TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 09/11 pp92–103
Volume 22, Number 1, February 2017

Design and Efficient Hardware Implementation Schemes for Non-Quasi-Cyclic LDPC Codes

Baihong Lin, Yukui Pei*, Liuguo Yin, and Jianhua Lu

Abstract: The design of a high-speed decoder using traditional partly parallel architecture for Non-Quasi-Cyclic (NQC) Low-Density Parity-Check (LDPC) codes is a challenging problem due to its high memory-block cost and low hardware utilization efficiency. In this paper, we present efficient hardware implementation schemes for NQC-LDPC codes. First, we propose an implementation-oriented construction scheme for NQC-LDPC codes to avoid memory-access conflict in the partly parallel decoder. Then, we propose a Modified Overlapped Message-Passing (MOMP) algorithm for the hardware implementation of NQC-LDPC codes. This algorithm doubles the hardware utilization efficiency and supports a higher degree of parallelism than that used in the Overlapped Message Passing (OMP) technique proposed in previous works. We also present single-core and multi-core decoder architectures in the proposed MOMP algorithm to reduce memory cost and improve circuit efficiency. Moreover, we introduce a technique called the cycle bus to further reduce the number of block RAMs in multi-core decoders. Using numerical examples, we show that, for a rate-2/3, length-15360 NQC-LDPC code with 8.43-dB coding gain for Binary Phase-Shift Keying (BPSK) in an Additive White Gaussian Noise (AWGN) channel, the decoder with the proposed scheme achieves a 23.8%–52.6% reduction in logic utilization per Mbps and a 29.0%–90.0% reduction in message-memory bits per Mbps.

Key words: Non-Quasi-Cyclic (NQC); Low-Density Parity-Check (LDPC) codes; decoder design; Modified Overlapped Message Passing (MOMP) algorithm; hardware utilization efficiency

1 Introduction

Low-Density Parity-Check (LDPC) codes, first discovered by Gallager[1] in 1962 and then reintroduced by Mackay and Neal[2] in 1996, are excellent channel codes with near-Shannon-limit error-correcting capability[3]. Quasi-Cyclic (QC) LDPC codes, the most popular class of the LDPC codes, have been widely considered and well implemented. These codes have acceptable hardware implementation complexity[4–6] and have been adopted by many standards, including the DVB-S2, IEEE 802.16e (WiMax), and CCSDS deep-space and near-Earth data communications. However, QC-LDPC codes are not the best coding schemes when approaching channel capacity, because they only provide a solution for making a trade-off between performance and implementation. Therefore, many researchers have turned to the construction of implementation-oriented Non-Quasi-Cyclic (NQC) LDPC codes, of which the efficient code scheme proposed in Refs. [7, 8] is an example.

Although the NQC-LDPC code in Refs. [7, 8] was

• Baihong Lin, Yukui Pei, and Jianhua Lu are with Department of Electronic Engineering, Tsinghua University, Beijing 100084, China. E-mail: linbaihong111@126.com; peiyk@tsinghua.edu.cn.
• Liuguo Yin is with School of Aerospace, Tsinghua University, Beijing 100084, and with the EDA Laboratory, Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China. E-mail: yinlg@tsinghua.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2016-03-11; accepted: 2016-05-23

reported to have low implementation complexity, two challenging problems arise in the high-speed decoder design. First, although the scheme with multiple decoder cores can enhance the decoder's throughput, it also sacrifices logic cells and the number of block RAMs. Second, the traditional partially parallel decoder using the Belief-Propagation (BP) algorithm has low hardware efficiency because of its serially iterative decoding scheme between the Variable-Node updating Processing (VNP) and the Check-Node updating Processing (CNP).

To overcome these problems, researchers have proposed several techniques including the modified Sum-Product Algorithm (SPA) and Overlapped Message Passing (OMP)[9–14]. The modified SPA and OMP algorithms are designed to improve the hardware utility efficiency (HUE). The former combines Variable-Node updating Units (VNUs) and Check-Node updating Units (CNUs) in the same hardware by changing the SPA formation[9,10], while the latter overlaps variable-node updates and check-node updates by taking advantage of the concurrency between these two processes. Obviously, the OMP technique greatly improves the architecture by enhancing the HUE and bringing in nearly double decoder throughput. But unfortunately, these techniques cannot be adopted for the code proposed in Refs. [7, 8], because the code construction method of the latter differs from those for which the techniques were proposed.

In this paper, we focus on a high-speed decoder design of the codes proposed in Refs. [7, 8]. To facilitate the implementation of the high-speed decoder, we first propose a code construction scheme that imposes mathematical constraints on the generated parameters to avoid memory-access conflict in the partly parallel decoder. Then, we propose a Modified Overlapped Message-Passing (MOMP) algorithm that overcomes the shortcoming of the OMP technique by doubling the hardware utilization efficiency and supporting a higher degree of parallelism than does the OMP algorithm. Furthermore, on the basis of the modified algorithm, we present architectures of single-core and multi-core decoders to reduce memory cost and improve circuit efficiency. We also introduce a technique called the cycle bus to further reduce the number of block RAMs in multi-core decoders. Finally, we provide numerical examples to demonstrate the effectiveness of our proposed solutions.

The rest of this paper is organized as follows. In Section 2, we present a class of irregular semi-random LDPC codes as well as a corresponding BP-based partially parallel decoder. Then, on the basis of the NQC semi-random LDPC codes, we propose an implementation-oriented code construction scheme in Section 3. In Section 4, we review the OMP technique and propose a modified algorithm. In Sections 5 and 6, we discuss the architectures of single-core and multi-core decoders, respectively, based on the proposed MOMP algorithm. In Section 7, we provide numerical examples to demonstrate the effectiveness of our proposed solutions. Finally, we draw our conclusions in Section 8.

2 Irregular Semi-random LDPC Codes and the Corresponding Partially Parallel Decoder

2.1 Irregular semi-random LDPC codes

There exists a class of semi-random LDPC codes, as proposed in Refs. [7, 8], the H-matrix of which can be represented as follows:

$$\boldsymbol{H}=\begin{pmatrix} \boldsymbol{J} & & \boldsymbol{\Pi}_{1,M_{\mathrm{b}}+1} & \cdots & \boldsymbol{\Pi}_{1,N_{\mathrm{b}}} \\ & \ddots & \vdots & \ddots & \vdots \\ & \boldsymbol{J} & \boldsymbol{\Pi}_{M_{\mathrm{b}},M_{\mathrm{b}}+1} & \cdots & \boldsymbol{\Pi}_{M_{\mathrm{b}},N_{\mathrm{b}}} \end{pmatrix},$$

wherein $\boldsymbol{J}=\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & \cdots & 1 & 1 \end{pmatrix}$ and $\boldsymbol{\Pi}_{i,j}$ denotes an $L \times L$ permutation matrix or a zero matrix. If we replace the square sub-matrix $\boldsymbol{\Pi}_{i,j}$ by an element, the $M_{\mathrm{b}} \times N_{\mathrm{b}}$ matrix is called a basic matrix. The row of the basic matrix is called the macro row while the column of the basic matrix is called the macro column.

In this paper, let $L = 2^n$, where n is a positive integer. We determine $\boldsymbol{\Pi}_{i,j}$ by a row parameter vector $(\theta_{i,j}, \tau_{i,j})$ as follows:

$$\mathrm{col} = (\theta_{i,j} + \tau_{i,j} \times \mathrm{row}) \bmod L \tag{1}$$

wherein (col, row) denotes that the none-zero element ranks at the row-th row and col-th column and $0 \leqslant x, y, \theta_{i,j}, \tau_{i,j} < L,\ 0 \leqslant i < M_{\mathrm{b}}, 0 \leqslant j < N_{\mathrm{b}}, \tau_{i,j} \bmod 2 = 1$. Equation (1) is also equivalent to the following:

$$\begin{cases} \text{row} = (\theta'_{i,j} + \tau'_{i,j} \times \text{col}) \bmod L, \\ (\theta_{i,j} + \tau_{i,j} \times \theta'_{i,j}) \bmod L = 0, \\ (\tau'_{i,j} \times \tau_{i,j}) \bmod L = 1, \\ 0 \leqslant \theta'_{i,j}, \tau'_{i,j}, \theta_{i,j}, \tau_{i,j} < L \end{cases} \tag{2}$$

wherein $(\theta'_{i,j}, \tau'_{i,j})$ denotes another parameter vector, called a column parameter vector.

Obviously, this is a class of QC-LDPC codes for $\tau'_{i,j} = 1$. But in this paper, the $\tau'_{i,j}$ parameters of the two sub-matrices are random and different, which make a class of NQC semi-random LDPC codes instead.

2.2 Partially parallel decoder

Figure 1 shows the partially parallel decoder of NQC semi-random LDPC codes. Messages transferred to the edge of the Tanner graph are stored in region A. We use the N_b VNUs in region C and the M_b CNUs in region B to calculate variable-to-check and check-to-variable messages, respectively. (Variable-to-check and check-to-variable messages are known as intra-messages). Their respective operations are called column and row operations.

To facilitate the implementation, matrix $\boldsymbol{J}$ is decomposed as follows:

$$\boldsymbol{J} = \boldsymbol{I} + \boldsymbol{I}',$$

wherein $\boldsymbol{I}' = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & \cdots & 1 & 0 \end{pmatrix}$ and $\boldsymbol{I}$ is an identity matrix. Hence, sub-matrix messages can be packaged into a RAM.

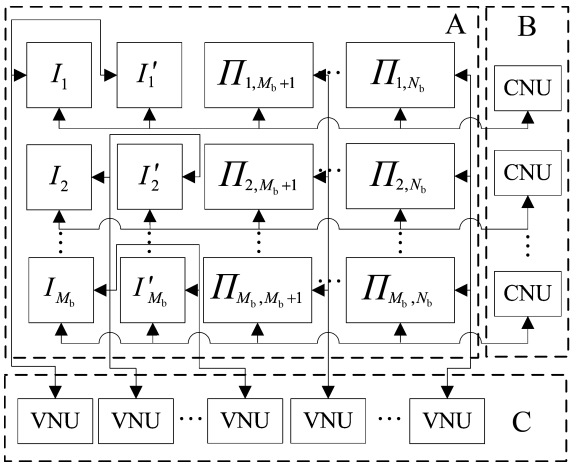

Fig. 1 The partially parallel decoder of the NQC semi-random LDPC codes[15].

When the BP-based decoder is working, the timing diagrams involve two alternative processes—variable-node and check-node updating processes. As noted above, these diagrams are inefficient because of their separate VNP and CNP[11, 12]. Furthermore, too many block RAMs are requested by the multi-core decoder. Many studies have been conducted to develop ways to offset the shortcomings of this architecture[11, 12, 15, 16].

3 Implementation-Oriented Code Construction Scheme

3.1 Code construction method

Our implementation-oriented code construction scheme is as follows:

(1) Construct a basic matrix as follows:

$$\boldsymbol{H}_\text{b} = \begin{pmatrix} 1 & & \Pi_{1,M_\text{b}+1} & \cdots & \Pi_{1,N_\text{b}} \\ & \ddots & \vdots & \cdots & \vdots \\ & 1 & \Pi_{M_\text{b},M_\text{b}+1} & \cdots & \Pi_{M_\text{b},N_\text{b}} \end{pmatrix},$$

wherein $\Pi_{i,j}$ denotes zero or one. The weight of each row is $\rho - 1$ as the same.

(2) Generate a random sequence as follows:

$$\{Y_i \mid 0 \leqslant i < M_\text{b}, Y_i = 2n + 1, \\ n \text{ is a nonnegative integer}, 0 \leqslant Y_i < \mu\},$$

wherein $\mu = 2^m$, $m = \lceil \log_2 \rho \rceil$. As shown, the sequence above contains M_b elements.

(3) Generate a sequence for the i-th row as follows:

$$\{\beta_{i,k} \mid \beta_{i,k} \text{ is a nonnegative integer}, \\ 0 \leqslant k < \rho, 0 \leqslant \beta_{i,k} < \mu, \beta_{i,1} = (\beta_{i,0} + 1) \bmod \mu\}.$$

The above sequence contains ρ elements that differ from each other.

(4) Generate the column-parameter-vectors-allowed set Ω_{i,j_k} for each none-zero sub-matrix $\boldsymbol{\Pi}_{i,j_k}$ by the following equations:

$$\Omega_{i,j_k} = \{(\theta'_{i,j_k}, \tau'_{i,j_k}) \mid \beta_{i,k+2} = \theta'_{i,j_k} \bmod \mu, \\ Y_i = \tau'_{i,j_k} \bmod \mu,\ k = 0, 1, \ldots, \rho - 3, \\ i = 0, 1, 2, \ldots, M_\text{b} - 1\}.$$

(5) Search the column parameter vectors in the allowed set Ω_{i,j_k} to expand the basic matrix so that the parity matrix has no girth-4. The expanding method is as follows:

(a) Expand the non-zero element on the i-th ($0 \leqslant i < M_\text{b}$) column of the basic matrix as the matrix $\boldsymbol{J}$.

(b) Expand the non-zero element on the other column as the matrix $\boldsymbol{\Pi}_{i,j}$, which is generated by the column parameter vector in the allowed set.

3.2 Property of the constructed codes

Before discussing the property of the constructed codes, we give several new mathematical concepts for the matrices as follows.

Definition 1 Let $\boldsymbol{B}$ to be a binary square matrix with size of L, in which there are ρ_i none-zero elements on the i-th row whose column positions are labeled as $c_{i,j}, 0 \leqslant j < \rho_i, 0 \leqslant c_{i,j} < L$.

If $\exists\ \rho \geqslant \max_i \rho_i$, $\forall i, 0 \leqslant i < L$, elements in the i-th sequence $\{\gamma_{i,j} \mid \gamma_{i,j} = c_{i,j} \bmod \rho,\ 0 \leqslant j < \rho_i\}$ are different from each other, then $\boldsymbol{B}$ is defined as a standard ρ-block-divided matrix and $\gamma_{i,j}$ are its divided factors.

Definition 2 Matrix $\boldsymbol{X} = (x_{i,j})_{L\times L}$ and matrix $\boldsymbol{Y} = (y_{i,j})_{L\times L}$ are two binary matrices. "$\vee$" denotes an operation defined as: $\boldsymbol{Z} = \boldsymbol{X} \vee \boldsymbol{Y} = (\max(x_{i,j}, y_{i,j}))_{L\times L}$.

Definition 3 There are ρ none-zero binary square sub-matrices in a macro row of the parity-check matrix, which are denoted as $\boldsymbol{X}_{i,j_k}, 0 \leqslant k < \rho, 0 \leqslant j_k < N_{\mathrm{b}}$. If there exist permutation matrices $\boldsymbol{P}_{j_0}, \boldsymbol{P}_{j_1}, \ldots, \boldsymbol{P}_{j_{\rho-1}}$ such that

$$\boldsymbol{A} = (\boldsymbol{X}_{i,j_0}\boldsymbol{P}_{j_0}) \vee (\boldsymbol{X}_{i,j_1}\boldsymbol{P}_{j_1}) \vee \cdots \vee (\boldsymbol{X}_{i,j_{\rho-1}}\boldsymbol{P}_{j_{\rho-1}}),$$

wherein $\boldsymbol{A}^{\mathrm{T}}$ is a standard μ-block-divided matrix ($\mu \geqslant \rho$), the macro row is defined as a μ-block-divided macro row $\boldsymbol{P}_{j_k}$ $(0 \leqslant k < \rho)$ are defined as the Row-divided Permutation (RP) matrices.

On the basis of the concepts above, we can easily find that each macro row of the constructed parity-check matrix is μ-block-divided. The proof is given in the following paragraph.

Assume that $\boldsymbol{J}$, $\boldsymbol{\Pi}_{i,j_1}$, $\boldsymbol{\Pi}_{i,j_2}$, ..., $\boldsymbol{\Pi}_{i,j_{\rho-2}}$ are the none-zero sub-matrices in the i-th macro row. Let $\boldsymbol{B}_i$ denote an $L \times L$ binary permutation matrix generated by the row parameter vector $(\beta_{i,0}, Y_i)$ and $\boldsymbol{P}_i = \boldsymbol{J}\boldsymbol{B}_i$. Obviously, $\boldsymbol{P}_i = \boldsymbol{B}_i \vee (\boldsymbol{I}'\boldsymbol{B}_i)$. $\boldsymbol{I}'\boldsymbol{B}_i$ is generated by the column parameter vector $(\beta_{i,1}, Y_i)$. We denote the column parameter vector of $\boldsymbol{\Pi}_{i,j_k}$ by $(\theta'_{i,j_k}, \tau'_{i,j_k})$, wherein $\theta'_{i,j_k} \bmod \mu = \beta_{i,k+2}$ and $\tau'_{i,j_k} \bmod \mu = Y_i$. $(\mathrm{row}_{(i,j_k)}, \mathrm{col}_{(i,j_k)})$ denotes that a none-zero element in the $\boldsymbol{\Pi}_{i,j_k}$ ranks at the $\mathrm{row}_{(i,j_k)}$-th row and $\mathrm{col}_{(i,j_k)}$-th column. μ is a factor of L, thus

$$\begin{cases} \gamma_{i,j_{k+2}} = \mathrm{row}_{(i,j_k)} \bmod \mu = \\ (\theta'_{i,j_k} + \tau'_{i,j_k} \times \mathrm{col}_{(i,j_k)}) \bmod L \bmod \mu = \\ (\beta_{i,j_{k+2}} + Y_i \times (\mathrm{col}_{(i,j_k)} \bmod \mu)) \bmod \mu, \\ \gamma_{i,j_0} = (\beta_{i,j_0} + Y_i \times \mathrm{col}_{(i,j_0)}) \bmod \mu, \\ \gamma_{i,j_1} = (\beta_{i,j_1} + Y_i \times \mathrm{col}_{(i,j_0)}) \bmod \mu, \\ 0 \leqslant k < \rho - 2 \end{cases} \tag{3}$$

Elements in the sequence $\{\beta_{i,j_k}, 0 \leqslant k < \rho\}$ differ from each other, so do elements in $\{\gamma_{i,j_k}, 0 \leqslant k < \rho\}$ when $\mathrm{col}_{(i,j_k)}$ $(0 \leqslant k < \rho - 2)$ of different matrices are as the same. Thus, the i-th macro row is μ-block-divided and its corresponding RP matrices are $\boldsymbol{B}_i, \boldsymbol{I}_{i,j_1}$, ..., $\boldsymbol{I}_{i,j_{\rho-2}}$.

Using the algorithm proposed in the next section, we can use the property of constructed codes to reduce the number of memory blocks in the decoder design.

4 OMP Technique and the Modified OMP Algorithm

4.1 Overlapped message passing schedule

To adopt the OMP algorithm for NQC semi-random LDPC codes, let us first review this technique. As is well known, the BP algorithm consists of two decoding processes: the VNP and the CNP. In general, these two processes may not overlap because they offer updated data to each other. However, studies have found that the effect of this data dependency could be reduced if the row and column operations followed proper sequences[11–13]. If these operation sequences are taken as a kind of matrix permutation, the schedule finds a permutation that could transform the square sub-matrix into an H-matrix or the H-matrix into a standard matrix (See Fig. 2), in which the bottom-left and top-right corners are the zero regions. Moreover, the VNP and CNP could be completely overlapped if the H-matrix is reconstructed in a specific mathematical pattern[11].

However, these proposed methods are meaningless

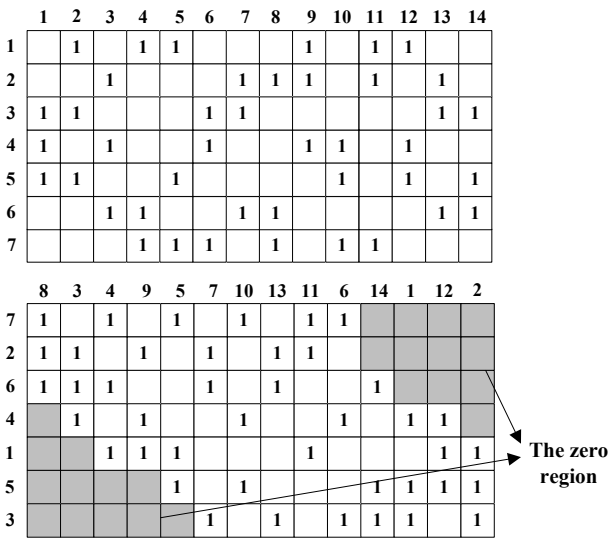

Fig. 2 OMP technique for a parity-check matrix H: matrix H (top) and permuted standard matrix (bottom).

when applied to NQC semi-random LDPC codes. As the $\tau'_{i,j}$ parameters are random in each sub-matrix, there are several problems, as follows:

- First, a compatible permutation may be hard to describe mathematically, so ROM is needed to store the permutation for implementation, which would increase hardware complexity.
- Second, even if a permutation matrix could be found, the OMP technique may not result in any significant hardware improvement because the zero region may be too small for the VNP and CNP to overlap to any great extent.

4.2 Modified overlapped message passing algorithm

To solve the above problems, we propose a modified OMP algorithm. We suppose that old message data will be used for decoding when the VNP and CNP overlap. C_j denotes the Log-Likelihood Ratio (LLR) channel information of the j-th variable node. $N(v_k)$ denotes a set of check nodes connected to the k-th variable nodes, while $N(c_k)$ denotes a set of variable nodes connected to the k-th check nodes. P denotes a parallelism parameter, and $P = 2^m$, $m \in \mathbf{N}$, $P < L$. $R_{c\to v}$ denotes the check-to-variable message whereas $Q_{v\to c}$ denotes the variable-to-check message. The modified OMP algorithm is shown in Algorithm 1.

Algorithm 1 Modified OMP Algorithm

1: **Initialize** all $R^{\text{old}}_{c\to v} = 0$, $Q^{\text{old}}_{v_j\to c_i} = C_j$;
2: **for** $i = 0$ to $L/P - 1$ **do**
3: **for** $j = 0$ to $N_{\text{b}} - 1$ **do**
4: **for** $h = 0$ to $P - 1$ **do**
5: $k = j \times L + i \times P + h$;
6: **for** $c_a \in N(v_k)$ **do**
7: **compute** $Q^{\text{new}}_{v_k\to c_a}$ with $R^{\text{old}}_{c\to v_k}$;
8: **end for**
9: **end for**
10: **end for**
11: **if** it's not the first iteration **or** $i \geqslant L/(2P)$ **then**
12: **for** $j = 0$ to $M_{\text{b}} - 1$ **do**
13: **for** $h = 0$ to $P - 1$ **do**
14: $k = j \times L + (i \times P + h + L/2) \bmod L$;
15: **for** $v_{\text{b}} \in N(c_k)$ **do**
16: **compute** $R^{\text{new}}_{c_k\to v_{\text{b}}}$ with $Q^{\text{old}}_{v\to c_k}$;
17: $R^{\text{old}}_{c_k\to v_{\text{b}}} = R^{\text{new}}_{c_k\to v_{\text{b}}}$;
18: **end for**
19: **end for**
20: **end for**
21: **end if**
22: **for** $j = 0$ to $N_{\text{b}} - 1$ **do**
23: **for** $h = 0$ to $P - 1$ **do**
24: $k = j \times L + i \times P + h$;
25: **for** $c_a \in N(v_k)$ **do**
26: $Q^{\text{old}}_{v_k\to c_a} = Q^{\text{new}}_{v_k\to c_a}$;
27: **end for**
28: **end for**
29: **end for**
30: **end for**
31: **if** Stopping rule is not satisfied **then**
32: Position=2;
33: **end if**

In this algorithm, Steps 7 and 16 compute $Q^{\text{new}}_{v_k\to c_a}$ and $R^{\text{new}}_{c_k\to v_b}$ with the same formulation as the offset min-sum algorithm[17] or the BP algorithm[4], but with a different message-passing schedule. The message passing of each square sub-matrix can be divided into three regions (see Fig. 3). In the first region, the message passing is the same as that of the BP algorithm. In the second region, check-to-variable messages updated in the n-th iteration are used to calculate the variable-to-check messages in the n-th iteration, whereas variable-to-check messages updated in the $(n-1)$-th iteration are used to calculate check-to-variable messages in the n-th iteration. In the third region, variable-to-check messages in the n-th iteration are calculated using the check-to-variable messages updated in the $(n-2)$-th iteration, whereas check-to-variable messages in the n-th iteration are calculated using the variable-to-check messages updated in the $(n+1)$-th iteration.

Compared to the OMP technique, the modified OMP algorithm completely overlaps the VNP and CNP without introducing any constraints on code construction. Furthermore, the parameter P can adjust the parallelism of the single-core decoder architecture. However, these message-passing tasks have different degrees of efficiency in their corresponding Tanner graphs. Moreover, when the variable-to-check messages are updated, old variable-to-check messages are used to calculate the check-to-variable messages, which implies that the variable-to-check and check-to-variable messages must be stored separately in simple two-port RAMs. In contrast to the BP vector partially parallel decoder, double memories are needed for intra-message storage.

5 Single-Core Decoder Architecture Implementation Analysis

In this section, we analyze the implementation of the single-core decoder architecture for the modified OMP algorithm. The first question is what is the single-core

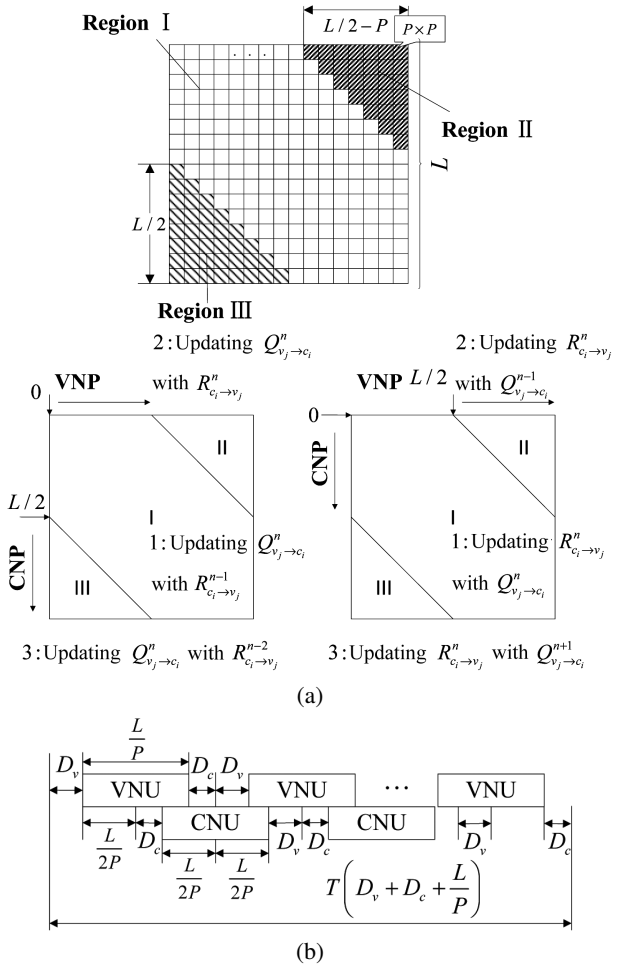

Fig. 3 The message passing process of modified OMP algorithm and its corresponding timing diagram: (a) The message passing process of modified OMP algorithm (top) and (b) the timing diagram of the single-core modified OMP decoder architecture (bottom).

decoder architecture? If the message memory capacity is designed specifically for a Tanner graph of LDPC codes, the corresponding decoder architecture is called a single-core decoder architecture. Otherwise, it is called a multi-core decoder architecture.

For the modified OMP algorithm, we implement the single-core decoder architecture based on the partially parallel decoder and the timing diagram of the modified OMP decoder architecture shown in Fig. 3b. In the timing diagram, T denotes the iteration of the algorithm. D_c and D_v denote the delay of the check-node updating process and the delay of the variable-node updating process, respectively. They have no effect on the message-passing algorithm because of the symmetry in the diagram. In contrast to the parameters in the timing diagram proposed in Ref. [11], D_c and D_v are determined by the message storage designs the VNU, and the CNU, which have no relation to the expanding parameter and are instead related to the degree of check nodes and variable nodes.

This implementation process is influenced by parameter P. In the following analysis, we discuss the architectures of $P = 1$ and $P > 1$.

5.1 Low-memory-cost decoder architecture when *P*=1

In Section 4, we explained that the modified OMP algorithm requires double memories to store intra-messages, in contrast to the vector BP decoder[16]. However, we know that in the offset min-sum algorithm, check-to-variable messages from a check node can be replaced by sign bits and three numbers, including the minimum number of variable-to-check messages, the second minimum number of variable-to-check messages, and the position of the minimum[6, 17]. These three numbers comprise a row vector. To simplify the check node updating process, if we adopt the offset min-sum algorithm in the modified OMP algorithm, it is possible to reduce the memory required for the intra-messages.

However, another problem remains. How can we design RAMs for row vectors without reducing the throughput of the decoder? When $P = 1$, the N_b column operation is carried out at the same time in a clock cycle. During this time, ρ check-to-variable messages are read from each macro row. For the LDPC codes proposed in Section 3, the i-th macro row of the given parity-check matrix is p-block-divided and its corresponding RP matrices are $\boldsymbol{B}_i, \boldsymbol{I}_{i,j_1}, \ldots, \boldsymbol{I}_{i,j_{\rho-2}}$. If column operations in the i-th ($0 \leqslant i < M_b$) macro column are carried out in the sequence $\{p_l, b_{k,p_l} = 1, \boldsymbol{B}_i = (b_{i,j})_{L\times L}, 0 \leqslant k < L\}$ and column operations in the other macro columns are executed in the same sequence $\{0, 1, ..., L-1\}$, the ρ check-to-variable messages will come from different rows in a macro row and their corresponding row labels will be different modulo μ. Hence, in general, row vectors in a macro row could be stored averagely in μ simple two-port RAMs with the size $L/\mu \times Q_\Lambda$ (L/μ is the RAM depth and Q_Λ is the width of the data bus). Q_Λ denotes the number of quantization bits for a row vector.

On the above basis, the row message unit is designed to store row vectors in the i-th macro row (see Fig. 4). In the structure, a row vector of the j-th row in the macro row output from the CNU will be stored in the (j mod μ)-th simple two-port RAM. Thus, a

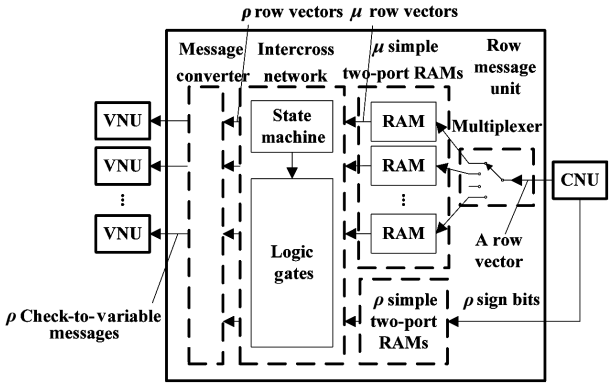

Fig. 4 The structure of row message unit.

multiplexer is needed to choose which RAM will be written. ρ sign bits from a CNU are stored in ρ simple $L \times 1$ two-port RAMs, and will be read at the same time as the row vectors. In the intercross network, there are μ kinds of state for the state machine, which is denoted as col mod μ. With the state machine, the k-th output port transfers row vectors from the γ_{i,j_k}-th RAM, $0 \leqslant k < \rho$. Finally, the message converter transforms row vectors and sign bits into check-to-variable messages, based on the position of the minimum. (If the current port is at the minimum position, the second minimum will be output. Otherwise the minimum will be output.)

Suppose that the number of quantization bits is the same (Q) for check-to-variable and variable-to-check messages. Then, the number of quantization bits for a row vector is $2Q-1+\log_2\mu$ ($\log_2\mu$ bits are for the minimum position). Compared to the vector BP decoder[18], (in the vector BP decoder, check-to-variable and variable-to-check messages share the same memory unit, thus the corresponding memory totals $M_{\mathrm{b}}\rho QL$, which is just half of the total storage space required for check-to-variable and variable-to-check messages.), the memory for intra-messages, after being reduced, totals:

$$\eta_{\mathrm{m}} = \frac{\mathrm{Memory}_{\mathrm{now}}}{\mathrm{Memory}_{\mathrm{BP}}} = \frac{M_{\mathrm{b}}\rho QL + M_{\mathrm{b}}L(2Q-1+\log_2\mu) + M_{\mathrm{b}}\rho L}{M_{\mathrm{b}}\rho QL} \approx 1+\frac{2}{\rho}+\frac{1}{Q} \tag{4}$$

This memory is about $1+\frac{2}{\rho}+\frac{1}{Q}$ times more than that of the vector BP decoder. With respect to throughput, to simplify the control logic, we let the initialization of the VNP and CNP take the same clock cycle as each iteration. In BP architecture, each iteration cycle totals $2L+D_v+D_c$ and in our proposed architecture it totals $L+D_v+D_c$. We assume that the iterations of the BP algorithm equal those of our proposed algorithm. Then, the total throughput is as follows:

$$\eta_{\mathrm{t}} = \frac{\mathrm{Throughput}_{\mathrm{now}}}{\mathrm{Throughput}_{\mathrm{BP}}} = \frac{(2L+D_v+D_c)T}{(L+D_v+D_c)T} = 1+\frac{L}{L+D_v+D_c} \tag{5}$$

When $L \gg D_v+D_c$, typically $\eta_{\mathrm{t}} \approx 2$ and $\eta_{\mathrm{t}} > \eta_{\mathrm{m}}$, which indicates that the throughput has doubled with only a slight increase in memory cost. Thus, compared to the vector BP decoder, our proposed low-memory-cost decoder architecture has a higher throughput memory ratio in the same iteration conditions.

5.2 High parallel decoder architecture ($P > 1$)

To further enhance the decoder throughput, we propose a high-parallel decoder architecture based on the modified OMP algorithm.

Obviously, $N_{\mathrm{b}}P$ VNUs and $M_{\mathrm{b}}P$ CNUs are needed to carry out the simultaneous $N_{\mathrm{b}}P$ column and $M_{\mathrm{b}}P$ row operations. A memory schedule in the low-memory-cost architecture is irrelevant in this circumstance. As such, a new memory distribution must be considered.

Let variable-to-check messages of a sub-matrix be stored generally in P simple two-port RAMs with the size $L/P \times Q$. These RAM groups are called V2C RAMs and the number of quantization bits (Q) for check-to-variable and variable-to-check messages are the same. The variable-to-check message in the i-th column of the sub-matrix is stored at the (i div P)-th address of the (i mod P)-th RAM. Similarly, check-to-variable messages are stored in C2V RAMs. However, the check-to-variable message on the j-th row of the sub-matrix is stored at the (j div P)-th address of the (j mod P)-th RAM. This memory distribution does not cause memory-access conflicts. We present a detailed analysis of this issue in the following paragraphs.

In the VNP, P column operations are carried out in the j-th macro column and their corresponding columns can be labeled with the addresses $\{nP, nP+1, nP+2, ..., nP+P-1\}$, $n \in \mathbf{N}$, $0 \leqslant n < L/P-1$. We assume that one of the non-zero sub-matrices in the macro column is on the i-th macro row. Thus, P check-to-variable messages are read from the i-th macro row. Let $(\theta'_{i,j}, \tau'_{i,j})$ denote the column parameter vector for the non-zero sub-matrix and R_k, $k = 0, 1, ..., P-1$,

denote the label of the RAM from which the P check-to-variable messages are read. P is a factor of L. Then

$$R_k = (\theta'_{i,j} + \tau'_{i,j} \times (nP + k)) \bmod L \bmod P = (\theta'_{i,j} + \tau'_{i,j} \times k) \bmod P \quad (6)$$

As is evident, $R_x \neq R_y$ if $x \neq y$. R_k has no relation to n. Thus, P check-to-variable messages are certainly from different RAMs and their corresponding RAMs do not change with n. The situation is similar for the variable-to-check messages.

From the above proof, we also find that $2M_b\rho P$ simple two-port RAMs are required for the check-to-variable and variable-to-check messages. Although total memory is not increased with P, the number of block RAMs, VNUs, and CNUs are P times more than that of the $P = 1$ architecture and the throughput totals are as follows:

$$\eta'_t = \frac{(L + D_v + D_c)}{(L/P + D_v + D_c)} = P - \frac{P(P-1)(D_v + D_c)}{L + P(D_c + D_v)} \quad (7)$$

Equation (7) shows that the throughput increases more slowly than P, but the VNU and CNU costs increase proportionally with P. Thus, when P increases, the VNU and CNU costs do not bring a correspondingly equal increase in the throughput. This means that the logic utility efficiency increases with P at first but then reduces when P becomes large.

6 Implementation Analysis of the Multi-core Decoder Architecture

The single-core decoder architecture described in Section 5 above serves as a basis for designing a high-speed decoder. However, we must keep in mind that this architecture has low logic-utility efficiency when P is large, despite the fact that memory is fully used. Furthermore, in some case such as FPGA implementation, there are not enough small block RAMs for the single-core decoder architecture. To solve these two problems, we propose a multi-core decoder architecture, which is classified into one of two categories, non-cooperative decoding architecture or cooperative decoding architecture, depending on whether the architecture adopts a cycle bus. In the following, we describe these two architectures in detail.

6.1 Non-cooperative decoding architecture

Figure 5 shows a non-cooperative decoding architecture, in which the received data streams are divided into M equal-sized chunks of code words

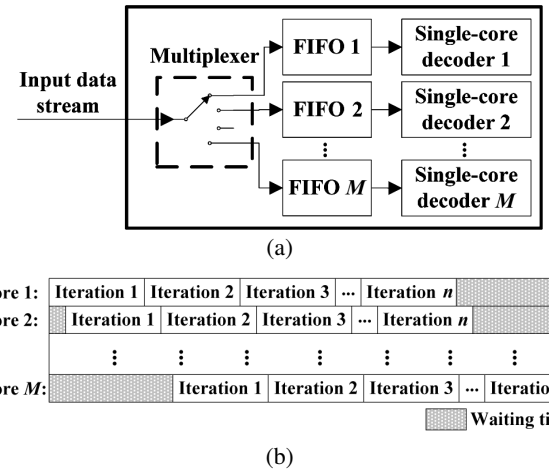

Fig. 5 Non-cooperative decoding architecture implementation: (a) Non-cooperative decoding architecture (top) and (b) Timing diagrams (bottom).

by the multiplexer. Each chunk is independently decoded by its corresponding single-core decoder. As such, the timing diagrams of the decoder cores are non-cooperative. In general, the rate of input data flow is faster than the throughput of the decoder core. Thus, a FIFO is needed to buffer the code words.

From the structure in Fig. 5, it is clear that this architecture is simple and that the single-core decoders do not influence each other, which improves the reliability of the implementation. However, the source cost is M times greater than that of the single-core architecture, despite the increase in throughput M times. The hardware efficiency remains the same as the increase in M. Although this is a popular engineering solution, it is not applicable in some FPGA implementation cases where block RAM resources are scarce.

6.2 Cycle bus and cooperative decoding architecture

To reduce the block RAMs, we introduce a cycle bus to the multi-core architecture. As shown in Fig. 6a, the data bus through the VNU groups, the message storage groups, and the CNU groups are called the cycle bus. In this architecture, the LLR storage groups, VNU groups, and CNU groups are simple units of the LLR storages, the VNUs, and the CNUs in different single-core decoders, respectively. However, the situation is different for message storage groups, in which the cycle bus combines RAMs from the same submatrix in different single-core decoders by enlarging the width of the RAM data port. For example, RAMs of size $L \times Q$

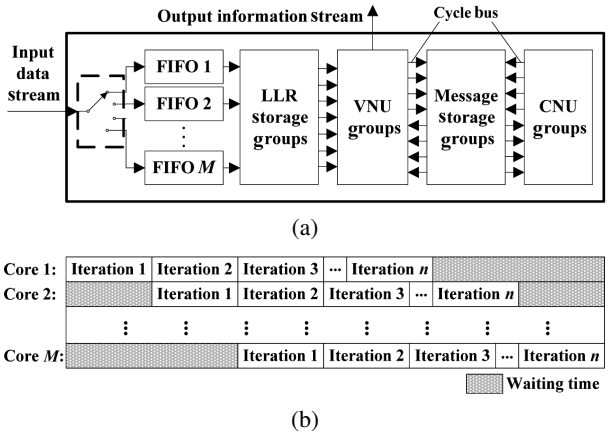

Fig. 6 Cooperative decoding architecture implementation: (a) Cooperative decoding architecture (top) and (b) Timing diagrams (bottom).

in M single-core decoders are combined with a RAM of size $L \times MQ$.

After this combination, the M message storage groups share an address-generating logic set for the parity-check matrix. Thus, different single-core decoders must operate in the same column and row in a submatrix when decoding, and the timing diagram becomes cooperative, as illustrated in the diagram in Fig. 6b. Even when a code word is prepared for a single-core decoder, it must wait to be decoded until the beginning of the next iteration. In the worst case, it will have to wait for a complete iteration cycle. So, a FIFO is required for each single-core decoder to buffer received code words during waiting periods.

This architecture greatly reduces the number of block RAMs. M single-core decoders feature the same block RAMs for message storage as those of single-core decoders. Furthermore, the logic cost is reduced by sharing address generators. On the other hand, the combined message storage increases the design difficulty and memory utility efficiency is not enhanced with the increase of M.

7 Numerical Examples

In this section, we give numerical examples to demonstrate the validity of our proposed methods.

First, we constructed a rate-2/3, length-15360 LDPC code based on the construction scheme described in Section 3. Its basic matrix has a size of 10×30, in which the weight of each row is 11 and the weight of each column ranges from 2 to 5. We generated the parity matrix by replacing each element in the basic matrix with a 512×512 square matrix. For the purposes of comparison, we also constructed the LDPC code proposed in Ref. [8] with a similar basic matrix. We refer to this second code as the original code.

Next, we simulated a performance evaluation with the modified OMP algorithm, and the BP algorithm over an Additive White Gaussian Noise (AWGN) channel with Binary Phase-Shift Keying (BPSK) modulations. We examined 1 000 000 code words in each simulation and set the number of decoding iterations to 19. The P values were 1, 2, 4, 8, and 16, respectively. There were six quantization bits for both the check-to-variable and variable-to-check messages. We designated 15 bits for a quantified row vector, in which five bits were for the minimum, five were for the second minimum, one was for the sum of all the input sign bits, and four were for the minimum position. Figure 7 shows the Bit-Error-Ratio (BER) simulation results, from which we find that our proposed code performs the same as the original code, which indicates an 8.43 dB coding gain for the BPSK in the AWGN channel compared to un-encoded circumstance. However, as shown above, our code is more beneficial for hardware implementation. In addition, E_b/N_0 increases with P under the same BER conditions, whereas P ranges from 1 to 16 and E_b/N_0 ranges from 2.33 dB to 2.4 dB at the output BER of 1×10^{-7}. We attribute this performance reduction to the expanding region of low-efficiency massage passing. Nevertheless, the MOMP algorithm sustained roughly a 0.2 dB performance reduction compared to the unquantified BP algorithm and the 0.07 dB reduction due to P is acceptable, considering its associated increases in throughput and hardware efficiency.

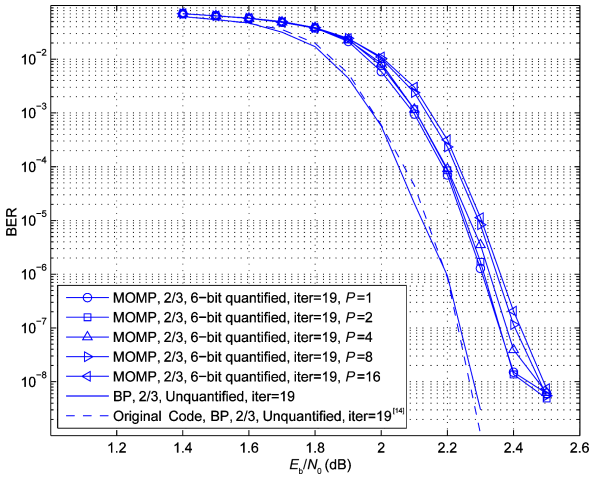

Fig. 7 Simulation result for the constructed code.

Third, we designed the single-core and multi-core decoders based on the modified OMP algorithm, which apply the above finite precision solution. To compare the different architectures, we set P and M to take different values. We synthesized the implementations on an Altera 5SGXMA7N2F40C2 FPGA, the results of which are shown in Tables 1 and 2.

We can identify five features from Tables 1 and 2, which can be summarized as follows:

(1) Compared to the traditional architecture in Ref. [8], our proposed architectures feature a 23.8%–52.6% reduction in logic utilization per Mbps. Furthermore, the reduction in the memory throughput ratio in single-core architectures ranges from 29.0% to 90.0%, which indicates better message memory utilization.

(2) In single-core architectures, the number of message memory bits of $P = 1$ is 30.3% lower than that of $P > 1$. Moreover, compared to the traditional architecture, the message-memory throughput ratio of $P = 1$ shows a 29.0% reduction.

(3) If we let ξ denote the ratio of the total message memory bits (Δ) and the throughput (π), the following equation shows that ξ decreases with P and is not related to M:

$$\xi(P) = \frac{\Delta}{\pi} = \frac{\Delta}{N_{\mathrm{b}} L F_{\mathrm{clock}}} \left(\frac{512}{P} + 21 \right) T = \alpha \left(\frac{512}{P} + 21 \right) \tag{8}$$

wherein $\alpha = \dfrac{\Delta}{N_{\mathrm{b}} L F_{\mathrm{clock}}} T$. For example, when $P = 1$, $\alpha = 5.83$ and when $P > 1$, $\alpha = 8.36$.

(4) For the same maximal throughput, the logic utilization per Mbps of the cooperative multicore architecture is lower than that of the non-cooperative architecture, and the reduction ranges from 8.9% to 19.9%. The logic utilization per Mbps of the cooperative multi-core architecture is also lower than that of the single-core architecture with the same parameter P, and the corresponding reduction ranges from 5.5% to 16.1%.

(5) In the single-core architecture, the connection between the number of message memory RAMs (ε) and P can be expressed as $\varepsilon = 220P$. In the multi-core non-cooperative architecture, $\varepsilon = 220MP$, whereas in multi-core cooperative architecture, $\varepsilon = 220P$.

The first of the five above features demonstrates the overall benefits of our proposed algorithm and architectures. In the second, we compare the $P = 1$ single-core architecture with traditional architectures. We attribute the relative reduction in the message-memory bits and message-memory throughput ratio to the memory storage schedule proposed in Section 5. The third feature highlights the fact that an increase in P can enhance memory utility efficiency. As $\xi'(P) = -512\alpha/P^2$, $\xi'(P)$ reduces correspondingly with the increase in P. Thus, message memory efficiency increases more slowly with P when P is large. The fourth and fifth features prove the advantage of the cycle bus technique. The former shows that cooperative architecture has higher logic utility efficiency, while the latter confirms that the number of message-memory

Table 1 FPGA implementation results for single-core architectures (Clock: 100 MHz, Iteration: 19, M=1).

Architecture	P	Maximal throughput (Mbps)	Logic utilization (in ALMs)	Total registers	Memory bits for message storage	Logic utilization per Mbps	Message memory bits per Mbps	Number of message memory RAMs
Traditional	—	77.36	9463	22 128	337 920	122.32	4368.15	110
Single-core	1	151.96	14 105	22 451	471 040	92.82	3099.76	270
	2	292.91	22 736	40 147	675 840	77.62	2307.33	440
	4	546.23	35 662	72 181	675 840	65.29	1237.28	880
	8	962.41	59 061	131 395	675 840	61.37	702.24	1760
	16	1554.66	106 475	236 374	675 840	68.49	434.72	3520

Table 2 FPGA implementation results for multi-core architectures (Clock: 100 MHz, Iteration: 19).

Architecture	P	M	Maximal throughput (Mbps)	Logic utilization (in ALMs)	Total registers	Memory bits for message storage	Logic utilization per Mbps	Message memory bits per Mbps	Number of message memory RAMs
Cooperative	1	2	303.92	23 666	39 037	1 351 680	77.87	4447.49	270
	4	2	1092.46	65 236	131 372	1 351 680	59.71	1237.28	880
	2	4	1171.64	77 549	132 585	2 703 360	66.19	2307.33	440
	8	2	1924.82	111 612	247 632	1 351 680	57.99	702.24	1760
Non-cooperative	1	2	303.92	28 325	44 823	1 351 680	93.20	4447.49	540
	4	2	1092.46	71 615	144 271	1 351 680	65.55	1237.28	1760
	2	4	1171.64	91 649	159 752	2 703 360	78.22	2307.33	1760
	8	2	1924.82	139 297	254 676	1 351 680	72.37	702.24	3520

block RAMs in the multi-core architecture can be reduced by the application of our proposed technique and its minimum is equal to that of single-core architecture with the same P. In general, P and M should be taken into account when designing decoder architecture, based on the fact that an FPGA and a cooperative architecture are superior to a non-cooperative architecture despite the complexity of its design.

8 Conclusion

In this paper, we presented efficient hardware implementation schemes for NQC LDPC codes. First, we proposed an implementation-oriented construction scheme for NQC-LDPC codes to avoid memory-access conflict in the partly parallel decoder. Then, we proposed an overlapped message-passing algorithm, which can double the throughput and enhance the hardware utility efficiency. On the basis of this algorithm, we proposed a single-core architecture with configurable parallelism and a multi-core architecture. We also introduced a technique, called the cycle bus, to reduce the number of block RAMs in the multi-core architecture, based on the classification of the multi-core architecture as either cooperative or non-cooperative. We used numerical examples to show that, for a rate-2/3, length-15360 NQC-LDPC code with an 8.43 dB coding gain for BPSK in an AWGN channel, the decoder with the proposed scheme achieves a 23.8%–52.6% reduction in logic utilization per Mbps and a 29.0%–90.0% reduction in message-memory bits per Mbps.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (Nos. 61101072 and 61132002), the new strategic industries development projects of Shenzhen city (No. ZDSY20120616141333842), and Tsinghua University Initiative Scientific Research Program (No. 2012Z10132).

References

[1] R. G. Gallager, Low-density parity-check codes, *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[2] D. J. C. MacKay and R. M. Neal, Near shannon limit performance of low density parity check codes, *Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[3] C. Sandu, I. Florescu, and C. Rotaru, Software simulation of ldpc codes and performance analysis, in *Proc. 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Beijing, China, 2014, pp. 162–165.

[4] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, Factor graphs and the sum-product algorithm, *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[5] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, Design of capacity-approaching irregular low-density parity-check codes, *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[6] Z. Wang, Z. Cui, and J. Sha, VLSI design for low-density parity-check code decoding, *Circuits and Systems Magazine*, vol. 11, no. 1, pp. 52–69, 2011.

[7] P. Li, W. K. Leung, and N. Phamdo, Low density parity check codes with semi-random parity check matrix, *Electronics Letters*, vol. 35, no. 1, pp. 38–39, 1999.

[8] Y. Pei, L. Yin, and J. Lu, Design of irregular ldpc codec on a single chip FPGA, in *Proc. 6th Int. Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication*, Beijing, China, 2004, pp. 221–224.

[9] Z. Wang, Y. Chen, and K. K. Parhi, Area efficient decoding of quasi-cyclic low density parity check codes, in *Proc. 2004 Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP'04)*, 2004.

[10] Z. Wang and Z. Cui, Low-complexity high-speed decoder design for quasi-cyclic LDPC codes, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 1, pp. 104–114, 2007.

[11] Y. Dai, Z. Yan, and N. Chen, Optimal overlapped message passing decoding of quasi-cyclic ldpc codes, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 5, pp. 565–578, 2008.

[12] Y. Chen and K. K. Parhi, Overlapped message passing for quasi-cyclic low-density parity check codes, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 6, pp. 1106–1113, 2004.

[13] I. C. Park and S. H. Kang, Scheduling algorithm for partially parallel architecture of ldpc decoder by matrix permutation, in *Proc. 10th Int. Symposium on Circuits and Systems*, 2005, pp. 5778–5781.

[14] J. Y. Park and K. S. Chung, Overlapped message passing technique with resource sharing for high speed CMMB LDPC decoder, *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1564–1570, 2011.

[15] B. Lin, Q. Li, Y. Pei, and L. Yin, High speed ldpc decoder design based on general overlapped message-passing architecture, in *Proc. 6th Int. Conf. on Ubiquitous and Future Networks*, Beijing, China, 2014, pp. 454–459.

[16] X. Chen, J. Kang, S. Lin, and V. Akella, Memory system optimization for fpga-based implementation of quasi-cyclic ldpc codes decoders, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 1, pp. 98–111, 2011.

[17] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X. Y. Hu, Reduced-complexity decoding of ldpc codes, *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[18] M. Rovini, N. E. L. Insalata, F. Rossi, and L. Fanucci, VLSI design of a high-throughput multi-rate decoder for structured LDPC codes, in *Proc. 8th Euromicro Conf. on Digital System Design*, 2005, pp. 202–209.

Baihong Lin received the BE degree from Beijing University of Aeronautics and Astronautics, China, in 2009. He is currently pursuing the PhD degree in Tsinghua University. His major research interests include LDPC encoders and decoders, image processing, medical imaging, computer vision, and machine learning.

Jianhua Lu received the BEng and MEng degrees from Tsinghua University, China, in 1986 and 1989, respectively, and the PhD degree in electrical and electronic engineering from Hong Kong University of Science and Technology. He has been with the Department of Electronic Engineering, Tsinghua University, since 1989, where he is currently a professor. He has published more than 180 technical papers in international journals and conference proceedings. His current research interests include broadband wireless communication, multimedia signal processing, satellite communication, and wireless networking. Dr. Lu has been an active member of several professional societies. He was a recipient of the Best Paper Award at the International Conference on Communications, Circuits and Systems in 2002 and ChinaCom in 2006, and received the National Distinguished Young Scholar Fund from the NSF Committee of China in 2006. He has served in numerous IEEE conferences as a member of Technical Program Committees and served as a Lead Chair of the General Symposium of IEEE ICC in 2008, as well as a Program Committee Co-Chair of the 9th IEEE International Conference on Cognitive Informatics. He is now an IEEE fellow.

Yukui Pei received the BEng and PhD degrees from Tsinghua University in 2002 and 2008, respectively. Now he is an associate professor with Department of Electronic Engineering, Tsinghua University. His research interests include channel coding, UWB, and wireless security.

Liuguo Yin is an associate professor in the School of Aerospace, Tsinghua University, China. He received the MEng and PhD degrees from Tsinghua University, China, in 2002 and 2005, respectively. From March 2005 to March 2007, he was a research assistant with the School of Aerospace, Tsinghua University. From April 2007 to March 2008, he was an ERCIM postdoctoral fellow with the Norwegian University of Science and Technology (NTNU), Trondheim, Norway. His research interests include channel coding, joint source-channel coding, aerospace communications, and wireless multimedia communication systems. He is a member of IEICE communications society and engineering sciences society.