Area-Efficient QC-LDPC Decoder Architecture Based on Stride Scheduling and Memory Bank Division

Bongjin KIM†, Nonmember and In-Cheol PARK†(a), Member

SUMMARY In this paper, an area-efficient decoder architecture is proposed for the quasi-cyclic low-density parity check (QC-LDPC) codes specified in the IEEE 802.16e WiMAX standard. The decoder supports all the code rates and codeword lengths defined in the standard. In order to achieve low area and maximize hardware utilization, the decoder utilizes 4 decoding function units, which is the greatest common divisor of the expansion factors. In addition, the decoder adopts a novel scheduling scheme named stride scheduling, which stores the extrinsic messages in non-sequential order to replace the conventional complex flexible permutation network with simple small-sized cyclic shifters and also minimize the number of memory accesses. To further minimize the complexity, the number of extrinsic memory instances for 24 block columns is reduced to 5 banks by identifying independent sets. All the memory instances used in the decoder are single-port memories which cost less area and price compared to dual-port ones. Finally, the decoding function units have partially parallel structure to make the decoding throughput sufficiently over the requirement of the WiMAX standard. The proposed decoder is synthesized with 49 K equivalent gates and 54,144 bits of memory, and the implementation occupies 0.40 mm² in a 65 nm CMOS technology.

key words: low-density parity-check (LDPC) codes, multi-mode decoder, low-area architecture, stride scheduling

1. Introduction Low-density parity-check (LDPC) codes, introduced in the early 1960s by R. Gallager [1], have not received much attention for over thirty years. However, since the rediscovery of the LDPC codes by Mackay [2], LDPC codes have become one of the most attractive topics both in academia and industry. In addition to excellent error correcting capability near the Shannon’s limit, LDPC codes can be decoded iteratively with low complexity, and provide high parallelism inherently. Due to these benefits, LDPC codes have been adopted in several wireless communication standards such as IEEE 802.11n (Wireless Local Area Network, WLAN) [3], IEEE 802.16e (Wireless Metropolitan Area Network, WiMAX) [4], and Digital Video Broadcasting (DVB-S2) [5].

Despite of the advantage of high parallelism, the fully parallel architecture [6] cannot be regarded as an appropriate solution for modern mobile applications because large memory and complex interconnection are required. Instead, the partially parallel architecture [7]–[9] has been considered as the tradeoff between the hardware complexity and the high throughput. Furthermore, the introduction of architecture-aware codes such as quasi-cyclic (QC) LDPC codes makes the partially-parallel decoders more suitable. The parity check matrices (PCMs) of QC-LDPC codes are composed of submatrices in such a way that all the rows within a submatrix can be processed in parallel. Moreover, the QC-LDPC code simplifies the interconnections among the processing units and the control of memory access. Partially parallel decoders can take these advantages in their implementations.

The QC-LDPC codes defined in the IEEE 802.11n and IEEE 802.16e standards support diverse codewords and code rates by making the submatrix size, which is called the expansion factor, expandable and providing several PCMs. Therefore, these wireless standards demand multi-mode decoders. In order to support multiple decoding cases while keeping the throughput high, the multi-mode decoder has been designed with a flexible permuter network as well as a large number of processor units [10], [11], leading to large area and power consumption.

In this paper, we present an area-efficient multi-mode decoder architecture for the QC-LDPC codes defined in the IEEE 802.16e standard. The proposed decoder includes a small number of processing units to reduce the hardware complexity, and employs the pipeline techniques to increase the decoding throughput. In addition, the check nodes are processed in a non-sequential order instead of the conventional sequential order. The non-sequential processing of check nodes, which is called stride scheduling in this paper, makes it possible to replace the complex flexible permuter network with simple cyclic shifters. Furthermore, the number of extrinsic memory banks is reduced by finding independent sets, which enables the decoder to use a small number of on-chip memory instances and plays a critical role in reducing the decoder area.

The rest of the paper is organized as follows. Section 2 reviews the QC-LDPC codes defined in the WiMAX standard and several related works. Section 3 proposes a new decoding architecture and describes how to achieve an area-efficient implementation. The implementation results are addressed in Sect. 4 and final remarks are made in Sect. 5.

2. Preliminaries

2.1 QC-LDPC Codes in IEEE 802.16e

The LDPC codes specified in the IEEE 802.16e standard...
are QC-LDPC codes [4]. As described above, the PCM of a QC-LDPC code is composed of submatrices each of which is a cyclically shifted identity matrix or a zero matrix. Due to the submatrix structure, all the rows within a layer or a block row that is as large as the submatrix size are independent so that they can be processed simultaneously without causing any conflict. This can be regarded as a great benefit to the partially parallel decoders and also to the layered decoding [12]. For the case of rate 1/2 and 2/3 codes, block rows can be rearranged such that adjacent block rows do not share any common columns. Such a rearrangement can make the layered decoding faster by processing the block rows in a pipelined manner.

The submatrix size is determined by the expansion factor $z_f$, which is a multiple of 4 and ranges from 24 to 96. Regardless of the code rate, every PCM is composed of 24 block columns, and thus the codeword length ranges from 576 to 2304. Figure 1 shows one of the base PCMs used in the IEEE 802.16e standard. The number inside the PCM denotes the shift amount of the identity matrix at the corresponding position, assuming the submatrix size is maximum, that is, $z_f = 96$. The maximum shift amount varies in proportional to the expansion factor. In addition, an empty submatrix indicates that a zero matrix is used at the position.

Table 1 summarizes the PCM sizes, and the row and column weights of all the QC-LDPC codes defined in the WiMAX, where $M_b$ and $N_b$ represent the number of block rows and block columns, respectively. The row weight is the number of variable nodes connected to a row, and the column weight is the number of check nodes relevant to a column.

### 2.2 Previous Works

To achieve high throughput, the fully parallel architecture [6] instantiates as many check-node units and variable-node units as the number of rows and columns of the PCM. As the number of decoding function units (DFUs) is large, however, the interconnection among them becomes significantly complex. Therefore, such a fully parallel decoder consuming huge area and power is inappropriate for portable applications.

The partially parallel architecture is widely employed as a practical solution. The number of DFUs employed in the conventional partially parallel decoder [11] is as many as the largest expansion factor, which is 96 in the WiMAX standard, in order to cope with multi-mode decoding operations. A unified architecture that can perform both the layered decoding [12] and the two-phase message passing (TPMP) [1] was proposed in [10]. Unlike the decoder in [11], this architecture integrates 96 variable-node units (VNUs) and 96 check-node units (CNUs), leading to large area. The decoder presented in [21] adopts 96 CNUs and 192 VNUs for dual-path decoding, and the decoder in [20] adopts 96 bit-serial DFUs in order to avoid the interconnection problem.

Furthermore, all the decoders include flexible permutation networks to support various input size and shift values. Logarithmic barrel shifters are used in [10], [20], and [21], where the outputs from two barrel shifters with an input size of 96 are chosen according to the proper expansion factor and shift amount. The numbers of 1-bit 2-to-1 multiplexers (MUXs) required for the shifters in [20] and [21] are 60,480 and 7,448, respectively. The shifters are replaced with a Benes network in [11]. Since the input size is as large as 96, the Benes network is still complex, as illustrated in Fig. 2. Assuming that a switch is composed of two 2-to-1 MUXs, the recursive structure of the 2^n-sized Benes network consists of numerous MUXs [13]. If the input size of the network is 128 and a word size is 6 bits, for example, the Benes network requires 9,884 1-bit MUXs. In addition to the above works, most of the previous works were implemented based on the flexible permuting network [14], [15]. As the maximum expansion factor of the WiMAX or WLAN standards is large, the network has a crucial influence on the overall area.

A decoder integrating 4 check-node units and 8 variable-node units was presented in [16]. Due to the relatively small number of processing units, the decoder does not necessitate a flexible permutation network. Though it is designed with low parallelism, it supports only a single mode, and uses numerous single-port and two-port register files instead of on-chip memories. More precisely, it inte-
grates 24 single-port register files to store the log-likelihood ratio (LLR) information and 76 two-port register files to store the intermediate messages. These register files make the decoder area large. As register files and on-chip memories take a large portion of the whole area, the number of storage instances should be minimized to achieve an area-efficient decoder. In the proposed architecture, the area cost is dramatically reduced by replacing the permuting network with simple cyclic shifters and reducing the number of storage instances.

3. Proposed Decoder Architecture

3.1 Overall Architecture

In minimizing the decoder area, it is critical to reduce the number of DFUs or the decoding parallelism. Besides, a small number of DFUs provides a good opportunity to reduce the size of shifting networks. Since the required throughput of the WiMAX standard is lower than 40 Mbps [17], it is clear that the parallelism does not need to be extremely high. For instance, let us consider the shortest code [17], it is clear that the parallelism does not need to be extremely high. For instance, let us consider the shortest code in which the expansion factor is 24 and the code rate is 1/2. Assuming that the maximum allowable iteration (MAI) is 10 and the clock period is 3 ns, the decoding throughput can be approximately estimated as follows:

\[
\text{Throughput} = \frac{z_f \times N_b \times R}{\text{MAI} \times M_b \times \text{clk}_{\text{sys}} \times T_{\text{CLK}}} \approx \frac{24 \times 24 \times 1/2}{10 \times 12 \times 24 \times \frac{7}{P_{\text{DFU}}} \times 3 \text{ ns}} \quad (1)
\]

where \(P\) is the number of DFUs integrated in the decoder, and \(P_{\text{DFU}}\) is the number of columns that a DFU can handle in a cycle, that is, the number of input data that a DFU can accept in a cycle. Therefore, the number of sub-iterations taken for a single layer of 24 rows is \(24/P\), and the number of cycles taken for a sub-iteration is \(7/P_{\text{DFU}}\) if all the rows have an equal weight of 7.

Under the constraint that the maximum required throughput is less than 40 Mbps, the product \(P \cdot P_{\text{DFU}}\) can be less than 10. Since the estimation is a rough approximation and it is undesirable to make the throughput close to the minimum requirement, the product is slightly loosened. Considering that the expansion factor \(z_f\) varies from 24 to 96 with a step size of 4, we set \(P\) to 4, which is the greatest common divisor (GCD) of the possible expansion factors. The GCD is a reasonable choice in the viewpoint of hardware utilization. If \(P\) is different from the GCD, some DFUs would be idle for some expansion factors.

According to the determined parallelism and the distribution of row weights, \(P_{\text{DFU}}\) is set to 4. As defined above, \(P_{\text{DFU}}\) is the number of column data accessed in a cycle by a DFU. In the previous works, the DFU is classified into two groups according to the number of data accessed in a cycle: serial-input DFU [11] and parallel-input DFU [16]. The serial-input DFU reads column data serially one at a time, whereas the number of column data accessed by the parallel-input DFU is as large as the entire row weight. For the serial-input DFU, \(P_{\text{DFU}}\) is 1. In order to provide a reasonable tradeoff between the throughput and the decoder area, the proposed decoder is designed with partially parallel DFUs that can process 4 input data in a cycle.

With the determined design parameters, the proposed decoder implements the layered decoding [6] based on the min-sum algorithm [18]. The layered decoding is one of the most widely used scheduling methods, since it can achieve almost twice faster speed of convergence compared to the traditional TPMP scheduling. The min-sum algorithm can reduce the decoding complexity at the cost of a little degradation of error correcting capability by approximating the update rule of the original sum-product algorithm (SPA). In addition, due to the approximation, the min-sum algorithm requires to store only the minimum and second minimum magnitudes of the variable-to-check (V2C) messages, the index of the minimum term, and the sign values of the check-to-variable (C2V) messages. As it can reduce the size of the intermediate storage, it is specifically effective in the design of a multi-mode decoder in which the row weight varies from case to case.

The proposed decoder employs a 4-stage pipelined structure consisting of read, pre-DFU, DFU, and write-back (WB) stages. The overall architecture is illustrated in Fig. 3, where the pipeline-stage boundaries are denoted by the horizontal shaded lines. As a partially parallel DFU processes 4V2C messages at a time, there are 4 paths from the a posteriori probability (APP) memories and also from the C2V memory. If the required maximum allowable iteration (MAI) is 10 and the clock period is 3 ns, the decoding throughput can be approximately estimated as follows:

\[
\text{Throughput} = \frac{z_f \times N_b \times R}{\text{MAI} \times M_b \times \text{clk}_{\text{sys}} \times T_{\text{CLK}}} \approx \frac{24 \times 24 \times 1/2}{10 \times 12 \times 24 \times \frac{7}{P_{\text{DFU}}} \times 3 \text{ ns}} \quad (1)
\]

where \(P\) is the number of DFUs integrated in the decoder, and \(P_{\text{DFU}}\) is the number of columns that a DFU can handle in a cycle, that is, the number of input data that a DFU can accept in a cycle. Therefore, the number of sub-iterations taken for a single layer of 24 rows is \(24/P\), and the number of cycles taken for a sub-iteration is \(7/P_{\text{DFU}}\) if all the rows have an equal weight of 7.

Under the constraint that the maximum required throughput is less than 40 Mbps, the product \(P \cdot P_{\text{DFU}}\) can be less than 10. Since the estimation is a rough approximation and it is undesirable to make the throughput close to the minimum requirement, the product is slightly loosened. Considering that the expansion factor \(z_f\) varies from 24 to 96 with a step size of 4, we set \(P\) to 4, which is the greatest common divisor (GCD) of the possible expansion factors. The GCD is a reasonable choice in the viewpoint of hardware utilization. If \(P\) is different from the GCD, some DFUs would be idle for some expansion factors.

According to the determined parallelism and the distribution of row weights, \(P_{\text{DFU}}\) is set to 4. As defined above, \(P_{\text{DFU}}\) is the number of column data accessed in a cycle by a DFU. In the previous works, the DFU is classified into two groups according to the number of data accessed in a cycle: serial-input DFU [11] and parallel-input DFU [16]. The serial-input DFU reads column data serially one at a time, whereas the number of column data accessed by the parallel-input DFU is as large as the entire row weight. For the serial-input DFU, \(P_{\text{DFU}}\) is 1. In order to provide a reasonable tradeoff between the throughput and the decoder area, the proposed decoder is designed with partially parallel DFUs that can process 4 input data in a cycle.

With the determined design parameters, the proposed decoder implements the layered decoding [6] based on the min-sum algorithm [18]. The layered decoding is one of the most widely used scheduling methods, since it can achieve almost twice faster speed of convergence compared to the traditional TPMP scheduling. The min-sum algorithm can reduce the decoding complexity at the cost of a little degradation of error correcting capability by approximating the update rule of the original sum-product algorithm (SPA). In addition, due to the approximation, the min-sum algorithm requires to store only the minimum and second minimum magnitudes of the variable-to-check (V2C) messages, the index of the minimum term, and the sign values of the check-to-variable (C2V) messages. As it can reduce the size of the intermediate storage, it is specifically effective in the design of a multi-mode decoder in which the row weight varies from case to case.

The proposed decoder employs a 4-stage pipelined structure consisting of read, pre-DFU, DFU, and write-back (WB) stages. The overall architecture is illustrated in Fig. 3, where the pipeline-stage boundaries are denoted by the horizontal shaded lines. As a partially parallel DFU processes 4V2C messages at a time, there are 4 paths from the a posteriori probability (APP) memories and also from the C2V memory. If the required maximum allowable iteration (MAI) is 10 and the clock period is 3 ns, the decoding throughput can be approximately estimated as follows:

\[
\text{Throughput} = \frac{z_f \times N_b \times R}{\text{MAI} \times M_b \times \text{clk}_{\text{sys}} \times T_{\text{CLK}}} \approx \frac{24 \times 24 \times 1/2}{10 \times 12 \times 24 \times \frac{7}{P_{\text{DFU}}} \times 3 \text{ ns}} \quad (1)
\]

where \(P\) is the number of DFUs integrated in the decoder, and \(P_{\text{DFU}}\) is the number of columns that a DFU can handle in a cycle, that is, the number of input data that a DFU can accept in a cycle. Therefore, the number of sub-iterations taken for a single layer of 24 rows is \(24/P\), and the number of cycles taken for a sub-iteration is \(7/P_{\text{DFU}}\) if all the rows have an equal weight of 7.

Under the constraint that the maximum required throughput is less than 40 Mbps, the product \(P \cdot P_{\text{DFU}}\) can be less than 10. Since the estimation is a rough approximation and it is undesirable to make the throughput close to the minimum requirement, the product is slightly loosened. Considering that the expansion factor \(z_f\) varies from 24 to 96 with a step size of 4, we set \(P\) to 4, which is the greatest common divisor (GCD) of the possible expansion factors. The GCD is a reasonable choice in the viewpoint of hardware utilization. If \(P\) is different from the GCD, some DFUs would be idle for some expansion factors.

According to the determined parallelism and the distribution of row weights, \(P_{\text{DFU}}\) is set to 4. As defined above, \(P_{\text{DFU}}\) is the number of column data accessed in a cycle by a DFU. In the previous works, the DFU is classified into two groups according to the number of data accessed in a cycle: serial-input DFU [11] and parallel-input DFU [16].
memories. Note that the de-permuter is excluded by using the same strategy as [11]. The usage of the read buffer and write buffer is addressed in the following subsection.

### 3.2 Stride Scheduling and Memory Structure

In the proposed decoder, the number of the inputs and outputs of the permuter is significantly reduced by employing small parallelism. The complexity of the permuter can be further reduced by applying a special scheduling. The scheduling in the decoding process is how to assign the rows within a layer to the DFUs. In the sequential scheduling which has been used conventionally, the assignment is interleaved [10], [14]. In other words, the top row, i.e., the 0-th row, is assigned to DFU0, the first row to DFU1 and so forth. Then, the fourth row is again assigned to DFU0, if the number of DFUs is 4. In general, DFU_i is in charge of the rows that have a remainder of i when the row indices are divided by P. The conventional sequential scheduling is depicted in Fig. 4, where the submatrix has a shift amount of 13. In Fig. 4(a), the submatrix consists of the data corresponding to rows 0, 6, 12 and 18. Figure 5(b) shows such a memory configuration, where an entry contains all the data corresponding to a group. More precisely, the i-th entry of the APP memory contains the data of row indices \(i, i + z_i/P, i + 2z_i/P, \) and \(i + 3z_i/P, \) where \( P \) is 4. This memory arrangement guarantees that the data being processed at the same time are stored into one entry, and this property is held independent of the cyclic shift amount of the submatrix. Therefore, the proposed stride scheduling makes it possible to access all the data required in the partially parallel processing at once, which is clearly an advantage over the conventional sequential scheduling.

In the viewpoint of practical implementation, there is another factor to be considered. During the process of a check node, the pipelined architecture shown in Fig. 3 accesses the APP memory two times; one for reading and the other for writing. As the two accesses are overlapped in the pipelined process, two access ports are required for the APP memory if implemented straightforwardly. Therefore, a simple method to implement the APP memory is to use the two-port memory. However, the two-port memory is associated with much higher cost than the single-port memory is. To reduce this overhead, a single-port memory is adopted but its entry size is enlarged twice to include two data groups. In case of 4 DFUs, for example, an entry of the APP memory is enlarged to store 8 data by merging two adjacent entries into one, as shown in Fig. 6. Two data sets of rows within a layer, which means that the indices of two adjacent check nodes being processed at the same time have the same stride of \( z_i/P \). This is the reason why the proposed scheduling is called stride scheduling.

To support the stride scheduling, the APP memory should store all the data to be accessed at the same time into one entry. The data being processed together can be considered as a group. There are \( z_i/P \) groups in a layer, and each group can be indexed by the modular value of the row indices. All the data in a group have the same remainder if the row index is divided by \( z_i/P \). For example, group 0 in Fig. 5(a) consists of the data corresponding to rows 0, 6, 12 and 18. Figure 5(b) shows such a memory configuration, where an entry contains all the data corresponding to a group. More precisely, the i-th entry of the APP memory contains the data of row indices \( i, i + z_i/P, i + 2z_i/P, \) and \( i + 3z_i/P, \) when \( P \) is 4. This memory arrangement guarantees that the data being processed at the same time are stored into one entry, and this property is held independent of the cyclic shift amount of the submatrix. Therefore, the proposed stride scheduling makes it possible to access all the data required in the partially parallel processing at once, which is clearly an advantage over the conventional sequential scheduling.

![Fig. 4](image1.png) Conventional scheduling. (a) Interleaved assignment of DFUs and (b) the APP memory configuration.

![Fig. 5](image2.png) Proposed stride scheduling. (a) Subsets each of which assigned to a DFU and (b) the APP memory configuration according to the group index.
are read in a cycle, and two data sets are written in the next cycle, which means that the read and write accesses are interleaved. As only one data set can be processed in a cycle, the other data set is temporarily stored into a buffer, as depicted in Fig. 6, where two buffers, a read buffer and a write buffer, are inserted to support the interleaved access. The proposed memory structure resolves the memory access conflict without employing the costly two-port memory. Based on our analysis, the area resolves the memory access conflict without employing the costly two-port memory. Based on our analysis, the area used to store the APP messages of the variable nodes [10].

As a straightforward approach, 24 on-chip memories can be used to store the APP messages of the variable nodes [10]. This method enables to simultaneously access all the APP messages that a row needs at any moment of the decoding process. However, integrating many memory modules leads to a large chip area because of the peripheral circuitry of a memory module such as row and column address decoders.

In the proposed decoder, each DFU has a partially parallel structure that accepts 4 input messages at a time. Therefore, at least 4 memory banks are required to make a DFU simultaneously access 4 messages coming from different variable nodes. Since the maximum row weight is dependent on the code rate, which is 7, 11, 15, or 20 in the WiMAX standard, a DFU takes at least 2, 3, 4, or 5 cycles to process a check node, respectively. The objective of memory bank division is to minimize the number of memory modules, while maintaining the number of memory accesses as small as possible. The number should be greater than or equal to 4, as mentioned above. The APP memory bank division can be achieved by finding independent sets in a graph.

In a graph $G(V, E)$, where $V$ is the set of all vertices and $E$ is the set of all edges, two vertices, $v$ and $w$, are called adjacent if $(v, w) \in E$. Let us consider a subset $S$ of $V$. The subset $S$ is called an independent set if $(v, w) \notin E$ for all $v, w \in S$ [19]. Figure 7(a) shows an example graph and an independent set of the graph. There is no direct edge between vertices 1, 6, 9, and 11 so that the vertices form an independent set in the graph.

The PCMs in the WiMAX standard can be represented in graphs. Every block column is mapped to a vertex. If two block columns are relevant to a block row, an edge connects the two vertices corresponding to the two block columns. Figure 7(b) depicts the graphical representation of the rate 1/2 WiMAX code. To avoid complex drawing, only the edges that have an endpoint at vertices 0, 3, and 6 are shown. It can be easily observed that there is no direct connection between vertices 0, 3, and 6. As a result, the three vertices can form an independent set.

Finding an independent set from a given graph can be accomplished by applying the recursive method presented in [19]. Let $v \in V$, and $A(v)$ be the set of vertices that are adjacent to $v$. Since any independent set either contains $v$ or does not, any independent set of $G$ is either $[v]$ combined with an independent set in $G(V-[v]-A(v))$ or an independent set in $G(V-[v])$. In Fig. 7(b), after selecting vertex 0 first and excluding all the adjacent vertices, vertex 3 can be chosen as the next entry. Excluding all vertices adjacent to vertex 3 among the remains, vertex 6 can be the final entry. As no vertex remains after excluding the vertices adjacent to vertex 6, an independent set is formed with vertices 0, 3, and 6.

By the definition of independent sets and the correspondence to the PCM, only one among the block columns in an independent set can be associated with the decoding process of a block row. In other words, a block row can be related to only one of the block columns included in an independent set. Therefore, the APP messages of the block columns in an independent set can be stored in the same
memory bank. In addition, as multiple cycles are required to process a row, multiple independent sets can be stored in the same memory bank as long as the independent sets do not have to be accessed at the same time. For example, in order to ensure a row to be processed in 2 cycles for the rate 1/2 code, at most 2 independent sets can be stored into the same memory bank.

Considering all the PCMs in the IEEE 802.16e standard, 24 block columns are successfully grouped into 5 memory banks. Regardless of the code rate, the APP memory banks can be configured in a fixed way, which simplifies the control of the decoder. Table 2 shows how the block columns are divided into 5 memory banks. Processing a row requires 2, 3, 4, and 5 reads when the code rate is 1/2, 2/3, 3/4, and 5/6, respectively.

The proposed memory bank division can save area significantly compared to conventional 24 memory banks. In our implementation with a 65 nm CMOS technology, the total area of the APP memories is 0.068 mm$^2$ which is 17.2% of the whole decoder area. The 24-banked APP memories would cost 0.268 mm$^2$, which is four times larger than our result. Since the number of DFU inputs is 4 in the proposed decoder, 4 or less memory banks are accessed at a time throughout the decoding process. A particular block column can be easily found in the proposed APP memory banks by using a small-sized look-up table that can be realized with a negligible overhead.

### 3.4 Partially Parallel DFU

The DFUs are responsible for finding the minimum (min) and the second minimum (min 2) values in magnitude among the input messages as well as finding the index of the minimum value and the sign values of all the outgoing messages. As mention above, in the proposed decoder, a DFU takes a partially parallel structure that processes 4V2C messages at a time. The partially parallel DFU can have several advantages over the conventional structures.

Compared to the fully parallel DFU [16], the partially parallel DFU costs less area and provides more flexibility for multi-mode operations. Due to the variable row weights summarized in Table 1, increasing the number of parallel inputs as large as the maximum row weight leads to huge logic area and low hardware utilization. In addition, the fully parallel DFU would increase the clock period, since it has to compare a large number of input messages in a cycle. On the other hand, the partially parallel DFU can achieve the higher throughput than the single-input serial DFU [11] does. The hardware complexity of the proposed decoder would be comparable to that of the decoder based on the single-input DFU. If the single-input serial DFUs are used, more DFUs have to be installed to assure the required throughput. Therefore, the proposed partially parallel DFU can be considered as a good tradeoff between the single-input DFU and the fully parallel DFU.

### 4. Implementation Results

Taking into account the pipeline structure, the decoding throughput can be estimated by (1). Figure 8 shows the throughputs resulting from all the code rates and expansion factors specified in the WiMAX standard. The maximum throughput is 149 Mbps, which is achieved when the code rate is 5/6 and the codeword length is maximal. Note that the throughput of the proposed decoder is sufficiently higher than the requirement of the standard for all the cases. The number of clock cycles per iteration varies from 144 to 624. Owing to the careful memory bank division, the number of cycles taken for a block row of the rate 2/3A code is identical to that of the 2/3B code. The throughputs of the two codes, therefore, are exactly the same. For the same reason, the throughput of rate 3/4A code is identical to that of 3/4B code.

During the decoding iterations, the decoder processes each layer separately, which means that the reading step of the next layer waits until the previous layer finishes its updates. As a result, the error correcting performance of the proposed decoder is not affected by the layer overlap problem. In case of the pipelined decoders that overlap the layer processes, a block column involved in two adjacent layers would cause memory access conflict, since the write operation of the first layer and the read operation of the second layer can be scheduled at the same time. This conflict degrades the decoding performance as a result. To separate the block row operations, 6, 8, and 10 pipeline bubbles are inserted between them, when the code rate is 2/3, 3/4, and 5/6, respectively.

In case of the rate 1/2 codes, the throughput expected by (1) varies from 49 Mbps to 69 Mbps, which is lower than the result shown in Fig. 8. To enhance the throughput of the rate 1/2 codes, the processing order of layers is altered in
such a way that adjacent two block rows do not share any common block column. This enables the decoder to overlap the processes of adjacent layers, or to start the next layer without waiting the previous one ends. Due to the stride scheduling and the APP memory structure, pipeline bubbles have to be inserted when the expansion factor is an odd multiple of 4, which results in the throughput degradation.

For the timing and area analysis, the proposed decoder is described in a hardware description language and realized with a 65 nm CMOS library. The maximum number of iteration is set to 10, considering both the decoding throughput and the error correcting performance. The maximum operating clock frequency is 400 MHz, i.e., $T_{CLK}$ is 2.5 ns. The synthesized gate count of the decoder including the datapath and control circuits is 49 K, and the design core occupies 0.40 mm² as shown in Fig. 9.

The decoder uses 54 Kbits of on-chip memory by employing 7-bit quantization for the APP messages and 5-bit quantization for the internal messages. As addressed in Sect. 3.3, 5 on-chip memories are used for the APP messages. The C2V messages that are associated with the decoding process at a certain moment always belong to the same block row. Therefore the C2V memory does not necessitate the bank division. As a result, a line of the C2V memory contains 4 C2V messages. The implementation includes two C2V memories, which is merely caused by the fact that the CMOS library used does not support the required width.

The implementation result is summarized in Table 3 and compared with several previous works. Although the proposed decoder is not more energy-efficient than the bit-serial decoder in [20], it demonstrates better energy-efficiency than the others and the best area-efficiency including the bit-serial decoder. The proposed decoder uses the smallest number of DFUs and on-chip memories, which significantly contributes to the area reduction. Though [16] uses a small number of DFUs, the decoder is viable only for specific circumstances because of its high parallelized structure, and it is not efficient for multi-mode operations. A low-area multi-mode decoder [14] presented for the WiMAX standard uses a large number of processing units and requires large-sized cyclic shifters. In addition, due to its sequential scheduling, the APP data required for the decoding are retrieved by invoking two read accesses using a double-rate clock. The proposed stride scheduling removes this overhead completely.

5. Conclusion

In this paper, an area-efficient QC-LDPC decoder architecture has been proposed for the IEEE 802.16e standard. To minimize the decoder area while meeting the throughput specification, the number of decoding function units is set to 4 which is the greatest common divisor of expansion factors. In addition, the decoder employs a novel scheduling method called stride scheduling to resolve the memory access problem under various shift amounts. As a result, the conventional flexible permuter network is replaced with simple cyclic shifters. To further reduce the area, the proposed decoder minimizes the number of on-chip mem-

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Implementation results and comparisons.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of DFUs</td>
<td>4</td>
</tr>
<tr>
<td>Operating Frequency (MHz)</td>
<td>400</td>
</tr>
<tr>
<td>Maximum Iterations</td>
<td>10</td>
</tr>
<tr>
<td>Maximum Throughput (Mbps)</td>
<td>148.9</td>
</tr>
<tr>
<td>Logic Gate Count</td>
<td>49K</td>
</tr>
<tr>
<td>(Datapath: 29K, Control: 20K)</td>
<td></td>
</tr>
<tr>
<td>Memory Bits</td>
<td>54,144</td>
</tr>
<tr>
<td>(APP: 16,128, C2V: 38,016)</td>
<td></td>
</tr>
<tr>
<td>Number of On-chip Memory Instances</td>
<td>7</td>
</tr>
<tr>
<td>Core Size (mm²)</td>
<td>0.40</td>
</tr>
<tr>
<td>Normalized Core Size (mm²²)</td>
<td>1.58</td>
</tr>
<tr>
<td>Nor. Area Eff. (Mbpss/mm²)</td>
<td>94.24</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>51</td>
</tr>
<tr>
<td>Normalized Power (mW)</td>
<td>192</td>
</tr>
<tr>
<td>Nor. Power Eff. (pl/bit/iteration)</td>
<td>68.5</td>
</tr>
</tbody>
</table>
ory instances by finding independent sets in a graph. The implementation results show that the proposed architecture achieves the smallest area, and the resulting throughputs are sufficiently higher than the requirement for all the code rates and code lengths specified in the standard.

Acknowledgements

This work was supported in part by the IT R&D program of MOTIE/KEIT [10035152, Energy Scalable Vector Processor-Primary Technology] and by the IC Design Education Center (IDEC).

References


Bongjin Kim received the B.S. and M.S. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2009 and 2010, respectively, where he is currently working toward the Ph.D. degree in the Department of Electrical Engineering, KAIST. His current research interest includes VLSI architectures for error correction codes.

In-Cheol Park received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea, in 1986, and the M.S. and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1988 and 1992, respectively. Since June 1996, he has been with the Department of Electrical Engineering, KAIST, where he is a Professor. Prior to joining KAIST, he was with IBM T.J. Watson Research Center, Yorktown, NY, from May 1995 to May 1996, where he researched on high-speed circuit design. His current research interest includes computer-aided design algorithms for high-level synthesis and VLSI architectures for general-purpose micro-processors.