PAPER

# Low-Complexity Memory Access Architectures for Quasi-Cyclic LDPC Decoders

**Ming-Der SHIEH**[†a)], ***Member***, **Shih-Hao FANG**[†], **Shing-Chung TANG**[††], ***and*** **Der-Wei YANG**[†], ***Nonmembers***

**SUMMARY** Partially parallel decoding architectures are widely used in the design of low-density parity-check (LDPC) decoders, especially for quasi-cyclic (QC) LDPC codes. To comply with the code structure of parity-check matrices of QC-LDPC codes, many small memory blocks are conventionally employed in this architecture. The total memory area usually dominates the area requirement of LDPC decoders. This paper proposes a low-complexity memory access architecture that merges small memory blocks into memory groups to relax the effect of peripherals in small memory blocks. A simple but efficient algorithm is also presented to handle the additional delay elements introduced in the memory merging method. Experiment results on a rate-1/2 parity-check matrix defined in the IEEE 802.16e standard show that the LDPC decoder designed using the proposed memory access architecture has the lowest area complexity among related studies. Compared to a design with the same specifications, the decoder implemented using the proposed architecture requires 33% fewer gates and is more power-efficient. The proposed new memory access architecture is thus suitable for the design of low-complexity LDPC decoders.

***key words:*** *error control coding, low-density parity-check (LDPC) codes, quasi-cyclic (QC) LDPC codes, partially parallel architecture, VLSI design*

## 1. Introduction

Low-density parity-check (LDPC) codes, first introduced by Gallager in 1962 [1], are linear block codes with very sparse parity check matrices. They can be represented as a bipartite graph, also called a *Tanner graph* [2], in which two kinds of node, variable nodes and check nodes, are used to denote the codeword bits and the parity bits, respectively. For LDPC decoder designs, fully parallel architectures [3], [4] and partially parallel architectures [5]–[8] are two common choices for VLSI implementations. Although the fully parallel architecture can obtain higher throughput, it may have complex interconnections between the check-node units (CNUs) and variable-node units (VNUs). Considering both the hardware cost and decoding throughput, most LDPC decoders are constructed using partially parallel architectures, which may use specific optimization techniques. For example, overlapped message passing algorithms [5], [6] were proposed for quasi-cyclic LDPC (QC-LDPC) codes to increase the hardware utilization of CNUs and VNUs. Kang [7] presented techniques to reduce the number of messages to be exchanged between CNUs and VNUs, thereby reducing the interconnection complexity. For practical applications such as the IEEE 802.16e system, the multi-mode decoder design for various codeword lengths have been investigated [8]. Recently, the layered decoding algorithm (LDA) [9], [10] has attracted much attention because of the reduced number of iterations as compared to conventional sum-product algorithm (SPA). In addition to single-rate designs, LDPC decoders for multi-rate applications have also been intensively studied [11]–[14].

For the partially parallel architecture, internal memory is required to store the extrinsic messages. To increase the parallelism of data access, the internal memory is usually divided into a set of small memory blocks. However, doing so may increase the required memory space because the peripherals of small memory blocks occupy a larger portion of memory area than do those of large memory blocks. This effect is shown in Fig. 1, whose results were obtained using the Artisan memory compiler [15]. Based on this observation, this paper presents an efficient method of selecting small memory blocks and then combining them into a larger one to save memory area. An efficient algorithm is used to deal with the memory access conflicts that result from memory merging management. For ease of explanation, the non-overlapped message passing architecture is adopted to describe the main idea of the proposed architecture. The same idea can be easily extended to the overlapped message passing architectures. Experimental results show that the proposed memory access architecture allows a significant reduction in the required memory area with a very limited degradation in the throughput rate. Compared to the related works, the proposed design has the lowest area complexity due to a significant reduction in the required memory area.

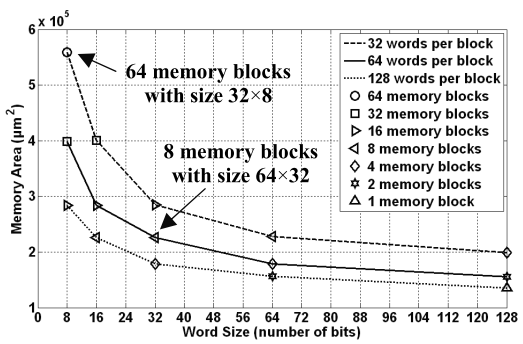


**Fig. 1** Comparison of required memory area using various sizes of memory blocks assuming a fixed memory size of 16 K bits.

Manuscript received April 4, 2011.
Manuscript revised August 30, 2011.

[†]The authors are with the Department of Electrical Engineering, National Cheng Kung University, Tainan City, Taiwan, R.O.C.
[††]The author is with Himax Technology, Tainan City, Taiwan, R.O.C.
a) E-mail: shiehm@mail.ncku.edu.tw
DOI: 10.1587/transinf.E95.D.549

The rest of this paper is organized as follows. Section 2 presents an overview of LDPC codes and decoders. Then, the proposed low-complexity memory access architecture is described in Sect. 3. Section 4 presents a simple memory merging algorithm for systematically selecting smaller memory blocks to be combined into a larger one. Section 5 shows the experiment results and performance evaluation of the proposed design. Finally, conclusions are given in Sect. 6.

## 2. Overview of LDPC Codes and Decoders

### 2.1 Quasi-Cyclic LDPC Codes

QC-LDPC codes, a special subset of LDPC codes, are widely used because their regular code structures are very suitable for hardware implementation. The parity-check matrix $\boldsymbol{H}$ of a $(J, K)$-regular QC-LDPC code includes $J$ block rows, $K$ block columns, and $J \times K$ circulant matrix $\boldsymbol{I}_{j,k}$, where $0 \leq j \leq J-1$ and $0 \leq k \leq K-1$. Each circulant matrix $\boldsymbol{I}_{j,k}$ is a $Z \times Z$ cyclically shifted identity matrix with an offset or shift value $S_{j,k}$. A $J \times K$ base matrix $\boldsymbol{H}_{base}$, which defines all the offset values $S_{j,k}$, can also be defined, where $0 \leq S_{j,k} \leq Z-1$. After that, a parity-check matrix $\boldsymbol{H}$ with a dimension of $M \times N$, where $M = J \times Z$ and $N = K \times Z$, can then be defined.

### 2.2 Decoding Algorithms

The kernel of sum-product algorithm (SPA) updates two kinds of extrinsic message, $L(r_{ji})$ and $L(q_{ij})$, which represent the check-to-variable message and the variable-to-check message, respectively. The subscripts $i$ and $j$ above stand for the $i$-th variable node and the $j$-th check node, respectively.

Before decoding is started, $L(q_{ij})$ is initialized based on the received symbol $y_i$. Assuming an additive white Gaussian noise (AWGN) channel with standard deviation $\sigma$, $L(q_{ij})$ is then initialized as the initial log-likelihood ratio (LLR) $L(c_i)$, computed as:

$$L(c_i) = 2y_i/\sigma^2 \tag{1}$$

During the check-node phase, $L(r_{ji})$ is updated based on $L(q_{ij})$, expressed as:

$$L(r_{ji}) = \left(\prod_{i' \in R_{j\backslash i}} \alpha_{i'j}\right) \cdot \phi\left(\sum_{i' \in R_{j\backslash i}} \phi(\beta_{i'j})\right) \tag{2}$$

where $\phi(x) = -\log(\tanh(|x|/2))$, $\alpha_{ij} = sign(L(q_{ij}))$, and $\beta_{ij} = |L(q_{ij})|$. The notation $R_{j\backslash i}$ denotes the set of variable nodes connected to the $j$-th check node excluding the $i$-th variable node.

Compared to the operations defined in the check-node phase, those in the variable-node phase are relatively simple. $L(q_{ij})$ and the estimated message $L(Q_i)$ can be updated using $L(r_{ji})$ and $L(c_i)$, expressed as:

$$L(q_{ij}) = L(c_i) + \sum_{j' \in \Lambda_{i\backslash j}} L(r_{j'i}) \tag{3}$$

and

$$L(Q_i) = L(c_i) + \sum_{j \in \Lambda_i} L(r_{ji}) \tag{4}$$

where $\Lambda_i$ stands for the set of check nodes connected to the $i$-th variable node and $\Lambda_{i\backslash j}$ contains the nodes in $\Lambda_i$ excluding the check node $j$. From the computed message $L(Q_i)$ in (4), the $i$-th estimated codeword bit $v_i$ can then be decided based on:

$$v_i = \begin{cases} 1, & \text{if } L(Q_i) < 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The decoding process terminates if the computed codeword $\boldsymbol{v} = [v_0, v_1, \ldots, v_{N-1}]$ satisfies the condition $\boldsymbol{v}\boldsymbol{H}^T = \boldsymbol{0}$ or the maximum number of iterations has been reached. Otherwise, the process goes back to the check-node phase and the next iteration is started.

### 2.3 Partially Parallel Architecture for QC-LDPC Decoding

From the $\boldsymbol{H}$ matrix for QC-LDPC, the corresponding partially parallel architecture of QC-LDPC decoding, as presented in [5], is illustrated in Fig. 2. It consists of $J$ CNUs and $K$ VNUs together with $J \times K$ memory blocks $M_{j,k}$, $0 \leq j \leq J-1$ and $0 \leq k \leq K-1$, used to store the extrinsic messages. Each CNU/VNU reads data from predefined memory blocks and then stores the updated messages to the same addresses of the memory blocks. It is assumed that the intrinsic messages $L(c_k)$ and the decoded codeword are stored in $K$ memory blocks $MC_k$ and $K$ memory blocks $MQ_k$, respectively.

Since the parity-check matrix of QC-LDPC codes is divided into a set of circulant matrices in which each column/row contains only one nonzero element, a straightforward addressing scheme is to map the nonzero elements in each submatrix $\boldsymbol{I}_{j,k}$ to the memory locations [11]. That is, the extrinsic message corresponding to the $i$-th row is stored in the $i$-th location of $M_{j,k}$. In this way, the mapping relationship between the memory address and the row/column

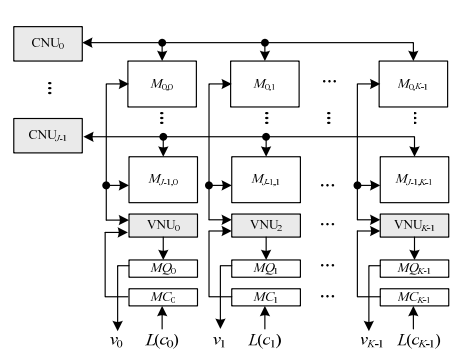


**Fig. 2** Partially parallel decoding architecture for the $\boldsymbol{H}$ matrix.

index can be easily obtained. Let $A(M_{j,k})$ denote an address in memory block $M_{j,k}$. The notation $RI(A(M_{j,k}))$ is used to represent the row index of $\boldsymbol{I}_{j,k}$ corresponding to $A(M_{j,k})$ in the check-node phase. Similarly, $CI(A(M_{j,k}))$ represents the column index of $\boldsymbol{I}_{j,k}$ corresponding to $A(M_{j,k})$ in the variable-node phase. Given the offset value $S_{j,k}$ and the dimension $Z$ of the square matrix $\boldsymbol{I}_{j,k}$, the following equations can be derived:

$$RI(A(M_{j,k})) = A(M_{j,k}) \tag{6}$$

and

$$CI(A(M_{j,k})) = (A(M_{j,k}) + S_{j,k}) \bmod Z. \tag{7}$$

Although the address generation unit (AGU) can be easily derived using (6) and (7), the hardware utilization of this architecture is only 50% if the two decoding phases are not overlapped. As stated in [5], the minimum waiting time between the two phases can be reduced by properly selecting the row starting index (RSI) and the column starting index (CSI) to obtain overlapped decoding phases, thereby increasing hardware utilization.

## 3. Proposed Partially Parallel Architecture with Combined Memory Blocks

As can be observed in Fig. 1, the total memory area required in LDPC decoders can be greatly reduced by merging the small memory blocks into larger ones. However, directly merging memory blocks may create memory access conflicts because each entry of the merged memory block now consists of multiple data, but these data may not be processed by CNUs/VNUs at the same time. Therefore, additional storage elements are needed to buffer the retrieved data. However, this results in extra hardware overhead, which may offset the benefits obtained from merging memory blocks. Note that a reduction in throughput rate may result from additional memory access, as described later. A similar problem occurs when the updated messages are written back to the merged memory blocks.

In this work, an efficient memory merging scheme is proposed. The resulting hardware requirement of QC-LDPC decoders, implemented using the partially parallel architecture, can be significantly reduced with very limited throughput degradation. The basic idea of the proposed scheme is to ensure that the retrieved data from the merged memory structure is processed immediately or with minimum delay by properly selecting memory blocks to be combined. For ease of explanation, a direct method, denoted as the block row merging scheme, is first presented to illustrate the main idea of our development. Then, a fast and efficient memory block selection (MBS) algorithm is proposed for selecting candidate blocks to be merged so that the number of extra buffers required from applying the direct method can be greatly reduced.

### 3.1 Block Row Merging Scheme

Assume that the $J \times K$ memory blocks shown in Fig. 2 are

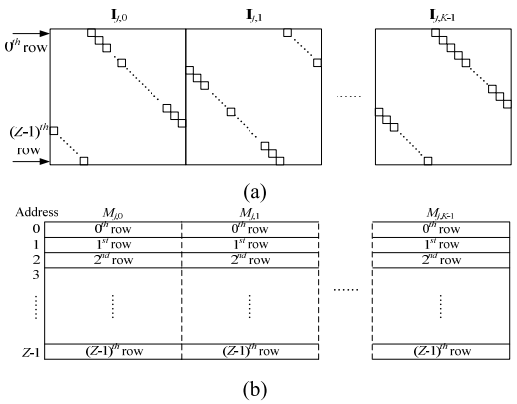


**Fig. 3** (a) $j^{\text{th}}$ block row of a parity-check matrix, and (b) memory arrangement based on the block row merging scheme.

used to store the extrinsic messages and that each $M_{j,k}$ is associated with a circulant matrix $\boldsymbol{I}_{j,k}$. A straightforward merging scheme is to combine all the distinct blocks in a block row, as depicted in Fig. 3 (a) for the $j$-th block row. Such an arrangement is referred to as the *block row merging scheme* in this work. Let $G_i$ denote the $i$-th memory group produced by merging a set of small memory blocks. Applying the block row merging scheme, exactly $J$ memory groups are obtained, with each group $G_i$ consisting of all $M_{i,k}$ for $0 \le k \le K-1$. Using the mapping relationship between the physical memory addresses and the nonzero elements of $\boldsymbol{I}_{j,k}$, the corresponding memory arrangement is illustrated in Fig. 3 (b), in which $K$ extrinsic messages in a row share the same memory address. Therefore, all $K$ messages are accessed at the same time.

According to the check-node operations, one can see that the $K$ messages accessed from the same row can be immediately processed by a CNU. In contrast, in the variable-node phase, because all the required messages in a column of $\boldsymbol{H}$ are stored in different memory groups that may not be accessed at the same time, buffer units are needed during the operation of VNUs.

The number of required buffers is determined by the life time of messages when they are read from the memory and then consumed in later time stages. A longer life time implies a large number of additional buffers, which in turn decreases the throughput rate in the variable-node phase. As a result, selecting appropriate starting addresses for the memory groups in the variable-node phase is very crucial for reducing the overhead in the buffer requirement and the degradation in throughput performance. How to choose a set of starting addresses for the memory groups to solve the problems in the variable-node phase is discussed in the next subsection.

### 3.2 Starting Address Determination

With the block row merging scheme, one can start the check-node phase from any address of the memory groups and each CNU accesses only one memory group. With-

out loss of generality, CNUs are assumed to access memory groups consecutively from the $0^{\text{th}}$ address; therefore, the RSI for each block row is also zero based on (6). During the variable-node phase, the required messages for each VNU are distributed in different memory groups, and the addresses of the memory groups corresponding to these messages may be different. This situation can be seen from (7). Similar to the check-node phase, each memory group in the variable-node phase is also accessed consecutively from a specific address, but its starting address is decided according to the given CSIs of original memory blocks. A simple but effective method of selecting a proper starting address for each memory group, derived by applying the block row merging scheme is describe below. A low-complexity memory access architecture with buffers for message alignment is thus obtained.

For ease of explanation, it is assumed that a set of CSIs has been obtained using existing techniques, such as those in [5], [6]. Let $SA_v(M_{j,k})$ denote the starting address of memory block $M_{j,k}$ in accordance with the given CSIs in the variable-node phase. Note that the value of $SA_v(M_{j,k})$ can be computed according to the CSI and (7). From the computed $SA_v(M_{j,k})$, a $J \times K$ matrix $\mathbf{\Omega}$ is constructed with its elements defined as:

$$(\mathbf{\Omega})_{j,k} = SA_v(M_{j,k}) \tag{8}$$

where $(\mathbf{\Omega})_{j,k}$ denotes the element at the $j$-th row and the $k$-th column of $\mathbf{\Omega}$. The matrix $\mathbf{\Omega}$ thus provides all the starting addresses of memory blocks. This information can be used to compute the distance between each pair of related starting addresses and to estimate the life times of retrieved messages. Therefore, the life times can be minimized by selecting proper starting addresses for the merged memory blocks. This in turn minimizes the number of additional buffers. Specifically, let $SA_v(G_j)$ be the starting address of memory group $G_j$ in the variable-node phase. As mentioned above, a good starting address $SA_v(G_j)$ can be calculated as:

$$SA_v(G_j) = \arg\min_a[\max_b((b-a) \bmod Z)] \tag{9}$$

where $a, b \in \mathbf{\Phi}_j$ and $\mathbf{\Phi}_j = \{SA_v(M_{j,k}) \mid M_{j,k} \in \boldsymbol{S}(G_j)\}$. $\boldsymbol{S}(G_j)$ is the set of memory blocks $M_{j,k}$ included in memory group $G_j$. That is, the argument $a$ which produces a minimum distance between $a$ and the other starting addresses of memory blocks in the same group is selected as the starting address of the memory group. The $SA_v(G_j)$ determined from (9) leads to a minimum number of redundant memory access in the variable-node phase.

To estimate the total number of additional buffers, the relative block delay $dR(M_{j,k})$ between $SA_v(M_{j,k})$ and $SA_v(G_j)$ is first computed. It is expressed as:

$$dR(M_{j,k}) = (SA_v(M_{j,k}) - SA_v(G_j)) \bmod Z \tag{10}$$

where $M_{j,k} \in \boldsymbol{S}(G_j)$. In addition, the maximum relative block delay of all $dR(M_{j,k})$ corresponding to $G_j$ is defined as:

$$\overline{dR}(G_j) = \max_{M_{j,k} \in \boldsymbol{S}(G_j)} (dR(M_{j,k})) \tag{11}$$

From the selected $SA_v(G_j)$, the value of $\overline{dR}(G_j)$ can be interpreted as the minimum number of delay elements required for memory group $G_j$ to line up the messages that can be directly retrieved from the original non-merging memory structure. For clarity, a $J \times K$ matrix $\mathbf{\Gamma}$ is also constructed with its elements defined as:

$$(\mathbf{\Gamma})_{j,k} = dR(M_{j,k}) \tag{12}$$

where $(\mathbf{\Gamma})_{j,k}$ denotes the element at the $j$-th row and the $k$-th column of $\mathbf{\Gamma}$.

As an example, consider the $2 \times 3$ base matrix $\boldsymbol{H}_{base}$ of a QC-LDPC code with block size $Z = 13$, as shown in Fig. 4 (a). Using the block row merging scheme, two memory groups, $G_0$ and $G_1$, are constructed as given in Fig. 4 (b). If the CSIs of the first to the last block columns are $\{10, 0, 0\}$, the $\mathbf{\Omega}$ matrix can be derived using (7) and expressed as:

$$\mathbf{\Omega} = \begin{bmatrix} 9 & 11 & 9 \\ 5 & 3 & 6 \end{bmatrix} \tag{13}$$

From (9), the starting addresses of memory groups are obtained as $SA_v(G_0) = 9$ and $SA_v(G_1) = 3$. With the derived $SA_v(G_0)$, $SA_v(G_1)$, and $\mathbf{\Omega}$ matrix, the $\mathbf{\Gamma}$ matrix is computed using (10) and (12) as:

$$\mathbf{\Gamma} = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 3 \end{bmatrix} \tag{14}$$

Therefore, the maximum block delays $\overline{dR}(G_0)$ and $\overline{dR}(G_1)$ defined in (11) are equal to 2 and 3, respectively.

### 3.3 Proposed Memory Access Architecture

The low-complexity memory access architecture obtained using the proposed block row merging scheme for the $2 \times 3$ base matrix in Fig. 4 (a) is depicted in Fig. 5. Note that the intrinsic memory which stores $L(c_i)$ is not shown in Fig. 5 for simplicity. In addition to the two memory groups, the decoder consists of two types of processing unit, CNUs and VNUs, and first-in first-out (FIFO) buffers associated with VNUs for message alignment, described later. Each message can be accessed from the assigned entry by passing through the data distributor and being written back in groups via the data combiner. Note that since the operations in the check-node phases are not affected by the proposed block row merging scheme, this study only focuses on the modifications in the variable-node phase. For clarity, the contents of memory groups $G_0$ and $G_1$ in the variable-node phase are detailed in Fig. 6, in which each entry of $G_0/G_1$ contains 3 messages and the symbol $r_{j,i}$ denotes the message

$$\mathbf{H}_{base} = \begin{bmatrix} 1 & 2 & 4 \\ 5 & 10 & 7 \end{bmatrix}$$

(a)

| | | | |
|---|---|---|---|
| $G_0$ | $M_{0,0}$ | $M_{0,1}$ | $M_{0,2}$ |
| $G_1$ | $M_{1,0}$ | $M_{1,1}$ | $M_{1,2}$ |

(b)

**Fig. 4** (a) Base matrix of a QC-LDPC code with block size $Z = 13$ and (b) memory groups after merging memory blocks in the same row.

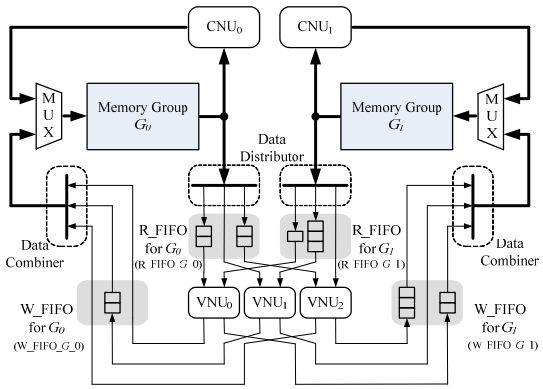


**Fig. 5** Low-complexity memory access architecture based on block row merging scheme for the QC-LDPC code shown in Fig. 4 (a).

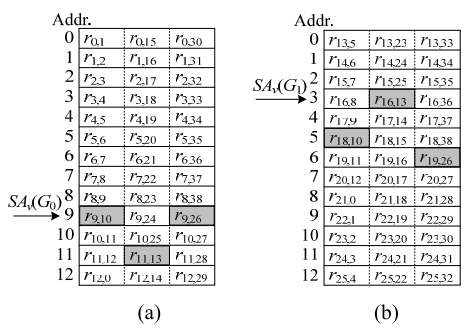


**Fig. 6** Contents of (a) memory group $G_0$; (b) memory group $G_1$ in the variable-node phase.

from check node $j$ to variable node $i$, as defined in Sect. 2.2. Therefore, the set of $r_{j,i}$ with the same subscript $i$ are messages to be processed by a VNU at the same time in the variable-node phase.

In the conventional memory access architecture, the six highlighted messages in Fig. 6, i.e., $r_{9,10}$, $r_{18,10}$, $r_{11,13}$, $r_{16,13}$, $r_{9,26}$, and $r_{19,26}$, are accessed at the same time in the variable-node phase. Using the proposed architecture, the three messages $r_{9,10}$, $r_{11,13}$, and $r_{9,26}$ are not read from $G_0$ simultaneously. Similar observations are true for $r_{18,10}$, $r_{16,13}$, and $r_{19,26}$ in $G_1$. To solve this problem, additional buffers, denoted as R_FIFO hereafter, are introduced to line up messages read from the memory groups for VNUs. R_FIFO_G_$j$ is used to denote the set of read FIFOs allocated for memory group $G_j$ and R_FIFO_G_$j$_$k$ is used to denote the $k$-th FIFO (counting from the left side) inside the R_FIFO_G_$j$. Specifically, the read buffer R_FIFO_G_0 (R_FIFO_G_1) associated with memory group $G_0$ ($G_1$) is used to rearrange the retrieved messages in the same order as that retrieved from conventional non-merged memory access architecture. For simplicity, R_FIFO is used to represent either R_FIFO_G_$j$ or R_FIFO_G_$j$_$k$ where appropriate.

From (11) and (12), one can easily show that the length of R_FIFO_$j$_$k$ associated with each output of the data distributor is minimized when assigned as $\overline{dR}(G_j) - dR(M_{j,k})$. Assuming the starting addresses $SA_v(G_0) = 9$ and $SA_v(G_1) = 3$, the length of required R_FIFO_$j$_$k$ is drawn in Fig. 5. Note that since the length of R_FIFO_$j$ is minimized individually for $G_j$, the times to start reading messages from $G_0$ and $G_1$ are skewed by $|\overline{dR}(G_1) - \overline{dR}(G_0)|$ time delays to synchronize the messages read from different memory groups. For example, the pairs of messages $(r_{9,10}, r_{18,10})$, $(r_{11,13}, r_{16,13})$, and $(r_{9,26}, r_{19,26})$, read from $G_0$ and $G_1$ at different time slots, appear at the inputs of $\mathrm{VNU}_0$, $\mathrm{VNU}_1$, and $\mathrm{VNU}_2$, respectively, simultaneously by using R_FIFO and assuming that the access times of $G_1$ and $G_0$ start from $t$ and $t + 1$, respectively. This process is repeated for consecutive retrievals of the remaining messages. Note that messages, such as $r_{9,24}$ and $r_{10,25}$ in $G_0$, retrieved before the highlighted rectangles in Fig. 6, are written back without modification the first time they are read from the memory group. That is, they are read twice before being processed by VNUs, which results in throughput degradation in the variable-node phase. This problem can be greatly reduced by employing the enhanced memory merging algorithm presented in Sect. 4.

Similar to the application of R_FIFO, the updated or bypassed messages have to line up again before being written back to memory groups. As a result, additional FIFOs, denoted as W_FIFO, are needed to align the messages for memory write operations. W_FIFO temporarily stores the messages outputted from VNUs and the aligned messages are then written back to memory groups through the data combiner. Because the length of R_FIFO_G_$j$_$k$ is equal to $\overline{dR}(G_j) - dR(M_{j,k})$, the length of W_FIFO_G_$j$_$k$ associated with each input of the data combiner can be assigned as $dR(M_{j,k})$. In this manner, all the messages read from $G_j$ exactly pass through $\overline{dR}(G_j)$ delay elements before they are written back to $G_j$. The summation of the lengths of R_FIFO_G_$j$_$k$ and W_FIFO_G_$j$_$k$ along a path $k$ is thus a constant, which is equal to $\overline{dR}(G_j)$. This indicates that extra memory access and throughput degradation can be reduced if $\overline{dR}(G_j)$ is minimized for each memory group. Note that although the values of CSIs affect the required FIFO length, how to choose good CSIs is not the focus of this work. The proposed starting address determination method and low-complexity memory access architecture can be used in conjunction with existing CSI determination methods. In summary, the proposed memory access architecture has the following advantages: (a) it can greatly reduce the required memory area by merging a set of small memory blocks with close addressing sequences into a larger one to relax the overhead of peripherals in the memory access architecture; (b) it is more power-efficient as compared to conventional LDPC architectures.

## 4. Memory Block Selection Algorithm

As discussed above, the memory access architecture derived using the proposed block row merging scheme can operate correctly by utilizing the inserted R_FIFO and W_FIFO. However, the required FIFO length may be large for parity-check matrices with large dimensions. An alternative is to group memory blocks with a maximum allowable FIFO length along each path to reduce the degradation in through-

put performance. A fast and efficient algorithm, called the MBS algorithm, for systematically selecting memory blocks to form memory groups is presented below.

Unlike the block row merging scheme, the memory blocks in a memory group selected by the proposed MBS algorithm may contain circulant matrices from different block rows and the number of selected blocks in a group is not restricted to a fixed number. As a result, the maximum relative block delay of memory group $G_i$ can be greatly decreased if the memory blocks to be combined are properly chosen. This reduces the required FIFO length in the low-complexity memory access architecture shown in Sect. 3. The proposed MBS algorithm is summarized as follows.

---

MBS Algorithm

---

*Input*: RSIs, CSIs, $L$
*Initialization*: $\mathbf{\Omega}$, $\boldsymbol{\omega} = \text{sort}(\mathbf{\Omega})$, $i = 0$
*Memory Block Selection*:
(A.1) $S_0 = min(\boldsymbol{\omega})$
(A.2) Include all $M_{j,k}$ with $SA_v(M_{j,k})$ satisfying $S_0 \leq SA_v(M_{j,k}) \leq S_0 + L$ in memory group $G_i$, and store the selected $SA_v(M_{j,k})$ in the vector $\boldsymbol{\omega}_i$.
(A.3) Remove all selected $SA_v(M_{j,k})$ in (A.2) from $\boldsymbol{\omega}$.
(A.4) *if* $(\boldsymbol{\omega} = \phi)$, $|G| = i + 1$ and go to *Memory Group Refinement*;
*else* $i = i + 1$ and go back to (A.1).
*Memory Group Refinement*:
(B.1) $i = 0$
(B.2) $N_0 = N(G_i, G_{i+1})$;
$N_d = N(D_i, D_{i+1})$, where $D_i = G_i - \{M_{j,k} \mid SA_v(M_{j,k}) = max(\boldsymbol{\omega}_i)\}$, and $D_{i+1} = G_{i+1} \cup \{M_{j,k} \mid SA_v(M_{j,k}) = max(\boldsymbol{\omega}_i)\}$;
$N_u = N(U_i, U_{i+1})$, where $U_i = G_i \cup \{M_{j,k} \mid SA_v(M_{j,k}) = min(\boldsymbol{\omega}_{i+1})\}$ and $U_{i+1} = G_{i+1} - \{M_{j,k} \mid SA_v(M_{j,k}) = min(\boldsymbol{\omega}_{i+1})\}$.
(B.3) $N_{min} = (N_0, N_d, N_u)$
(B.3.1) *if* $(N_{min} = N_d)\&(\overline{dR}(D_{i+1}) \leq L)$, move $M_{j,k}$ with $SA_v(M_{j,k}) = max(\boldsymbol{\omega}_i)$ from $G_i$ to $G_{i+1}$, move $max(\boldsymbol{\omega}_i)$ from $\boldsymbol{\omega}_i$ to $\boldsymbol{\omega}_{i+1}$, let $i = i - 1$ if $i \neq 0$, and go back to (B.2);
*else* go to (B.3.2).
(B.3.2) *if* $(N_{min} = N_u)\&(\overline{dR}(U_i) \leq L)$, move $M_{j,k}$ with $SA_v(M_{j,k}) = min(\boldsymbol{\omega}_{i+1})$ from $G_{i+1}$ to $G_i$, let $i = i - 1$ if $i \neq 0$, move $min(\boldsymbol{\omega}_{i+1})$ from $\boldsymbol{\omega}_{i+1}$ to $\boldsymbol{\omega}_i$, and go back to (B.2);
*else* go to (B.3.3).
(B.3.3) *if* $(i \neq |G| - 2)$, $i = i + 1$ and go back to (B.2);
*else* go to *Output*.
*Output*: $G_i$ and $\boldsymbol{\omega}_i$ for $0 \leq i \leq |G| - 1$.

---

The proposed MBS algorithm consists of two main steps: memory block selection and memory group refinement. In the former, memory blocks are sorted according to their computed starting addresses and then the sorted blocks are partitioned into groups subject to the constraint $L$, which is denoted as the maximal allowable FIFO length. In the latter, the number of required delay elements is reduced by migrating memory blocks between adjacent groups without violating the constraint $L$. The incremental improvement in the number of delay elements is guided by iteratively evaluating the cost function $N(G_i, G_{i+1})$ for every possible move. The cost function $N(G_i, G_{i+1})$ is defined as:

$$N(G_i, G_{i+1}) = \overline{dR}(G_i) \times |G_i| + \overline{dR}(G_{i+1}) \times |G_{i+1}| \quad (15)$$

where $|G_i|$ is the number of memory blocks in $G_i$. The value of $N(G_i, G_{i+1})$ indicates the total number of required delay elements associated with $G_i$ and $G_{i+1}$. Since all the elements in $\boldsymbol{\omega}_i$ are less than those in $\boldsymbol{\omega}_{i+1}$ in each iteration, the MBS algorithm has the choice of either moving the memory block(s) in $G_i$ with $max(\boldsymbol{\omega}_i)$ down to $G_{i+1}$ or moving those in $G_{i+1}$ with $min(\boldsymbol{\omega}_{i+1})$ up to $G_i$. The values of $N_d = N(D_i, D_{i+1})$ for downward movement and those of $N_u = N(U_i, U_{i+1})$ for upward movement are then compared to the $N_0 = N(G_i, G_{i+1})$ of the current partition to keep the solution with the lowest cost. For a successful move, the iteration traces back one step by reducing the index $i$ by one because the content of $G_i$ has been altered now. Otherwise, the iteration proceeds by increasing the index by one until the final group is reached.

In summary, the MBS algorithm reduces the number of required delay elements to reduce the FIFO overhead by allowing merged memory blocks to be selected from different block rows. The pre-defined maximum allowable FIFO length controls the maximum throughput degradation; a small $L$ makes degradation negligible in practical applications of LDPC codes with large dimensions. A larger $L$ leads to a smaller number of memory groups with an increase in the number of FIFOs and throughput degradation. There is a tradeoff among the area reduction from merging memory blocks, the increase in FIFO overhead, and the throughput degradation. A proper value of $L$ can be chosen based on the requirements of target LDPC decoder.

## 5. Experiment Results and Comparisons

A LDPC decoder design for IEEE 802.16e systems with a code rate-1/2 parity-check matrix and a maximum block size of $Z = 96$ was coded in Verilog hardware description language and synthesized using Synopsys tools. The LDPC decoder consists of 19 modes with a code length ranging from 576 to 2304. The key settings of the proposed LDPC decoder design are: (i) the RSIs are assumed to be zero for all the block rows and the CSIs are given as $\{70, 75, 28, 95, 2, 0, 34, 87, 91, 24, 88, 1, 26, 4, 53, 7, 50, 63, 59, 52, 49, 46, 41, 25\}$ for block columns counting from the left side, (ii) the maximum allowable FIFO length $L$ is set to 4 in the MBS algorithm to group memory blocks, (iii) the min-sum algorithm is employed with 8 quantization bits for comparison with results in, (iv) the data path contains 12 conventional CNUs and 24 VNUs with bypass mode. Note that since the selection of CSIs is not the main focus of this work and the prototype design adopts non-overlapped message passing architecture, the CSIs were randomly generated for simplicity. Using the

**Table 1** Comparison of LDPC decoders.

| | Blanksby [3] | Kang [7] | Zhang [9] | Liu [12] | Shih [8] | This work |
|---|---|---|---|---|---|---|
| Architecture | Fully Parallel | Partially Parallel | Layered Decoding | Self-Routing | Partially Parallel | Partially Parallel |
| Multi-Mode | No | No | No | Yes (114 modes) | Yes(19 modes) | Yes(19 modes) |
| Parity Check Matrix | Irregular | (3,6)-regular | Quasi-Cyclic | Quasi-Cyclic | Quasi-Cyclic | Quasi-Cyclic |
| Block Length | 1024 | 1024 | 2304 | 576-2304 | 576-2304 | 576-2304 |
| Code Rate | 1/2 | 1/2 | 1/2 | 1/2, 2/3A, 2/3B 3/4A, 3/4B, 5/6 | 1/2 | 1/2 |
| Iterations | 64 | 8 | 10 | 20 | 8 | 8 |
| Quantization Bits | 4-bit | – | 6-bit | 6-bit | 8-bit | 8-bit |
| # PEs (CNUs/VNUs) | 512/1024 | 16/32 | 12/24 | 192/96 | 4/8 | 12/24 |
| Technology | 0.16μm | 0.18μm | 90nm | 90nm | 0.13μm | 0.13μm |
| Frequency | 64MHz | 200MHz | 950MHz | 150MHz | 83.3MHz | 100MHz |
| Throughput Rate | 1Gbps | 582Mbps | 2.2Gbps | 105Mbps | 60Mbps | 135Mbps |
| Gate Count | 1750k | 457k | – | 970k(1) | 420k | 280k |
| Area ($mm^2$) | 52.5$mm^2$ | 6.29$mm^2$ | 2.9$mm^2$ (Core Size) | 6.25$mm^2$ | 8.29$mm^2$ (Die Size) 4.45$mm^2$ (Core Size) | 1.89$mm^2$ (Core Size) |
| Power Dissipation | 690mW | – | 870mW | 264mW | 52mW | 61.3mW |

(1) Logic part requires 380k gates and the memory part requires 590k gates.

**Table 2** Area comparison of storage units.

| Storage Units | Area ($\mu m^2$) | | Improvement |
|---|---|---|---|
| | Proposed | [8] | |
| Intrinsic Memory | 104,909 | 225,428 | 53.46% |
| Extrinsic Memory | 663,592 | 1,161,490 | 42.87% |
| FIFO | 99,200 | - | - |
| Total | 867,701 | 1,386,918 | 37.44% |

MBS algorithm with $L = 4$, the derived memory architecture consists of 16 memory groups with each group including 1 to 8 memory blocks, and the total number of extra delay elements for message alignment is 248.

The example design, which is implemented in TSMC 0.13 $\mu$m CMOS technology, can operate at 100 MHz and has a core area of 1.89 $mm^2$. The post-layout simulation indicates that the power consumption is 61.3 mW when operated at 100 MHz and that the throughput is 135 Mb/s. The total area requirement expressed in terms of gate count is 280 k, including the memory groups and logic circuit. The proposed decoder design has the smallest core size among related studies. Moreover, the number of required memory address generators and their interconnections is also reduced in the proposed memory access architecture.

A comparison of the proposed LDPC decoder design with related designs is shown in Table 1. Although the specifications of the designs in Table 1 are very different, the proposed design has the lowest area complexity when implemented in the same specification due to a significant reduction in required memory area. In particular, when compared with the design in [8], which has similar specifications as ours, the decoder implemented using the proposed low-complexity memory access architecture can achieve about 33% reduction in gate count. As shown in Table 2, the improvement in the extrinsic and intrinsic memory areas over the conventional design [8] is about 53.46% and 42.87%, respectively. Taking in account the additional FIFOs of size 99,200 $\mu m^2$, the overall improvement in storage units over the conventional design becomes 37.44%. Note that since the intrinsic memory is only used for VNUs, the problem arises in merging extrinsic memory blocks does not exist in the merged intrinsic memory. Thus we can merge all the intrinsic memory blocks without introducing additional FIFOs. Moreover, from the normalized power efficiency (NPE) defined in [16]:

$$\text{NPE} = \frac{\text{Throughput Rate}}{\text{Power Consumption}} \times \left(\frac{\text{Technology}}{0.13\,\mu\text{m}}\right)^2$$

one can show that the proposed design is almost twice as efficient as that in [8]. NPE evaluates the throughput rate a LDPC decoder can obtain per unit of power consumption. As a result, the proposed merged memory access architecture also leads to a power-efficient solution. The work [13] focuses on designing a high-throughput multi-rate QC-LDPC decoder by adopting dual-path and fully-overlapped techniques to improve its throughput rate. The design in [13] thus has higher throughput rates than the proposed one because it can deal with the nonzero matrices from two rows at the same time. This implies that the high throughput rate is achieved at the price of more hardware requirements. On the contrary, the proposed design, which tries to merge small memory blocks into large ones, is targeted at designing a low-complexity LDPC decoder based on the proposed memory merging technique. As a result, the area complexity of our design is smaller than that of [13].

Since additional FIFOs are inserted in our design, the VNU takes $(Z + 2L)$ instead of $Z$ cycles to complete the operations in the variable-node phase. For $Z = 96$ and $L = 4$, this results in about $2L/(2Z) = 4\%$ degradation in the overall throughput performance as compared to the conventional design assuming that the same number of computational cycles are needed in both the check-node and variable-node phases. The reason why the throughput rate of [3] is much higher than ours is that a fully parallel architecture is considered in their design. Similarly, the partially parallel architecture with overlapped check-node and variable-node phases in [7] incorporates more processing elements than ours does to increase its throughput rate. Note that since the main focus of this work is on exploring the benefit obtained from merging memory blocks, the prototype design

adopts the non-overlapped message passing scheme for ease of explanation. Note that the proposed method can be easily extended to the overlapped message passing design also. Recently, several LDPC decoding architectures suitable for multi-rate applications have been presented in [11]–[14]. The proposed method can also be applied to support all 114 modes in such an application to further reduce the hardware requirement.

To determine the effect of $L$ on the number of required delay elements and the partition of memory groups, the area consumptions of both FIFOs and memory groups, reported from the Artisan memory compiler, was plotted as a function of $L$ for comparison. Figure 7 plots the synthesized results of FIFO area, extrinsic memory area, intrinsic memory area, and their summation using the same design parameters as those described above. The figure reveals that the required FIFO area increases almost linearly with the value of $L$. This is due to the memory groups generated from the MBS algorithm using large $L$ possibly resulting in a large value of relative block delay defined in (11), thereby leading to higher FIFO requirements. The trend of required memory area is similar to that in Fig. 1 because more memory blocks are included in each memory group for large $L$. As a result, the advantage of merging memory blocks is offset by the additional FIFOs for large values of $L$. Note that the intrinsic memory area is independent of $L$ since the original intrinsic memory can be merged directly without the aid of the MBS algorithm. The MBS algorithm only deals with merging proper extrinsic memory blocks to minimize the total FIFO requirement. From Fig. 7, a feasible choice of $L$ for minimizing the area requirement of the present LDPC decoder design ranges from 3 to 6. These values of $L$ are usually much smaller than the block size of the parity-check matrices of QC-LDPC codes in practical applications so that the degradation in throughput performance can be negligible using the proposed method. For the special case when $L$ is equal to zero, each memory group contains only one kind of $SA_v(M_{j,k})$ after the MBS algorithm is applied; thus, no additional FIFOs are required in the proposed architecture. This implies that no throughput degradation exists when $L = 0$.

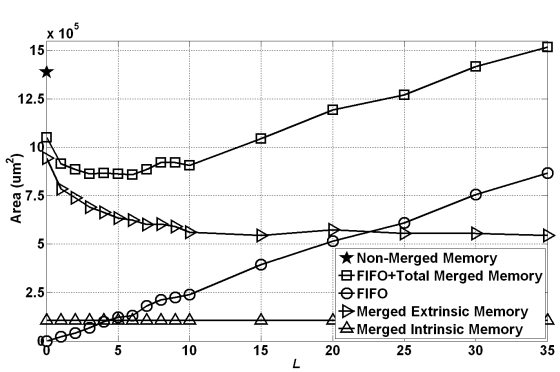


**Fig. 7** Areas of required FIFO and memory groups for the code rate-1/2 parity check matrix defined in IEEE 802.16e systems with a block size of $Z = 96$.

## 6. Conclusion

A low-complexity memory access architecture was presented for the design and implementation of QC-LDPC decoders. In the proposed architecture, smaller memory blocks are merged into memory groups to reduce the impact of peripherals inside the memory. The conflict of multiple message accesses that results from merging memory blocks is successfully resolved by adding buffers to synchronize the retrieved messages. An efficient memory block selection algorithm was also proposed to minimize the required buffer size. Experiment results show that the QC-LDPC decoder designed using the proposed memory access architecture significantly reduces the area requirement and also provides greater power efficiency than that achieved in related studies.

## Acknowledgments

This research described in this paper was supported in part by the National Science Council of R.O.C. under contract NSC 96-2221-E-006-296-MY3.

### References

[1] R.G. Gallager, "Low-density parity-check codes," IRE Trans. Inf. Theory, vol.IT-8, no.1, pp.21–28, Jan. 1962.
[2] R. Tanner, "A recursive approach to low complexity codes," IEEE Trans. Inf. Theory, vol.27, no.5, pp.533–547, Sept. 1981.
[3] 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.
[4] N. Onizawa, T. Hanyu, and V.C. Gaudet, "Design of high-throughput fully parallel LDPC decoders based on wire partitioning," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.18, no.3, pp.482–489, March 2010.
[5] Y. Cheng and K.K. Parhi, "Overlapped message passing for quasi-cyclic low-density parity check codes," IEEE Trans. Circuits Syst. I, Regular Papers, vol.51, no.6, pp.1106–1113, June 2004.
[6] Y. Dai, Z. Yan, and N. Chen, "Optimal overlapped message passing decoding of quasi-cyclic LDPC codes," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.16, no.5, pp.565–578, May 2008.
[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, Regular Papers, vol.53, no.5, pp.1045–1056, May 2006.
[8] X.Y. Shih, C.Z. Zhan, and A.Y. Wu, "An 8.29 mm$^2$ 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 $\mu$m CMOS process," IEEE J. Solid-State Circuits, vol.43, no.3, pp.672–683, March 2008.
[9] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," IEEE J. Sel. Areas Commun., vol.27, no.6, pp.985–994, Aug. 2009.
[10] J. Jin and C.Y. Tsui, "An energy efficient layered decoding architecture for LDPC decoder," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.18, no.8, pp.1185–1195, Aug. 2010.
[11] L. Zhang, L. Gui, Y. Xu, and W. Zhang, "Configurable multi-rate decoder architecture for QC-LDPC codes based broadband broadcasting system," IEEE Trans. Broadcast., vol.54, no.2, pp.226–235, June 2008.
[12] C.H. Liu, S.W. Yen, C.L. Chen, H.C. Chang, C.Y. Lee, Y.S. Hsu, and S.J. Jou, "An LDPC decoder chip based on self-routing network

for IEEE 802.16e applications," IEEE J. Solid-State Circuits, vol.43, no.3, pp.684–694, March 2008.
[13] B. Xiang and X. Zeng, "A 4.84 $mm^2$ 847-955 Mb/s 397 mW dual-path fully-overlapped QC-LDPC decoder for the WiMAX system in 0.13 $\mu$m CMOS," Proc. IEEE Symp. VLSI Circuits (VLSIC), pp.211–212, June 2010.
[14] D. Oh and K.K. Parhi, "Low-complexity switch network for reconfigurable LDPC decoders," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.18, no.1, pp.85–94, Jan. 2010.
[15] Artisan Components, Sunnyvale, CA, "Artisan standard library register file generator user manual," 2004.
[16] H.Y. Hsu, J.C. Yeo, and A.Y. Wu, "Multi-symbol-sliced dynamically reconfigurable reed-solomon decoder design based on unified finite-field processing element," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.14, no.5, pp.489–500, May 2006.

**Ming-Der Shieh** received the B.S. degree in electrical engineering from National Cheng Kung University, in 1984, the M.S. degree in electronic engineering from National Chiao Tung University, Taiwan, in 1986, and the Ph.D. degree in electrical engineering from Michigan State University, East Lansing, in 1993. From 1988 to 1989, he was an engineer at United Microelectronic Corporation, Taiwan. From 1993 to 2002, he was with the faculty of Department of Electronic Engineering, National Yunlin University of Science & Technology. He received the teaching award in 1998 and was the department chairman from 1999 to 2002. Since 2002, he has been with the Department of Electrical Engineering, National Cheng Kung University, where he is currently a professor. His research interests include VLSI design and testing, VLSI for signal processing, and digital communication. He was the program co-chair and general co-chair of Asian Test Symposium in 2004 and 2009, respectively, and the chair of Tainan Chapter of IEEE Circuits and Systems from 2009 to 2010. He is now the associate editor of IEEE Transaction on Circuits and Systems: Part I.

**Shih-Hao Fang** received the B.S. degree in electronic engineering from National Kaohsiung University, Taiwan, in 2005. He is pursuing the Ph.D. degree from National Cheng Kung University, Taiwan since 2006. His primary research areas were blind channel estimation, channel coding, and MIMO detection. He is now the chair of IEEE student branch at National Cheng Kung University.

**Shing-Chung Tang** received the B.S. degree in electronic engineering from National Kaohsiung University, Taiwan, in 2007, and the M.S. degree in electrical engineering from National Cheng Kung University, Taiwan, in 2009. He is currently a system engineer at Himax Technology, Taiwan. His primary research areas include hardware design of channel coding and OCP protocol.

**Der-Wei Yang** received the B.S. and M.S. degree in electronic engineering from National Cheng Kung University, Taiwan, in 2007 and 2008, respectively. He is pursuing the Ph.D. degree from National Cheng Kung University, Taiwan. His research interests include hardware design of channel coding, system integration and verification, and 3D stereo video processing.