國立中正大學　　通訊工程學系

專題研究報告

National Chung Cheng University

Department of Communication Engineering

Independent Study Report

適用於 5G NR 接收機的 LDPC Codes 平行處理解碼器之研究

Study on Parallel LDPC Decoder for 5G NR Receiver

專題生：蘇沛錦

Student：Pei-Jin Su

指導教授：李昌明 教授

Advisor：Doctor Chang-Ming Lee

中華民國一百一十年七月

July, 2021

ABSTRACT

LDPC (Low-Density Parity Check) codes have attracted lots of interest in the field of channel coding since they are characterized by the performance close to Shannon limit with high throughput, and the great parallelism in decoding. LDPC codes are suitable for parallel decoding with recursive computation due to sparsity in the parity check matrix. Therefore, LDPC codes have already been considered in several wireless communication standards, such as IEEE 802.11n (Wi-Fi) and IEEE 802.16e (WiMAX), and so on. Comparing with 4G long-term evolution (LTE), the forward-error-correction (FEC) coding of 5G New Radio (NR) gives the promise to fulfill the requirements for high throughput, reliable communication, and low-latency. Instead of Turbo codes, LDPC codes are considered for data channel in 5G NR standards. A design of efficient parallelized decoder for 5G NR is proposed in this report. The key feature of the architecture is the efficient chunk, which stores the variable-to-check messages and reduces the memory space of the check-to-variable messages.

Due to the structure of quasi-cyclic (QC) in 5G NR LDPC codes, we also introduce the parallelization of message computation with column-by-column design in the LDPC decoding for 5G NR. However, the memory access collisions occur in the process of parallel decoding when the base matrix has the same offset of the submatrices in the same row. This problem leads to low throughput and needs additional latency to realize the decoding. In this report, we proposed a pre-procession to assign all columns in the base matrix into different column sets and confirm columns with the same offset would not be assigned in the same set. Through the pre-procession, all columns in different column sets can be computed in parallel, and we also prevent the memory access collision and optimize the throughput. Finally, we defined the memory allocation of the efficient chunk for 5G NR LDPC parallel decoding. To avoid unnecessary memory access collision, the experimental results of pre-procession show that all columns from the parity check matrix of BG1 (the first base graph) for lifting size Z_c = 256 are assigned to three column sets. The case of BG2 (the second base graph) for the same Z_c are with two column sets.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The chapter provides the basic introduction to a digital communication system and our motivation for the research on parallelized 5G NR LDPC decoder. The most important process for a digital communication system consists of source coding, channel coding, and modulation. The channel coding scheme is used to protect the desired messages from transmission error in a noisy channel. The LDPC codes belong to one type of error control code, and they are considered for channel coding in the 5G NR standard. The quasi-cyclic structure of LDPC codes is especially suitable for parallelized decoding under hardware implementation. The purpose of the research is to introduce the efficient chunk mechanism and column-by-column computing method in the design of a parallel LDPC decoder for a 5G receiver.

1.1 Digital Communication System

A digital communication system represents the procedure for a sequence of digital messages conveyed in a point-to-point or point-to-multipoint channel. Figure 1-1 shows the system is mainly composed of three basic components: a transmitter, a receiver, and the channel. At first, the source messages are compressed before the modulation in the process of source coding, then the error control encoder adds additional control information bits to the compressed messages. Completed the source and channel encoding, the message is modulated with a specific carrier and transmitted over the noisy channel. Then, the receiver at the channel output extracts the target messages from the received signal through demodulation. The demodulated messages are corrupted by the channel noise during the propagation and therefore the error control decoder can solve the degradation which occurs in the communication system.

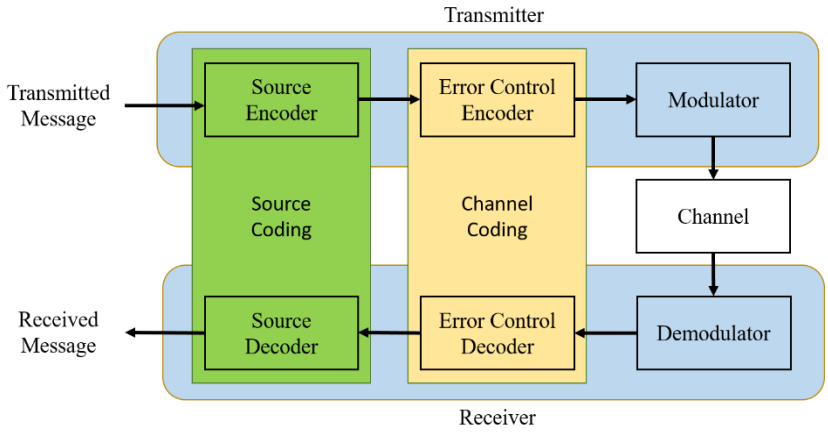


Figure 1-1 The block diagram of the digital communication system.

To conserve the transmission time and bandwidth, the source coding usually consists of two parts: redundancy removal and data compression. The former erases the redundancy bits in the message, and the size of the message will be shortened by the latter for high transmission efficiency. The purpose of channel coding is to detect the presence of error bits and obtain the error control information. The channel codes can be classified according to different conditions. They are divided into the error-detection codes and the error-correction codes based on distinct applications in the system. Error-detection codes are used to detect the position of error bits and return feedback (usually called ARQ) for retransmission. Error-correction codes enable to get the error control properties and try to correct the error caused by the noise in the channel. We can also distinguish the channel codes into block codes and convolution codes according to their structure. The main theme of discussion in the report is LDPC codes which belong to the block codes.

1.2 Motivation

Recently, the technological progress in the fifth generation (5G) communication brings the innovation of data transmission in different kinds of industry and everyone's life. The latest release of 5G NR access standardization finalized by the 3rd Generation Partnership Project (3GPP) usher the advent of transition in error control codes. For today's 4G LTE network techniques, Turbo codes are the primary physical channel coding scheme for data channels in the specifications [1]. However, the new cellular communication system needs to ensure filling the requirements for ubiquitous connections among thousands of mobile devices, machines, and sensors in the future, and therefore the channel coding solution holds promise to satisfy the needs. These aforementioned requirements include high throughput, low power consumption, rate compatibility, and support for hybrid automatic repeat request (HARQ). During the evaluation of many error control codes including 4G's Turbo codes, LDPC codes are ultimately adopted to replace Turbo codes as the data channel coding scheme for 5G NR standardization [2] by 3GPP.

The Shannon limit or Shannon channel capacity was described by Shannon in 1948 [3], the related evaluations of different channel codes were widely discussed in the channel coding field. Shannon capacity guarantees that error-free communication is achievable only if the signal-to-noise ratio (SNR) exceeds this theoretic limit for a coded system with code rate R. LDPC codes were originally proposed by Gallager [4] but were limited by complex computation due to difficult hardware implementation then. They were rediscovered that the performance of LDPC codes is close to the Shannon capacity by Mackay and Neal in the 1990s [5]. The estimated distance from

the performance to the Shannon limit is merely 0.0045 dB. LDPC codes have attracted lots of interest in their related research since then. They are also considered in many wireless standards, such as IEEE 802.11n and IEEE 802.16e [6] [7].

Message passing is the method of decoding widely used for LDPC codes, which acquires decoded messages by propagating them between variable nodes and check nodes. Due to many advantages of LDPC codes, they have not only well performance near to the Shannon limit but also great parallelization in decoding. These advantages consist of the sparse parity check matrix (H matrix) and the structure of quasi-cyclic. We propose a design of parallelized decoder for 5G NR LDPC codes in the report. The target of high throughput is achievable by the proposed parallel decoding scheme thanks to the concept of efficient chunk and column-by-column computing in [8]. The benefits of our proposed parallelized decoder architecture for 5G NR LDPC are the vacant memory space of the check-to-variable messages and improved throughput. Besides, we also propose a pre-processing to avoid memory collisions for 5G NR LDPC codes. It conduces to compute the column sets efficiently in parallelism.

Chapter 2 Details of 5G NR LDPC Codes

The first chapter briefly explains the structure of the digital communication system and the motive of a proposed parallel LDPC decoder architecture for 5G NR. This chapter will describe the details about the LDPC codes and the structure of 5G NR LDPC at first. Then, the parallelized QC-LDPC decoding algorithm scheme for IEEE 802.16e is considered in the last section. In Section 2.1, the content presents the definition of LDPC codes, the Tanner graph which is used as a graphical presentation for the *H* matrix, and the procedure of message passing algorithm based on the bipartite graph. Section 2.2 presents the main characteristic of 5G NR LDPC codes which consists of the base graph BG1 and BG2. Then, Section 2.3 details the concept of memory optimization in the QC-LDPC decoder for WiMAX which was introduced in [8]. It is composed of the structure of efficient chunk and the column-by-column decoding method.

2.1 Elementary Knowledge of LDPC Codes

Consider an (*n*, *k*) LDPC codes, which means that a message of *k* bits is encoded into the *n* bits codeword. The characteristic of the LDPC codes is the low density of 1's in the sparse parity check matrix. The design of sparsity in the *H* matrix based on density evolution (DE) breaks up the data dependence in the process of decoding, which increases the probability of correcting error bits in the received codeword. We usually verify the construction of the parity check matrix before the start of LDPC encoding or decoding (See Figure 2-1). Given a valid *n* bits codeword *c*, which satisfies the equation:

$$H \times c = 0 \tag{1}$$

The codeword *c* for *k* bits message *m* is

$$c = m \times G \tag{2}$$

where the generator matrix G can be derived from the parity check matrix with Gaussian elimination or the RU method [9].

$$G = \left[\begin{array}{ccc|cccc} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]_{4\times 7} = \left[P \mid I^4\right] \qquad H = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}\right]_{3\times 7} = \left[I^3 \mid P^T\right]$$

$$c = m \times G \qquad\qquad H \times c = 0$$

Figure 2-1 The parity check matrix (*H*) and generator matrix (*G*) of (7, 4) LDPC codes.

Tanner proposed a graphical description of the H matrix with a bipartite graph, which is called the Tanner graph [10]. The bipartite graph consists of two sets of nodes based on the *H* matrix as presented in Figure 2-2. We define the top nodes (square) as the check nodes, which correspond with the rows of the *H* matrix. The foot ones (circle) of the bipartite graph are called variable nodes, which correspond to the columns in the matrix. The edge is connected between two variables based on the position of a nonzero entry in the parity check matrix. Tanner graph is useful to illustrate the mathematic computation in the decoding algorithm, which is known as the message passing algorithm (MPA). The algorithm describes the process of conveying and updated messages which represent the probability of 1 or 0 on the corresponding bit in the received codeword. The decision rule of associated bits is based on the probability computed at the end of each iteration.

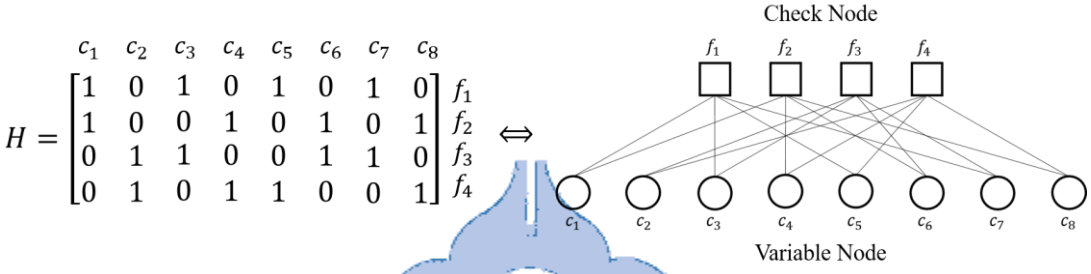


Figure 2-2 Tanner graph of (8, 4) LDPC parity check matrix.

Message passing algorithm, also called Sum-Product Algorithm (SPA), is commonly used in the field of decoding, which makes a decision based on the computation of a posteriori probability. The main feature of the algorithm is the belief propagation between two nodes which reduces the error probability of the associated bit in the received code from the channel. The process of propagation consists of two ways of message transmission from one node to the opposite. Tanner graph illustrates the information which represents the probability passing and updating the message on the associated node as described in Figure 2-3 In the graphical representation, the message of probability is updated to the check node from the variable node. We define the associated message as the variable-to-check message (q_{ij}). Then the information from the check node is conveyed to the variable node. Similarly, the one is called check-to-variable message (r_{ji}). It requires two processing units for complete computation on an iteration. The unit which deals with the variable-to-check messages in Figure 2-3 (a) is called the variable node function unit (VNFU). The case of the check-to-variable messages in Figure 2-3 (b) is the check node function unit (CNFU).

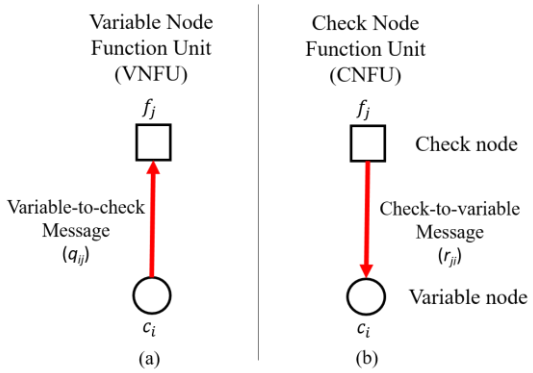


Figure 2-3 Two ways of message passing in Tanner graph.

Given a received codeword $y = \{y_1, y_2, \ldots, y_n\}$, the algorithm decides whether the i^{th} codeword c_i is 1 or 0 base on the calculation of posterior probability. Suppose the BPSK modulation is applied over the AWGN channel, the received i^{th} bit message is $y_i = x_i + \omega_i$ where x_i is the modulated bit and ω_i is the zero-mean AWGN noise with σ^2. Consider the case of BPSK signal, the i^{th} modulated bit is $x_i \in \{-1, +1\} = (-1)^{c_i}$, where $c_i \in \{0, 1\}$ is the transmitted bit in the codeword. In the case of the zero-mean AWGN channel with variance σ^2, the posterior probability is given as

$$P(x_i = x|y_i) = \frac{1}{1+e^{-2y_i x/\sigma^2}} \tag{4}$$

We consider the rule decision based on the likelihood ratio (LR) of the posterior probability $P(c_i = 0 \mid y_i)$ to $P(c_i = 1 \mid y_i)$ expressed as $l(c_i)$. To simply computation and reduce complexity, we compute $l(c_i)$ with logarithm expressed as

$$L(c_i) \equiv \log\left(\frac{P(c_i = 0|y_i)}{P(c_i = 1|y_i)}\right) = \log\left(\frac{P(x_i = +1|y_i)}{P(x_i = -1|y_i)}\right) \tag{5}$$

We verify the definition of variables introduced in a description of the algorithm at first.

- $R_j = \{i: h_{ji} = 1\}$, the set consisting of column index i whose entry at position (i, j) equals 1 in the parity check matrix.
- $R_{j\backslash i} = \{i': h_{ji} = 1\}$, the set excluding the column index i.
- $C_i = \{j: h_{ji} = 1\}$, the set consisting of row index j whose entry at position (i, j) equals 1 in the parity check matrix.
- $C_{i\backslash j} = \{j: h_{ji} = 1\}$, the set excluding the row index j.
- Q_i : the sum of passing messages with probability on variable node c_i.

The following indicates a procedure for decoding:

Step 1. Initialize the input of the variable node c_i and update the information to check node f_j. (Consider the transmission over the AWGN channel with variance σ^2)

$$L(q_{ij}) = L(c_i) = \frac{2y_i}{\sigma^2} \tag{6}$$

Step 2. The incoming messages from different variable nodes are revised and conveyed

to the associated variable node.

$$L(r_{ji}) = \left(\prod_{i' \in R_{j\backslash i}} \alpha_{ij}\right) \times \phi\left(\sum_{i' \in R_{j\backslash i}} \phi(\beta_{i'j})\right) \quad (7)$$

In equation (7), we define the operator

$$\phi(x) \equiv -\log\left(\tanh\left(\frac{1}{2}x\right)\right) = \log\left(\frac{e^x+1}{e^x-1}\right) \quad (8)$$

, α_{ij} is a sign operator of $L(q_{ij})$: $\alpha_{ij} \equiv \text{sign}\,(L(q_{ij}))$, and β_{ij} is the scale operator: $\beta_{ij} \equiv |L(q_{ij})|$.

Step 3. Prepare the update message for the next iteration.

$$L(q_{ij}) = L(c_i) + \sum_{j' \in C_{i\backslash j}} L(r_{j'i}) \quad (9)$$

Step 4. Compute the value for all i:

$$L(Q_i) = L(c_i) + \sum_{j \in C_i} L(r_{ji}) \quad (10)$$

Step 5. Make a decision based on the value from equation (10) for all i:

$$\hat{c}_i = \begin{cases} 1, & L(Q_i) < 0 \\ 0, & else \end{cases} \quad (11)$$

We keep carrying out the algorithm from Step 2 to Step 5 until the maximum number of iterations is reached or the fulfillment of $\hat{c} \times H = 0$. In the report, we introduce one of the well-known approximation algorithms, which is called the Normalized-Min-Sum Algorithm (NMSA) to reduce the complexity and achieve the performance similar to the Sum-Product Algorithm. The algorithm simplifies the equation (7) as

$$L(r_{ji}) = \left(\prod_{i' \in R_{j\backslash i}} \alpha_{ij}\right) \times \alpha \times \min_{i' \in R_{j\backslash i}} \beta_{i'j} \quad (12)$$

where α is called the normalization factor, which is adopted at about 0.75 in general.

2.2 Structure of 5G NR LDPC Codes

5G NR LDPC codes apply the quasi-cyclic structure in its construction of parity check matrix (H). The complete H matrix is composed of numerous zero matrices of size $Z_c \times Z_c$ or circular shift identity matrices of the same size on the basis of a protograph, which is officially called a base graph (BG) in the specification for 5G NR [2]. Each entry in the base graph represents the shift value $P_{i,j}$: The element of value -1 is replaced by an all zero matrix; the element of positive value is replaced by a circular permutation matrix, which is obtained from the identity matrix cyclically shifted from right $P_{i,j}$ times. The value of $P_{i,j}$ is given by the modulo operation:

$$P_{i,j} = \begin{cases} -1, & if \;\; P_{i,j} = -1 \\ mod(V_{i,j}, Z_C), & else \end{cases} \quad (13)$$

where $V_{i,j}$ is defined as the (i,j)-th shift coefficient of the base graph in [2] Table 5.3.2-2 and Table 5.3.2-3. Notably, the $P_{i,j}$ of value 0 denotes the identity matrix and the case

of -1 indicates the null matrix (zero matrix).

In the 5G NR specification, there are two base graphs: the first base graph (BG1) and the second base graph (BG2) used to construct the codeword with different information block sizes and the full range of code rates, which indicates the characteristic of rate-compatible. The usage of these two base matrices mainly depends on the code rate ($R = N/K$) and information block size (K). In general, we apply BG1 for large code block size ($500 \leq K \leq 8448$) and high code rate ($1/3 \leq R \leq 8/9$), whereas BG2 is targeted for small code block size ($40 \leq K \leq 2560$) and low code rate ($1/5 \leq R \leq 2/3$) [11]. The desired code block size and the code rate dominate the choice.

Table 2-1 The basic parameters of base graph for 5G NR LDPC.

Basic Parameters	**BG1**	**BG2**
Matrix size	46×68	42×52
Number of information column	22	10
Maximum code block size (K)	8448 ($=22 \times 384$)	3840 ($=10 \times 384$)
Minimum code rate (R)	1/3	1/5

Some basic parameters of BG1 and BG2 for 5G NR LDPC are given in Table 2-1. The base matrix BG1 is similar in the structure to BG2. Figure 2-4 gives a detailed sketch of the base matrix BG1, whose columns are divided into three parts: information columns, core parity columns, and extension columns. In the same way, the rows of a base graph are partitioned into core check rows and extension check rows. It is worthy to note that the first two columns (gray columns) of the base graph are punctured information columns that are never transmitted.

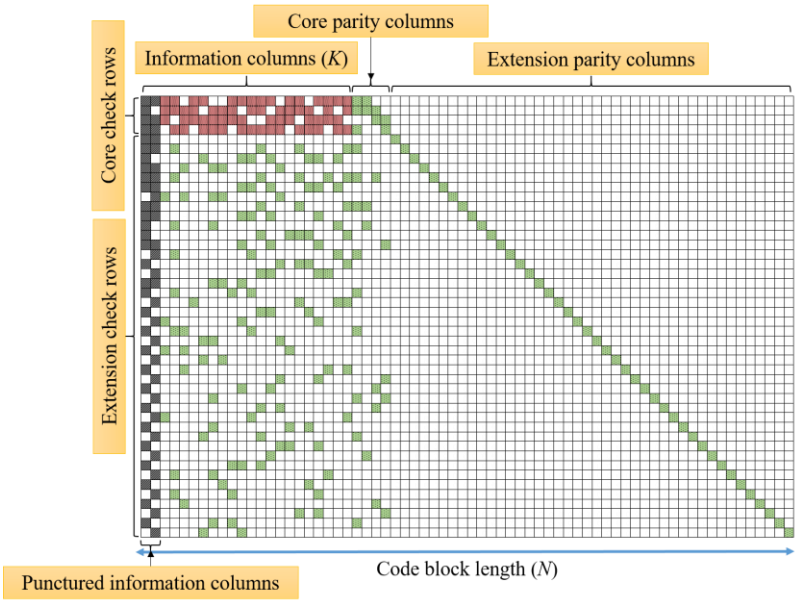


Figure 2-4 The structure of BG1 for 5G NR LDPC. (Each colored square denotes a circular

permutation matrix with different shift value $P_{i,j}$, whereas the space one corresponds to zero matrix.)

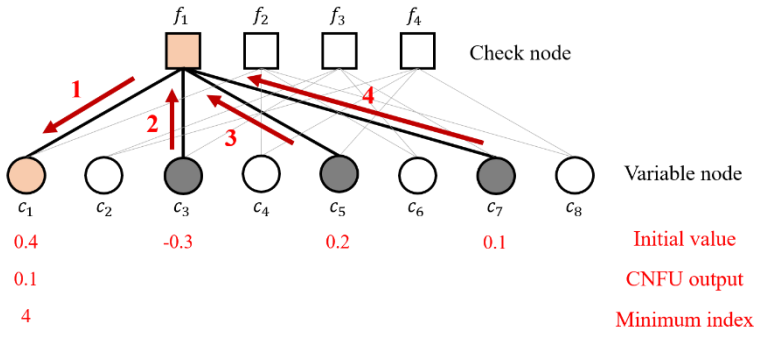
For base graphs BG1 and BG2, we compute the (i, j)-th shift value by the modulus operation of shift coefficient $V_{i,j}$ and Z_c in the first place. The variable Z_c contained in the equation (13) is defined as the lifting size, which of value is derive from $Z_c = a \times 2^j$ for $a = \{2, 3, 5, 7, 9, 11, 13, 15\}$ and $0 \leq j \leq 7$. For both base graph BG1 and BG2, these 51 different lifting size is divided into eight sets according to the parameter as listed in Table 2-2. Each element in the eight sets is designed for a specific permutation matrix with different shift coefficients in the 5G NR specification TS 38.212 [2].

Table 2-2 The relationship between the set index and lifting sizes Z_c for 5G NR LDPC.

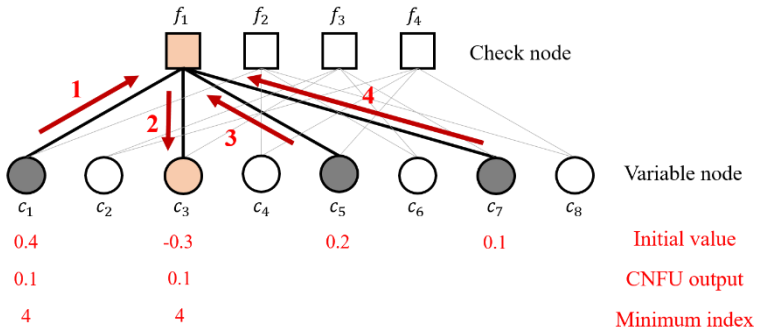
Set index	Set of lifting sizes (Z_c)
Set 0 ($a = 2$, $0 \leq j \leq 7$)	{2, 4, 8, 16, 32, 64, 128, 256}
Set 1 ($a = 3$, $0 \leq j \leq 7$)	{3, 6, 12, 24, 48, 96, 192, 384}
Set 2 ($a = 5$, $0 \leq j \leq 6$)	{5, 10, 20, 40, 80, 160, 320}
Set 3 ($a = 7$, $0 \leq j \leq 5$)	{7, 14, 28, 56, 112, 224}
Set 4 ($a = 9$, $0 \leq j \leq 5$)	{9, 18, 36, 72, 144, 288}
Set 5 ($a = 11$, $0 \leq j \leq 5$)	{11, 22, 44, 88, 176, 352}
Set 6 ($a = 13$, $0 \leq j \leq 4$)	{13, 26, 52, 104, 208}
Set 7 ($a = 15$, $0 \leq j \leq 4$)	{15, 30, 60, 120, 240}

2.3 Memory Optimization in QC-LDPC Decoder for WiMAX

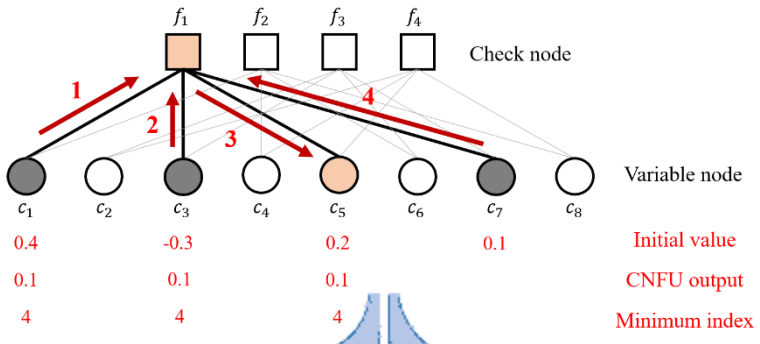
We usually store the check-to-variable messages and variable-to-check messages separately into the memory in the design of the Min-Sum decoder. However, the idea of spending lots of memory on all messages leads to a waste of storage space. To avoid such waste, the efficient chunk architecture is proposed in [8].



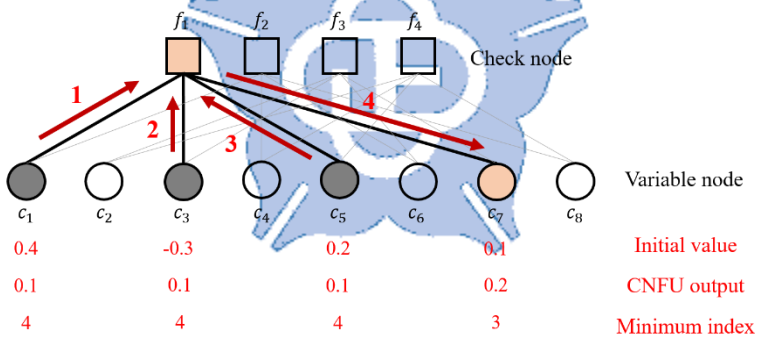
(a)



(b)



(c)

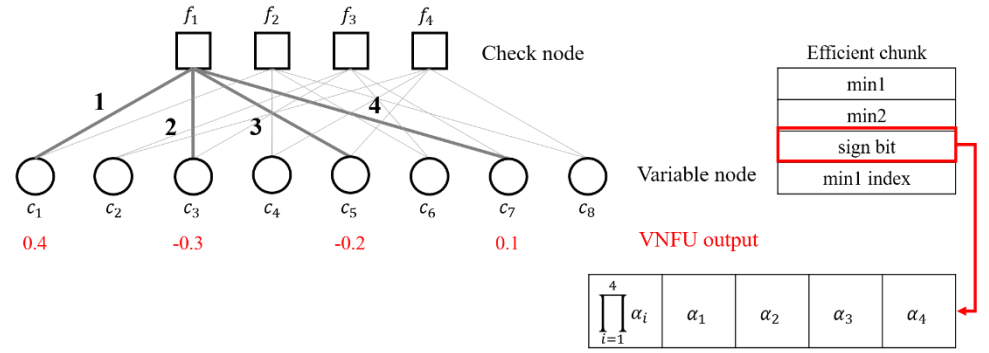


(d)

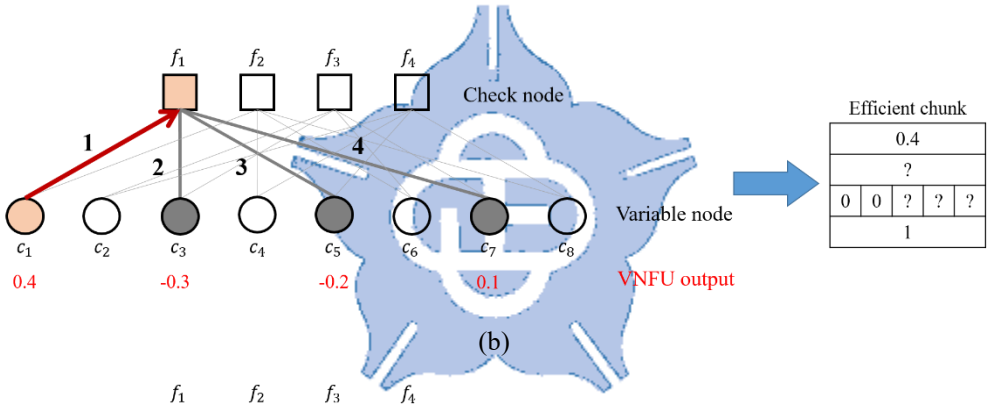
Figure 2-5 Updating all check-to-variable messages on the check node f1 with (8, 4) H matrix.

The procedure of computing the check-to-variable messages on the check node f_1 in equation (12) is shown in Figure 2-5. At first, we initialized all likelihood ratio values of the variable nodes which are connected to the check node f_1: 0.4, -0.3, 0.2, and 0.1 separately. We derive the updated message on the associated variable node through the processing element CNFU according to the NMSA decoding algorithm. There are only two values of the check-to-variable message: 0.1 and 0.2, then it is worth noting that we just need to store the first minimum (min1) and the second minimum (min2) in the memory. Furthermore, we also memorize the index of the minimum likelihood ratio from the initial variable node, and we use the representation of numbers that are

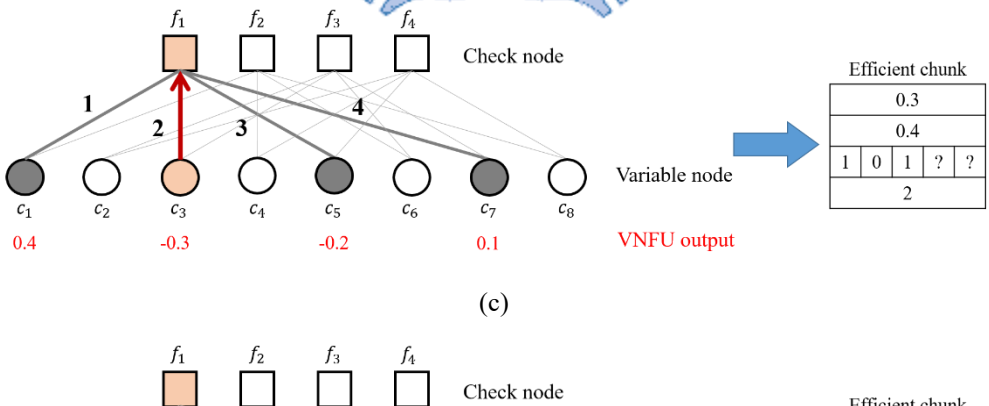
attached to the edges to record the positions of the connected variable nodes in Figure 2-5. The main idea of the efficient chunk is to store: min1 value, min2 value, the index of min1, and sign bits of LR values on all variable nodes as described in Figure 2-6. We can also predict the check-to-variable messages via computing the variable-to-check messages and updating to the efficient chunk at the current iteration (See Figure 2-6).



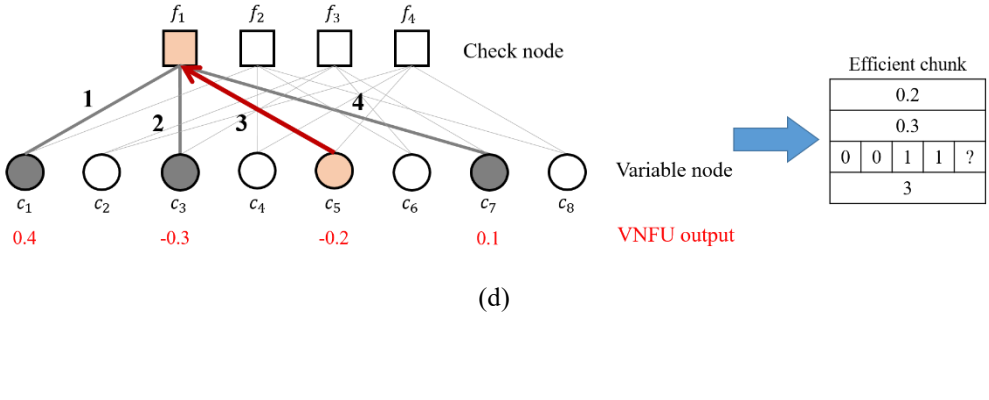
(a)



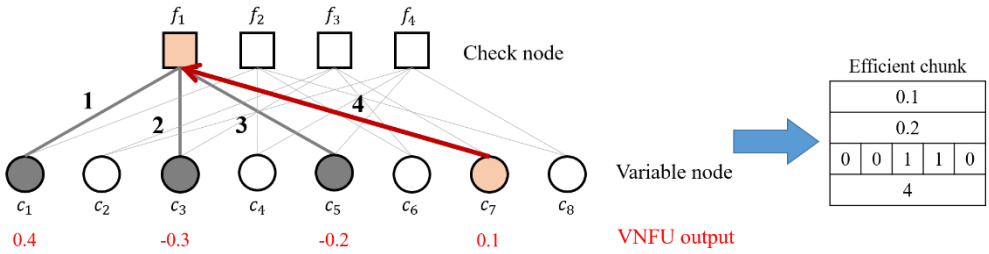
(b)



(c)



(d)



(e)

Figure 2-6 Simplifier stores the variable-to-check messages into the efficient chunk.

The simplifier unit is used to update these new variable-to-check messages to the efficient chunk once the computation is finished. Since we apply the efficient chunk to record the information which is derived from the current VNFU computation, the check-to-variable messages are required to recover for the next iteration. For incorporation of the efficient chunk mechanism in the NMSA decoding algorithm, we modify the equation (12) described in Section 2.1:

- If the variable node c_i is not in the position of min1, then

$$L(r_{ji}) = \left(\prod_{i' \in R_{j\backslash i}} \alpha_{ij}\right) \times \alpha \times \min 1 \tag{14}$$

- If the variable node c_i is in the position of min1, then

$$L(r_{ji}) = \left(\prod_{i' \in R_{j\backslash i}} \alpha_{ij}\right) \times \alpha \times \min 2 \tag{15}$$

Besides, the belief sum of variable nodes is updated by adding the recovered check-to-variable messages from equation (14) or (15). The VNFU computes the new variable-to-check messages for the next iteration according to the following equation:

$$L(Q_i) = L(c_i) + \sum_{j \in C_i} L(r_{ji}) \tag{16}$$

$$L(q_{ij}) = L(Q_i) - L(r_{ji}) \tag{17}$$

The quasi-cyclic (QC) structure of LDPC codes is beneficial to implement a column-by-column decoding algorithm in [8], which enables great parallelism in the design of the decoder. In column-by-column decoding, each submatrix implements an update on the messages between the variable node and check nodes simultaneously. Initially, the likelihood ratio values on all variable node's input are updated to the associated check node. Then messages of the check nodes are stored into the corresponding row of an efficient chunk. The contents in the efficient chunk will be recovered to the correct check-to-variable messages. Next, the variable node function unit (VNFU) computes the belief sum $L(Q_i)$ and variable-to-check message $L(q_{ij})$ with the column-by-column method in one process as shown in Figure 2-7. During the Z_c-th process, all messages on variable nodes are computed and updated to the efficient chunk for the next iteration.

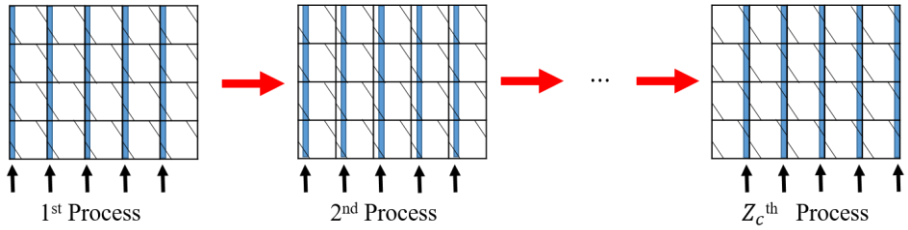


Figure 2-7 The messages on variable node can be computed with column-by-column method.

The QC-LDPC codes are considered for the wireless standard IEEE 802.16e (WiMAX) due to the feature of parallelism in an efficient hardware implementation. The architecture of efficient chunk and the method of column-by-column computing are applied in the design of the QC-LDPC decoder for WiMAX in [8]. However, the repeated offset elements of the protograph for IEEE 802.16e LDPC cause update conflict in the efficient chunk. To avoid such writing conflict in the memory, the pre-process is also introduced to separate the protograph into several sets of the column in [8]. Each column set is composed of the selected column to prevent memory collision from happening. Above all, the column sets can be processed in parallel, which results in low decoding latency and high throughput.

Chapter 3 Parallelized Decoder for 5G NR LDPC Codes

The detailed information on LDPC codes is introduced in the previous chapter. It also consists of the structure of 5G NR LDPC and the memory optimization for the WiMAX LDPC. We propose a parallelized decoder architecture of 5G LDPC which applies the memory arrangement of efficient chunk and the column-by-column computing method since the 5G NR LDPC codes belong to the quasi-cyclic structure. However, the rows with repeated shift value P_{ij} in the base graph result in the memory access collision, which will cause unnecessary decoding latency and low throughput.

Section 3.1 explains how the proposed pre-procession use to prevent the memory access conflict. All columns of the base graph are assigned into the appropriate column sets. Section 3.2 presents the memory structure of efficient chunk which is applied in the hardware architecture of the proposed parallel 5G LDPC decoder. Finally, the design of parallelized LDPC decoder is described in Section 3.3. The section also introduces the function of multiple processing units in the proposed decoder.

3.1 The Memory Conflict Prevention for 5G NR LDPC

Since the 5G NR LDPC codes belong to the structure of quasi-cyclic, we can implement parallelized decoding with the architecture of efficient chunk and the computing method by column-by-column. We organize a memory space as the efficient chunk to store all computed variable-to-check messages in a row of parity check matrix. For the next iteration, the content of the efficient chunk will be recovered to correct check-to-variable messages for an update on all variable nodes in the process of VNFU. In addition to the arrangement of required messages in the efficient chunk, we can implement an efficient parallelized decoding with the column-by-column method due to the structure of quasi-cyclic in 5G NR LDPC codes. However, some submatrices with the same shift values (P_{ij}) will cause memory access collision in the process of updating the computed messages to corresponding efficient chunks in parallel. For example, we consider BG1 with Z_c = 256, which is presented with the positive shift value of (i, j)-th entry P_{ij} in Table 3-1 (See Figure I-1 in Appendix I for a graphical description). There are several elements of the base graph with equal shift value in the same row which are marked with different colors in the table.

Table 3-1 The memory conflict problem in BG1 with Z_c = 256.

H_{BG1} (Z_c=256)												
Row i	Column j	P_{ij}	Row i	Column j	P_{ij}	Row i	Column j	P_{ij}	Row i	Column j	P_{ij}
1	1	250	6	1	205	16	2	96	30	19	84
	2	69		2	236		11	65		26	6
	3	226		4	194		14	63		52	0
	4	159		13	231		19	75	31	1	216
	6	100		17	28		26	179		11	73
	7	10		22	123		38	0		14	120
	10	59		23	115	17	2	64		25	9
	11	229		28	0		4	49		53	0
	12	110	7	1	183		12	49	32	2	95
	13	191		7	22		21	51		8	177
	14	9		11	28		23	154		23	172
	16	195		12	67		39	0		26	61
	17	23		14	244	18	1	7		54	0
	19	190		18	11		15	164	33	1	221
	20	35		19	157		17	59		13	112
	21	239		21	211		18	1		15	199
	22	31		29	0		22	144		25	121
	23	1	8	1	220		40	0		55	0
	24	0		2	44	19	2	42	34	2	2
2	1	2		5	159		13	233		3	187
	3	239		8	31		14	8		12	41
	4	117		9	167		19	155		22	211
	5	124		15	104		20	147		56	0
	6	71		30	0		41	0	35	1	127
	8	222	9	1	112	20	1	60		8	167
	9	104		2	4		2	73		16	164
	10	173		4	7		8	72		18	159
	12	220	9	13	211		9	127		57	0
	13	102		17	102		11	224	36	2	161
	15	109		20	164		42	0		7	197
	16	132		22	109	21	1	151		13	207
	17	142		23	241		4	186		23	103
	18	155		25	90		10	217		58	0
	20	255		31	0		12	47	37	1	37
	22	28	10	1	103		23	160		15	105
	23	0		2	182		43	0		16	51
	24	0		11	109	22	2	249		19	120
	25	0		12	21		6	121		59	0
3	1	106		14	142		17	109	38	2	198
	2	111		18	14		21	131		14	220
	3	185		19	61		22	171		24	122
	5	63		21	216		44	0		60	0
	6	117		32	0	23	1	64	39	1	167
	7	93	11	2	98		13	142		10	151
	8	229		3	149		14	188		11	157
	9	177		5	167		18	158		13	163
	10	95		8	160		45	0		61	0
	11	39		9	49	24	2	156	40	2	173
	14	142		15	58		3	147		4	139
	15	225		33	0		11	170		8	149
	16	225	12	1	77		19	152		20	0
	18	245		2	41		46	0		62	0
	19	205		13	83	25	1	112	41	1	157
	20	251		17	182		4	86		9	137
	21	117		22	78		5	236		18	149
	25	0		23	252		12	116		63	0
	26	0		24	22		23	222	42	2	167
4	1	121		34	0		47	0		4	173
	2	89	13	1	160	26	2	23		10	139
	4	84		2	42		7	136		19	151
	5	20		11	21		8	116		64	0
	7	150		12	32		15	182	43	1	149
	8	131		14	234		48	0		5	157
	9	243		19	7	27	1	195		25	137
	11	136		35	0		3	243		65	0
	12	86	14	1	177		5	215	44	2	151
	13	246		4	248		16	61		17	163
	14	219		8	151		49	0		19	173
	15	211		21	185	28	2	25		26	139
	17	240		24	62		7	104		66	0
	18	76		36	0		9	194	45	1	139
	19	244	15	1	206		50	0		8	157
	21	144		13	55	29	1	128		10	163
	22	12		16	206		5	165		23	173
	23	1		17	127		20	181		67	0
	26	0		18	16		22	63	46	2	149
5	1	157		22	229		51	0		7	151
	2	102		37	0	30	2	86		11	167
	27	0	16	1	40		15	236		68	0

Table 3-2 Color representation related to the rows with repeated shift values in Table 3-1.

Row	Column	Repeated Value	Background Color
2	23, 24, 25	0	RED
3	6, 21	117	YELLOW
	15, 16	225	BLUE
	25, 26	0	GREEN
15	1, 16	206	ORANGE
17	4, 12	49	PINK
40	20, 62	0	BROWN

Table 3-1 describes the structure of BG1 with Z_c = 256, and another entry with value -1 in the graphical presentation (See Figure I-1 in Appendix I.) but not in the table represents a zero matrix of size $Z_c \times Z_c$. The rows with a repeated value of P_{ij} are denoted in different colors as shown in Table 3-2. If one row has multiple repeated cyclically shifted values, we use two or more distinct colors to separate them. For example, the 3rd row has 3 repeated values, and one of them is 117 in the 6th and 21st columns. Then the entries with the same offset 117 in the row are marked with red color in the background. The element for 225 in the 15th and 16th columns are labeled in blue. In the case of 0, the entries at (3, 25) and (3, 26) are denoted with green color.

We can find that the 2nd, 3rd, 15th, 17th, and 40th rows have some cyclically shifted identity matrices with the same offset value in Table 3-2. If the row has the same shift values, the memory conflict will occur in the corresponding efficient chunk. Then the hardware requires additional clock cycles to deal with the memory collision, which results in decoding latency and low throughput. To avoid unnecessary memory access collision, we proposed a pre-procession to assign all columns in the base matrix into different column sets and confirm columns with the same offset would not be assigned in the same set. It also guarantees that each set is equal in columns during the process. The main concept of the proposed pre-procession consists of four phases in Figure 3-1:

- **Phase 1.**
  Confirm the maximum number of column sets (N) and find out all columns with the same shift value in a row.
- **Phase 2.**
  Assign the extension parity columns of the parity check matrix into N column sets.
- **Phase 3.**
  Assign the main columns of the parity check matrix into N column sets.
- **Phase 4.**
  Solve the problem of the row with all zero matrices in the set by swapping whose columns with other sets.

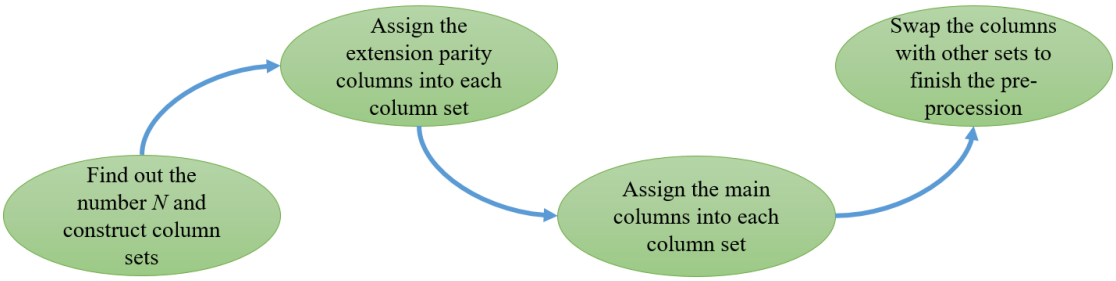


Figure 3-1 Four phase of the pre-procession to avoid memory collusion.

In the initial phase of the pre-procession, we count the repeated shift value in the same row and choose the number of column sets N. We define the variable $n_{d,j}$ as the number of the repeated shift value $d \in \{0, 1, 2, \ldots, Z_c - 1\}$ in the j-th row. Then the value of $n_{d,j}$ is counted for all rows in the base graph successively. We use N to store the maximum number of repeated offset in one row. For an example of BG1 with $Z_c = 256$ in Table 3-1, we find that the 2nd and 3rd rows both have three repeated values corresponding to 0 and 117 separately. Since the 2nd row is in advance of the 3rd row, we choose the maximum value of $n_{d,2}$ in the 2nd row for the number of sets N. When the value of N is decided to be 3, we create three column sets and put the 23rd, 24th, and 25th columns into each set in order. Figure 3-2 illustrates the process of finding the value of N and constructing each set individually.

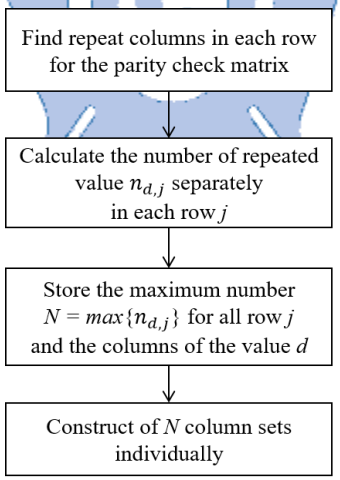


Figure 3-2 Finding the number of column sets N in an initial phase.

Next, all columns in the base graph are allocated to the corresponding sets. The process of arranging column vectors in each set is composed of two parts which are the extension columns and main columns of the parity check matrix. At first, we consider the extension parity columns in the second phase. The block of extension parity columns is similar to the identity matrix which consists of ones on the diagonal and zeros elsewhere. Since the extension parity part never arises the memory access conflict,

we assign these columns into the sets in advance. The range of extension parity columns in BG1 is from 27 to 68 (Total 42 columns) as shown in Figure 3-4 (a) which is marked with a red frame. For BG2 in Figure 3-4 (b), the range of the extension parity columns is from15 to 52 (Total 38 columns). It's worth noting that we should make the number of columns in each set equal during the process of distribution (See Figure 3-3).

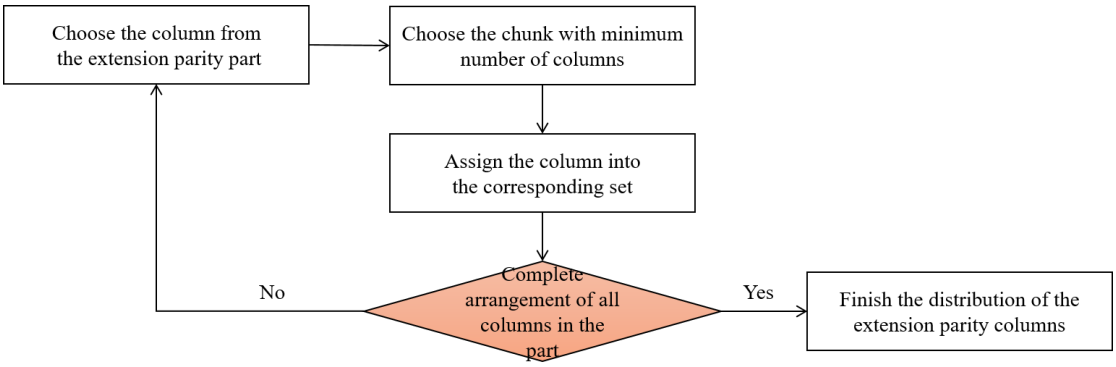
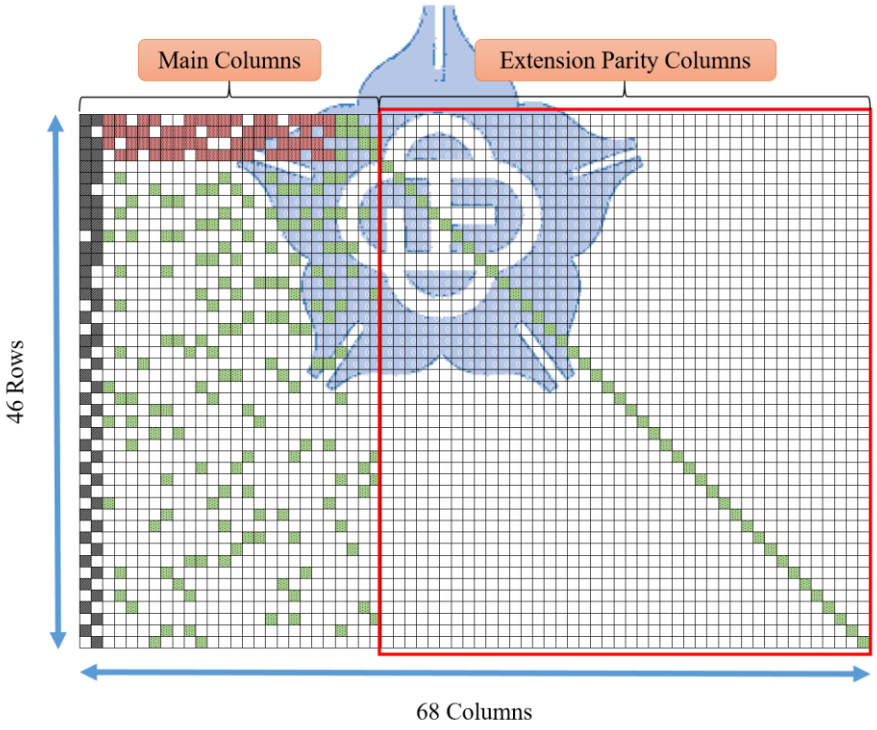
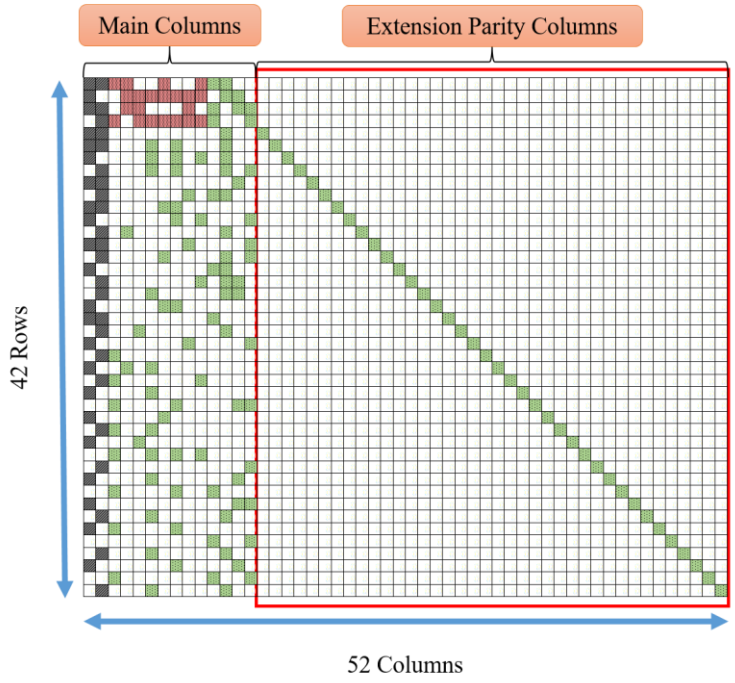


Figure 3-3 The allotment of extension parity columns is made in the second phase.



(a) There 26 main columns and 42 extension columns in BG1



(b) There 14 main columns and 38 extension columns in BG2

Figure 3-4 The columns of BG1 and BG2 are individually partitioned into two parts.

In the third phase, we make arrangements for the main columns which are composed of information columns and core parity columns. How to assign the remaining columns into the corresponding column sets is an important problem since most memory access conflicts happen in these. For BG1 in Figure 3-4 (a), the unassigned columns are out of the red frame mark and the number of them is 26, whereas BG2 has 14 remaining columns for distribution in Figure 3-4 (b). To describe the allotment of the main columns, we introduce the following variables in Figure 3-5:

- s_j : The state describes the existence of the all zero row j in the column set.
- n_k : Number of nonzero entries for the main column k.

We choose the column set with minimum columns at first in Figure 3-5, then we check the state of all zero rows in the set s_j. If the j-th row has all zero elements in the set, we set the value of state as 1. On the contrary, the value will be 0 when the row has a nonzero element. Since the j-th row is an all zero row in the set, we try to find the remaining column m which has the most nonzero elements. Then we check whether the selected column m causes memory collusion in the set. Despite the set has no all zero rows j, we still select the remaining column k and assign it into the set until there are no unassigned columns. When all columns of the base graph are assigned to the corresponding column sets, we count the nonzero elements in each row for every set. We try to make no all zero rows in each column set since the row with all zero elements indicates that it will do nothing in the process of column-by-column computing.

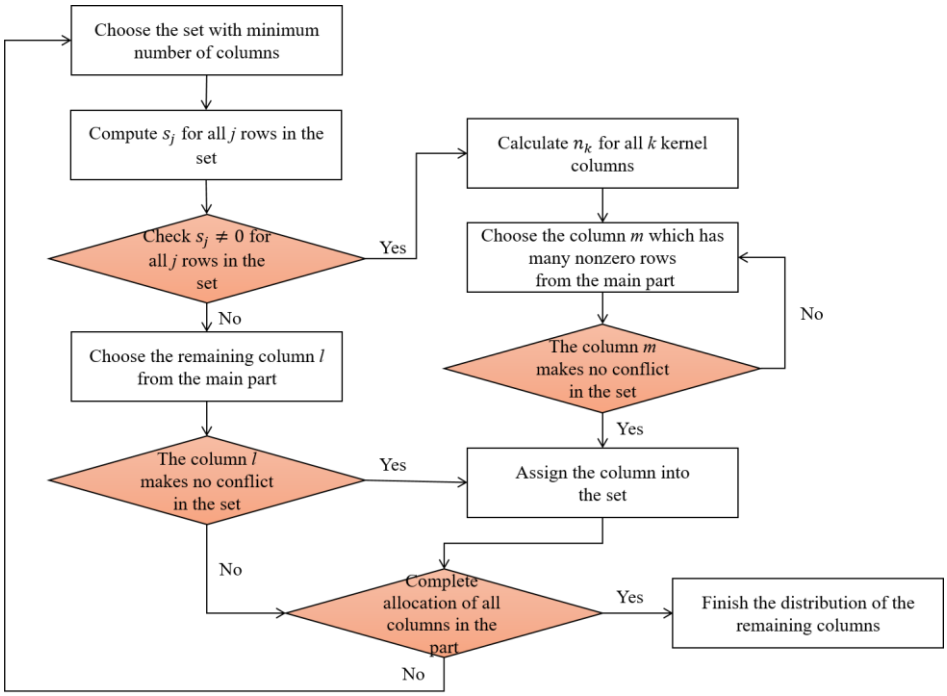


Figure 3-5 Assign the remaining columns into each sets in the third phase.

Take BG1 with $Z_c = 256$ in Table 3-1 as an example, the index of assigned columns in each set is described in Figure 3-6. We also count the number of nonzero elements in each row for all column sets. In Figure 3-7, Set 2 has zero values in the 5th, 29th, and 43rd rows. The 28th and 36th rows both have all zero values in Set 3. The value of 0 in Figure 3-7 is marked with red color. To avoid the set has an all zero row, the method of swapping the columns between two sets is introduced in the final phase.

Index of column in each set		
Set 1	Set 2	Set 3
23	24	25
68	67	66
65	64	63
62	61	60
59	58	57
56	55	54
53	52	51
50	49	48
47	46	45
44	43	42
41	40	39
38	37	36
35	34	33
32	31	30
29	28	27
2	14	19
18	8	22
13	12	10
5	17	4
1	7	11
21	16	3
26	9	15
	6	20

Figure 3-6 The index of assigned row in each column set.

Number of nonzero elements in each row			
Row index	Set 1	Set 2	Set 3
1	5	7	7
2	5	7	7
3	6	6	7
4	8	6	5
5	2	0	1
6	4	2	2
7	4	3	2
8	3	2	2
9	4	2	4
10	5	2	2
11	2	2	3
12	4	3	1
13	3	2	2
14	2	2	2
15	3	3	1
16	4	1	2
17	3	1	2
18	2	2	2
19	3	1	2
20	2	2	2
21	2	2	2
22	3	2	1
23	3	1	1
24	1	1	3
25	4	1	1
26	1	2	2
27	2	2	1
28	2	2	0
29	2	0	3
30	2	1	2
31	2	1	2
32	3	1	1
33	2	1	2
34	2	1	2
35	2	2	1
36	3	2	0
37	2	1	2
38	1	2	1
39	2	1	2
40	2	1	2
41	2	1	1
42	1	1	3
43	3	0	1
44	2	1	2
45	2	2	1
46	2	1	1

Figure 3-7 The number of nonzero elements in each row for all column sets.

To solve the problem that there exist all zero rows in the column set, we take the method of swapping two columns in different sets. When the set has an all zero row, we look for the column in the extension parity part with nonzero shifted value in the same row and make an exchange of them. Take the result of BG1 with $Z_c = 256$ for example, we find that the 5th row has zero value in Set 2 as shown in Figure 3-7, which means that the decoding unit does not work in the entire row. To prevent the condition and maximize the resource utilization, we look for the extension column which has a nonzero shifted value in a certain row to swap. Finally, we know that the 27th column has 0 in the 5th row in Figure 3-8. It is worth noting that the value 0 represents the offset for the $Z_c \times Z_c$ identity matrix in the base graph, and you should not confuse it with the number of nonzero elements in each row for the set. The 27th column is already assigned in Set 3 and therefore we need to choose one column in Set 2 to be exchanged with the desired column 27 in Set 3.

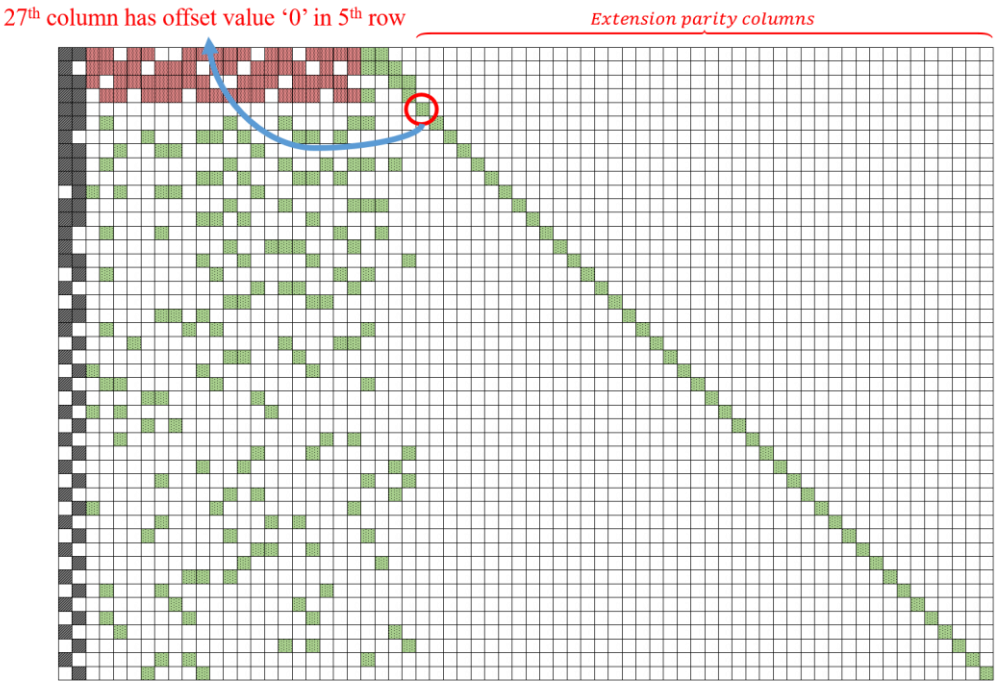


Figure 3-8 The extension column 27 has offset value 0 in the 5th row.

When we make sure the desired column (27th column in Set 3), now we need to find out which column in Set 2 to be swapped. Next, we count the number of nonzero values in the certain row which is the index of the offset '0' for all assigned extension parity columns in Set 2. We use the number of nonzero items in the corresponding row for all extension parity columns which is assigned in Set 2 as the reference for choosing the column to be exchanged with the 27th column (See Figure 3-9).

The row counting for extension columns in Set 2	
Column Index	Row Counting
64	1
61	1
55	1
52	1
46	1
67	2
58	2
49	2
43	2
40	2
31	2
28	2
37	3
34	3

Figure 3-9 The corresponding row counting for all assigned extension parity columns in Set 2.

Figure 3-9 describes the column index from the 27th to 68th column in Set 2 and the numbers of nonzero items in the corresponding row for the extension parity columns. The column index is sorted by the value of row counting, and we make a decision based on the maximum of row counting. If we find the maximum which means that the column is suitable for swap, the corresponding column will be exchanged with the desired column to solve the problem of all zero rows. In the case of BG1 with $Z_c = 256$, the maximum of row counting is 3 in the 34th and 37th columns. However, we choose column 34 which is marked with a red circle in Figure 3-9 as the target for the swap at first. Before we exchange the column with the desired column 27 in Set 3, we need to check that the 34th column would make memory conflict with columns in Set 3. If the column has other duplicate values with other columns in the same row for Set 3, then we choose the second maximum 37th column and repeat the above process to check whether there exists repetition until the swap is completed as described in Figure 3-10.

To describe the process of exchanging column between two sets, the following variables in Figure 3-10 are introduced:

- j : The index of an all zero row in the set s_1.
- i : The extension parity column whose offset 0 in the j^{th} row from another chunk s_2.
- n_k: The total number of nonzero values in the corresponding row for the column k in the original chunk. (Note: The column k is an extension parity columns)
- m : The suitable column from chunk c_1 for swapping into the set c_2 according to the values of n_k.

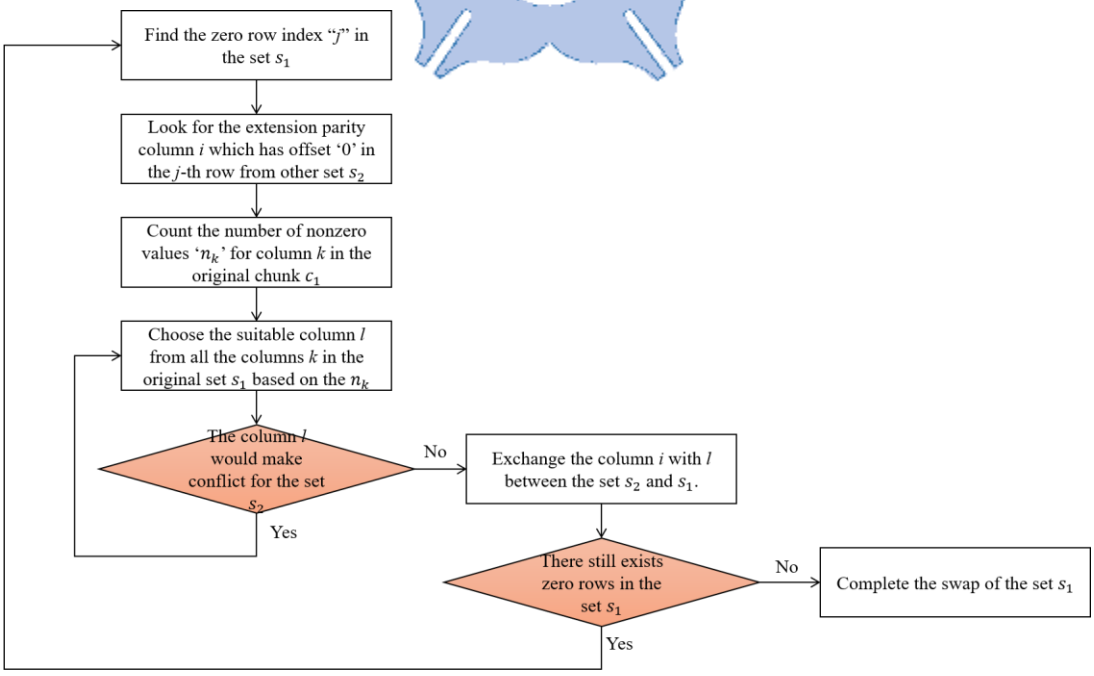


Figure 3-10 The final process of swapping the column with other set to prevent an all zero row.

According to the process of swap which is shown in Figure 3-10, the case of BG1 with Zc = 256 has three column sets as described in Figure 3-11 (a) eventually and we compute the row counting again for all sets in (b). We can find that Set 2 still has an all zero row which is denoted with a red circle, and we will discuss it in the next chapter.

Index of column in each set		
Set 1	Set 2	Set 3
23	24	25
68	67	66
28	64	63
62	61	60
59	30	57
56	55	54
53	52	37
34	49	48
47	46	45
44	43	42
41	40	39
38	51	36
35	33	27
32	31	58
29	65	50
2	14	19
18	8	22
13	12	10
5	17	4
1	7	11
21	16	3
26	9	15
	6	20

(a)

Number of nonzero elements in each row			
Row index	Set 1	Set 2	Set 3
1	5	7	7
2	5	7	7
3	6	6	7
4	8	6	5
5	2	0	1
6	5	1	2
7	4	3	2
8	3	3	1
9	4	2	4
10	5	2	2
11	2	3	2
12	5	2	1
13	3	2	2
14	2	2	2
15	3	2	2
16	4	1	2
17	3	1	2
18	2	2	2
19	3	1	2
20	2	2	2
21	2	2	2
22	3	2	1
23	3	1	1
24	1	1	3
25	4	1	1
26	1	2	2
27	2	2	1
28	1	2	1
29	2	1	2
30	2	1	2
31	2	1	2
32	3	1	1
33	2	1	2
34	2	1	2
35	2	2	1
36	3	1	1
37	2	1	2
38	1	2	1
39	2	1	2
40	2	1	2
41	2	1	1
42	1	1	3
43	2	1	1
44	2	1	2
45	2	2	1
46	2	1	1

(b)

Figure 3-11 (a) The index of assigned column in each column set after the swap. (b) The number of nonzero elements in each row for all sets after the swap.

3.2 The Memory Structure of Efficient Chunk

Since the 5G NR LDPC codes are designed in the structure of quasi-cyclic, we propose the memory architecture of efficient chunk for updating the variable-to-check messages in the parallel decoding. The computed messages are stored in the temporary register, and they will be recovered for the check-to-variable messages at the next iteration. Due to the characteristic of quasi-cyclic structure and the use of efficient chunks, each variable function unit (VNFU) in a submatrix can implement the NMSA decoding algorithm with the column-by-column method. Finally, we use a processing unit simplifier to update the data in each row to the corresponding chunk register. The data which is stored in the efficient chunk consists of four basic components: min1, min2, the index of min1, and the sign bits. Figure 3-12 shows the structure of the chunk.

Address	min1	min2	Index of min1	Sign bits
0	8 bits	8 bits	5 bits	20 bits
1	8 bits	8 bits	5 bits	20 bits
2	8 bits	8 bits	5 bits	20 bits
3	8 bits	8 bits	5 bits	20 bits
⋮	⋮	⋮	⋮	⋮
11774	8 bits	8 bits	5 bits	20 bits
11775	8 bits	8 bits	5 bits	20 bits

Figure 3-12 The memory structure of efficient chunk for BG1 with $Z_c = 256$.

For different H matrices, the structure of the corresponding efficient chunk needs to be adjusted properly for the storage of the messages in each row. Figure 3-12 shows the efficient chunk register for the base graph BG1 with $Z_c = 256$. The total number of columns in the H matrix based on the protograph is 68×256 = 17408, and the matrix has 46×256 = 11776 rows in total. Each row of the efficient chunk is used to store the updated message at the output of the simplifier unit. The min1 and min2 represent the maximum absolute value of the variable-to-check message ($|L(q_{ij})|$) and the second one separately. The data of min1 and min2 are stored with 8bits, whose 2 bits represent the integer part of the value and the remaining 6 bits represent the decimal part [8]. The index of min1 stores the position of the shifted submatrix which outputs the minimum value. Since the maximum degree of the H matrix with $Z_c = 256$ is 19, we use 5 bits to describe the index. The last part of the chunk is the sign bits. The most shifted submatrices in the H matrix is 19 (Maximum degree = 19) and therefore the sign part of the maximum value is denoted in 20 bits, whose 1bit represents the result of XOR computation from them. Finally, the memory address is from 0 to 1175 and each row in the efficient chunk register consists of 41bit for the data storage.

3.3 Design of Parallel LDPC Decoder

Figure 3-13 describes the design of the proposed parallel LDPC decoder. The main architecture of the decoder is composed of the following basic components:

- Intrinsic LLR register: The register is used to keep the initial data which is the LLR value at the input of the decoder before the decoding process.
- Check node function unit (CNFU): The function of the unit is to compute the check node messages ($L(r_{ij})$) and update them to the efficient chunk at the first iteration.
- Efficient chunk: The memory space stores the required element of the previously computed variable-to-check messages ($L(q_{ij})$) for the next iteration.
- Recovery unit: The processing unit works on the recovery of the check-to-variable

message ($L(r_{ij})$) based on the efficient chunk for the variable node processing.

- Variable node function unit: Compute the belief sum value ($L(Q_i)$) for decoding decision and the variable-to-check message for the simplifier unit to update.
- Decision unit: Decide whether the decoded message $\hat{c}$ which is corresponded to the belief sum satisfies the condition: $\hat{c} \times H = 0$.
- Simplifier: The processing unit updates the desired data which consists of min1, min2, min1 index, and sign bits on the efficient chunk for the next iteration.

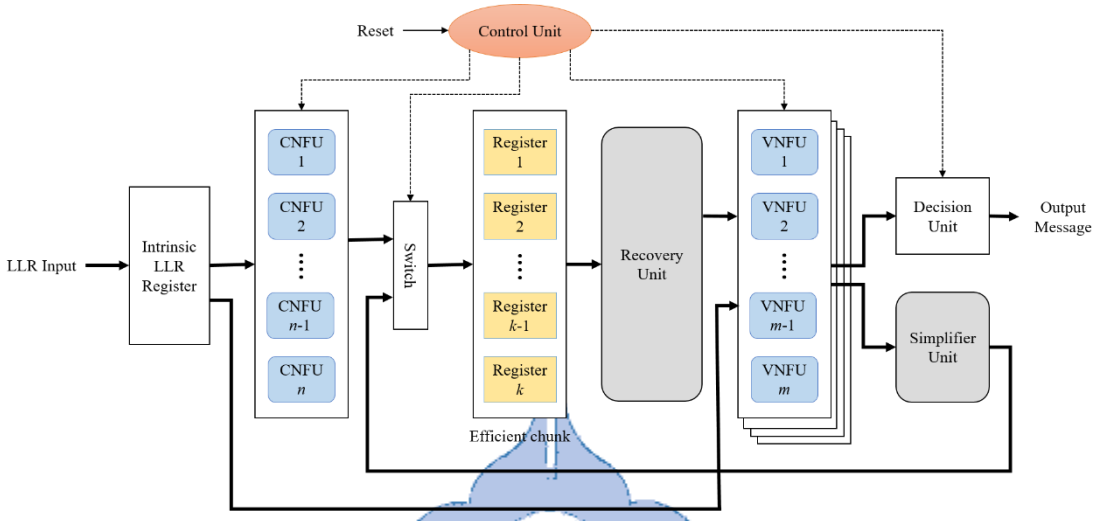


Figure 3-13 The proposed 5G NR LDPC decoder architecture.

In the process of 5G NR LDPC decoding, the intrinsic LLR values are stored into the intrinsic register before the first iteration. Then, the CNFU computes the check-to-variable messages from the initial data according to the equation (12) at first. The desired elements of the message are sent to the efficient chunk for storage. The recovery unit will prepare the corresponding check-to-variable message which is based on the equations (14) and (15) for the following variable node processing. It is worth noting that the recovery of a correct message needs the previously computed data from the efficient chunk. Next, the VNFU computes belief sum which is the sum of the recovered messages as described in equation (16). The variable-to-node messages are also computed from the message ($L(r_{ij})$) excluding itself conveyed message by VNFU.

Due to the structure of quasi-cyclic in 5G NR LDPC, the process of VNFU can be implemented with the column-by-column method. The computation of variable node computation will spend merely Z_c times during the process and therefore each column of the protograph can implement decoding algorithm in parallel. Finally, the simplifier will collect the desired information from the VNFU computation and update it to the chunk for the next iteration. The process of LDPC decoding stops when the corresponding messages derived from the decision unit satisfy the termination condition ($\hat{c} \times H = 0$) or the process exceeds the defined maximum iterations.

Chapter 4 Experimental Results

This chapter presents the experimental result of assigned columns in each set for 5G BG1 and BG2 with Z_c = 256 through the pre-procession which is described in Chapter 3. Section 4.1 introduces the final allotment for BG1 with Z_c = 256 which are processed in Section 3.1 and also explains the reason why Set 2 in the result still has an all zero rows after the swap. In Section 4.2, we consider the second base graph (BG2) with the same lifting size (Z_c) and also provide its final result through the pre-procession.

4.1 The Result of Pre-procession for 5G NR LDPC BG1 with Z_c = 256

The purpose of this section is to discuss the result of the proposed pre-procession for 5G NR LDPC BG1 with Z_c = 256 when all columns in the protograph are assigned into the associated column sets to prevent the memory access conflict. The procedure of the pre-procession has been already described in Section 3.1. Next, we will analyze the allotment result of BG1 with Z_c = 256 which is shown in Table 3-1, whose notation is in Table 3-2. During the third phase of pre-procession, all columns of the base graph are already assigned into three column sets as Figure 3-6 presents, and we count the number of nonzero elements for different rows in each column set (See Figure 3-7). However, we find that Set 2 and Set 3 have several all zero rows which are not computed during the decoding process. The situation leads to a waste of the associated processing unit which is unused at the iteration, and it also reduces the efficiency of the parallel decoding algorithm. To avoid certain conditions, the aim of swapping in the last phase is to exchange one column in the set with a selected column from another set.

Figure 3-11 shows the final allotment result for BG1 with Z_c = 256. The column indexes of every column set are described in (a), and we count the number of nonzero elements in one row for all sets again in (b). The result shows that the value of row counting for Set 2 is 0 in the fifth row, which implies that the fifth row of Set 2 is still an all zero row. Therefore, we explain the reason why the unwanted situation of all zero rows arises in the final result. We observe that the 27th column of the extension parity part has a 0 offset at the 5th row in BG1 which is described in Figure 4-1. However, the 27th column of BG1 with Z_c = 256 is assigned into Set 3. It is worth mention that the number of nonzero elements in the fifth row in Set 3 is 1 which is derived from the 27th column. When Set 2 swap the desired column with Set 3, and it results in that Set 3 ends up in the same situation as Set 2. This explains the reason why Set 2 still has an all zero row after the swap.

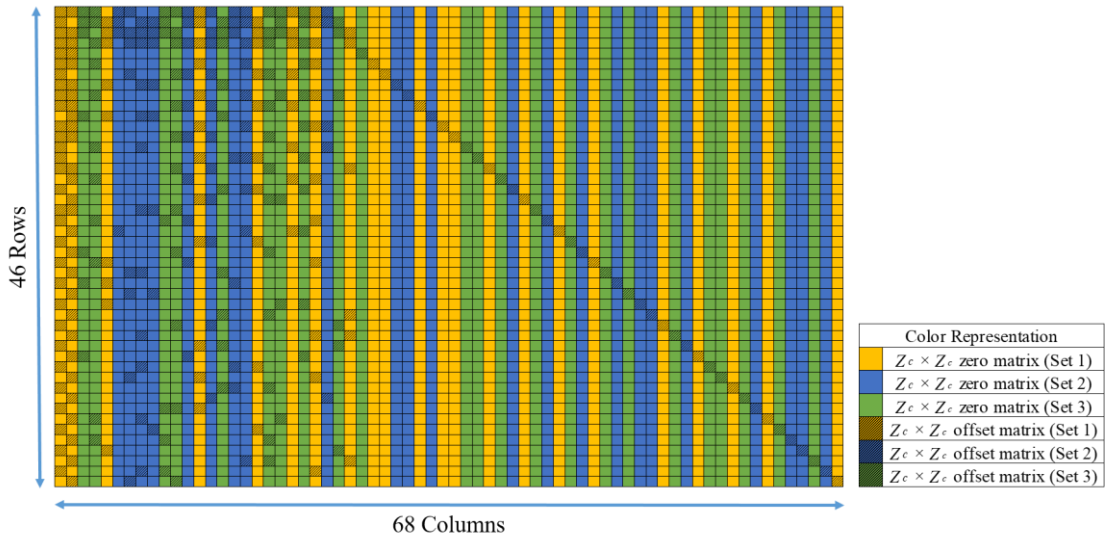


Figure 4-1 The final result of assigned column in each set for BG1 with $Z_c = 256$.

Figure 4-1 is a simplified graphical presentation of BG1, which shows the final allotment of columns in the associated sets for the base graph with $Z_c = 256$. The different background colors are used to symbolize the assigned columns in each set. The columns of Set 1 are marked by the orange background color. The blue one is used to denote the columns in Set 2, and the assigned columns in Set 3 are highlighted in green. The empty square represents a zero matrix of size 256×256 (Note that the current lifting size: $Z_c = 256$), whereas the square with slash denotes the different cyclically shifted matrix of the same size. Finally, the 22 columns are assigned into Set 1. Both Set 2 and Set 3 have the same number of assigned columns, whose value equals 23. The practical result of all column sets is presented in Appendix I. For the numbers of row counting in each set shown in Figure 3-11 (b), it can be observed that the maximum values of the estimated row counting are concentrated in the first four rows of these sets. This can be explained by the fact that the base graph has dense nonzero elements in the block, surrounded by the core check rows and information columns.

4.2 The Case of BG2 with $Z_c = 256$

Next, we consider the case of BG2 with the same lifting size: $Z_c = 256$. The second base graph of size 42×52 is suitable for the small block size and low code rate in channel coding. We find that BG2 with $Z_c = 256$ also has the unwanted situation of the repeated shift values, which results in the memory access collision when it executes the decoding algorithm in parallel. Therefore, Table 4-1 illustrates the base matrix with the positive shift value of (i, j)-th entry P_{ij}. The different repeated elements in the same row are highlighted in multiple colors, whose representations are described in Table 4-2.

Table 4-1 The memory conflict problem in BG2 with Z_c = 256.

H_{BG2} (Z_c=256)								
Row i	Column j	P_{ij}	Row i	Column j	P_{ij}	Row i	Column j	P_{ij}
1	1	9	10	20	0	26	1	228
	2	117	11	1	11		6	156
	3	204		2	185		36	0
	4	26		7	0	27	3	29
	7	189		8	117		8	143
	10	205		21	0		13	160
	11	0	12	1	11		14	122
	12	0		8	236		37	0
2	1	167		10	210	28	1	8
	4	166		14	56		7	151
	5	253		22	0		38	0
	6	125	13	2	63	29	2	98
	7	226		4	111		3	101
	8	156		12	14		6	135
	9	224		23	0		39	0
	10	252	14	1	83	30	1	18
	12	0		2	2		5	28
	13	0		9	38		40	0
3	1	81		14	222	31	3	71
	2	114		24	0		6	240
	4	44	15	2	115		8	9
	5	52		7	145		10	84
	9	240		12	3		41	0
	11	1		14	232	32	2	106
	13	0		25	0		14	1
	14	0	16	1	51		42	0
4	2	8		11	175	33	1	242
	3	58		12	213		6	44
	5	158		26	0		13	166
	6	104	17	2	203		43	0
	7	209		10	142	34	3	132
	8	54		12	8		8	164
	9	18		13	242		11	235
	10	128		27	0		44	0
	11	0	18	2	254	35	1	147
	14	0		6	124		13	85
5	1	179		12	114		14	36
	2	214		13	64		45	0
	12	71		28	0	36	2	57
	15	0	19	1	220		6	40
6	1	231		7	194		12	63
	2	41		8	50		46	0
	6	194		29	0	37	1	140
	8	159	20	1	87		3	38
	12	103		2	20		8	154
	16	0		11	185		47	0
7	1	155		30	0	38	11	219
	6	228	21	2	26		14	151
	8	45		5	105		48	0
	10	28		12	29	39	2	31
	12	158		31	0		6	66
	17	0	22	1	76		12	38
8	2	129		9	42		49	0
	6	147		14	210	40	1	239
	8	140		32	0		8	172
	12	3	23	2	222		13	34
	14	116		3	63		50	0
	18	0		33	0	41	3	0
9	1	142	24	1	23		11	75
	2	94		4	235		14	120
	13	230		6	238		51	0
	19	0		34	0	42	2	129
10	2	203	25	2	46		6	229
	9	205		3	139		12	118
	11	61		10	8		52	0
	12	247		35	0			

Table 4-2 Color representation related to the rows with repeated shift values in Table 4-1.

Row	Column	Repeated Value	Background Color
1	11, 12	0	RED
2	12, 13	0	GREEN
3	13, 14	0	BLUE
4	11, 14	0	YELLOW
41	3, 51	0	ORANGE

Table 4-1 shows the structure of BG2 with $Z_c = 256$, and it merely denotes the non-negative offset values in the base graph. See Appendix II for a complete and graphical description that comprises the element of -1. Table 4-2 indicates what position is the memory access collision for BG2 with $Z_c = 256$. There are five repeated offsets in different rows of the base graph, and they are highlighted in multiple background colors. It can be observed that the first four rows and the 41st one have the same value of repeated offset as 0. Therefore, we assigned all columns of the base graph to the associated column sets through the proposed pre-procession. They are assigned equally in two sets. Figure 4-2 (a) illustrates the column index of each set, and we also count the number of nonnegative elements in each row again to avoid all zero rows.

Index of column in each set	
Set 1	Set 2
11	12
52	51
50	26
48	47
46	45
44	43
42	41
40	39
17	37
36	35
34	33
32	22
30	29
28	27
49	25
24	23
31	21
20	19
18	38
16	15
1	2
10	8
3	6
7	9
4	5
13	14

(a)

Number of noonzero elements in each row		
Row index	Set 1	Set 2
1	6	2
2	5	5
3	4	4
4	4	6
5	1	3
6	2	4
7	3	3
8	1	5
9	2	2
10	2	3
11	2	3
12	2	3
13	1	3
14	2	3
15	1	4
16	2	2
17	2	3
18	2	3
19	2	2
20	3	1
21	1	3
22	2	2
23	1	2
24	3	1
25	2	2
26	2	1
27	2	3
28	2	1
29	1	3
30	2	1
31	2	3
32	1	2
33	2	2
34	3	1
35	2	2
36	1	3
37	2	2
38	2	1
39	1	3
40	3	1
41	2	2
42	1	3

(b)

Figure 4-2 (a) The index of assigned column in each set after the pre-procession. (b) The number of nonzero elements in each row for all column sets.

Figure 4-2 (b) indicates that both Set 1 and Set 2 have at least one nonnegative cyclically shifted value in each row, which implies that there are no all zero rows in these column sets. It also explains that the swapping phase plays an important role in the proposed pre-procession. We use a simplified graphical description Figure 4-3 to show the assigned columns in each set. They are highlighted with two background colors. Set 1 takes the orange color to label its columns, and the blue one is used to denote them in Set 2. The complete description of each set is shown in Appendix II.

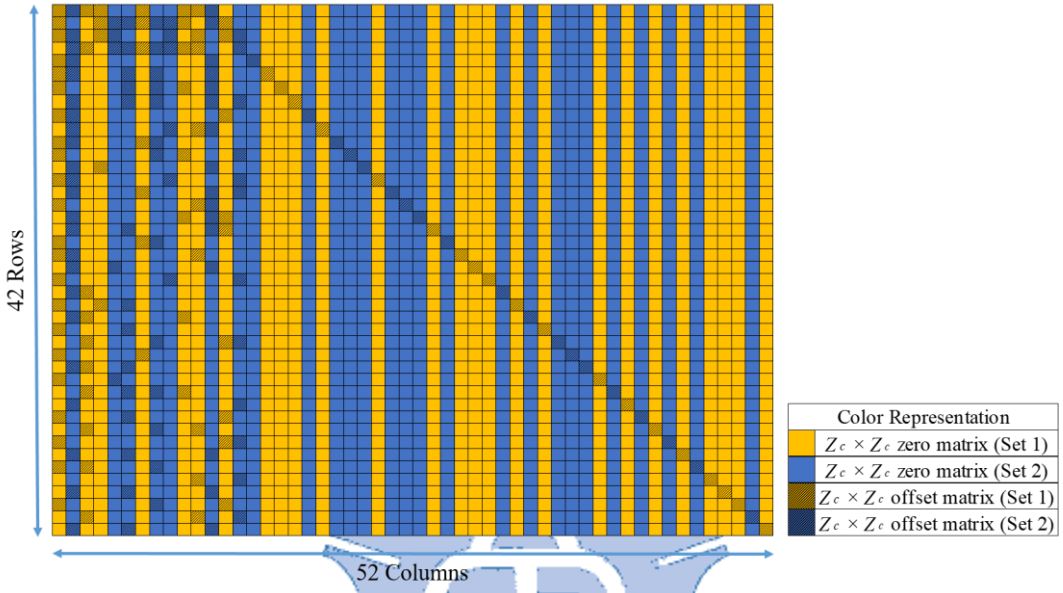


Figure 4-3 The final result of assigned column in each set for BG2 with $Z_c = 256$.

Finally, we compare the number of repeated offsets in both BG1 and BG2 with the same lifting size $Z_c = 256$. It can be observed that BG2 has fewer memory conflicts than BG1. The base graph BG2 is considered for the small block size and low code rate, and therefore most of the cyclically shifted values are distinct in the same row due to its smaller size. We find that all of the repeated offsets in BG2 are 0, which indicates that other positive values appear once in the same row. Moreover, each assigned column set has the maximum value of its row counting in one of the first four rows for BG1 and BG2. It can be explained that the first four rows of these two base graphs are considered as the core parity rows, which result in that there is a group of nonnegative cyclically shift values in this region. When all columns of the base graph are assigned into the associated sets, the maximum values must lie in one of the first four rows.

Chapter 5 Conclusion

Section 5.1 concludes this report and Section 5.2 provide the prospects for the future.

5.1 Conclusion

This report aims to design an efficiently parallelized LDPC decoder for 5G NR. We employ the simplified min-sum algorithm NMSA as our main LDPC decoding algorithm to implement the iterative computation. The efficient chunk scheme is introduced into the hardware architecture. It provides a temporary register to store the computed variable-to-check messages, and which can be converted to the desired check-to-variable messages by the recovery units. The parallelization of message computation with column-by-column design is also introduced in the proposed LDPC decoding for 5G NR due to its quasi-cyclic structure. Each submatrix of the base graph can be implemented on the parallelization of message computation. This enables to reduce the decoding latency and increase the data throughput. However, the memory access conflicts exist in the submatrices with the same offsets when the corresponding processing units compute the iterative decoding algorithm in parallel. It takes the proposed decoder additional delay time to solve it. Therefore, we propose a pre-procession to assign all columns of the 5G NR base graph into different column sets. The prevention of memory access collision is crucial to verify no repeated columns in each set. Finally, we take both BG1 and BG2 with the lifting size $Z_c = 256$ for examples. The result shows that all columns of BG1 are assigned into three column sets. The case of BG2 is with two sets through the pre-procession. We also count the number of nonnegative offsets in each row for these column sets. This implies the utilization of the processing unit. The method of swapping is proposed in the final phase of the pre-procession to optimize the utilization. The results show that BG1 has only an all zero row in Set 2 after the swap. Similarly, the case of BG2 is optimized in the same way.

5.2 Future Prospects

The main feature of our proposed parallel decoder for 5G NR LDPC is the memory architecture of efficient chunks and the parallelization of message computation with column-by-column design. We expect that they can be employed for the design of the non-binary LDPC decoder in the future. It is worth noting that memory access conflicts may happen in this case. Therefore, we consider the proposed pre-procession in the report as the reference for solving the uncertain problem in the near future.

Appendix

I. The Memory Access Conflicts Problem in BG1 with Z_c = 256

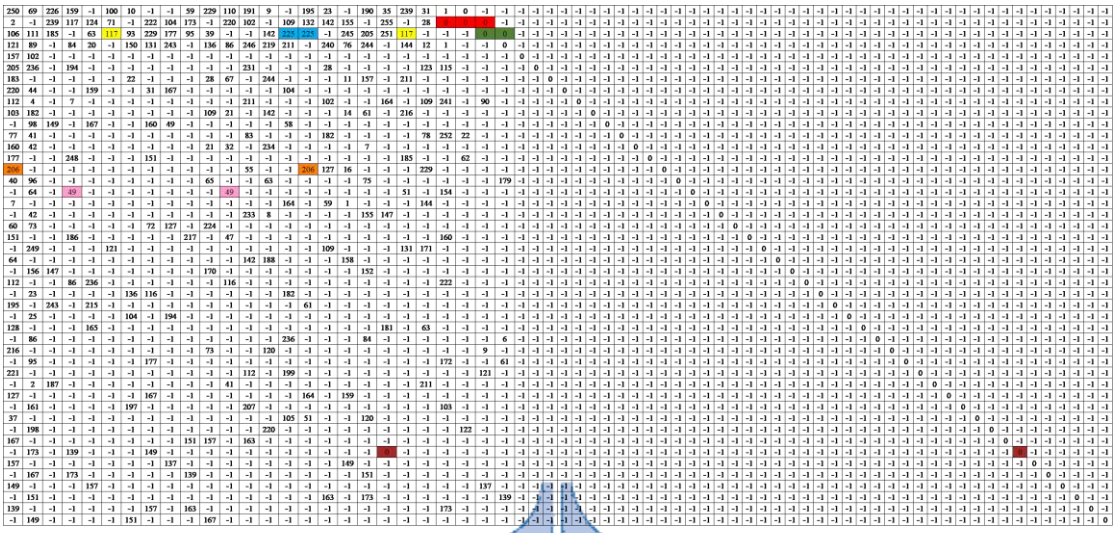


Figure I-1 The memory conflict problem in BG1 with Z_c = 256.

Figure I-1 illustrates the condition of memory access collision in base graph BG1 with lifting value Z_c = 256. There are five rows with the same cyclically shifted value, and they are denoted in multiple background colors as shown in Table I-1. The 2nd, 15th, 17th, and 40th only have one repeated value but the 3rd row has three different duplicate offsets. The rows with the same shifted value will cause the memory conflict during the process of updating the variable-to-check message in the efficient chunk. The following figures present the practical result of Set 1, Set 2, and Set 3, and each column of the set represents the assigned column through the pre-procession. Set 1 has 23 columns in Figure I-2. The element -1 of the base matrix denotes a zero matrix of size 256×256.

Table I-1 Color presentation related to the rows with repeated shift values in Figure I-1.

Row	Column	Repeated Value	Background Color
2	23, 24, 25	0	RED
3	6, 21	117	YELLOW
	15, 16	225	BLUE
	25, 26	0	GREEN
15	1, 16	206	ORANGE
17	4, 12	49	PINK
40	20, 62	0	BROWN

Set 1																						
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	69	-1	191	-1	250	239	-1
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	155	102	124	2	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	111	245	-1	63	106	117	0
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	89	76	246	20	121	144	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	102	-1	-1	-1	157	-1	-1
115	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	236	-1	231	-1	205	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	11	-1	-1	183	211	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	44	-1	-1	159	220	-1	-1
241	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	4	-1	211	-1	112	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	182	14	-1	-1	103	216	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	98	-1	-1	167	-1	-1	-1
252	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	41	-1	83	-1	77	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	42	-1	-1	-1	160	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	177	185	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	16	55	-1	206	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	96	-1	-1	-1	40	-1	179	
154	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	64	-1	-1	-1	-1	51	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	7	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	42	-1	233	-1	-1	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	73	-1	-1	-1	60	-1	-1	
160	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	151	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	249	-1	-1	-1	-1	131	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	158	142	-1	64	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	156	-1	-1	-1	-1	-1	-1	
222	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	236	112	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	23	-1	-1	-1	-1	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	215	195	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	25	-1	-1	-1	-1	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	165	128	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	86	-1	-1	-1	-1	-1	6	
-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	216	-1	-1	
172	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	95	-1	-1	-1	-1	-1	61	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	112	-1	221	-1	-1	
-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	159	-1	-1	127	-1	-1	
108	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	161	-1	207	-1	-1	-1	-1	
-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	37	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	198	-1	-1	-1	-1	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	163	-1	167	-1	-1	
-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	173	-1	-1	-1	-1	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	149	-1	-1	157	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	167	-1	-1	-1	-1	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	157	149	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	151	-1	-1	-1	-1	-1	139	
173	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	139	-1	-1	
-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	149	-1	-1	-1	-1	-1	-1	

Figure I-2 The final assigned columns of Set 1 for BG1 with Z_c = 256.

Set 2 has 23 assigned columns which are shown in Figure I-3.

Set 2																						
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	9	-1	110	23	10	195	-1	100
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	222	220	142	-1	132	104	71
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	142	229	-1	-1	93	225	177	117
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	219	131	86	240	150	-1	243	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	28	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	244	-1	67	-1	22	-1	-1	-1
-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	31	-1	-1	-1	-1	167	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	102	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	142	-1	21	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	160	-1	-1	-1	-1	49	-1
22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	182	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	234	-1	32	-1	-1	-1	-1	-1
62	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	151	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	127	-1	206	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	63	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	49	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	59	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	72	-1	-1	-1	-1	127	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	47	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	109	-1	-1	-1	121
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	188	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	116	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	116	-1	-1	136	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	61	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	104	-1	194	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	120	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	177	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	41	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	167	-1	-1	-1	164	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	197	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	51	-1	-1
122	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	220	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	149	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	137	-1
-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	163	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	157	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	151	-1	-1	-1

Figure I-3 The final assigned columns of Set 2 for BG1 with Z_c = 256.

Figure I-4 presents that Set 3 also has 23 assigned columns in total.

Set 3																						
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	190	31	59	159	229	226	-1	35
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	28	173	117	-1	239	109	255
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	205	-1	95	-1	39	185	225	251
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	244	12	-1	84	136	-1	211	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	123	-1	194	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	157	-1	-1	-1	28	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	104	-1
90	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	109	-1	7	-1	-1	-1	164
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	61	-1	-1	-1	109	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	149	58	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	78	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	7	-1	-1	-1	21	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	248	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	229	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	75	-1	-1	-1	65	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	49	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	144	-1	-1	-1	-1	164	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	155	-1	-1	-1	-1	-1	-1	147
-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	224	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	217	186	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	171	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	152	-1	-1	-1	170	147	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	86	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	182	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	243	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	63	-1	-1	-1	-1	-1	181
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	84	-1	-1	-1	-1	-1	236	-1
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	73	-1	-1	-1
-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
121	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	199	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	211	-1	-1	-1	187	-1	-1
-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	120	-1	-1	-1	-1	-1	105	-1
-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	151	-1	157	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	139	-1	-1	-1	0
-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	151	-1	139	173	-1	-1	-1	-1
137	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	173	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	163	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	167	-1	-1	-1

Figure I-4 The final assigned columns of Set 3 for BG1 with Z_c = 256.

II. The Memory Access Conflicts Problem in BG2 with Z_c = 256

Figure II-1 is a graphical description to illustrate the memory collision problem of BG2 with the same Z_c, and multiple background colors in Table II-1 are used to denote the repeated elements of each row in the base graph.

9	117	204	26	-1	-1	189	-1	-1	205	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
167	-1	-1	166	253	125	226	156	224	252	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
81	114	-1	44	52	-1	-1	-1	240	-1	1	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	8	58	-1	158	104	209	54	18	128	0	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
179	214	-1	-1	-1	-1	-1	-1	-1	-1	-1	71	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
231	41	-1	-1	-1	194	-1	159	-1	-1	-1	103	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
155	-1	-1	-1	-1	228	-1	45	-1	28	-1	158	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	129	-1	-1	-1	147	-1	140	-1	-1	-1	3	-1	116	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
142	94	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	230	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	203	-1	-1	-1	-1	-1	-1	205	-1	61	247	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	185	-1	-1	-1	-1	0	117	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	-1	-1	-1	-1	-1	-1	236	-1	210	-1	-1	-1	56	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	63	-1	111	-1	-1	-1	-1	-1	-1	-1	14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
83	2	-1	-1	-1	-1	-1	-1	38	-1	-1	-1	-1	222	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	115	-1	-1	-1	-1	145	-1	-1	-1	-1	3	-1	232	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
51	-1	-1	-1	-1	-1	-1	-1	-1	-1	175	213	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	203	-1	-1	-1	-1	-1	-1	-1	142	-1	8	242	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	254	-1	-1	-1	124	-1	-1	-1	-1	-1	114	64	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
220	-1	-1	-1	-1	-1	194	50	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
87	20	-1	-1	-1	-1	-1	-1	-1	-1	185	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	26	-1	-1	105	-1	-1	-1	-1	-1	-1	29	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
76	-1	-1	-1	-1	-1	-1	-1	42	-1	-1	-1	-1	210	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	222	63	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
23	-1	-1	235	-1	238	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	46	139	-1	-1	-1	-1	-1	-1	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
228	-1	-1	-1	-1	156	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	29	-1	-1	-1	-1	143	-1	-1	-1	-1	160	122	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1	151	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	98	101	-1	-1	135	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	-1	-1	-1	28	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	71	-1	-1	240	-1	9	-1	84	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	106	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
242	-1	-1	-1	-1	44	-1	-1	-1	-1	-1	-1	166	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	132	-1	-1	-1	-1	164	-1	-1	235	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1
147	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	85	36	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1
-1	57	-1	-1	-1	40	-1	-1	-1	-1	-1	63	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1
140	-1	38	-1	-1	-1	-1	154	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	219	-1	-1	151	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1
-1	31	-1	-1	-1	66	-1	-1	-1	-1	-1	38	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1
239	-1	-1	-1	-1	-1	-1	172	-1	-1	-1	-1	34	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1
-1	-1	0	-1	-1	-1	-1	-1	-1	-1	75	-1	-1	120	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1
-1	129	-1	-1	-1	229	-1	-1	-1	-1	-1	118	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0

Figure II-1 The memory conflict problem in BG2 with Z_c = 256.

Table II-1 Color representation related to the rows with repeated shift values in Figure II-1.

Row	Column	Repeated Value	Background Color
1	11, 12	0	RED
2	12, 13	0	GREEN
3	13, 14	0	BLUE
4	11, 14	0	YELLOW
41	3, 51	0	ORANGE

Table II-1 describes that there are five memory access conflicts in the base graph BG2 with Z_c = 256. All of the first four rows and the 41st row have the same repeated offset 0 in two distinct columns, and they are highlighted in different background colors. The first four rows use the red, green, blue, and yellow background colors respectively to denote their repeated columns. The orange one represents the case of the 41st row. Therefore, we assigned all columns of the base graph into two column sets through the pre-procession which is presented in Chapter 3. Figure II-2 shows that Set 1 has 26 columns. Each column of the matrix corresponds to the assigned column from the base graph. Similarly, Set 2 also owns 26 columns which are described in Figure II-3.

Set 1

0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	9	205	204	189	26	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	167	252	-1	226	166	0
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	81	-1	-1	-1	44	0
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	128	58	209	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	179	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	231	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	155	28	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	142	-1	-1	-1	-1	230
61	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	11	-1	-1	0	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	11	210	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	111	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	83	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	145	-1	-1
175	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	51	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	142	-1	-1	-1	242
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	64
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	220	-1	-1	194	-1	-1
185	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	87	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	76	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	63	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	23	-1	-1	-1	235	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8	139	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	228	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	29	-1	-1	160
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8	-1	-1	151	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	101	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	18	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	84	71	-1	-1	-1
-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	242	-1	-1	-1	-1	166
235	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	132	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	147	-1	-1	-1	-1	85
-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	140	-1	38	-1	-1	-1
219	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	239	-1	-1	-1	-1	34
75	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure II-2 The final assigned columns of Set 1 for BG2 with Z_c = 256.

Set 2																									
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	117	-1	-1	-1	-1	-1
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	156	125	224	253	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	114	-1	-1	240	52	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8	54	104	18	158	0
71	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	214	-1	-1	-1	-1	-1
103	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	41	159	194	-1	-1	-1
158	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	45	228	-1	-1	-1
3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	129	140	147	-1	-1	116
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	94	-1	-1	-1	-1
247	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	203	-1	-1	205	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	185	117	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	236	-1	-1	-1	56
14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	63	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	-1	38	-1	222
3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	115	-1	-1	-1	-1	232
213	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	203	-1	-1	-1	-1	-1
114	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	254	-1	124	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	50	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	20	-1	-1	-1	-1	-1
29	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	26	-1	-1	-1	105	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	42	-1	210
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	222	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	238	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	46	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	156	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	143	-1	-1	-1	122
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	98	-1	135	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	28	-1
-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	9	240	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	106	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	44	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	164	-1	-1	-1	-1
-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	36
63	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	57	-1	40	-1	-1	-1
-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	154	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	151
38	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	31	-1	66	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	172	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	120
118	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	129	-1	229	-1	-1	-1

Figure II-3 The final assigned columns of Set 2 for BG2 with Z_c = 256.

REFERENCE

[1] 3rd Generation Partnership Project. Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding. TS 36.212, V 16.5.0 Release 16. Mar. 2021.

[2] 3rd Generation Partnership Project. Technical Specification Group Radio Access Network; NR; Multiplexing and channel coding. TS 38.212, V 16.5.0 Release 16. Mar. 2021.

[3] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379-423, July. 1948.

[4] Gallager, R. G., “Low Density Parity Check Codes,” M.I.T. Press, 1963.

[5] D. J. C. MacKay, “Good error-correcting codes based on very sparse matrices,” *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399-431, Mar. 1999.

[6] IEEE, “IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput,” in IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009) , vol., no., pp.1-565, 29 Oct. 2009.

[7] IEEE, “IEEE Standard for Local and Metropolitan Area Networks - Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems - Amendment for Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands,” in IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor 1-2005 (Amendment and Corrigendum to IEEE Std 802.16-2004) , vol., no., pp.1-822, 28 Feb. 2006.

[8] 陳建宇, IEEE 802.16e LDPC Codes 高效能解碼器之設計與實作, 國立中正大學通訊工程研究所碩士論文，July 2012

[9] Tram Thi Bao Nguyen, Tuy Nguyen Tan and Hanho Lee, "Efficient QC-LDPC Encoder for 5G New Radio," *Electronics*, Vol. 8, Issue 6, Jun. 2019.

[10] R. Tanner, “A recursive approach to low complexity codes,” *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533-547, Sept. 1981.

[11] T. Richardson and S. Kudekar, "Design of Low-Density Parity Check Codes for 5G New Radio," *IEEE Communications Magazine*, Vol. 56, Issue 3, pp. 28-34, March. 2018.

[12] D. Hui, S. Sandberg, Y. Blankenship, M. Andersson and L. Grosjean, "Channel

Coding in 5G New Radio: A Tutorial Overview and Performance Comparison with 4G LTE," *IEEE Vehicular Technology Magazine*, Vol. 13, Issue 4, pp. 60-69, Dec. 2018.

[13] Jung Hyun Bae, Ahmed Abotabl, Hsien-Ping Lin, Kee-Bong Song and Jungwon Lee, "An overview of channel coding for 5G NR cellular communications," APSIPA Transactions on Signal and Information Processing, Vol. 8, Jun. 2019.