^{-1}+2^{-2}$. This choice facilitates simple implementation of the normalization operations. In the figure, the output of a Min cell is the minimal value of its two inputs.

A SMTC unit converts a CNU output in sign-magnitude representation to 2's-complement representation before sending it to a BNU. It is because that finding the absolute minimal value is easier by using sign magnitude representation than the 2's complement representation, while 2's complement representation is more suitable for addition and subtraction operations. Hence, the 2's complement data are converted back to the sign magnitude representations by using the TCSM modules after the BNU operations. Besides, for the consideration of the best balanced circuit delay time, SMTC and TCSM modules are included in CNUs and BNUs, respectively.

3.3.4 BNU Set

Since both BNE set 1 and BNE set 2 will share the same BNU set, they should be optimally arranged in the matrix partition, as shown in Fig. 4. Specifically, 10 and 14 BNE subsets are allocated to BNE set 1 and set 2, respectively. Since each BNE subset comprises 24 BNEs, BNE set 1 and set 2 contain 24×10 and 24×14 BNEs, respectively. Based on this arrangement, the proposed BNU sets consists of 24×10 parallel BNUs shared by the two BNE sets. The 24×10 BNUs are basically one-to-one mapped to the 24×10 BNEs of BNE set 1, and they can be reconfigured to execute all the 24×14 BNEs of BNE set 2.

The complete BNU-set architecture is very similar to the complete CNU-set architecture shown in Fig. 8, except that the CNU subsets are replaced by BNU subsets and there are ten BNU subsets with six or three inputs. Besides, Input1 and Input2 to the BNU set are from MMU0, and BNU set has an addition output bus connected to CMVP unit. The BNU architecture is a direct mapping from BNE set 1. Consequently, according to Fig. 4, there are ten BNU subsets and their numbers of inputs are the same as those of BNE set 1. Table 1 shows the detailed mapping between the BNU subsets and BNE set 1 and set 2. Inside the BNU set, as mentioned before, a six-input adder of the 6th, 8th and 10th BNU subsets (for executing BNE set 1 operations) is efficiently



Channel value
In1
In2
In3
In4
In5
In6
7
TCSM
Out1
Out2
Out3
Out4
Out5
Out6
Bit-node message of 6-input BNU (to CMVP unit)
Bit-node message of 2-input BNU (to CMVP unit)

Fig. 10 The six-input BNU architecture in the proposed LDPC decoder.

designed to perform three independent two-input addition operations (for executing BNE set 2 operations), and a three-input adder in executing BNE set 1 is directly employed as a two-input adder for executing BNE set 2, as shown in Table 1. Figure 10 shows the six-input BNU architecture. On the other hand, all the BNUs inside the 5th, 7th and 9th BNU subsets have three inputs, are capable of executing both the three-input BNEs and the two-input BNEs. Without affecting the execution results, those unused third inputs are fed with the zero value for executing the two-input BNEs. Besides, due to the size difference of BNE set 1 and set 2, the results of the 1st and 4th BNU subsets for executing BNE set 2 will not be fed to MMU1.

Notice that the output word length will be clipped to fit the input word length after all the add/sub operations for avoiding word length mismatch.

3.3.5 CMVP Unit

In practical realization, for the consideration of low-power and low-complexity designs, fixed-point implementation instead of floating-point implementation is adopted. However, fixed-point realizations always cause quantization errors. As such, decoding performances will be significantly degraded. To remedy this problem, intuitively, a fixed-point quantization with enough word length is needed. However, doing this way will significantly increase the hardware area and power consumption. To alleviate the problem, recently we proposed a post-processing algorithm in [14] which enhances the decoding correctness of the existing decoding algorithms (including SPA, MSA and modified MSA) by defining the following additional auxiliary decoding operation as briefly explained below. The performance of this algorithm approaches that of the floating-point MSA with small hardware overhead. Please refer to [14] for detailed introduction of the algorithm.

3.3.5.1 Concept of the CMVP Algorithm

The CMVP algorithm consists of the following three different processes, including the confidence test, the persistency test and the majority vote to help decode the message bits:

***Confidence test*:** In a decoding process, the LLR of a message bit (as defined by Eq. (4)) is utilized to decide whether the message bit should be decoded as 0 or 1. Intuitively, we expect that a message bit would be very likely wrongly decoded if its absolute LLR value is close to zero. Therefore, the CMVP algorithm defines a confidence threshold C to assist the judgment of the correctness of a tested bit, i.e., if the absolute LLR value of a tested bit in the latest iteration is less than the confidence threshold, the tested bit will be treated as an unreliable bit.
***Persistency test*:** By intuition and observation, one can expect that the later iteratively bit-node messages are more credible than the earlier ones. Hence, CMVP algorithm considers the tested bit as reliable, if its last P iteratively bit-node messages (obtained from the adopted decoding algorithm, such as SPA, MSA or modified MSA) remain constant; otherwise this bit will be regarded as an unreliable bit.

After the confidence test and the persistency test, the unreliable bits (i.e., those message bits remained to be decided) will be subject to the following majority vote operation to finalize their bit values.
***MajorityVote*:** This process is intended to enhance the correctness probability of a tested bit; the proposed CMVP algorithm further performs the majority vote on the latest MV iteratively bit-node messages of the unreliable bit after the confidence and persistency tests. Then, the majority vote result is the final decoded value of the unreliable bit.

As a result, the proposed decoder needs to store the last MV or P (depending on which one is larger) iterative hard-decision values of the bit-node messages for CMVP unit. The CMVP algorithm is summarized below.

3.3.5.2 Detailed Flow of the CMVP Algorithm

At the end of conventional LDPC decoding operations (by using SPA, MSA or modified MSA), CMVP algorithm is performed according to the following six steps. Notice that the last MV or P iterative hard-decision values of the bit-node messages obtained from the adopted decoding algorithm have been stored in the memory of CMVP unit before executing CMVP algorithm.

Step 1) Initialize: Set C, P and MV to some prescribed proper values. Set the value of bit count variable N_b to 1. Denote L_{Nb} as the LLR value of the N_b-th bit in Eq. (4).
Step 2) Test the confidence: If $|L_{Nb}| \geq C$, set this bit to the latest decoding result and go to Step 5.
Step 3) Test the persistency: If the last P iterated decoding results of the N_b-th bit are all the same, set this bit to the latest decoding result and go to Step 5.
Step 4) Perform majority vote: Execute the majority vote based on the last MV decoding results of the tested bit, if it has more 1 than 0, set this bit to 1, and vice versa.
Step 5) Check termination condition: $N_b = N_b + 1$. If N_b is less than or equal to the code length, go to Step 2.
Step 6) End of the algorithm: Output the decoded bits.

Figure 11 shows the flow chart of the CMVP algorithm

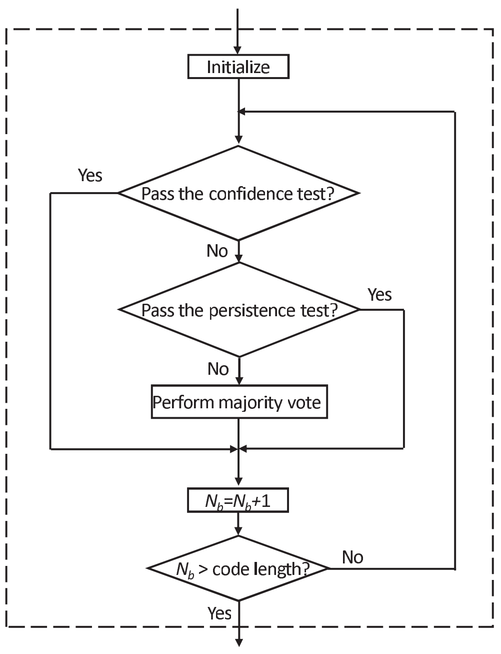


Fig. 11 The flow chart of the CMVP algorithm.

which corresponds to the CMVP block in Fig. 2. Notice that this algorithm can be flexibly adjusted by considering different combinations of the design factors, such as including all the testing factors (i.e., C, MV and P) or any two factors out three possible factors or only one factor for low complexity design requirements.

In designing CMVP architecture, we need to decide the optimal C, MV and P values first. Since it is hard to derive the optimal values by theoretical analysis, computer simulations were conducted to properly decide the values. Simulation results show that MV=3 and C=1 achieve the best performance among all the tested settings. In this case, there is no need to consider P, because the maximum P value is equal to the trivial value of one according to the constraint that P should be smaller than half of MV value and larger than 1. Figure 12 compares the performances of the CMVP algorithm (assuming MV=3 and C=1) combined with the modified MSA and the modified MSA alone. As shown, the proposed algorithm achieves better performances than the modified MSA by more than 0.2 dB.

It is very easy to test the condition if $|L_{Nb}| \geq C$ by using TCSM unit, because if we select C=1 with [7:5] quantization scheme mentioned before, (i.e. C will be '0100000' in sign magnitude representation), we only need to check the second bit from the MSB of TCSM output for each bit node. Note that the MSB of TCSM output is the sign bit. The combined TCSM and CMVP architecture is shown in Fig. 13 where thick and thin lines represent the multi-bit and single-bit buses, respectively. In the figure, there are two 3×1-bit memory units (because MV=3) which respectively store the last three iterative hard-decision values of the bit-node messages $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$ from BNUs. The control signal "Codeword no." is used to select the right memory unit to be accessed at the right time.

4. Implementation

As mentioned before, the proposed design processes two different codewords concurrently without any stalls. It takes four clock cycles to complete a decoding iteration for each codeword, including two cycles for CNUs and two cycles for BNUs. For channel value loading, each codeword takes one extra cycle. Since the maximum iteration number of decoding a codeword is 10 in the proposed design, the total amount of clock cycles needed to complete the decoding of two different codewords is 1+1+10*4=42 cycles. Moreover, since the synthesized clock frequency is 400 MHz, the decoding throughput is 400*[1152*(1/2)]/42≈5.5 Gbps.

The proposed LDPC decoder is compared with some other designs, as listed in Table 2. Notice that the 2nd and 3rd columns are the proposed decoders with and without the CMVP algorithm, respectively. As mentioned before, adopting CMVP algorithm can further improve the decoding performance by more than 0.2 dB in this design case. One can see that the area and power overheads of the CMVP algorithm are 4K gates and 2 mW, which only occupy 0.7% and 1% of the total area and power consumption, respectively.

Since the compared designs are implemented with different IC process technologies, code structures, code lengths, code rates, word lengths and iteration numbers, it is hard to do a precise comparison. However, some observations still can be concluded. As shown, compared with the fully parallel designs [4], [5], the proposed design operates with a smaller iteration number and a longer word length. It is for the consideration of good trade-off between hardware complexity and decoding performance as described in the beginning of Sect. 3. Besides, the throughput of 5.5 Gbps is much higher than 1.024 Gbps of [4] and 3.3 Gbps of [5], at the same time it only costs 2/5 area of [4] and slightly smaller than that of [5].

Compared with the partially parallel design in [7], the proposed design has a slightly larger area (in terms of gate counts), probably because that the decoder of [7] only decodes a codeword at a time and needs no additional memory control circuit to handle the second codeword. However, note that [7] does not specifies the word length of its design and the word length could have noticeable impact on the area size. Regarding the throughput performance, the new design is five times that of [7].

As mentioned in Sect. 3.2, the proposed structure is

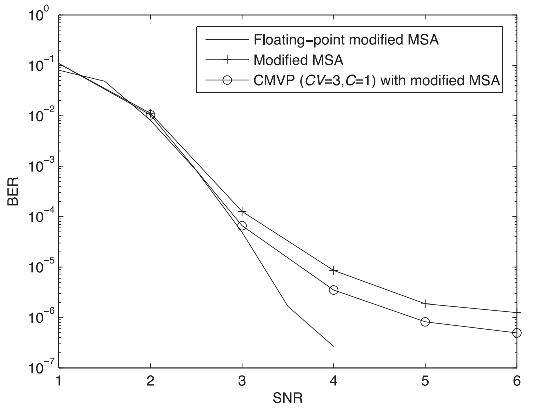


Fig. 12 Fixed-point simulations of the proposed CMVP algorithm combined with the modified MSA, iteration number=10.

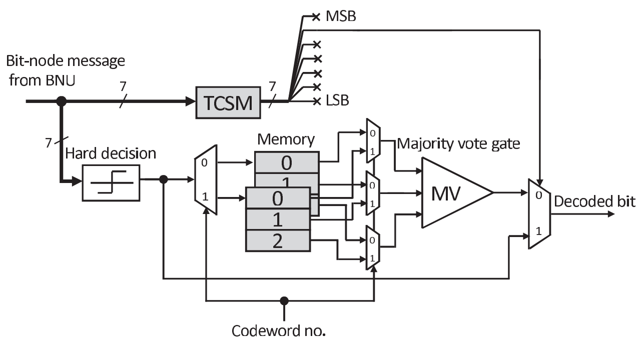


Fig. 13 The Combined [7:5] TCSM and CMVP unit, C=1 and MV=3.

Table 2 Comparison of LDPC decoders.

	Proposed-1	Proposed-2	[4]	[5]	[7]	[8]	[17]
Code length	576	576	1024	660	1024	1200	2304
Code rate	1/2	1/2	1/2	8/11	1/2	3/5	1/2
Word length (bits)	7	7	4	4	N/A	6	5
Iteration number	10	10	64	15	8	8	10
Technology (*nm*)	90	90	160	130	180	180	90
Clock rate (*M*Hz)	400	400	64	300	200	64	950
Power (*m*W)	117@0.9V	115@0.9V	690@1.5V	1408@1.2V	N/A	644@1.8V	870@1.0V
Area (*K* gate count)	653	649	1750	690	543	1150	645
Throughput (*M*bps)	5500	5500	1024	3300	985	3330	2200
Decoding algorithm	CMVP+Modified min-Sum	Modified min-sum	Sum-product	Modified min-sum	Sum-product	Min-sum	Modified min-sum
Scheme	Partially parallel	Partially parallel	Fully parallel	Fully parallel	Partially parallel	Partially parallel	Partially parallel

based on an optimized decoding schedule and special memory arrangement. The proposed design requires less memory sub-blocks and multiplexers than the designs of [8]. Besides, the designed six-input BNU can be dynamically adjusted to execute three two-input BNEs at the same time. Hence, the BNUs attain high area utilization in executing BNE set 1 and BNE set 2. These design features help the proposed decoder to achieve 100% hardware utilization rate which is more efficient than [8].

Compared with the design of [17] assuming the same IC process technology and iteration number, the proposed design has a much higher throughput than 2.2 Gbps of [17], even thought the clock rate of the proposed design is 400 MHz which is much slower than 950 MHz of [17]. Besides, the power consumption of proposed design is also much less than the design of [17]. Overall, in terms of time and area performances, the proposed designs are better than the compared designs.

5. Conclusion

A partially parallel, high-throughput and area-efficient LDPC decoder architecture has been proposed in the work. Owing to an optimized scheduling scheme and combined MMU design for message passing, the proposed design achieve 100% utilization rate in CNU and BNU, while require smaller memory size than the existing designs. Based on the structure, a decoder for the (576,288) LDPC code of 802.16e standard is realized. In addition, in order to reduce the decoding error and compensate the performance loss due to fixed-point realization of an LDPC decoder, a CMVP algorithm is employed in the proposed design with very small hardware overhead. The design concepts and techniques can also be well applied to other LDPC decoder designs with different specifications.

Acknowledgments

This work is supported in part by the grants NSC 95-2219-E-009-004 and MOEA 95-EC-17-A-01-S1-048, Taiwan.

References

[1] R.G. Gallager, Low-density parity-check codes, MIT Press, MA, 1963.

[2] R. Tanner, "A recursive approach to low complexity codes," IEEE Trans. Inf. Theory, vol.IT-27, no.3, pp.533–547, Sept. 1981.

[3] C.J. Howland and A.J. Blanksby, "Parallel decoding architectures for low density parity check codes," Proc. IEEE ISCAS'01, vol.4, pp.742–745, May 2001.

[4] A.J. Blanksby and C.J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," IEEE J. Solid-State Circuits, vol.37, no.3, pp.404–412, March 2002.

[5] A. Darabiha, A.C. Carusone, and F. Kschischang, "A 3.3-Gbps bit-serial block-interlaced min-sum LDPC decoder in 0.13-μm CMOS," Proc. IEEE CICC'07, pp.459–462, San Jose, California, Sept. 2007.

[6] Z. Cui and Z. Wang, "Area-efficient parallel decoder architecture for high rate QC-LDPC codes," Proc. IEEE ISCAS'06, pp.5107–5110, May 2006.

[7] S.H. Kang and I.C. Park, "Loosely coupled memory-based decoding architecture for low density parity check codes," IEEE Trans. Circuits Syst. I, vol.53, no.5, pp.1045–1056, May 2006.

[8] C.-C. Lin, K.-L. Lin, H.-C. Chang, and C.-Y. Lee, "A 3.33 Gb/s (1200,720) low-density parity check code decoder," Proc. IEEE ESSCIRC'05, pp.211–214, Sept. 2005.

[9] I.C. Park and S.H. Kang, "Scheduling algorithm for partially parallel architecture of LDPC decoder by matrix permutation," Proc. IEEE ISCAS'05, pp.5778–5781, May 2005.

[10] Y. Chen and K.K. Parhi, "Overlapped message passing for quasi-cyclic low-density parity check codes," IEEE Trans. Circuits Syst. I, vol.51, no.6, pp.1106–1113, June 2004.

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

[12] X.Y. Hu, E. Eleftheriou, D.M. Arnold, and A. Dholakia, "Efficient implementation of the sum-product algorithm for decoding LDPC codes," Proc. IEEE GLOBECOM'01, vol.02, pp.1036–1036E, Nov. 2001.

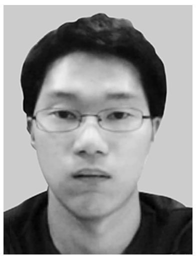
[13] J. Chen and M.P.C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," IEEE Trans. Commun., vol.50, no.3, pp.583–587, March 2002.

[14] J.H. Hung and S.G. Chen, "A low-complexity high-performance decoding algorithm for fixed-point LDPC decoder," Proc. IEEE ICSPCS'08, pp.1–5, Dec. 2008.

[15] Part 16: Air Interface for Fixed and Mobile BroadbandWireless Access Systems Amendment for Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands, IEEE P802.16e-2005, 2005.

[16] J.H. Hung and S.G. Chen, "A systematic optimized comparison algorithm for fast LDPC decoding," Proc. IEEE ISSPIT'07, pp.922–926, Dec. 2007.

[17] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-ldpc codes," IEEE J. Sel. Areas Commun., vol.27, no.6, pp.985–994, Aug. 2009.



Jui-Hui Hung received his B.S. degree from National Chi Nan University, Taiwan, in 2005 and M.S. degree in electrical engineering, from the Institute of Electronics, National Chiao Tung University. He is currently a Ph.D. student at the same Institution. His research interests include digital signal processing, channel coding, VLSI architecture and bio-information.

Sau-Gee Chen received his B.S. degree from National Tsing Hua University, Taiwan, in 1978, M.S. degree and Ph.D. degree in electrical engineering, from the State University of New York at Buffalo, NY, in 1984 and 1988, respectively. During 2003 and 2006, he was director of Institute of Electronics, Department of Electronics Engineering, National Chiao Tung University, Taiwan. Currently, he is a professor at the same organization. His research interests include digital communication, multi-media computing, digital signal processing, and VLSI signal processing. He has published more than 100 conference and journal papers, and holds several US and Taiwan patents.