Reduced-Complexity VLSI Architectures for Binary and Nonbinary LDPC Codes

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Sangmin Kim

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Professor Gerald E. Sobelman, Adviser

August, 2010

Acknowledgements

I like to thank a number of people for their contribution to my PhD study. First of all, I would like to express my sincerest gratitude to my adviser, Professor Gerald E. Sobelman, for his guidance and support throughout my research at University of Minnesota. Without him, this dissertation would not have been possible. I would also like to thank Professor Keshab K. Parhi, Professor Emad S. Ebbini, and Professor Paul Garrett for support as my committee members.

I would like to thank Electronics and Telecommunications Research Institute (ETRI) for supporting a portion of the research.

I thank to Juyul Lee and Heon Hwa Cheong for their assistance and encouragement in improving my papers, including journal and conference publications, along with many discussions. I also thank my friends at this University and elsewhere for their assistance to continue my studies.

Above all, I wish to express my deepest thanks to my parents and brothers for their immeasurable love and encouragement. With their continuous support, the preparation of this thesis would have been possible.

Abstract

This thesis proposes efficient algorithm and architecture aspects for binary and nonbinary low-density parity-check (LDPC) codes by developing optimal quantization approaches, decoding algorithms, decoding schedules and switch networks based on the characteristics of specific codes. To provide a quantitative comparison with previous work, including design performance and cost, we implement and analyze our architectures using a Field Programmable Gate Array (FPGA) platform. The decoding of LDPC codes uses soft information, so it is important to analyze the error correcting performance with fixed-point computations. An adaptive quantization scheme to select suitable input values for the min-sum based decoding algorithm is given. Our simulation results show that it gives good error correcting performance compared with the conventional method. A reduced-complexity LDPC layered decoding architecture is proposed using an offset permutation scheme in the switch networks. Then, a switch network for the code rates defined in the IEEE 802.15.3c standard is optimized by reducing the number of control bits and eliminating unnecessary switch elements. We implement a 672-bit, rate-1/2 irregular LDPC code on a Xilinx Virtex-4 FPGA device and this design achieves an information throughput of 822 Mb/s at a clock speed of 335 MHz a maximum of 8 iterations. We propose an improved nonbinary decoding algorithm with a threshold factor to increase the performance of LDPC decoders. Implementing nonlinear functions as small look-up table leads us consider the dynamic range of the nonlinear functions in order to take more precisely into account the effect of finite precision computation. Finally, an efficient VLSI architecture for a nonbinary LDPC decoder will be presented.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An error-correcting code (ECC) or forward error correction (FEC) code is a system of adding redundant data to a message such that it can be recovered by a receiver even when errors were introduced, either during the transmission (or recording) over a channel (e.g. telephone lines, internet cables, fiber-optic lines, high frequency channels, and cell phone channels), or on storage (e.g. hard drives, diskettes, CD-ROMs, DVDs, flash memory systems, and solid state memory). Many communication or storage channels are subject to channel noise, and thus errors may be introduced during transmission from the transmitter and a receiver. Therefore, error correction techniques have been one of the most significant parts in modern communication systems.

FEC is used in error control strategies for a one-way system, while automatic repeat request (ARQ) is employed in error detection and retransmission for a two-way system. In an ARQ system, when errors are detected at the receiver, a request is sent for the transmitter to retransmit the message, and repeat requests continue to be sent until it is correctly received or the error persists beyond a predetermined number of retransmissions. ARQ is appropriate if the channel has unknown and varying capacity (e.g. internet). However, ARQ results in possibly increased latency due to the retransmissions. In addition, when the channel error rate is high, retransmissions must be sent too frequently, and the system throughput can be lowered by an ARQ system. Therefore, most of the coded systems in today use some form of FEC in a one-way communication system as illustrated in Fig. 1.1 [1][2].

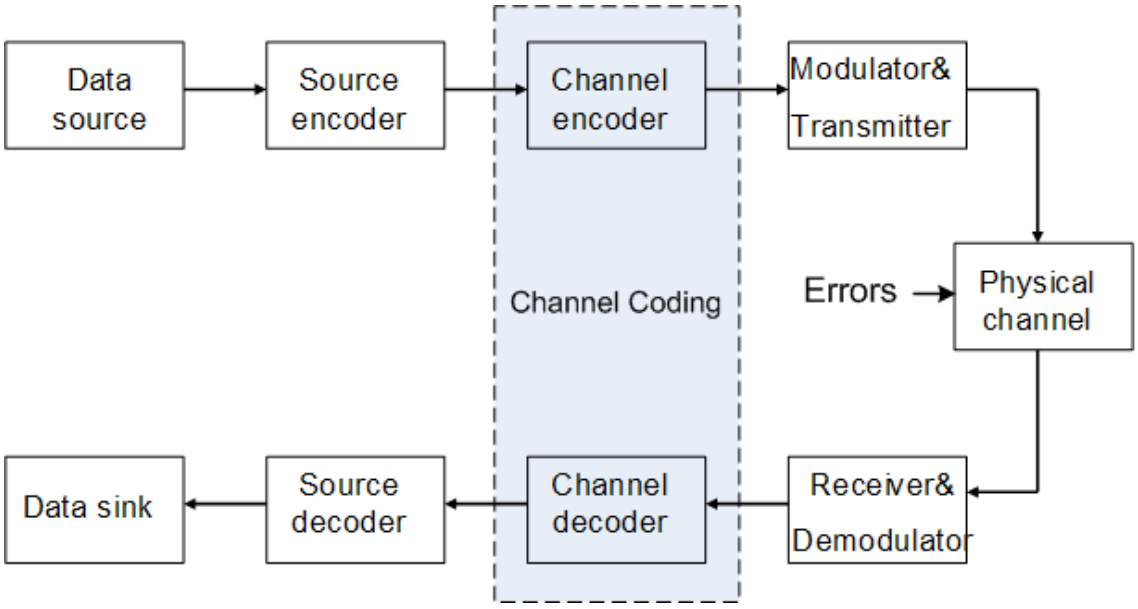


Figure 1.1: Block diagram of a general communication system.

ECCs are structurally distinguished between convolutional codes and block codes. Convolutional codes are processed on bit-by-bit basis, while block codes are processed on a block-by-block basis. Viterbi decoders are known as the optimal decoding for convolutional codes and are easily implemented in vary-large-scale integration (VLSI) hardware. Examples of block codes are repetition codes, Hamming codes, and Reed-Solomon codes. Recently, turbo codes and low-density parity-check (LDPC) codes were constructed to provide almost optimal efficiency. In 2008, LDPC beat convolutional turbo codes to be the FEC scheme for the ITU-T (International Telecommunication Union) G.hn (common name for a new home network technology) standard [3]. Long (binary) LDPC codes with iterative decoding based on belief propagation have shown to achieve an error performance only a fraction of a decibel away from the Shannon limit [68]. However, binary LDPC codes have weaknesses when the code length is small or when high order modulation is applied. In the past few years, the performance of binary LDPC codes has been

improved by an extension to a high-order Galois field (GF(q), where q is a prime number or a power of a prime number). For this class of LDPC codes, which are referred to as nonbinary LDPC codes, all elements in the parity-check matrix are elements over GF(q). It is shown that nonbinary LDPC codes also have superior performance for burst errors. However, this improvement is achieved at the cost of increased decoding complexity.

The main advantages of LDPC codes over turbo codes are their lower decoding complexity and lower error floor at the desired range of operation. In addition, LDPC codes do not need a long interleaver to achieve good error performance and their decoding is not trellis based. Therefore, they are being widely used in wireless communication and network standards and storage devices.

This dissertation deals with efficient high-throughput VLSI architectures for both binary and nonbinary LDPC codes. As communication devices get smaller and need higher data rates with high reliability to meet the high demand for multimedia transmission technologies, efficient low-complexity high-throughput implementation is of great importance for LDPC decoders. We will give a more detailed motivation in the next section.

1.1 Motivation

LDPC codes have attracted much attention because of their excellent error correcting performance and inherently parallelizable VLSI implementation. Therefore they are being widely used in communication standards, such as Digital Video Broadcasting-Satellite-Second Generation (DVB-S2) [4], IEEE 802.3an (10GBase-T) Ethernet [5], IEEE 802.16e Worldwide Interoperability for Microwave Access (WiMAX) [6][23], IEEE 802.11n Wireless Local Area Network (WLAN) [7] and IEEE 802.15.3c Millimeter Wave Wireless Personal Area Networks (WPANs) [24], and storage devices [8][9], such as hard drives, solid state drives and flash memory systems. LDPC codes over finite fields GF(q = 2), which are referred to as binary LDPC codes, have been shown to approach Shannon-limit performance for very long code length [10][11]. For moderate code

lengths, on the other hand, the error performance can be improved by increasing q. One of the most challenging issues in decoding LDPC codes over nonbinary field GF(q) is the computational complexity.

From the hardware engineering perspective, for the development for algorithms and architectures of iterative error correcting codes such as both binary and nonbinary LDPC codes, an important issue is the co-design of algorithms and architectures for achieving a high-throughput low-complexity LDPC encoder/decoder for the specific applications. In this dissertation, we make efforts to improve the decoding performance and reduce the computational complexity of such decoders. We now point out algorithm developments and low complexity architectures for both binary and nonbinary LDPC codes decoders.

There have been a significant amount of studies of decoding algorithms for binary LDPC codes. It is well know that iterative belief propagation (BP) or the sum-product algorithm can achieve the best decoding performance. *Probabilities* or *beliefs*, which are usually represented as real number values, in the belief propagation algorithm are propagated through the structure of LDPC codes. Therefore, it is very important to analyze the finite precision effects on the performance of LDPC codes. This behavior analysis can provide the optimal performance in determining finite word lengths of the decoder as far as the tradeoffs between error performance and hardware complexity are concerned. In this dissertation, we deal with adaptive quantization schemes in the approximated decoding (such as min-sum) algorithm considering scaling effects to improve the performance of an LDPC decoder.

All of the LDPC codes in the above communication standards are based on "Architecture-Aware LDPC codes" [25] or "Block-LDPC codes" [26]. The parity-check matrix **H** of these codes is partitioned into block-columns and block-rows, which are particularly suitable for VLSI implementations by simplifying memory access and message passing. Therefore, partially parallel implementations are being usually used in designing decoders of structured LDPC codes. In many

cases, the structured LDPC codes for most standard wireless communication systems adopt different code rates and block sizes depending on the channel circumstances. A flexible LDPC decoder is desirable in order to satisfy the requirements of wireless communication systems. In this dissertation, we will develop a flexible high-throughput LDPC decoder architecture using a block-parallel scheduling scheme.

In the partially parallel designs, conventional decoders use a bidirectional network or two switch networks for shuffling and reshuffling messages, which results in increasing the hardware complexity. Therefore, it is necessary to develop efficient solutions for LDPC decoders, to be capable of reducing the implementation cost. There are several designs, which are targeted for only one parity-check matrix **H** or specific array code, using one switch network. Our purpose is to develop a design that is suitable for multiple code rates and for different codeword sizes.

In order to reduce the complexity of the BP algorithm for decoding nonbinary LDPC codes, the BP algorithm in the logarithm domain is performed. However, the logarithm and exponential computations used in the check node units may incur overflows in the soft information due to the finite word-length. Investigation of the optimal word-length for nonbinary LDPC decoders should be introduced by selecting the proper word-length for BP algorithms in the logarithm domain.

1.2 Background

LDPC codes are also known as Gallager codes, in honor of Robert G. Gallager, who proposed the LDPC concept in his doctoral dissertation at MIT in 1960 [10]. It was not feasible to implement algorithms for LDPC codes at the time they were developed. Therefore, LDPC codes were forgotten until they are rediscovered by Mackay [12]. LDPC codes are linear block codes obtained from sparse bipartite graphs. We can represent LDPC codes as matrices or bipartite

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

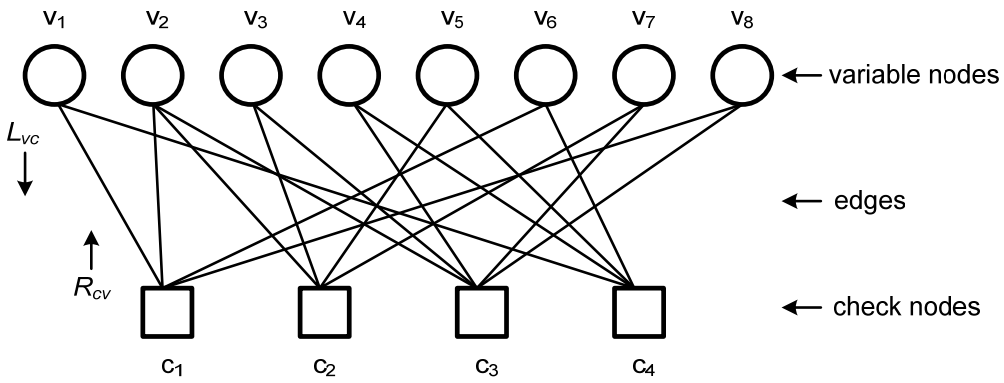


Figure 1.2: Parity-check matrix **H** and bipartite graph of binary LDPC code.

graphs (graphical representation). Fig. 1.2 shows a parity-check matrix **H** and a corresponding bipartite graph (called a Tanner graph). In the matrix representation of a code, each column corresponds to one of the variable nodes while each row corresponds to one of the check nodes. In the bipartite graph, variable nodes indicate bits of a codeword and check nodes indicate check equations. An edge for connecting one variable node to one check node in the bipartite graph is indicated by "1" in the parity-check matrix **H**.

A general class of decoding algorithm for LDPC codes is called message passing algorithms, which are iterative algorithms. The main reason for this name is that messages are passed from check nodes to variable nodes, and from variable nodes back to check nodes. An important aspect is that the message that is sent from a variable node v to a check node c must not involve the

message sent in the previous step from c to v. This is true for messages passed from check nodes c to variable nodes v.

The messages such as L_{vc} and R_{cv} in Fig. 1.2 represent *probabilities* or *beliefs*. The algorithm is also known as *belief propagation* and the LDPC codes can be decoded using iterative belief propagation (BP). In detail, the message R_{cv} passed from c to v is the *probability* or *belief* that v node has certain information (values) given all the messages passed to c node in the previous step from variable nodes other than v. On the other hand, the message passed from v to c is the probability or *belief* that c node has certain information given all the messages passed to v node in the previous step from variable nodes other than c. It is easy to work with likelihoods, or even log-likelihoods, instead of using probabilities to represent messages. In BP, likelihood functions are recursively computed by each node in the graph, and a message containing this information is transmitted along each edge.

One of the great promises of this algorithm is that it can in principle be implemented by fully parallel hardware. In such a scheme, the graph would be laid out as two-dimensional VLSI architecture. Each node in the graph would be instantiated by a hardware module that is able to carry out a simple computation, and each edge would be instantiated by a wire connecting the variable node to the check node. Another advantage is the ability to pipeline the decoder for high-speed implementations in order to reduce the path delay at the cost of registers and latency..

1.2.1 BP and Min-Sum Decoding Algorithms for Binary LDPC codes

In the BP decoding algorithm, messages are denoted by R_{cv} for extrinsic messages from the check node c to the variable (bit) node v and by L_{vc} for extrinsic messages from the variable node v to the check node c. The update operations at the check nodes and variable nodes can be expressed as in equations (1.1) and (1.2), respectively.

$$R_{cv} = -\prod_{n\in N(c),n\neq v} sign(L_{nc}) \cdot \Psi\{\sum_{n\in N(c),n\neq v} \Psi(L_{nc})\} \tag{1.1}$$

$$L_{vc} = \sum_{m\in M(v),m\neq c} R_{mv} - y_v \tag{1.2}$$

$N(c)$ and $M(v)$ denote the set of positions of the columns of **H** and the set of position of the rows of **H** such that $N(c) = \{v|\mathbf{H}_{c,v}=1\}$ and $M(v) = \{c|\mathbf{H}_{c,v}=1\}$, respectively. In equation (1.2), the Psi-function, $\Psi(x) = \log(\tanh(|x/2|))$, is a nonlinear function and y_v is the prior log-likelihood ratio (LLR) given by $2r_v/\sigma^2$, where r_v is the additive white Gaussian noise (AWGN) channel output and σ^2 is the noise variance. At every iteration (one iteration consists of equations (1.1) and (1.2)), the soft decoding result for each bit is determined as follows:

$$L_v = \sum_{c\in M(v)} R_{cv} - y_v \tag{1.3}$$

In the normalized min-sum algorithm, the check node update equation, R_{cv}, is shown as follows.

$$R_{cv} = \alpha \cdot \left(\prod_{n\in N(c),n\neq v} sign(L_{nc})\right) \cdot \min_{n\in N(c),n\neq v} |L_{nc}| \tag{1.4}$$

where α is a scaling factor, which depends on the structure of **H**. The check node update in the offset min-sum algorithm can be represented as in equation (1.5):

$$R_{cv} = \left(\prod_{n\in N(c),n\neq v} sign(L_{nc})\right) \cdot \max\left(\min_{n\in N(c),n\neq v} |L_{nc}| - \beta, 0\right), \tag{1.5}$$

where the offset min-sum algorithm reduces the magnitude by a positive constant β. The decoding algorithm stops if the estimated bits, L_v, satisfy all the parity check equations or if the maximum number of iterations has been reached.

1.2.2 LDPC Decoding Schedules

In this subsection, let us consider a decoding schedule scheme, which plays an important role in the decoding convergence of both binary and nonbinary LDPC codes. Variable nodes and check nodes exchange messages according to a pre-determined schedule. A scheme of determining the update order of extrinsic messages (edge messages) is called a scheduling scheme. This affects the convergence speed based on the iterations of the decoder.

There are two scheduling schemes, which are the standard message passing schedule and the layered decoding schedule. In the standard message passing schedule, all variable node update equations cannot start their computation until all check nodes pass new messages through their edges and *vice versa*. In other words, variable node and check node computations as shown in Fig. 1.3 occur sequentially. In contrast to the standard message passing schedule, the layered decoding schedule, described in [41] and [27], processes the rows or columns of the parity check matrix in layers or groups. This achieves an approximately twice as fast decoding convergence due to the use of intermediate variable-node or check-node message values.

Fig. 1.4 (a) ~ (d) show the layered decoding schedule using column by column updates. Suppose that each message L_{vc} is initialized to y_v (input LLR). To send L_{vc} messages (bold arrow lines) corresponding to the v_1 node, as shown in Fig. 1.4 (a), check nodes (c_1, c_4) should be previously computed by using L_{vc} messages (indicated by dotted lines) related to the check nodes. Fig. 1.4 (b) shows the processing of the second variable node (v_2) through the updated check nodes (c_1, c_2, c_3). The updated L_{vc} (from v_1 to c_1) message is used for generating a new check node (c_1) operation. The processing of the third and fourth variable nodes (v_3, v_4) is shown in Fig. 1.4 (c) and (d). This processing is repeated for the other variable nodes ($v_5 \sim v_8$). After finishing all variable nodes ($v_1 \sim v_8$) in the bipartite graph, the soft decoding result (L_v) for each bit is determined. A row by row update schedule is the converse of the column by column updates.

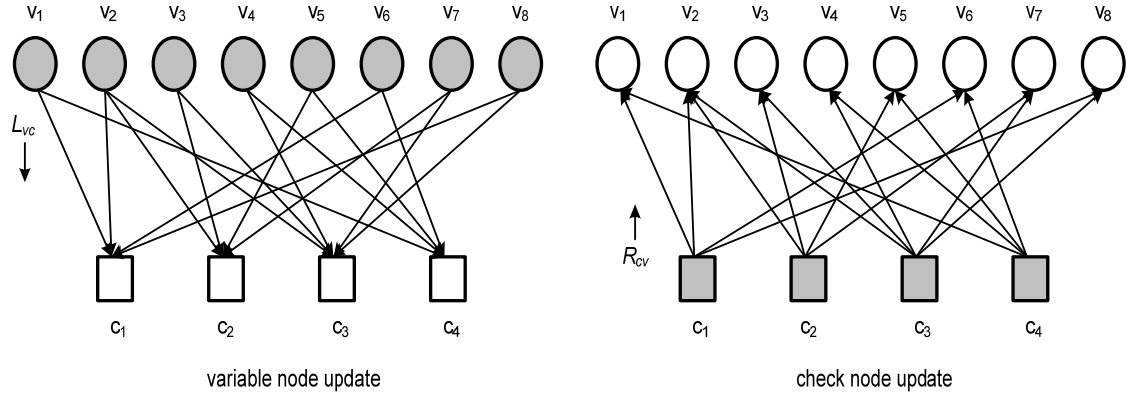


Figure 1.3: Standard message passing schedule for the LDPC iterative decoding algorithms.

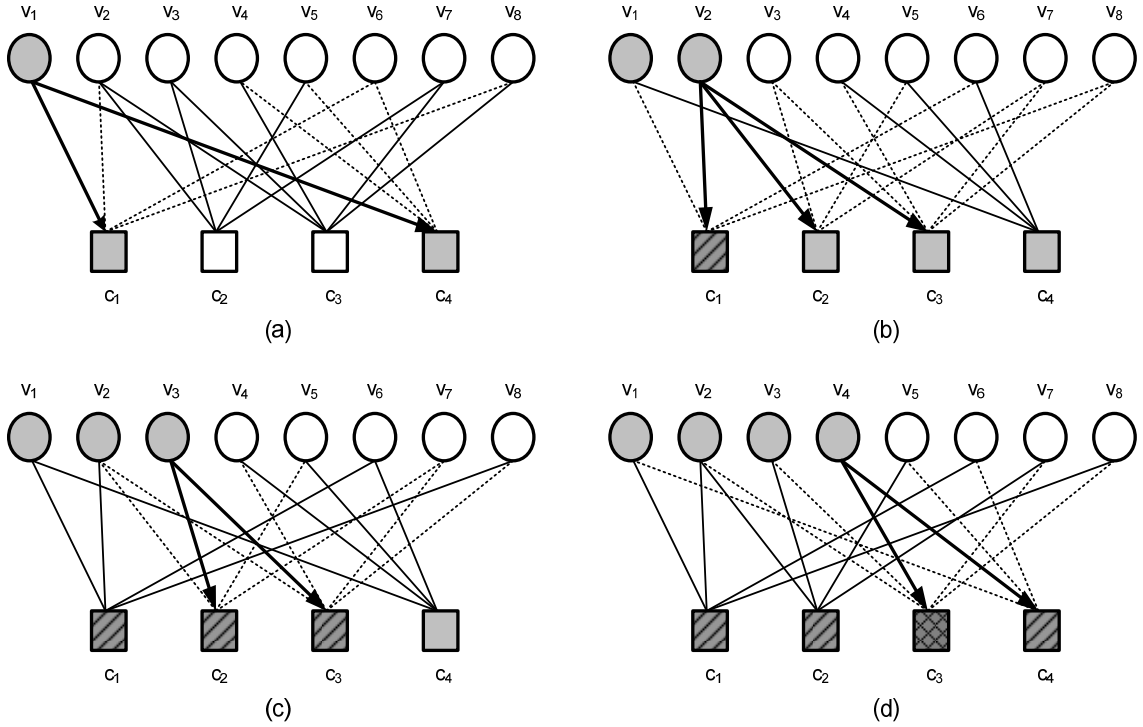


Figure 1.4: Layered decoding schedule for the LDPC iterative decoding algorithms.

1.2.3 Nonbinary LDPC Decoding

In an LDPC code over GF(q) = {0, 1,…, q – 1}, where q is a prime number or a power of a prime number, each entry $h_{m,n}$ in a sparse parity-check matrix **H** of size $M \times N$ is one of the q elements in GF(q). In particular, an LDPC code over GF($q = 2^p$), which is an extension field of GF(2), groups p bits into an element of this field.

In general, a nonbinary LDPC code like the binary LDPC code illustrated in Fig. 1.2, can be expressed using a bipartite graph which is represented by variable nodes, check nodes and edges connecting the variable nodes and the check nodes with each other. A variable node in the nonbinary LDPC codes is a random variable of GF($q = 2^p$), and a message passed through an edge is a vector with size of 2^p.

Nonbinary LDPC codes can be decoded with the BP algorithm using an iterative message-passing algorithm with an increase in decoding complexity. The BP algorithm for nonbinary LDPC codes is not a direct generalization of the binary case because the nonzero values of the parity-check matrix **H** are not binary. Let $\mathbf{C} = [c_1\ c_2\ \ldots\ c_N]^T$ denote the transmitted codeword, where c_n corresponds to a symbol defined over GF(2^p), for $1 \le n \le N$. In other words, a codeword of the nonbinary LDPC code, **C**, is a vector having a length of N and including elements of GF(q), and satisfies (1.6):

$$\sum_{n=1}^{N} h_{m,n} \otimes c_n = 0 \bmod p(x), \quad \forall m \in \{1,...,M\} \tag{1.6}$$

Suppose that a mth row of **H** includes four non-zero elements and the four non-zero elements are, $h_{m,1}$, $h_{m,2}$, $h_{m,3}$, and $h_{m,4}$. Then, codeword **C** satisfies (1.6), as follows:

$$\left(h_{m,1} \otimes c_1\right) \oplus \left(h_{m,2} \otimes c_2\right) \oplus \left(h_{m,3} \otimes c_3\right) \oplus \left(h_{m,4} \otimes c_4\right) = 0 \bmod p(x) \tag{1.7}$$

where $\oplus$ and $\otimes$ represent additive and multiplicative operations, respectively, on GF($q = 2^p$) and $p(x)$ in the modulo operator is a degree p primitive polynomial of GF($q = 2^p$). In this sense, the variable nodes needed to perform the BP algorithm on a check node are not the codeword symbols alone, but the codeword symbols multiplied by nonzero values of the parity-check matrix **H**. Therefore, the equation (1.7) is more complicated than the binary case because we have to consider the nonbinary parity check matrix elements. Moreover, each coded symbol has q likelihoods associated with it. From the hardware engineering perspective, a processing unit that connects the two variable nodes c_n and $h_{m,n} \otimes c_n$ is equal to a permutation of the message values. For example, we can calculate the message from the check node m to variable node $n = 1$ by first substituting all possible nonbinary elements into the coded symbols that satisfy the parity check equation when $c_1 = x$, that is:

$$\left(h_{m,2} \otimes c_2\right) \oplus \left(h_{m,3} \otimes c_3\right) \oplus \left(h_{m,4} \otimes c_4\right) = h_{m,1} \otimes c_1 , \tag{1.8}$$

and then computing the message of each sequence. The permutation that is used in (1.8) corresponds to the multiplication of $h_{m,n}$ from node c_n to node $h_{m,n} \otimes c_n$, and to the division of the indices $h_{m,n}$. This leads to the concept of a convolution of all incoming messages, just as in the binary case, to update check nodes. The decoding problem is to find the most probable vector **L** such that **HL** = 0 mod $p(x)$, where $\mathbf{L} = [L_1\ L_2\ \ldots\ L_N]^T$ is a received vector through a channel and 0 is defined over GF($q = 2^p$).

1.3 Contributions of the Thesis

This thesis focuses on VLSI implementation of both binary and nonbinary LDPC decoders with algorithmic improvements and low-complexity architectures. The contributions of this dissertation are discussed next.

Chapter 2: Adaptive quantization schemes in the normalized min-sum decoding algorithm considering scaling effects to improve the performance of irregular LDPC decoder are introduced. We discuss the finite precision effects on the performance of irregular LDPC codes and propose optimal finite word lengths of variables over an SNR. For floating point simulation, it is known that in the normalized min-sum or offset min-sum algorithms the performance of a min-sum based decoder is not sensitive to scaling in the log-likelihood ratio (LLR) values. However, when considering the finite precision for hardware implementation, the scaling affects the dynamic range of the LLR values. The proposed adaptive quantization approach provides the optimal performance in selecting suitable input LLR values to the decoder as far as the tradeoffs between error performance and hardware complexity are concerned.

Chapter 3: A flexible high-throughput LDPC decoder architecture that can support different code rates and block sizes in wireless applications such as IEEE 802.11n, IEEE 802.16e, and IEEE 802.15.3c standards is introduced. The proposed architecture is based on a block-parallel scheduling scheme using a layered decoding method. To achieve higher throughput, check node-based processes are implemented in a fully parallel architecture and the memory is partitioned into a number of banks. System flexibility is achieved by allowing the check node-based units and the memory banks to be configured according to the code rate and block size of the LDPC code of interest.

Chapter 4: A reduced-complexity LDPC layered decoding architecture is proposed using an offset permutation scheme in the switch networks. This method requires only one shuffle network, rather than the two shuffle networks which are used in conventional designs. In addition, we use a block parallel decoding scheme by suitably mapping between required memory banks and processing units in order to increase the decoding throughput. The proposed architecture is realized for a 672-bit, rate-1/2 irregular LDPC code on a Xilinx Virtex-4 FPGA device. The design achieves an information throughput of 822 Mb/s at a clock speed of 335 MHz with a maximum of 8 iterations.

Chapter 5: An improved quantization procedure for fast Fourier transform (FFT)-based decoding of nonbinary LDPC codes is introduced. In particular, quantization effects in the exponential and logarithm functions are considered. The dynamic range of the quantized data is investigated in order to reduce the word length used in the system and the resulting look-up table sizes needed for those functions. The proposed offset-based approach utilizes the relative magnitudes of the quantized data to reduce the dynamic range under a given quantization. The resulting decrease in look-up table size is achieved without sacrificing the decoding performance.

Chapter 6: The finite precision effects of nonbinary LDPC decoding algorithms in the probability or mixed domain has been less extensively studied. For a practical implementation, we show how to achieve the improved decoding performance by using an offset-based method and proper scaling techniques in an FFT-based BP decoder. In addition, we propose novel FFT-based BP decoder architectures to balance the computation load between the main processing units. The results show a 53 % reduction in the number of required FPGA slices compared to a standard FFT-based BP architecture.

Chapter 2

Adaptive Quantization in Min-Sum based Irregular LDPC Decoder

2.1 Introduction

There are a variety of decoding algorithms, such as the iterative belief propagation (BP) algorithm, the Log-BP algorithm, the min-sum algorithm, and the normalized or offset min-sum algorithm [15][16][17][18]. The BP algorithm has a good decoding performance but requires a large hardware complexity. The min-sum algorithm can significantly reduce the hardware complexity at the cost of performance degradation, where complex computations at the check nodes are approximated by using comparators and multiplexers, thereby reducing the area and the power consumption of the decoder. Recently, the normalized or offset min-sum algorithm with scaling factors has been preferred for many practical applications since it offers comparable decoding performance compared to that of Log-BP for regular LDPC codes [19].

In [20], [21] and [22], novel versions of the min-sum algorithm and adaptive quantization effects of the Log-BP algorithm are respectively proposed. The two papers [20], [21] apply normalization factors depending on the bit node degree in the extrinsic message or down-scaling factors to the intrinsic message, respectively. The min-sum algorithm with a few additional computations in [20] reduces the magnitude of the extrinsic information in order to avoid early saturation states at the bit nodes. In [21], the variable nodes use the down-scaled intrinsic

information iteratively to compensate the quantization errors at the bit nodes caused by finite precision. In other words, using down-scaling factors decreases the prior LLR iteratively as the number of decoding iterations increases since the absolute magnitude of the prior LLR usually grows larger in the high SNR region. In this chapter, we propose adaptive quantization schemes in the normalized min-sum decoding algorithm with scaling effects to improve the performance of irregular low-density parity-check (LDPC) decoders.

The rest of the chapter is organized as follows. In Section 2.2, the characteristics of IEEE 802.16e LDPC codes are introduced. We provide the background of the normalized min-sum decoding algorithm and the conventional Log-BP algorithm. In Section 2.3, we show the finite precision effects through the normalized min-sum decoding algorithm with a variable number of quantization bits. We then investigate the quantization effects in the min-sum based decoder without estimated channel SNR for the IEEE 802.16e application. In Section 2.4, we propose an adaptive quantization scheme for the min-sum decoding algorithm to improve the decoder performance. Finally, our conclusions are presented in Section 2.5.

2.2 Background of LDPC Codes and Normalized Min-Sum Decoding

2.2.1 Block Irregular LDPC Codes for WirelessMAN

The IEEE 802.16e, also referred to WirelessMAN [23], is a standard for mobile access where orthogonal frequency division multiplexing (OFDM) is adopted. The LDPC codes standardized in IEEE 802.16e consist of the same style of blocks with different cyclic shifts. The block irregular LDPC codes in IEEE 802.16e have competitive performance and provide flexibility and low encoding/decoding complexity.

Each base matrix in the block LDPC codes has 24 block columns and (1- code rate) × 24 block rows. The expansion factor Z is equal to N/24 for code length N, and Z ranges from 24 to 96 in increments of 4. For example, the code with length N = 1920 has the expansion factor Z = 80. There are four code rates (1/2, 2/3, 3/4, and 5/6) and six different code classes spanning four different code rates.

2.2.2 System Model

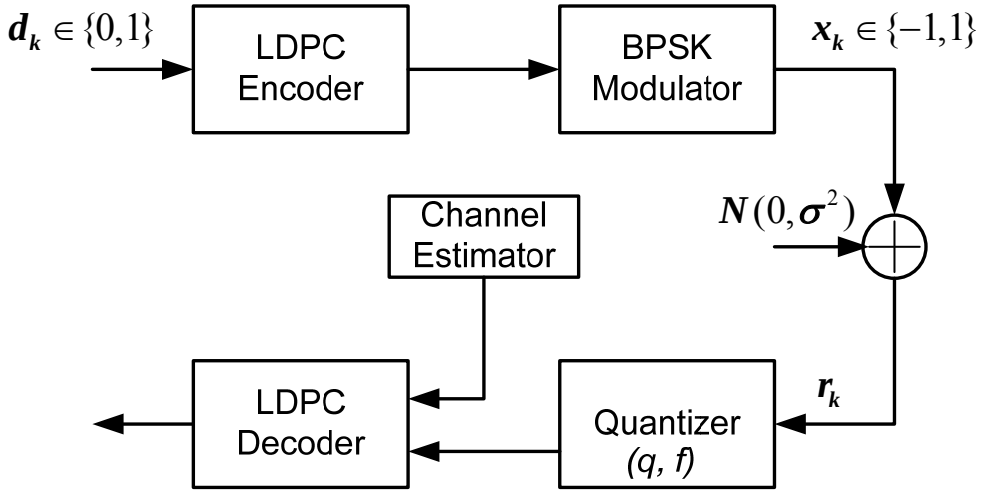


Figure 2.1: System model.

A block diagram of the communication system considered in this paper is given in Fig. 2.1. The LDPC encoder converts an information bit sequence d_k to an encoded bit sequence, where d_k is the k^{th} bit of the block frame. For simplicity, consider a binary modulator which maps the coded symbols {0, 1} into the channel symbols x_k = {-1, 1}. Then, additive white Gaussian noise (AWGN) is added to the transmitted signal by the channel. The received signal r_k is digitized by a given quantizer and the estimated SNR information is fed into the input of the LDPC decoder with received signal r_k.

2.2.3 Normalized Min-Sum Decoding Algorithm

In the normalized min-sum algorithm, which can be considered as an approximation of the BP algorithm, there are two kinds of computation units, check node units (CNUs) and variable node units (VNUs). Messages are denoted by R_{cv} for extrinsic messages from the check node c to the variable node v and by L_{vc} for extrinsic messages from the variable node v to the check node c. The update operation at the check nodes in the normalized min-sum algorithm can be expressed as follows:

$$R_{cv} = \alpha \cdot S_{cv} \cdot \min_{n \in N(c), n \neq v} |L_{nc}| \tag{2.1}$$

$$S_{cv} = \prod_{n \in N(c), n \neq v} sign(L_{nc}) \tag{2.2}$$

where α is a correction factor, S_{cv} stands for the sign part of R_{cv}, and $N(c)$ denotes the set of variable nodes connected to the check node c. The update extrinsic message (R_{cv}) from a check node to a variable node is equal to the minimum reliability of the incoming L_{vc} extrinsic messages from other nodes. In the case of an implementation using the normalized min-sum algorithm, the memory storage element corresponding to CNU stores the smallest value, second smallest value, and the index of the edge providing the incoming message of least value. Compared to the BP algorithm, the CNU in the normalized min-sum algorithm has the advantage of reducing the size of a Look Up Table (LUT), which is required to implement Eq. (2.3) in Log-BP algorithm.

$$|R_{cv}| = 2 \tanh^{-1}\left(\prod_{n \in N(c), n \neq v} \tanh\left(\frac{|L_{nc}|}{2} \right) \right) \tag{2.3}$$

The update operation at the variable nodes is the same as in BP and can be expressed as follows:

$$L_{vc} = \sum_{m \in M(v), m \neq c} R_{mv} - y_v \tag{2.4}$$

$$L_v = \sum_{c \in M(v)} R_{cv} - y_v \tag{2.5}$$

where $M(v)$ denotes the set of check nodes connected to the variable node v and y_v is the prior LLR given by $2r_v/\sigma^2$, where r_v is the AWGN channel output and σ^2 is the noise variance. For the AWGN channel, it is known that the min-sum based algorithms such as the normalized min-sum or offset min-sum are insensitive to scaling the VNU computations in (2.4), (2.5) because the scaling factor σ^2 does not affect the output of the CNU in (2.1). Therefore, the prior LLR, y_v, values can be computed as the received value r_v. The decoding algorithm stops if either the estimated codewords satisfy all the parity check equations or the maximum number of iterations is reached.

2.3 Finite Precision Effects on Normalized Min-Sum Decoder for Irregular LDPC Codes

In the implementation of normalized min-sum LDPC decoding, effects due to finite precision should be considered because they degrade the error performance of systems. The quantization effects are related to the fixed-point number format that is used in the processing of intrinsic and extrinsic messages in the decoder. Moreover, the hardware complexity and decoding performance depend on the fixed number format. In this work, we assume that irregular LDPC codes are modulated by BPSK and transmitted over an AWGN channel. For simplicity, we use one correction factor for all check nodes in the normalized min-sum algorithm.

We use the notation (q, f) to represent a quantization scheme in which q bits are used for total bit size and f bits are used for fractional values. In a uniform quantization scheme, a signed fixed-point number format has a quantization precision of 2^{-f} with a maximum value of $2^{q-f-1} - 2^{-f}$ and a minimum value of -2^{q-f-1}. To analyze the quantization effects of the normalized min-sum algorithm, we consider the received values to be clipped symmetrically at a given maximum and minimum value in the uniform (q, f) quantization scheme. For BPSK and an AWGN channel, the received values are distributed with a Gaussian distribution around the transmitted signal {-1, 1}. More than

99% of the occurring values are covered by limiting the dynamic-range of the received channel values to [-4, 4]. Values above the maximum or below the minimum are clipped in both the CNU and VNU.

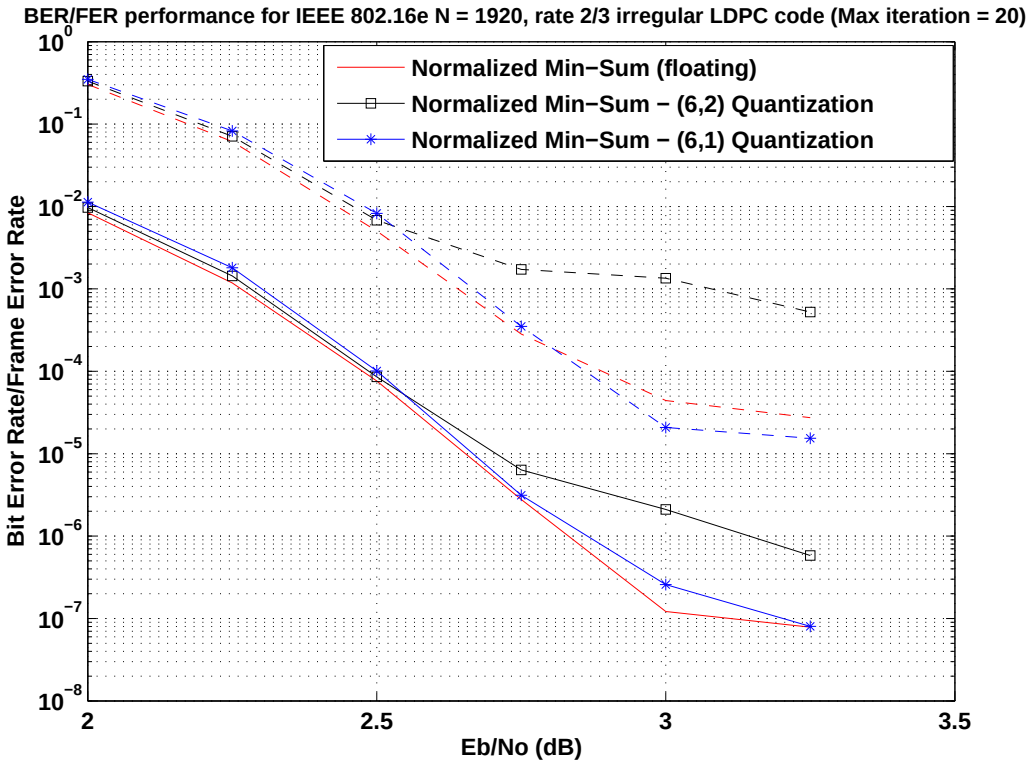


Figure 2.2: Performance of the (1920, 1280) irregular code implemented using quantization schemes where solid lines correspond to BER and dashed lines correspond to FER.

The performances of the (1920, 1280) irregular LDPC code with floating point and several quantization schemes are shown in Fig. 2.2. In this chapter, we limit the word length as 6 to investigate the saturation and quantization effects. It is shown that for q = 6, the (6, 1) quantization has the best performance. We can see that the difference between (6, 1) and (6, 2) quantization is quite significant in high SNR region. In other words, the performance gain using (6, 1) quantization compared with (6, 2) is more than 0.4dB at BER = 7×10^{-7}. It is known that the normalized or offset min-sum decoding does not need any channel information and works with

just the received values as inputs. In order to analyze the effect of channel information on the normalized min-sum decoding, we select the input to the decoder to be $2r_v/\sigma^2$. As the SNR increases, (6, 2) quantization scheme is not sufficient to cover the distribution of $2r_v/\sigma^2$ because the dynamic range of $2r_v/\sigma^2$ is larger than that of (6, 2) quantization scheme.

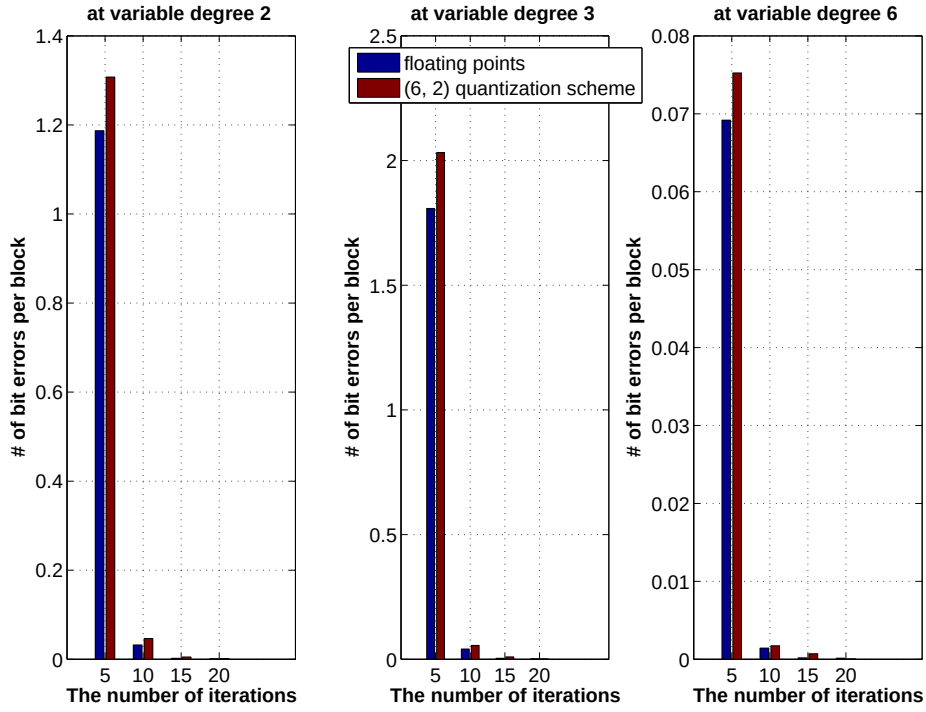


Figure 2.3: Number of bit errors per block at the variable nodes {2, 3, 6} for (1920, 1280) irregular LDPC code.

In [20] and [21], two modified versions of the normalized min-sum algorithm are presented and a behavioral analysis on extrinsic message states is studied as the number of iterations increases. The degree distribution polynomials of our code are $\lambda(x) = 0.2917x^2 + 0.5x^3 + 0.2083\,x^6$ with respect to the variable nodes and $\rho(x) = x^{10}$ with respect to the check nodes. Fig. 2.3 shows the number of bit errors at the variable nodes of degree {2, 3, 6} after some number of iterations at SNR = 2.75 dB. From Fig. 2.3, the convergence speed of correcting bit errors at variable nodes of

degree 6 is faster than other variable nodes of degree 2 and 3, although variable nodes of degree 3 contain more bit errors than that of the others. The percentage reduction in bit errors on variable nodes of degree {2, 3, 6} is shown in Fig. 2.4. After 10 iterations, the percentage reduction in bit errors in both floating and fixed point implementations decreases. In other words, the number of

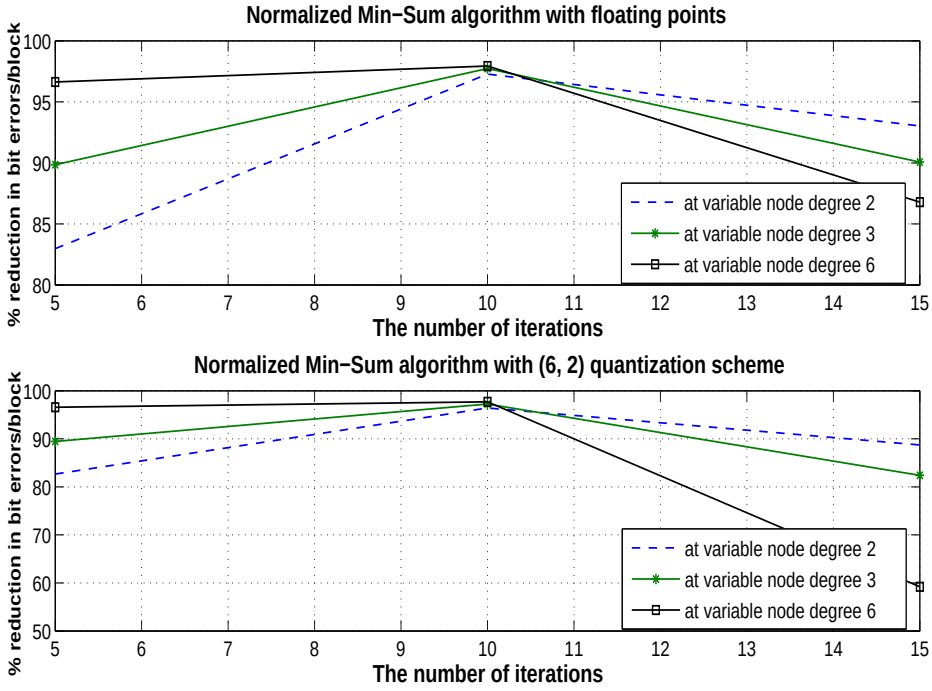


Figure 2.4: Percentage reduction in bit errors after 5, 10, 15 iterations.

bit errors remains unchanged after a certain number of iterations and the extrinsic messages have no effect on decoding performance. From the above observation, distinct down-scaling factors [21], which are determined by the degree of the variable nodes, is used on the intrinsic messages ($2r_v/\sigma^2$) in order to reduce the strength effects of the intrinsic magnitude on the extrinsic information L_{vc}. Instead of using the down-scaling factors on the intrinsic messages, our proposed quantization scheme uses the received value (r_v) as the inputs to intrinsic messages in a high SNR region. This quantization method helps to improve the performance of irregular LDPC decoders without additional hardware complexity, which will be discussed in Section 2.4.

2.4 On Implementation of Adaptive Quantization in Normalized Min-Sum Algorithm

In this section, we analyze quantization effects on the performance of an irregular LDPC decoder. The study presented in the last section led us to consider the dynamic range of the prior LLR in order to take more precisely into account the effect of finite precision on the intrinsic data. Quantization of incoming prior LLR data significantly affects the decoding performance and it should be analyzed in order to design an efficient LDPC decoder in terms of hardware complexity and decoding performance. In the case of floating point simulations, the error performance of a normalized or offset min-sum algorithm does not vary with SNR estimation. However, when considering the finite precision of a hardware implementation, scaling affects the dynamic range of LLR values. At high SNR, quantization effects are reduced by used r_v rather than $2r_v/\sigma^2$. In that case, a (6, 2) quantization scheme is sufficient.

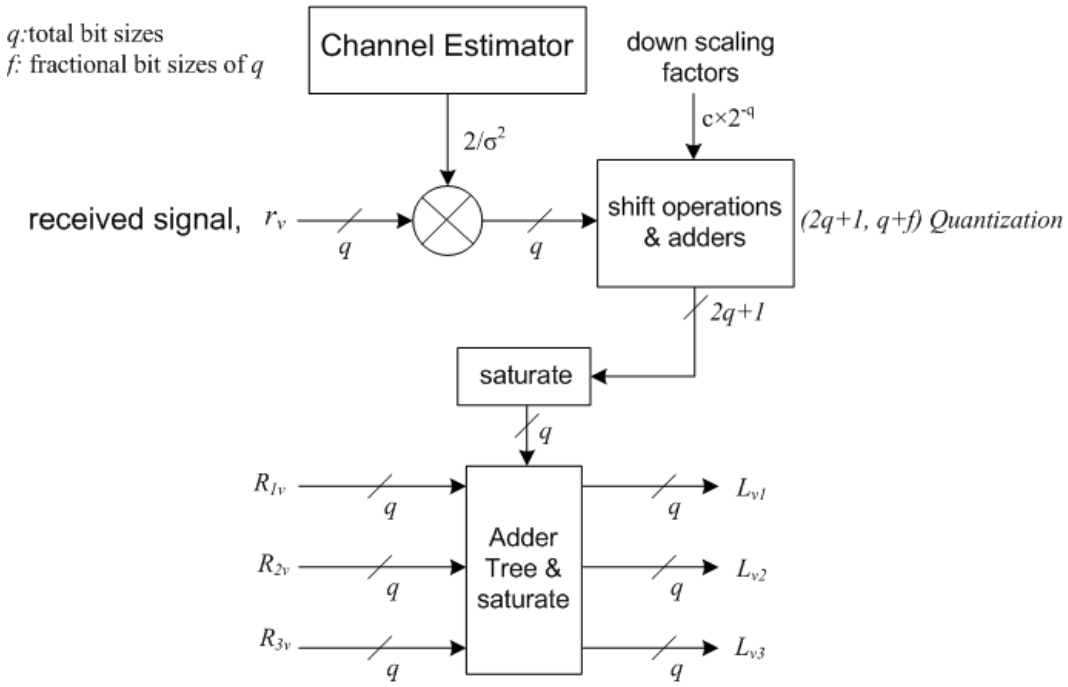


Figure 2.5: Degree-3 VNU architecture for (6, 2) quantization simulation.

Fig. 2.5 shows a VNU architecture for simulating various quantization schemes on the normalized min-sum decoding algorithm. The architecture needs additional hardware ((2q + 1)–bits adders and shift operators) so that it holds the precision of intrinsic message computations in order to use down-scaling factors. In our work, we use the same VNUs excluding down-scaling factors with shift operations and adder blocks. In Fig. 2.6, we present the performances of the

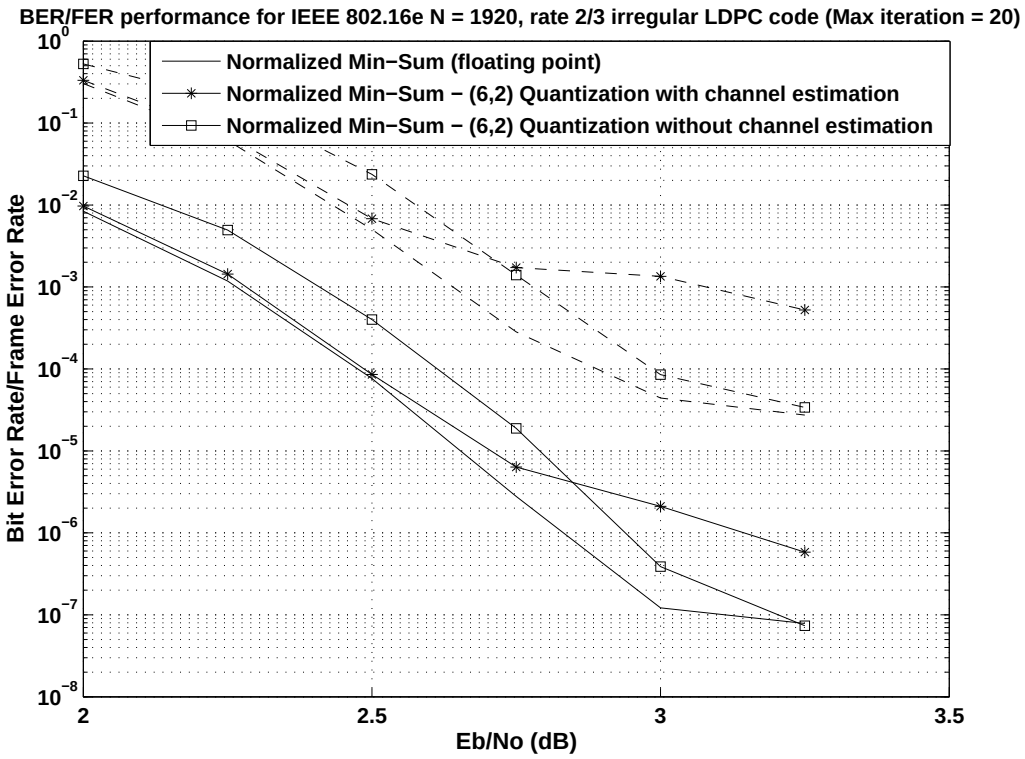


Figure 2.6: Effect of scaling intrinsic messages by excluding SNR estimation on the (6, 2) quantization scheme.

normalized min-sum algorithm with the (6, 2) quantization scheme without SNR estimation and down scaling factors. Based on our simulation results, the SNR estimation does not help the normalized min-sum decoding performance at high SNR levels while the min-sum based algorithms need to know the channel state information at low SNR region. Considering dynamic

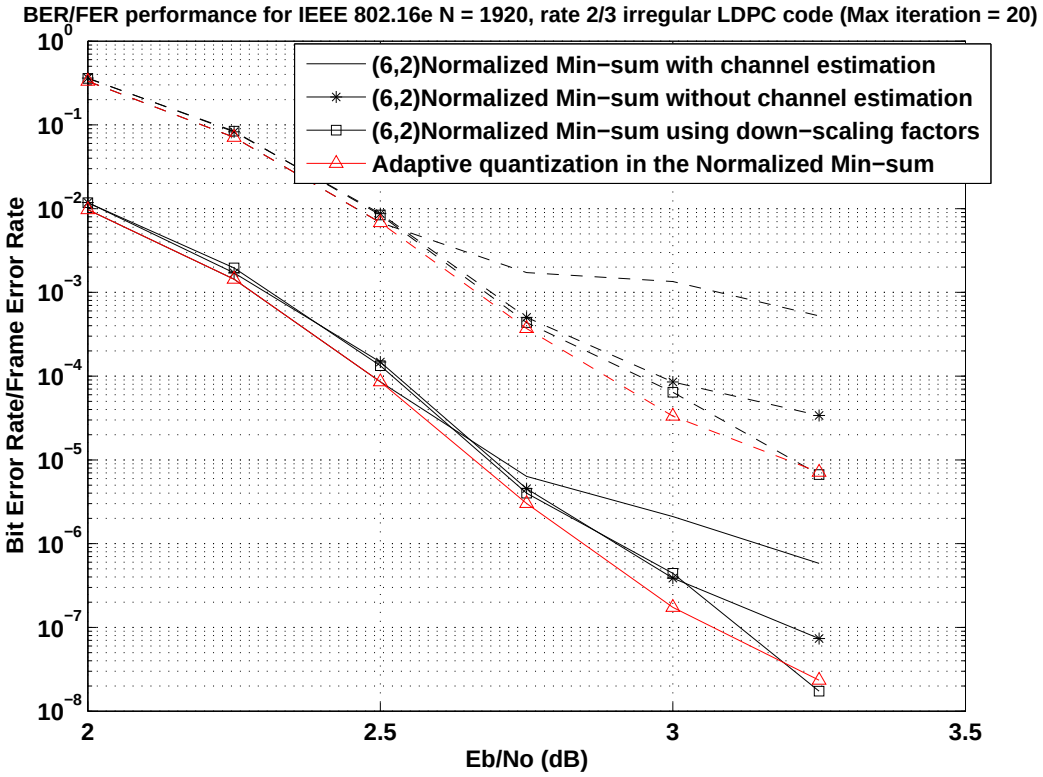


Figure 2.7: Performance comparison between ref [21] and the adaptive quantization scheme where solid lines correspond to BER and dashed lines correspond to FER.

range of LLR received inputs, an adaptive quantization scheme can be used in a normalized min-sum based decoder.

With the adaptive quantization in the normalized min-sum algorithm, y_v in Eq. (2.4) can be expressed as

$$y_v = \begin{cases} \dfrac{2r_v}{\sigma^2}, & SNR \le C \ \ dB \\ r_v, & SNR > C \ \ dB \end{cases} \tag{2.6}$$

where (6, 2) and (6, 3) quantization schemes are used at SNR $\le C$ and SNR $> C$, respectively, and where C is a given value, depending on the specific LDPC code and code rate. In our case, C (2.75 *dB*) is obtained from extensive simulation. To implement Eq. (2.6), a simple multiplexer is

required and the output component of r_v is used for generating (6, 3) quantization at a high SNR level. A (6, 2) quantizer including a signed multiplication needs to achieve a conversion from (6, 3) to (6, 2) quantization. The performance of the adaptive quantization against several fixed point implementations of LDPC decoder is shown in Fig. 2.7. The simulation results show that the adaptive quantization using optimal input LLR values in an LDPC decoder provides much better BER and FER performance than the conventional (6, 2) quantization scheme. Moreover, it performs slightly better than a (6, 2) quantization scheme with down-scaling factors.

2.5 Conclusion

In this chapter, we have investigated the quantization effects on decoding performance of irregular LDPC codes for WMAN applications. We have performed simulations on a (1920, 1280) irregular LDPC code to achieve the optimal finite word-lengths of variables for the normalized min-sum algorithm. In the simulations, up to 2×10^6 block codewords are simulated for each high SNR data point. We have proposed an adaptive quantization for the normalized min-sum algorithm. Computer simulation results show that the proposed quantization scheme, which depends on the dynamic range of LLR input values and uses suitable LLR input values to the decoder, achieves much better performance than the conventional (6, 2) quantization scheme.

Chapter 3

Flexible LDPC Decoder Architecture for High-Throughput Applications

3.1 Introduction

Low Density Parity Check (LDPC) codes have attracted much attention because of their excellent error correcting performance and inherently parallelizable VLSI implementation. Therefore, they are being widely used in communication standards such as DVB-S2, IEEE 802.16e and IEEE 802.11n. In addition, mmWave (millimeter wave) Wireless Personal Area Networks (WPANs) described by the IEEE 802.15.3c Working Group [23] are considering LDPC codes as the preferred choice for forward error correction (FEC). All of the LDPC codes in the above standards are based on "Architecture-Aware LDPC codes" [25] or "Block-LDPC codes" [26]. The parity-check matrix H of these codes is partitioned into block-columns and block-rows, which are particularly suitable for VLSI implementations by simplifying the memory access and utilizing a well developed switching network.

LDPC codes are decoded using an iterative message-passing algorithm, consisting of a row operation and a column operation (called the two-phase message passing algorithm), over a graph-based representation of the codes. A method of determining the update order between the row operations and the column operations is called a scheduling. Various scheduling schemes have been proposed, such as a flooding schedule and a serial or layered schedule [27]. The flooding

schedule updates all row operations after updating all column operations and vice versa, while the layered schedule updates the row (column) operations by sending an immediately updated column (row) message. The layered decoding schedule can reduce the number of iterations by almost 50% without performance degradation compared to the flooding decoding schedule. In other words, it achieves approximately twice as fast decoding convergence due to the use of intermediate check-node (or variable-node) message values. A low complexity LDPC decoder architecture using the layered decoding schedule was developed in [28].

A semi-parallel or block-serial architecture of a layered LDPC decoder has been presented in the literature [29]-[31] to increase the convergence speed and to reduce latency. However, it has low decoder throughput due to its block-serial scheduling architecture. In this chapter, we propose a check node-based processor (CNBP) architecture suitable for improving decoding throughput while achieving system flexibility, which is necessary for next-generation mobile communication systems. A novel architecture based on block-parallel operations (simultaneously processed group by group) using a layered decoding schedule is developed that uses parallel memory accesses. The rest of the chapter is organized as follows. In Section 3.2, we provide the background for block-LDPC codes and the layered decoding schedule. In Section 3.3, we propose a block-parallel LDPC decoder based on a novel architecture for the check node-based processor. In addition, system flexibility will be described which allows reconfiguration of the LDPC decoder. Finally, our conclusions are presented in Section 3.4.

3.2 Background of Layered Decoding Schedule

After giving a brief introduction to Block-LDPC codes, the layered decoding schedule is addressed in this section. The layered decoding schedule allows the use of efficient block-serial decoder architectures. Although the block-serial decoder architecture is efficient for achieving system

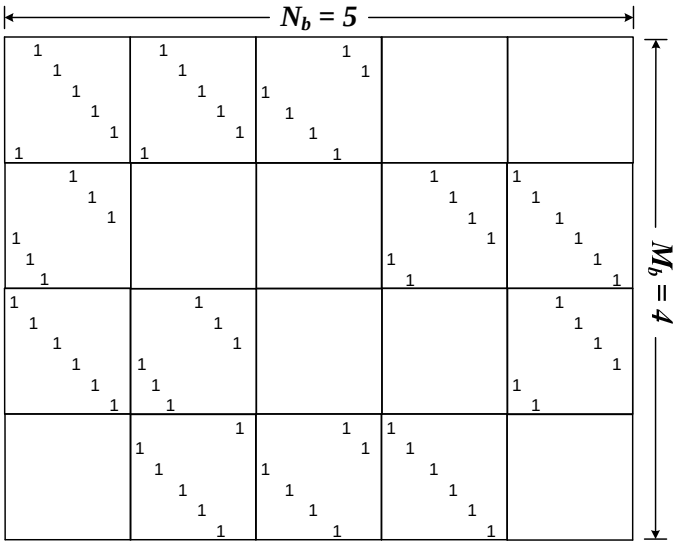


Fig. 3.1: An example of the *4* × *5* base matrix $\mathbf{H_b}$ where the size of each sub-matrix *z* is 6 and empty squares correspond to all-zero matrices.

flexibility, its throughput is limited due to the serial architecture of the message processing units. For example, a multi-edge type vector LDPC decoder, as proposed by Richardson [32], can be implemented at low hardware complexity but it has a relatively low decoder throughput.

3.2.1 Block-LDPC Codes

Block-LDPC codes described in several IEEE standards have constraints or structures which can be exploited in implementing both the encoder and decoder. For example, the IEEE 802.15.3c LDPC codes shown in [23] consist of blocks with different cyclic shifts, and can support very low complexity systematic encoders and low complexity, highly parallelizable decoders. The $M_b \times N_b$ base matrix $\mathbf{H_b}$ with $M_b = M/z$ and $N_b = N/z$, where M is the number of parity check equations, N is the code length, and z is the sub-matrix size, in the IEEE 802.15.3c LDPC codes have 32 columns of blocks and (1− code rate) × 32 rows of blocks. Table 3.1 summarizes 3 code prototypes with various row and column parameters, as defined by the standard. For example, at rate 1/2 for N =

Table 3.1: IEEE 802.15.3c LDPC code prototypes

Code	1/2	3/4	7/8
Rows	16	8	4
Columns	32	32	32
Row degree	{5, 6, 7, 8}	{13,14,15,16}	{29,30,31,32}
Column degree	{4, 3, 2, 1}	{4, 3, 2, 1}	{4, 3, 2, 1}

672, the parity check matrix has $M_b = 16$ layers, the size of the permutation sub-matrix is $z = 21$, and the column weight of each layer is at most 1. An example of the 4 × 5 base matrix $\mathbf{H_b}$ is shown in Fig. 3.1.

3.2.2 Layered Decoding Schedule

A brief overview of the decoding algorithm is provided to describe the layered decoding scheme and the architectural issues.

In the iterative message passing algorithm, messages are denoted by R_{cv} (row operation) for extrinsic messages from the check node c to the variable node v and by L_{vc} (column operation) for extrinsic messages from the variable node v to the check node c. The update operations at the check nodes and variable nodes can be expressed as in equations (3.1) and (3.2), respectively.

$$R_{cv} = -\prod_{n \in N(c), n \neq v} sign(L_{nc}) \cdot \Psi\{\sum_{n \in N(c), n \neq v} \Psi(L_{nc})\} \tag{3.1}$$

$$L_{vc} = \sum_{m \in M(v), m \neq c} R_{mv} - y_v \tag{3.2}$$

$N(c)$ and $M(v)$ denote the set of positions of the columns of **H** and the set of position of the rows of **H** such that, $N(c) = \{v|\mathbf{H}_{c,v}=1\}$ and $M(v) = \{c|\mathbf{H}_{c,v}=1\}$, respectively. In equation (3.2), the Psi-function, $\Psi(x) = \log(\tanh(|x/2|))$, is a nonlinear function and y_v is the prior log-likelihood ratio (LLR) given by $2r_v/\sigma^2$, where r_v is the AWGN channel output and σ^2 is the noise variance. At every iteration (one iteration consists of equations (3.1) and (3.2)), the soft decoding result for each bit is determined as follows:

$$L_v = \sum_{c \in M(v)} R_{cv} - y_v \tag{3.3}$$

In contrast to the iterative message passing algorithm using a flooding schedule, the layered decoding schedule processes the M_b^{th} row (or N_b^{th} column) of **H** in layers or groups, In our work, we use a horizontal layer decoding scheme for application to a check node-based processor. For each variable node v inside the current M_b^{th} row, R_{cv} in equation (3.1) is computed and is immediately used for the next layer. Instead of using L_{vc} messages, variable node messages for each column block are used to update the R_{cv} messages on the fly, thus avoiding the need to maintain additional memory for the L_{vc} messages. A more detailed description of the layered decoding schedule is given in [27] and [28]. We propose a block-parallel LDPC decoder by reformulating the check node-based computation of the horizontal layered decoding schedule for improved throughput, as will be discussed in the next section.

3.3 Flexible LDPC Decoder Architecture

The proposed decoder architecture is based on a multi-edge type vector LDPC decoder [32], but it has been reformulated to increase the throughput and to achieve system flexibility. In [32], a vector of z processors (z check/variable node processors) operates on a macro column/row sequentially with one sub-matrix. For instance, the block-serial decoder needs at least the

V1	V2	V3	V4	V5
m1	m4	m7	0	0
m2	0	0	m9	m11
m3	m5	0	0	m12
0	m6	m8	m10	0

Fig. 3.2: Memory data of C2V_MEM and VN_MEM, highlighted in blue letters (m1, m4, m7) and red letters (V1, V2, V3, V4, V5), respectively.

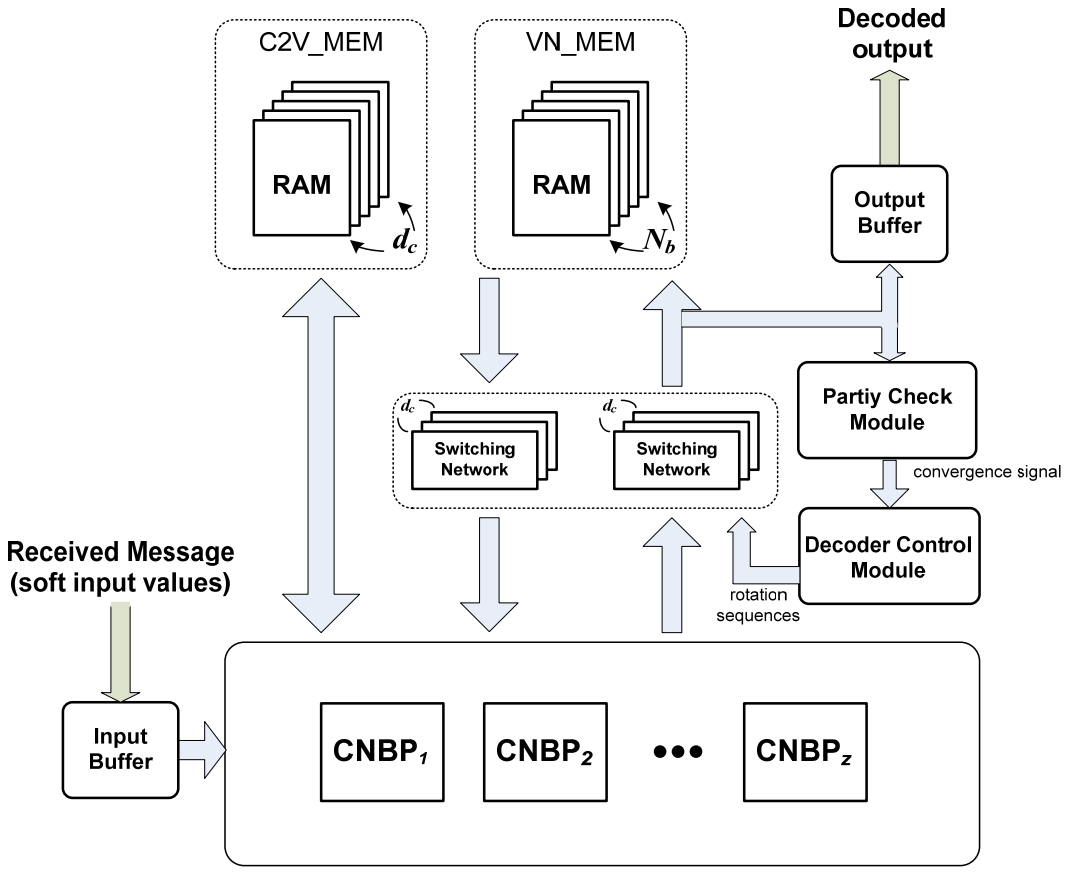


Fig. 3.3: Overall block-parallel LDPC decoder architecture.

maximum check node degree (d_c = 3) number of clock cycles to process three messages (m1, m4, m7), as shown in Fig. 3.2. In the proposed architecture, all messages (m1, m4, m7) can be simultaneously processed in a single clock cycle, which will considerably improve the throughput of the decoder. As depicted in Fig. 3.3, the proposed block-parallel LDPC decoder mainly consists of two memory blocks for storing messages, check node-based processors (CNBPs) for processing intermediate messages, switching networks (SNs) for routing messages, a parity check module and a decoder control module. Fig. 3.4 shows an example processing for the first row (m1, m4, m7) of a 4 × 5 $\mathbf{H_b}$ matrix, showing the relationship between messages and memory blocks through the SNs. The architecture of a CNBP suggests the use of parallel structures for achieving faster decoding convergence of the layered decoding schedule. Let m(i) (i = 1, 2,…, z) represent the i-th element of each message vector. For each element i of each message vector per row block, the number of inputs in the CNBP depends on the value of d_c. A variable node memory (VN_MEM) block includes $N_b \times z \times K$ bit values with K-bit precision corresponding to one edge. The VN_MEM bank N_b's are used to read/write variable-to-check messages while the CNBP performs the block row (m1(i), m4(i), m7(i)) processes shown in Fig. 3.3. In other words, the CNBP simultaneously processes several block edges adjacent to the M_b^{th} block check node. A check-to-variable memory (C2V_MEM) block stores $L \times z \times K$ bit values, where L indicates the number of non-zero integers in the base matrix $\mathbf{H_b}$. The C2V_MEM block is partitioned into d_c banks. A C2V_MEM bank address selects sets of the d_c banks to be read or written.

A switch network (SN) that implements rotations of the input message vector is available in the Benes [33] network. In our work, 2 × d_c SNs are required for switching message outputs from the CNBP to the VN_MEM, and for switching messages output from the VN_MEM to the CNBP. In addition, a specific memory that is responsible for storing pre-computed routing patterns should be able to provide for different code rates and block sizes.

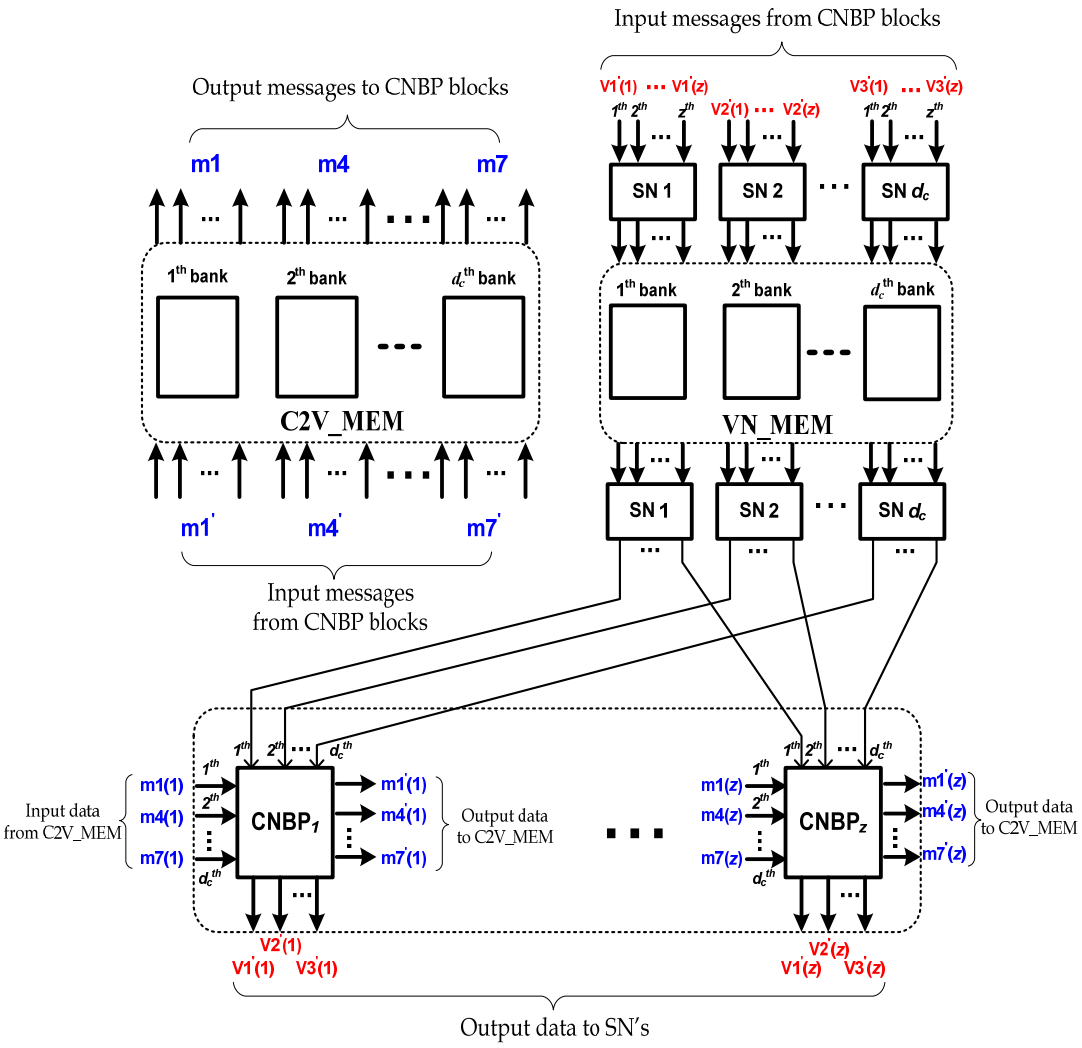


Fig. 3.4: Dataflow graph of the proposed block-parallel LDPC decoder architecture.

In order to clarify the higher throughput provided by our proposed block-parallel LDPC decoder, the throughput of the decoder can be estimated as:

$$Throughput \approx \frac{R \times N \times fclk_{\max}}{iterations \times M_b}, \tag{3.4}$$

where R is the code rate, $fclk_{\max}$ is the maximal clock frequency, and M_b is the number of block rows corresponding to R. This approximate throughput is not related to the total number of message edges, L, whereas the throughput in a block-serial decoder depends on L.

For implementing the CNBP in a fully parallel architecture, the layered decoding schedule [27] could be equivalently reformulated as follows.

Initialization: $R_{cv} = 0, \ \forall c$ and $\forall v \in N(c)$ (3.5)

$$Q_v = y_v + \sum_{m \in M(v)} R_{mv}, \ \forall v \tag{3.6}$$

Iteration: $\forall c$ in the current layer l (l=1, 2,…,M_b)

$$R'_{cv} = \Psi \left\{ \sum_{n,m \in N(c), n \neq v} \Psi \left(Q_n - R_{cm} \right) \right\} \tag{3.7}$$

$$Q'_v = R'_{cv} + \left(Q_v - R_{cv} \right) \tag{3.8}$$

Note that the R'_{cv} term in (3.7) and Q'_v term in (3.8) are most recently updated by using values R_{cv} and Q_v in the previous layer. An example of the algorithm with the 4 × 5 base matrix $\mathbf{H_b}$ is shown in Fig. 3.5. This decoding scheme describes messages to be exchanged from two memory blocks, which are C2V_MEM and VN_MEM, leading to the high throughput decoder. By applying the proposed decoding architecture to various structured LDPC codes, we can reduce the number of processing cycles per iteration.

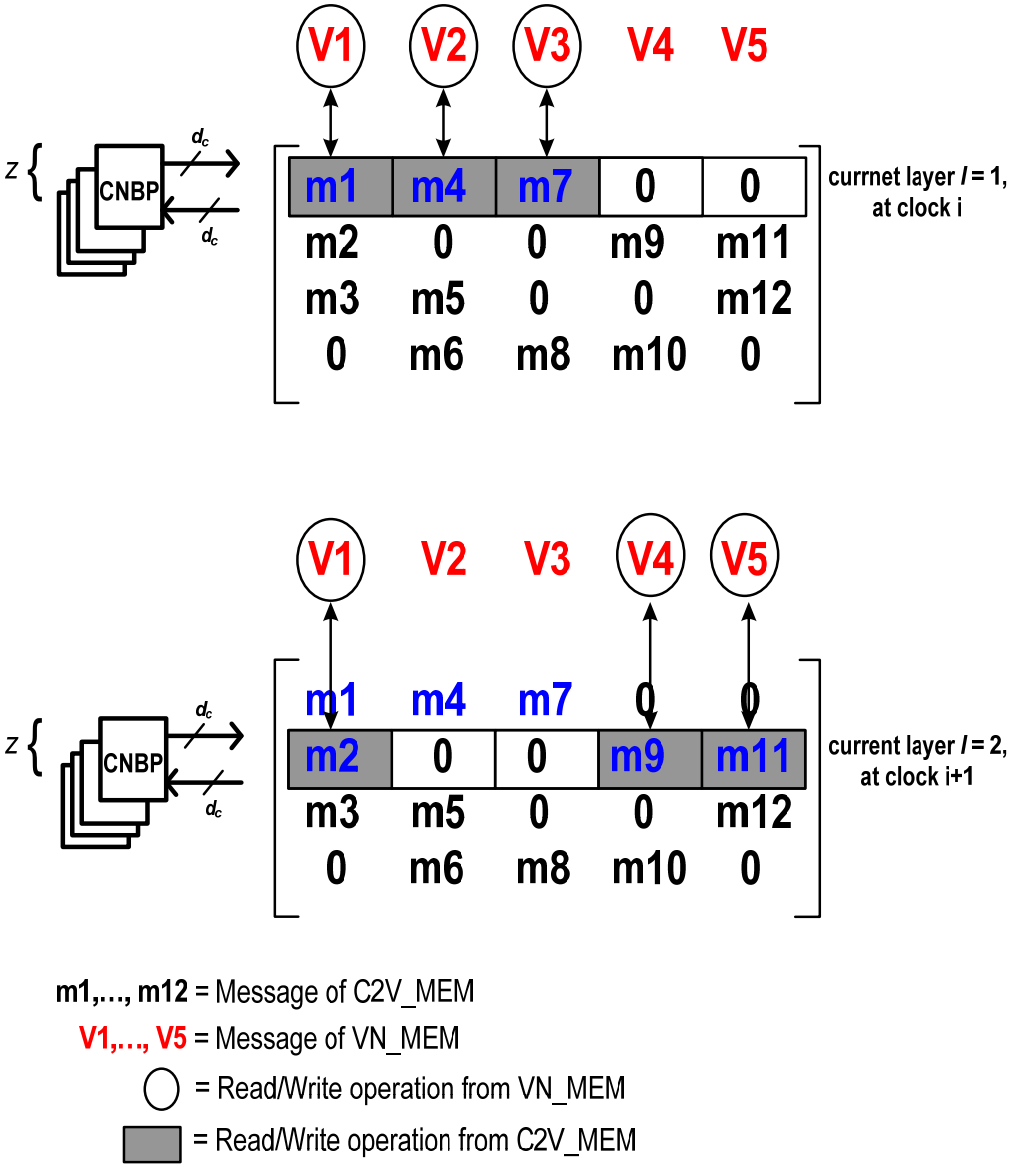


Fig. 3.5: Illustrative example of the block parallel LDPC decoder based on the proposed CNBPs.

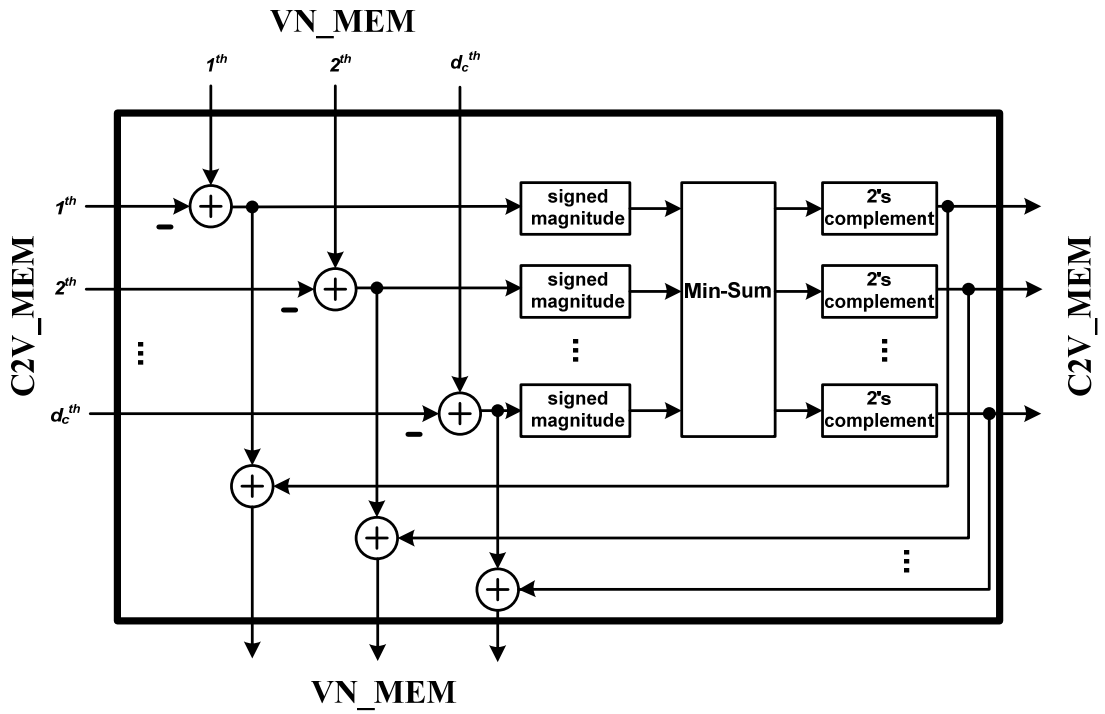


Fig. 3.6: Architecture of the check node-based processor – CNBP.

Fig. 3.6 shows the structure of the check node-based processor using the normalized min-sum algorithm which is an approximation algorithm of the above equations (3.7) ~ (3.8) and which reduces the decoding hardware complexity. The Min-Sum module is responsible for selecting first and second minimum values and the CNBP module can apply the scaling operations, similar to the adaptive quantization method in [34]. This fully parallel architecture simultaneously reads R_{cv} messages from C2V_MEM block and Q_v messages from VN_MEM block through SNs. Moreover, *Read* and *Write* operations are simultaneously performed in the dual-port memory banks. The signed magnitude and 2's complement convert blocks are efficient for the Min-Sum module and the addition/subtraction required for calculating intermediate messages, respectively. System flexibility in terms of the supported block sizes and code rates can be achieved by the control unit without modifying CNBPs or memory blocks. The required components of CNBP, C2V_MEM, and VN_MEM are accessed through a multiplexer or fed as zero value for unused inputs.

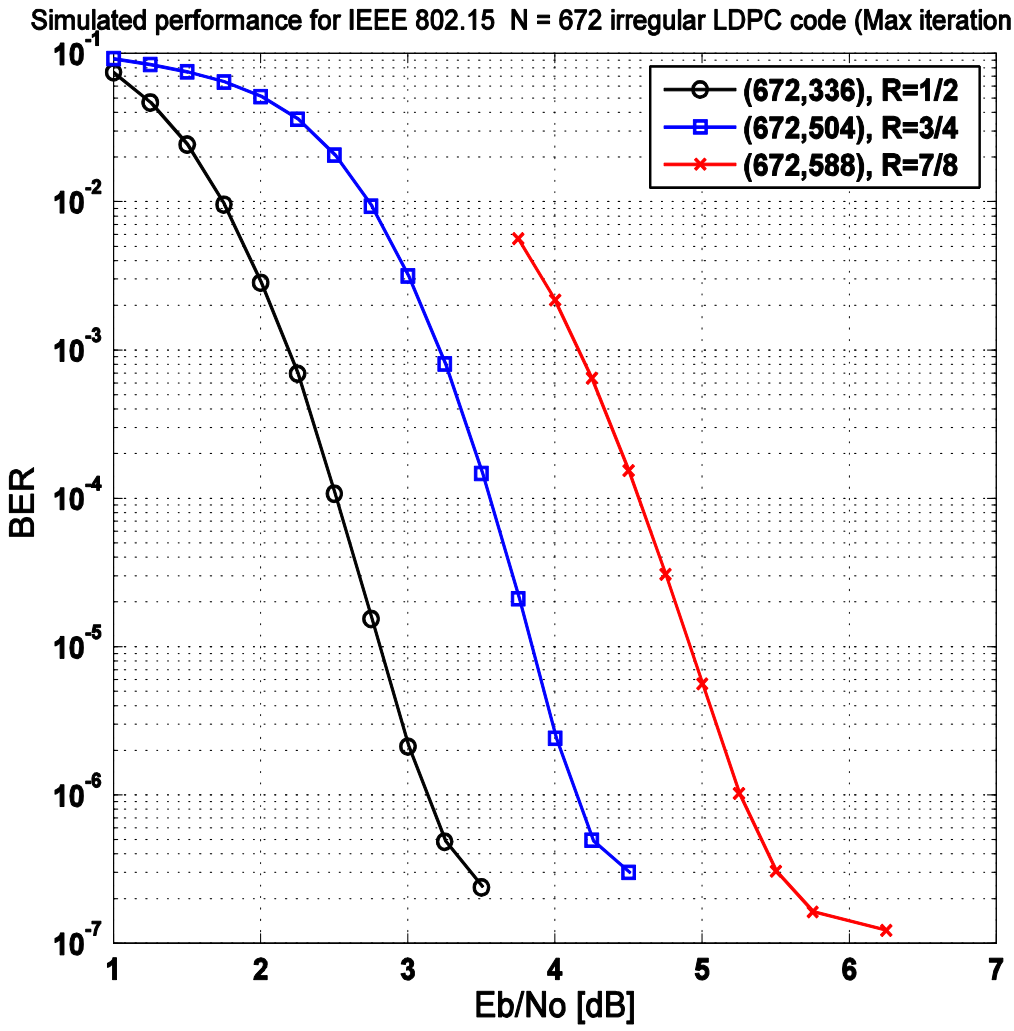


Fig. 3.7: BER performance of the layered, modified min-sum algorithm.

Figs. 3.7 and 3.8, which use the normalization factor α = 0.875, show the layered, modified min-sum decoding performance using a maximum of 10 iterations and floating point arithmetic, simulated from low to high code rates. The bit error rate (BER) and the frame error rate (FER) for rate-1/2, 3/4, and 7/8 are shown in Figs. 3.7 and 3.8, respectively.

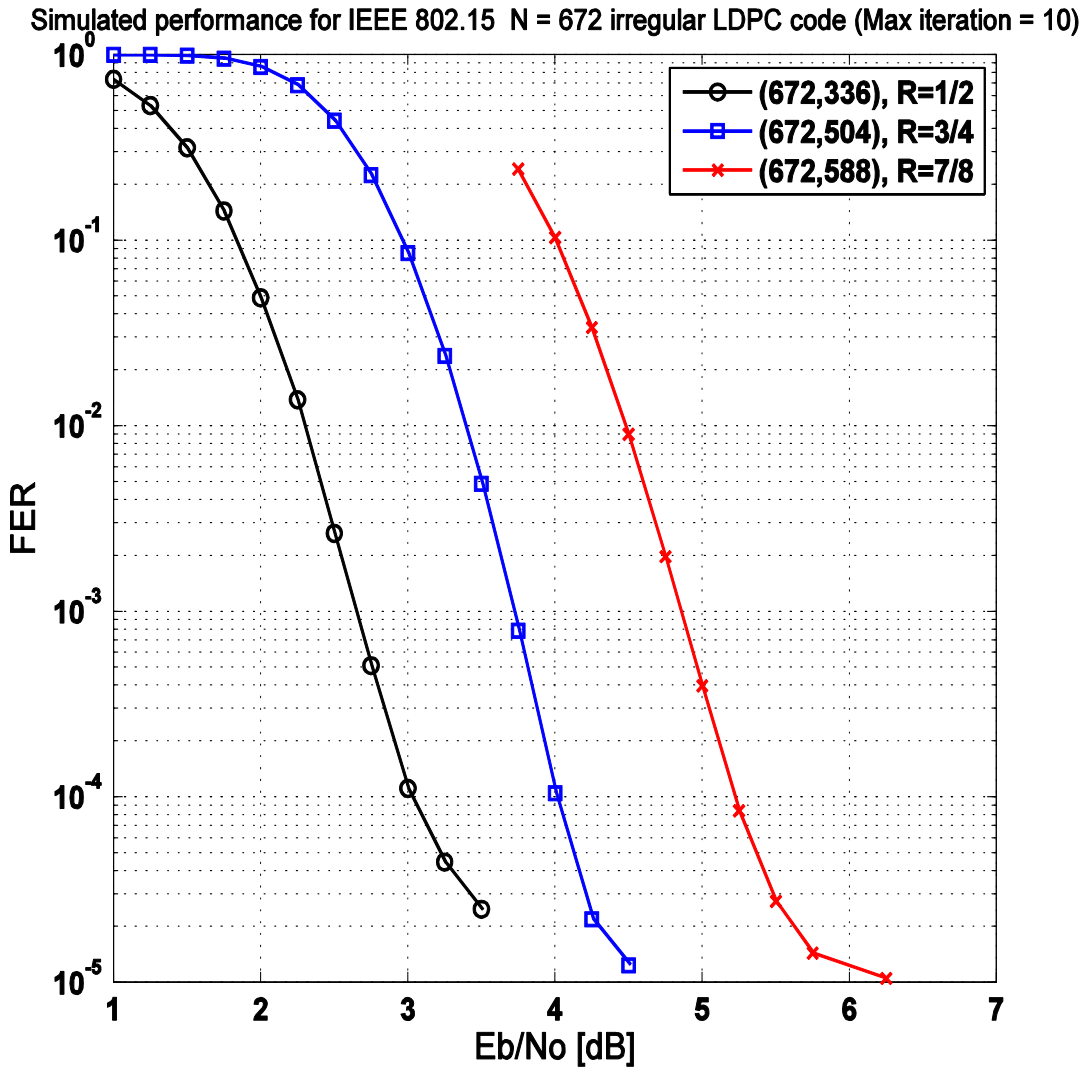


Fig. 3.8: FER performance of the layered, modified min-sum algorithm.

These simulation results are used to trade off between complexity, speed and decoding performance and provide a benchmark for determining the data width to be used in the overall decoder architecture. Fig. 3.9 compares the average number of iterations required by the layered, decoding Min-Sum algorithm. This result demonstrates its characteristic of quick convergence after only a limited number of iterations.

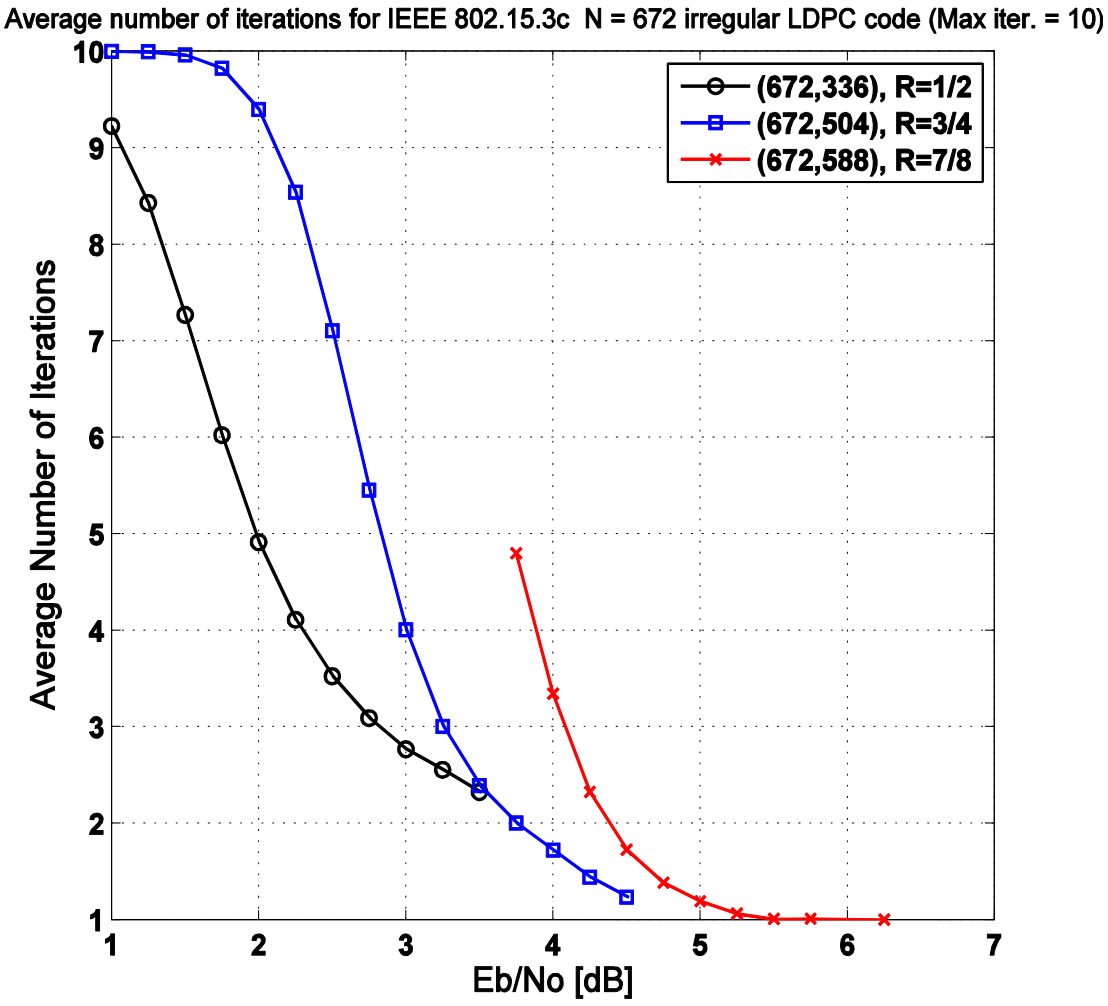


Fig. 3.9: Average number of iterations of the layered, modified min-sum algorithm.

3.4 Conclusion

A flexible, high-throughput LDPC decoder architecture is presented for supporting different code rates and block sizes in wireless applications. The proposed CNBP architecture is suitable for block-parallel implementation and the overall decoder can achieve higher throughput than a block-serial scheduling scheme.

Chapter 4

A Reduced-Complexity Architecture for LDPC Layered Decoding Schemes

4.1 Introduction

The basic decoder design [35] for achieving the highest decoding throughput is to allocate processors corresponding to all check and variable nodes, together with an interconnection network. In this fully-parallel decoder architecture, the hardware complexity due to the routing overhead is very large. Therefore, much of the work on LDPC decoder design has been directed towards achieving optimal trade-offs between hardware complexity and decoding throughput. In particular, a time-multiplexed or folded approach [36], which is known as a partially parallel decoder architecture, has been proposed.

Recently, several partially parallel decoder designs[37]–[43] for "block structured LDPC codes" or "architecture-aware LDPC codes" have been developed using elements such as check node units (CNUs), variable node units (VNUs), and interconnection networks between CNUs and VNUs. These approaches lead to decoders having a reduced number of clock cycles per iteration, which results in higher decoding throughput. In [38], the sum and sign accumulation unit for the CNU is used in computing a portion of each row while the VNU computes each column. The overlapped decoding scheme exploited in [39] for high-rate LDPC codes is similar to the method in [38] except that a CNU computes a portion of a row by accumulating partial results. To achieve

a faster convergence compared to the overlapped, two-phase decoding scheme, turbo-decoding message-passing [40] or a layered decoding [41] schedule and architecture for regular structured codes have been proposed. However, conventional layered decoders use a bi-directional network or two switch networks for shuffling and reshuffling messages, which increases the hardware complexity.

Designs utilizing one data shifter or a cyclic shifter have been introduced in [42] and [43], respectively. However, the proposed data shifter in [42] is targeted for only one parity-check matrix **H** since its interconnection mapping is fixed by the shift values in the first block row of **H**. In [43], the overall decoder is designed specifically for a (3, 6) array code. In contrast, our objective is to create a design that is suitable for multiple code rates and different codeword sizes. Specifically, we propose a reduced-complexity LDPC decoder architecture for use in layered decoding having an offset generating algorithm to decrease the interconnection complexity with no degradation in the decoding throughput.

The remainder of this chapter is organized as follows. In Section 4.2, we briefly describe the layered decoding scheme and present a block parallel layered decoder, which is suitable for high-throughput applications. In Section 4.3, we propose an algorithm for generating offset values for the switch network so as to reduce the interconnection complexity. Hardware complexity comparisons with previous designs are given in Section 4.4. An FPGA implementation of a 672-bit, rate-1/2 irregular LDPC decoder is summarized in Section 4.5. Moreover, we present a functional verification of our LDPC decoder, a state transition diagram of the top control, and the architecture of the optimized switch network in Section 4.6. Our conclusions are presented in Section 4.7.

4.2 Layered Block Parallel Decoder Architecture

4.2.1 Layered Decoding Scheme

Structured regular or irregular LDPC codes are described by an $M_b \times N_b$ base matrix $\mathbf{H_b}$ with $M_b = M/z$ and $N_b = N/z$, where M is the number of parity check equations, N is the code length and z is the size of a square sub-matrix. The parity check matrix $\mathbf{H}$ of a structured LDPC code can be viewed as the concatenation of constituent codes [40], where the number of constituent codes is equal to M_b. The dataflow of a typical layered decoder is shown in Fig. 4.1. Let $\mathbf{R} = \left[\mathbf{r}_1\ \mathbf{r}_2 \cdots \mathbf{r}_{M_b}\right]^T$ denote the check-to-variable messages, where $\mathbf{r}_k$ corresponds to a constituent code of $\mathbf{H}$ for $1 \le k \le M_b$. $\mathbf{Q}^{(k)}$ and $\mathbf{Q}^{(k+1)}$ are the previously decoded soft output value and the newly decoded soft output value used for updating the next block row, respectively. $\mathbf{L}^{(k)}$ denotes the variable-to-check message which has entered the decoding update block, and $\mathbf{r}_k^+$ represents the updated check-to-variable message at the kth block row. The updated check-to-variable message $\mathbf{r}_k^+$ can be expressed as [41]:

$$\mathrm{r}_k^+ = -\prod \operatorname{sign}(\mathrm{L}^{(k)}) \cdot \Psi\left\{\sum \Psi\left(\mathrm{L}^{(k)}\right)\right\}, \tag{4.1}$$

$$\text{where } \Psi(x) = \log\left(\tanh\left(\left|\frac{x}{2}\right|\right)\right).$$

For notational simplicity, we omit the indices denoting the set of positions of the columns connected to all check nodes within the kth block row. In Fig. 4.1, the decoding update block, which was presented as a check node-based processor (CNBP) in [44], can be implemented for any decoding algorithm such as approximations of BP.

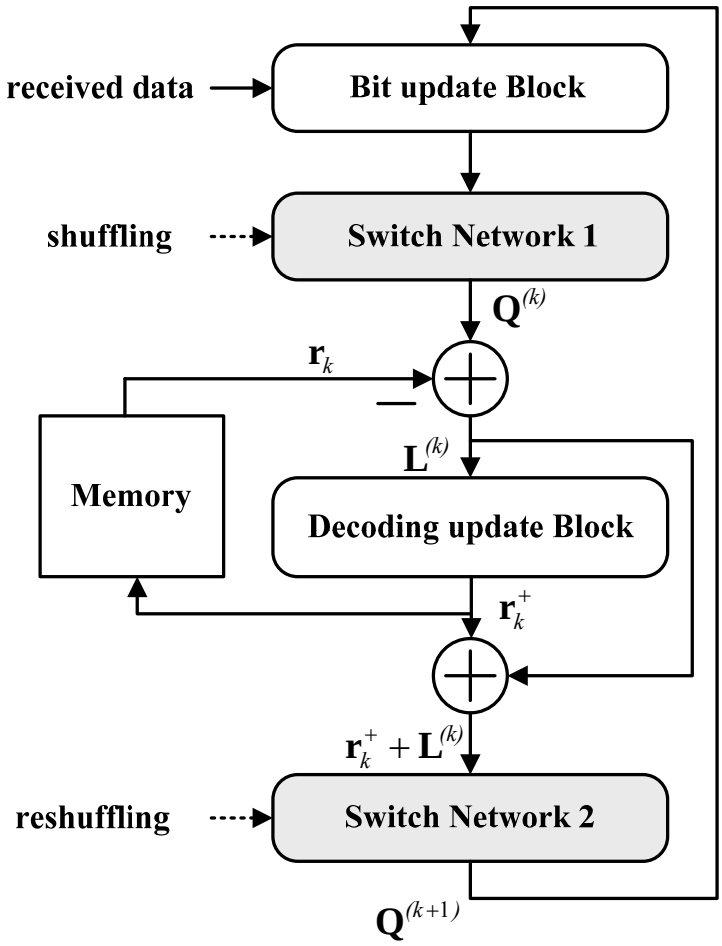


Fig. 4.1: Dataflow of a typical layered decoder.

After the initialization of the layered decoder is achieved using the soft values from the channel in the bit update block, the decoder starts updating messages corresponding to the first constituent code ($\mathbf{r}_1$). The switch network (SN) 1 shuffles the channel soft values based on the permutation information obtained from $\mathbf{r}_1$. The shifted messages $\mathbf{Q}^{(1)}$ from SN 1 and the check-to-variable messages $\mathbf{r}_1$ read from memory are used to compute the variable-to-check messages $\mathbf{L}^{(1)}$. The decoding update block computes the check-to-variable messages $\mathbf{r}_1^+$ based on $\mathbf{L}^{(1)}$ and stores $\mathbf{r}_1^+$ back into memory. The updated posterior messages are computed by adding the recently updated check-to-variable messages to the variable-to-check messages, then reshuffled through SN

2 and finally stored as $\mathbf{Q}^{(2)}$ in the bit update block. This updated soft output value $\mathbf{Q}^{(2)}$ is used to compute messages corresponding to the next constituent code ($\mathbf{r}_2$). Decoding for a constituent code ($\mathbf{r}_k$) or for the complete $\mathbf{H}$ is called one sub-iteration or one iteration, respectively.

4.2.2 Block Parallel Decoder Architecture

In this subsection, a reduced-complexity block-parallel decoder is described for layered decoding. Compared to the decoder structures presented in [38]–[41], this decoder architecture has the following unique characteristics: 1) We generate offset shifting values for shuffling and reshuffling messages so that the proposed decoder needs to use only SN 1 rather than two SNs; and 2) The number of memory banks for check-to-variable messages is configured to be a row weight of $\mathbf{H}$. The first characteristic is achieved by observing that the operation of the SNs for shuffling and reshuffling messages is overlapped during the updating of the constituent codes of $\mathbf{H}$. In other words, the SN 2 block in Fig. 4.1 reshuffles updated output messages corresponding to $\mathbf{r}_k$ until the decoder reaches the end of one iteration for the complete $\mathbf{H}$. At the end of a sub-iteration the recently updated outputs are shuffled by SN 1 for the next constituent code. Therefore, the two consecutive operations, reshuffling and shuffling, are not necessary to compute the decoded output within a sub-iteration and this provides an opportunity for reducing the complexity of the interconnections. The second characteristic is used to simultaneously process all messages corresponding to $\mathbf{r}_k$ in one clock cycle.

The dataflow of the layered decoder architecture based on the above two characteristics is shown in Fig. 4.2. The decoding steps are almost the same as in the conventional decoding with the exception of the ordering patterns in the bit update block and the offset permutations through SN 1. Let $\mathbf{P}^{(k)}$ and $\mathbf{P}^{(k+1)}$ be the previous and updated soft outputs, respectively. Note that $\mathbf{P}^{(k+1)} = \Pi(\mathbf{Q}^{(k+1)})$ is a permutation of $\mathbf{Q}^{(k+1)}$. The top-level architecture using the layered mode with offset

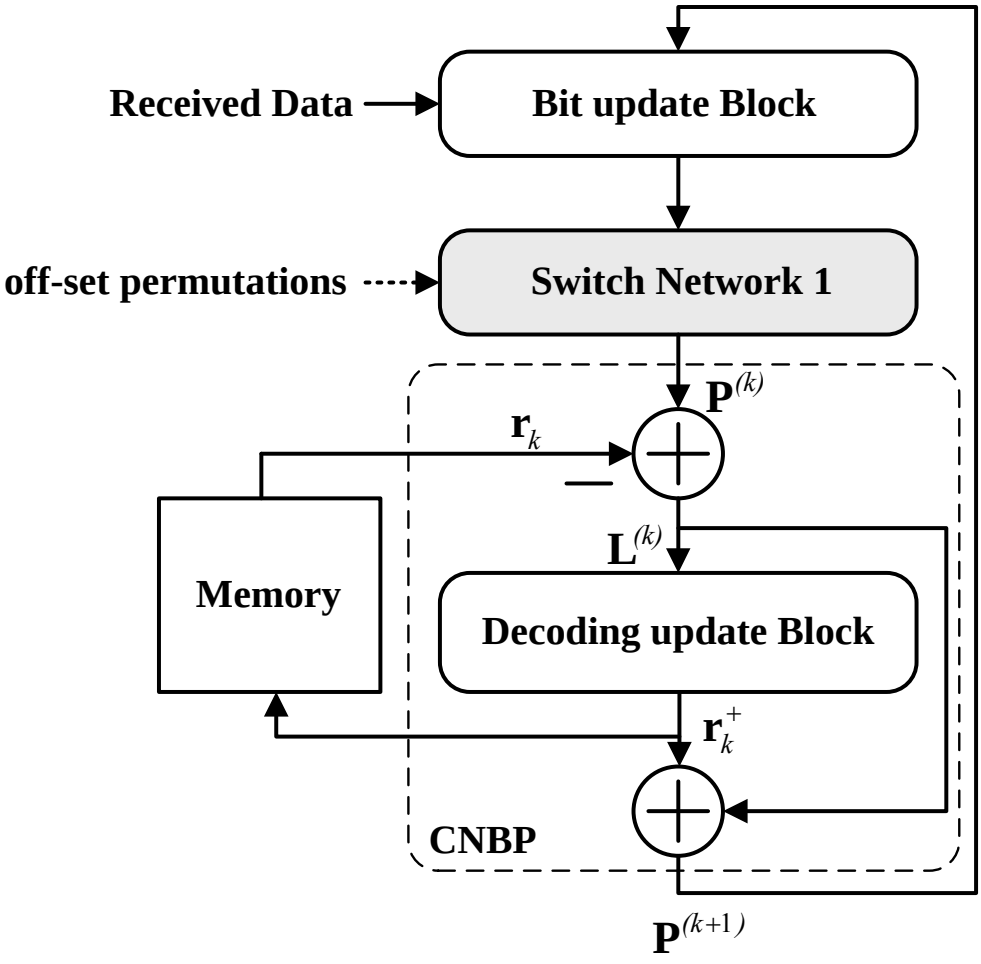


Fig. 4.2: Modified dataflow with offset permutations.

permutations for SN 1 is illustrated in Fig. 4.3. During an initialization operation, the incoming soft message is shifted into the bit updating register array. Then, the registered MUX block simultaneously loads the required messages into SN 1. Following that, SN 1 rotates the input messages by the amount of the offset permutations. The check-to-variable messages and the rotated variable messages loaded into the CNBP blocks are then computed for newly updated check-to-variable messages and rotated soft output messages. The check-to-variable messages are stored in the memory, and the rotated soft output messages replace the previous messages in the bit-update register array.

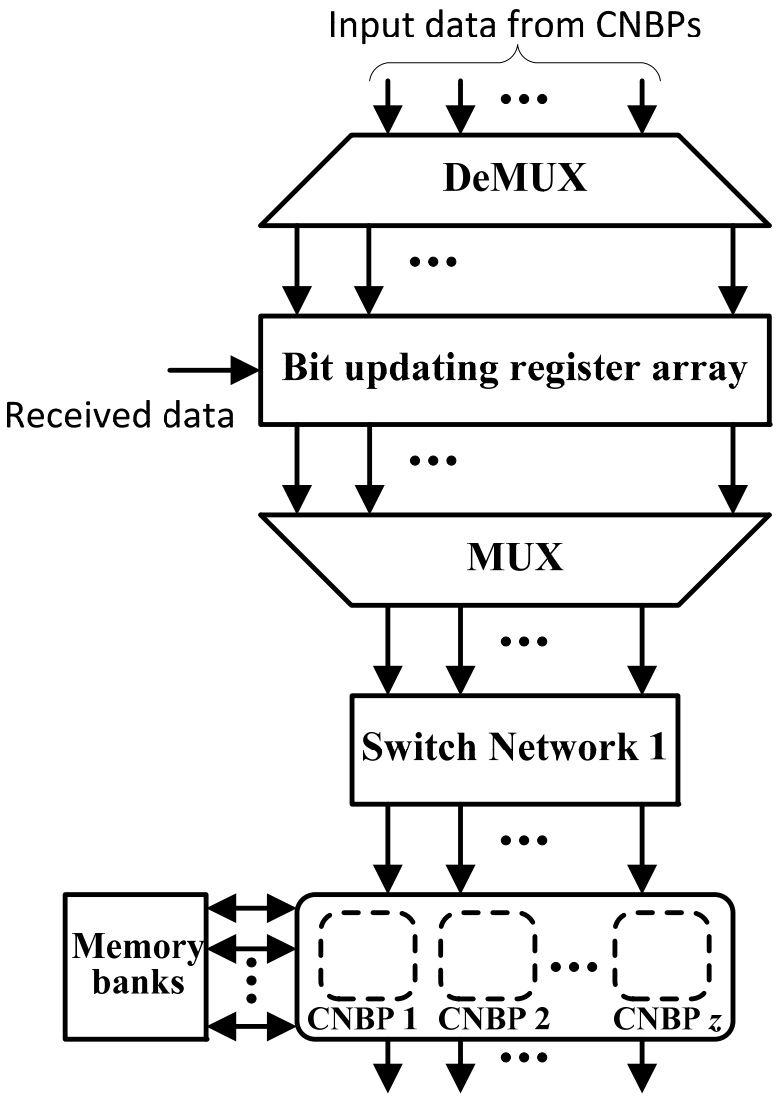


Fig. 4.3: Block parallel layered decoder architecture.

4.3 Algorithm for Generating Offset Values of Switch Network

We present a novel algorithm to generate offset values for switch networks which leads to the elimination of SN 2. A decoder design using this proposed algorithm decreases the hardware cost by removing redundant shifting operations.

In general, indices of the base matrix $\mathbf{H_b} = (h_{m,n})$ are usually represented by cyclically shifting the columns of the identity matrix $\mathbf{I}_{z\times z}$ to the right or left by $h_{m,n}$ places, where $h_{m,n} \in \{0, 1, \ldots, z - 1\} \cup \{-1\}$, for $m = 1, \ldots, M_b$ and $n = 1, \ldots, N_b$, in which '-1' represents null (i.e., all-zero) submatrices. We denote two $M_b \times N_b$ matrices of precomputed cyclic shifts for SN 1 and 2 as $\mathbf{A} = (a_{m,n})$ and $\mathbf{B} = (b_{m,n})$, respectively, where $a_{m,n}, b_{m,n} \in \{0, 1, \ldots, z - 1\} \cup \{-1\}$, for $m = 1, \ldots, M_b$ and $n = 1, \ldots, N_b$. The required cyclic shifting values of **A** and **B** can be set according to either shuffling or reshuffling operations for SN 1 or SN 2, respectively. All integer elements i, where $i \in \{0, 1, \ldots, z - 1\}$, of **A** and **B** can be stored in a dedicated look-up table (LUT). The proposed algorithm for generating offset shifting values can be described as follows:

Algorithm 1

---

for $n = 1 : N_b$

$m \leftarrow 1$

while $a_{m,n} == -1$

$s_{m,n} \leftarrow -1$

$m \leftarrow m + 1$

end

$s_{m,n} \leftarrow a_{m,n}$

$m \leftarrow m + 1$

while $m \le M_b$

$l \leftarrow m - 1$

while $b_{l,n} == -1$ and $l \ge 1$

$l \leftarrow l - 1$

end

$s_{m,n} = a_{m,n} \oplus b_{l,n}$

$m \leftarrow m + 1$

end

end

---

where $a \oplus b = \begin{cases} -1, & a = -1 \textbf{ or } b = -1, \\ (a+b) \textbf{mod z}, & otherwise \end{cases}$

In the above, we indicate that each element $s_{m,n}$ of the matrix $\mathbf{S} = (s_{m,n})$ is an offset shifting value. Therefore, we need only use SN 1 with offset shifting information $s_{m,n}$, which exploits the characteristic structure of the layered decoding scheme. Based on a given set of $s_{m,n}$ shifting values, we can reduce the amount of hardware required by removing any unnecessary 2 × 2 switches from the switching network. To compute the parity check equations using the hard decoded output $\mathbf{x} = [x_1, x_2, \ldots, x_N]^T$, which is the same as determining if $\mathbf{H} \cdot \mathbf{x} = \mathbf{0}$, the shifting information for performing the parity check equations and for sorting correctly the hard decision output is needed after each iteration. The shifting information $\mathbf{D} = (d_n)$, for $n = 1, \ldots, N_b$, is described in Algorithm 2:

Algorithm 2

for $n = 1 : N_b$

$m \leftarrow M_b$

$d_n \leftarrow b_{m,n}$

while $d_n == -1$

$m \leftarrow m - 1$

$d_n \leftarrow b_{m,n}$

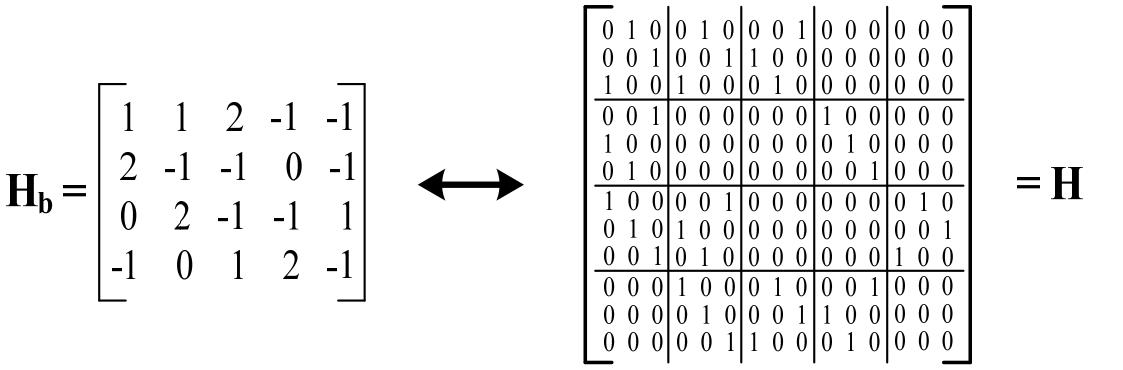
end

end

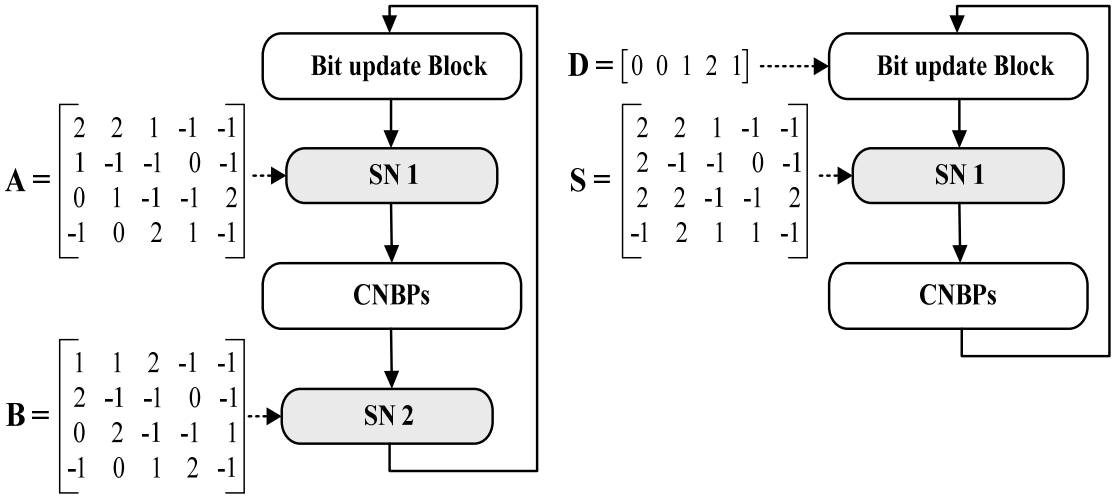
Given the shifting information **D** and the hard decisions **x**, we can pre-compute the output ordering information $\mathbf{y} = \mathbf{E} \cdot \mathbf{x}$, where **E** can be written as:

$$\mathbf{E} = \begin{bmatrix} \boldsymbol{p}^{d_1} & \mathbf{O} & \cdots & \mathbf{O} \\ \mathbf{O} & \boldsymbol{p}^{d_2} & & \vdots \\ \vdots & & \ddots & \mathbf{O} \\ \mathbf{O} & \cdots & \mathbf{O} & \boldsymbol{p}^{d_{N_b}} \end{bmatrix}$$

Here, **O** indicates a $z \times z$ zero matrix, and $\boldsymbol{p}^j$ is obtained from the $\mathbf{I}_{z \times z}$ by cyclically shifting the columns to the right by j elements. From the output ordering information **y**, the decoded data can be easily mapped to the output ports of this decoder without extra hardware cost.



(a)



(b) (c)

Fig. 4.4: Example of generating offset values for the switch networks. (a) 4 × 5 $\mathbf{H_b}$. (b) Typical layered decoder. (c) Proposed decoder.

EXAMPLE: The base matrix $\mathbf{H}_b$ in Fig. 4.4 (a) is a 4 × 5 array of 3 × 3 cyclicly-shifted identity or all-zero matrices. The control signals represented in matrix form, **A** and **B**, for SN 1 and SN 2 as shown in Fig. 4.4 (b) can be determined based on the elements of the $\mathbf{H}_b$. We compute offset control signals and decoded output mapping information, which are illustrated in Fig. 4.4(c), by using the proposed Algorithms 1 and 2.

4.4 Hardware Complexity Comparison

In layered decoding architectures the number of memory bits is reduced by nearly 50% and the number of iterations for achieving the same error rate is also reduced by almost 50% compared with traditional decoder designs [41]. To show the low complexity of the block parallel processor in the layered decoding scheme, we compare it with different decoder architectures.

Table 4.1: Key Component Characteristics for Three Different Designs

	Design A [37]		Design B [38]			Proposed scheme
	CNU	VNU	CNU	VNU	SSAU	CNBP
LUT	8	4	1	4	1	16
Adder	15	8	1	8	1	31
Ex-OR	15	-	1	-	1	15
SM-2's	-	4	-	4	-	8
2's-SM	-	4	1	-	1	8
Registers	-	-	-	-	2	25
No. of functional unit	84	168	336	84	336	21

The irregular LDPC code in the IEEE 802.15 standard has a 16 by 32 $\mathbf{H}_b$ with $z = 21$, so that its parity check matrix $\mathbf{H}$ has 16×21 rows and 32×21 columns. Tables 4.1 and 4.2 present, for three different designs, characteristics of the key components used and the estimated total number of hardware resources required, respectively (Note that designs A [37] and B [38] do not use layered decoding). In Table 4.1, design A is obtained using a folding factor of 4 for both the CNUs and the

Table 4.2: Estimated Total Hardware Resources for Three Different Designs

	Design A [37]	Design B [38]	Proposed scheme
LUT	1,344	1,008 (100%)	336 (33%)
Adder	2,604	1,344 (100%)	651 (48%)
Ex-OR	1,260	672 (100%)	315 (47%)
SM-2's	672	336 (100%)	168 (50%)
2's-SM	672	336 (100%)	168 (50%)
Registers	-	672 (100%)	525 (78%)

VNUs in a time-multiplexed approach and it requires 8 clock cycles to complete one decoding iteration. Design B uses a set of sum and sign accumulation units (SSAUs) in addition to the CNUs and VNUs. In this design, the CNUs and SSAUs are fully parallel while the VNUs have a folding factor of 8, and it requires 9 clock cycles to complete one decoding iteration.

To perform a fair comparison with the hardware complexity of designs A and B, we use a standard Log-BP structure in the CNBP as shown in Fig. 4.5. For simplicity, sign-magnitude (SM), 2's complement (2's) and exclusive-or (xor) units are not shown in the figure. Pipeline registers are inserted to provide the same critical path in all 3 designs, which is equal to the path from a LUT to an 8-input adder tree block. For the **H** matrix considered here, there are no data dependencies between adjacent layers while updating posterior messages. For other **H** matrices having such dependencies, stalls could be used to avoid conflicts in the pipeline.

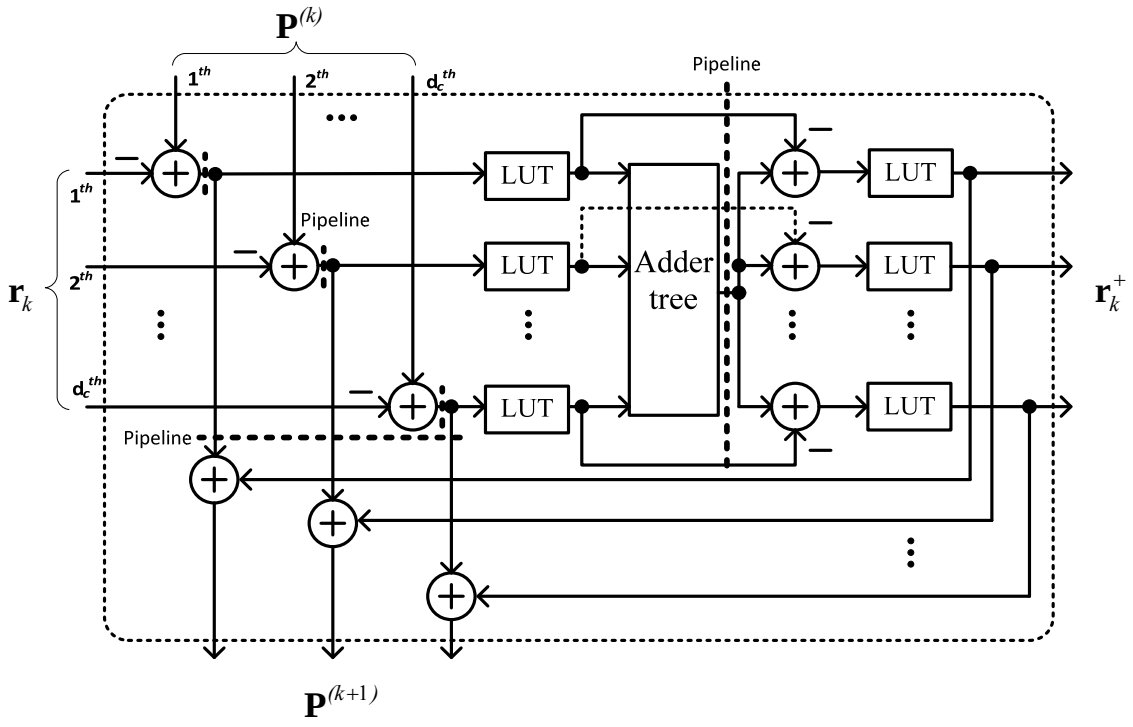
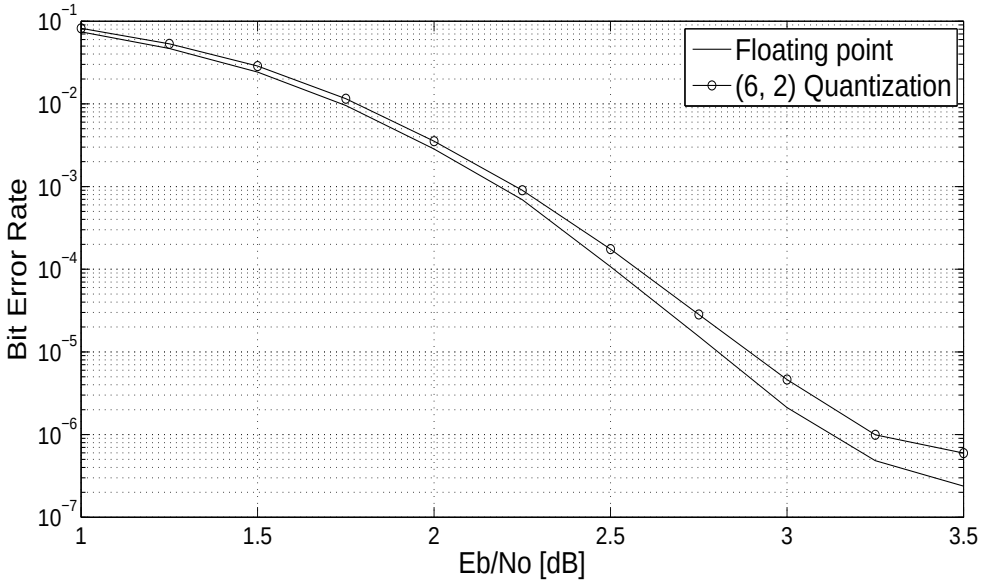


Fig. 4.5: Simplified diagram of a block parallel layered decoding unit, the check-node based processor (CNBP).

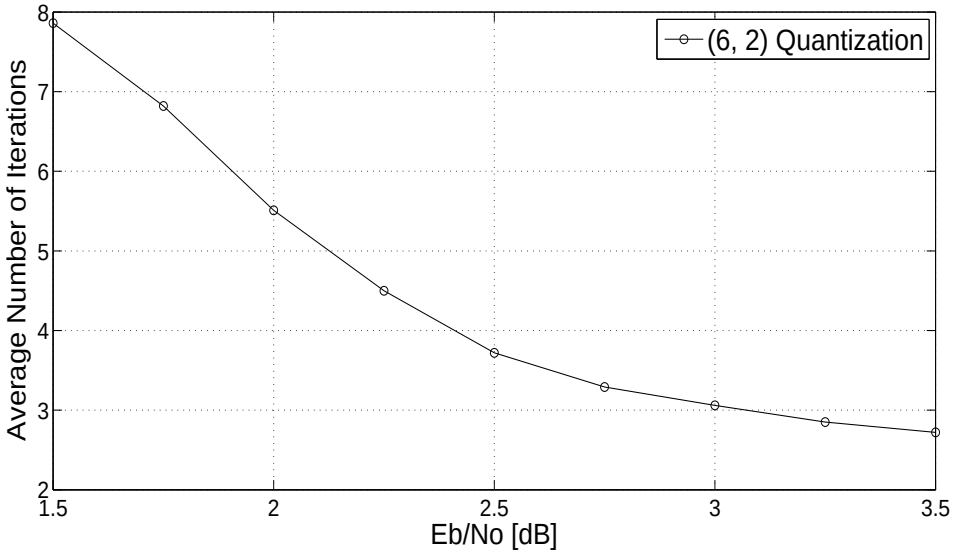
The decoding throughput can be approximated as:

$$\text{Throughput} \approx \frac{N \times f_{\text{clk}} \times R}{N_{\text{clk}} \times N_{\text{iter}} + N_{\text{latency}}}, \tag{4.2}$$

where f_{clk} is the clock frequency, R is the code rate, N_{clk} is the number of clock cycles required for an iteration, N_{iter} is the average number of iterations and N_{latency} is the number of clock cycles due to the pipeline latency. Note that a layered decoding scheme needs only about half the average number of iterations compared with designs A and B in order to achieve same error rate. Therefore, the proposed design uses about twice as many clock cycles per iteration (i.e., 16 clock cycles vs. 8 for design A and 9 for design B) with no throughput degradation. As shown in Table 4.2, the hardware complexity of the decoding processing units and the amount of memory required for the proposed design is significantly smaller than for either design A or B.



(a)



(b)

Fig. 4.6: Simulated performance for $N = 672$, rate-1/2 irregular LDPC code. (Maximum number of iterations = 8). (a) Bit error rate for the proposed design. (b) Average number of iterations using the (6, 2) quantization.

4.5 Implementation Results

We designed an N = 672 (data length = 336), rate-1/2 irregular LDPC decoder based on the proposed offset control scheme for the SN using a block parallel architecture. The min-sum decoding algorithm, which is a modified version of the standard Log-BP algorithm, is exploited in the CNBP, which has four pipeline stages. Based on the simulated performance results of Fig. 4.6 (a), we use a (q, f) = (6, 2) quantization scheme, where q and f are the total bit size and the number of fractional bits, respectively. Furthermore, our decoder typically needs only 3 iterations to converge at a signal-to-noise ratio of 3 dB, as shown in Fig. 4.6 (b).

Our SN uses a Benes network [13] in which the unnecessary switches have been removed, and it uses three pipeline stages in order to reduce the critical path delay. As a result, the critical path of the pipelined SN is three 2 × 2 switches. This decoder was implemented on the Xilinx Virtex-4 xc4vlx200 FPGA. To provide a fair comparison, we also implemented a conventional layered decoding design for the same code using the same quantization and pipelining techniques. The synthesis results for both designs are given in Table 4.3. The proposed decoder, using only a single SN, results in 9.3% reduction in the number of slices with no degradation in the decoding throughput.

The proposed decoder has a pipeline latency of 9 clock cycles, of which 7 cycles are due to the SN 1 and CNBP blocks and the other 2 cycles are due to the registered MUX and DEMUX blocks. The information decoding throughput is estimated to be approximately (335 MHz) × 336 / (8 × 16 + 9) = 822 Mb/s based on the maximum clock frequency of 335 MHz (from the synthesis timing report) and using a maximum of 8 decoding iterations and the pipeline latency of 9 cycles. The estimated gate count (14 adders, 20 muxes, and 11 xor gates per VNU and CNU) in [42] is based on a different code, i.e. a 3456-bit, rate 1/2, (3, 6)-regular code, i.e. having a column weight of 3

Table 4.3: Xilinx Virtex4 xc4vlx200 FPGA Synthesis Results

Resource	Conventional layered decoding	Proposed scheme	Improvement
Slices	29.763	27,003	9.27%
Slice Flip Flops	26,281	23,206	11.7%
4 input LUTS	51,298	45,409	11.5%
Block RAMs	32	32	-
Throughput	798 Mb/s	822 Mb/s	3.01%

and a row weight of 6. The design in [9] requires 12 clock cycles per iteration, while our decoder design requires only 3 clock cycles per iteration. However, the estimated gate count of our design (21 adders, 31 muxes, and 11 xor gates) is higher than that of [42].

4.6 Functional Verification of LDPC Decoder

In this section, we will show the state transition diagram of the top control block, the hardware architecture of the control module and the hardware complexity reduction method used in the switch network. Finally, functional verification of our LDPC decoder is given.

4.6.1 Architecture of the Control Module

The control module generates memory addresses (*ADDA, ADDB* and *WEB*) for reading and writing, and several control signals such as the shift signal (*SHIFT_OK*) for shifting intrinsic soft input messages into the input shift-registers block, the present state signal (*P_STATE*) for entering the input initialization or decoding processing states, the iteration signal (*ITER*) for counting the number of iterations, the control signal (*CS*) bits for the cyclic-shifted identity permutations, the count signal (*COUNT*) for tracking layers of the base matrix $\mathbf{H_b}$ while the LDPC decoder is in the decoding processing state, and the decoding termination signal (*DECODING_DONE*) to indicate if the decoded outputs have been obtained within the maximum number of allowed iterations. Fig. 4.7 shows the state transition diagram of our top control module. In the Input initialization state, the intrinsic soft input messages are shifted into the input shift-registers block. We assume that the number of received codeword elements per clock cycle is 21. Therefore, 32 clock cycles are required to obtain one complete codeword (i.e., 21*32 = 672). After 32 clocks for shifting the soft input messages, the top control module enters the Decoding processing state.

Fig. 4.8 shows the block diagram of the control module. The COUNTER module generates the memory read/write address (*ADDA* and *ADDB*) and write enable (*WEB*) signal for the 32 RAMs. The Iteration Logic generates the number of iterations based on the *COUNT* and *P_STATE* signals.

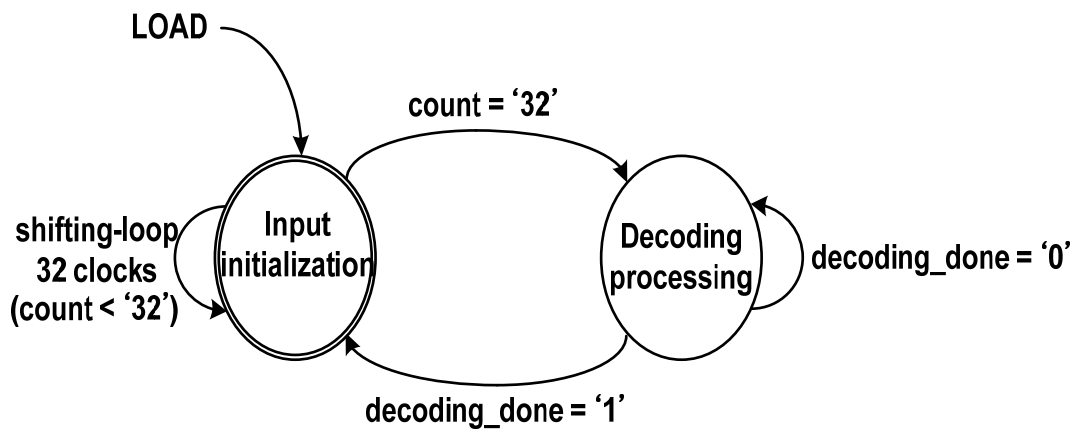


Fig. 4.7: State transition diagram of the top control module.

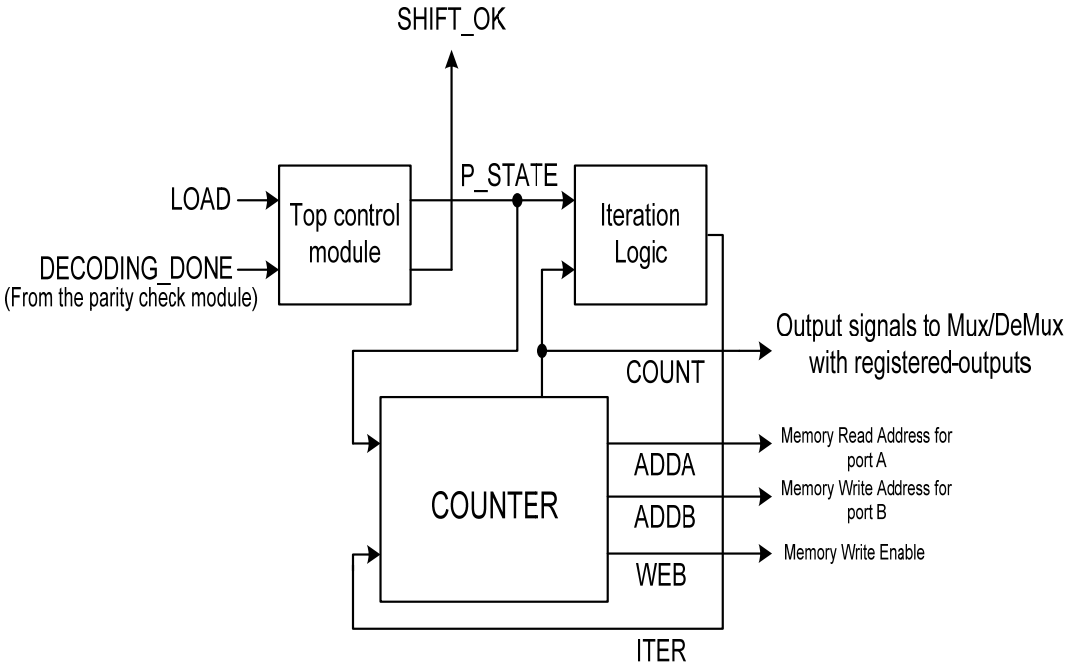


Fig. 4.8: Block diagram of the control module.

4.6.2 Architecture of the Optimized Switch Network

There are several types of switching networks, such as Banyan networks [47], Benes networks [48] and 64 × 64 dual bi-directional networks [40], which have been used in LDPC decoders. In this chapter, we use an optimized switch network that is a modification of the Benes network.

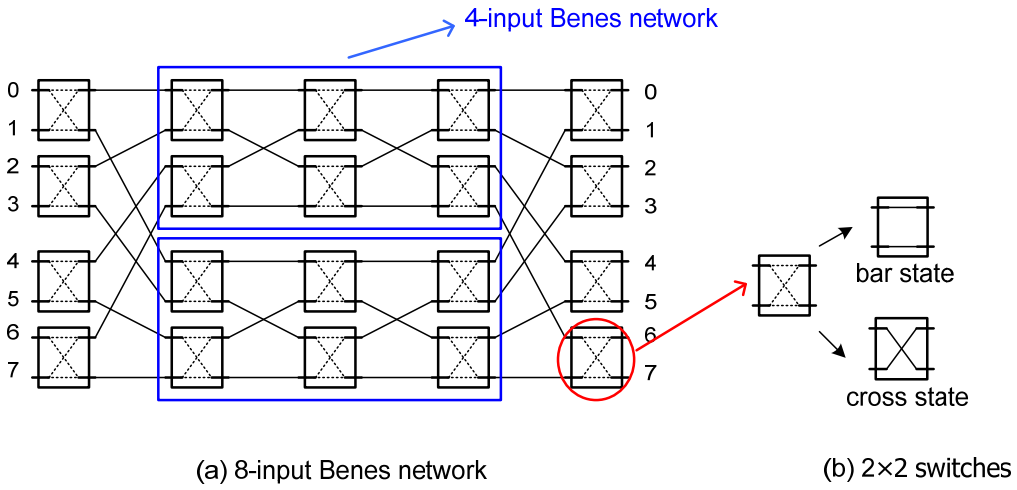


Fig. 4.9: Switch network structure.

Fig. 4.9 (a) and (b) show an 8-input Benes network structure and 2×2 switches, respectively. Each switch element can be in either in the bar state (when the control signal is 0) or in the cross state (when the control signal is 1), as shown in Fig. 4.9 (b). To control all the 2×2 switches in the 8-input Benes network, control bits must be provided for 28 output combinations. However, there are only a limited number of cyclic shifts in the parity check matrix H for IEEE 802.15.3c. Therefore, it is sufficient to provide a limited set of control signals to implement the required set of cyclic shifts. Recently, a controller design for reconfigurable LDPC decoders has been presented in [48]. Instead of using their reconfigurable barrel shifters, we find the required 2×2 switches and reduce the control bits for a set of known cyclic shifted permutations by incorporating the algorithm [48] into the characteristics of the structured parity check matrices. In Fig. 4.10, a 32-

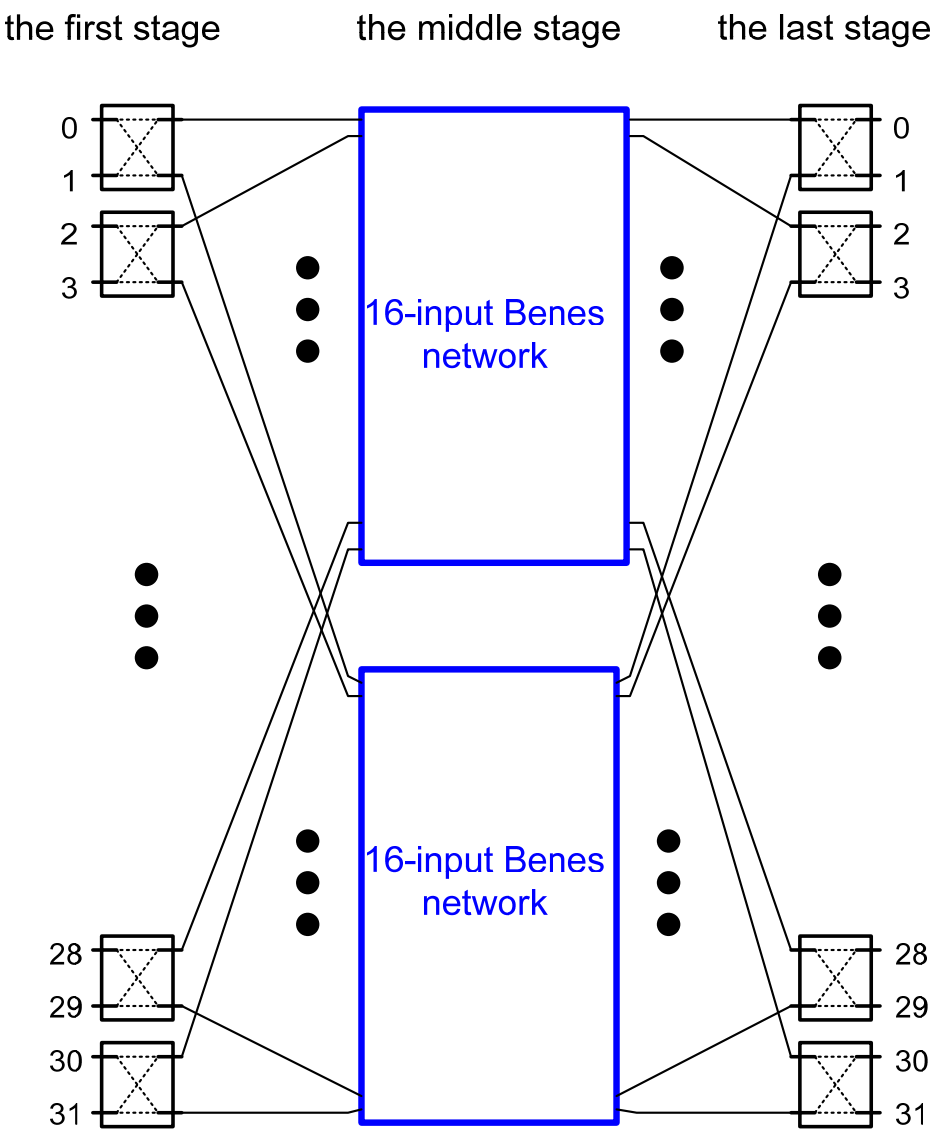


Fig. 4.10: 32-input Benes network.

input Benes network including two 16-input Benes networks is shown. In the Benes network for an LDPC decoder, the control signals are stored in a dedicated look-up table (LUT). For example, the control signals for each cyclic shifted permutation in the millimeter wave 60-GHz wireless personal area networks would require 144 bits (i.e., 144 2×2 switches in Fig. 4.10). However, in our optimized switch network, only 68 2×2 switches are required to implement the optimized

switch network. Moreover, the control signals in the middle stage and the control signals in the last stage only need a smaller number of bits, namely 35 bits and 1 bit, respectively. The control signals corresponding to the results of our computer simulation are given in Table 4.4. The colored parts in Table 4.4 indicate the reduction of the control signals.

Table 4.4: Control signals for the optimized switch network.

Cyclic shift index	Control bits in the first stage	Control bits in the middle stage	Control bits in the last stage
0	**{10{1'b0}}**	**{48{1'b0}}**	**{10{1'b0}}**
1	**10'b00000000_00**	**48'b00000_000000_00000_00001_00000001_00000_00111_0000011111**	**10'b11111111_11**
2	**10'b00000000_01**	**48'b00000_000000_00000_01001_00010001_00_110_0111_1111111111**	**10'b00000000_00**
3	**10'b00000000_01**	**48'b00001_000000_00000_01101_00010101_00_111_1111_1111100000**	**10'b11111111_11**
4	**10'b00000000_11**	**48'b00001_000000_00000_11101_01010101_11_111_1111_0000000000**	**10'b00000000_00**
5	**10'b00000000_11**	**48'b00001_000000_00000_11111_01010111_11_111_1000_0000011111**	**10'b11111111_11**
9	**10'b00000011_11**	**48'b00011_010101_00001_11111_11111110_00000_00111_0000011111**	**10'b11111111_11**
11	**10'b00000111_11**	**48'b00111_010101_01101_11111_11101010_00_111_1111_1111100000**	**10'b11111111_11**
12	**10'b00001111_11**	**48'b00111_010101_11101_11111_10101010_11_111_1111_0000000000**	**10'b00000000_00**
14	**10'b00011111_11**	**48'b00111_011101_11111_11111_10001000_11_001_1000_1111111111**	**10'b00000000_00**
15	**10'b00011111_11**	**48'b01111_011111_11111_11111_10000000_11_000_0000_1111100000**	**10'b11111111_11**
16	**10'b00111111_11**	**48'b01111_111111_11111_11111_00000000_00_000_0000_0000000000**	**10'b00000000_00**
17	**10'b00111111_11**	**48'b01111_111111_11111_11110_00000001_00000_00111_0000011111**	**10'b11111111_11**
18	**10'b01111111_11**	**48'b01111_111111_11111_10110_00010001_00_110_0111_1111111111**	**10'b00000000_00**
19	**10'b01111111_11**	**48'b11111_111111_11111_10010_00010101_00_111_1111_1111100000**	**10'b11111111_11**

4.6.3 Functional Verification of the Implemented LDPC Decoder

We show the timing diagrams of the implemented LDPC decoder, which was simulated in Verilog using ModelSim. As an example of functional verification, we have designed and tested the proposed decoder architecture using the length-672 and rate-1/2 irregular LDPC code. We fix the number of soft bits at 6. There are four major processing modes of the layered LDPC decoder, which can be described as follows.

1) *Initialization mode*: During an initialization operation, the incoming soft message is shifted into the bit updating register array in 32 clock cycles.

2) *Read/Switch Operation mode*: During a read operation, i) the content of the C2V_MEM memories at the address on the ADDA inputs becomes valid at the output ports of the MUX with registered-outputs block, and ii) the content of the input shift-registers are loaded into the registered-output of the MUX when ITER = 1 and COUNT = 0. During a switch operation, the optimized switch network rotates the input message vector by an amount depending on the COUNT value.

3) *Computation Operation mode:* During a computation operation, the CNBPs need to fetch and compute data simultaneously. This operation requires two clock cycles in order to balance the critical-path delay between CNUs and VNUs.

4) *Write Operation mode:* During a write operation, the content of the C2V_MEM at the location specified by the address on the ADDB inputs is replaced by the value on the output ports of the DeMUX with registered-outputs block.

The modules have been verified using C++ at the algorithm level and Verilog at the architecture level. In other words, our Verilog simulations were found to match the results of the C++ simulations.

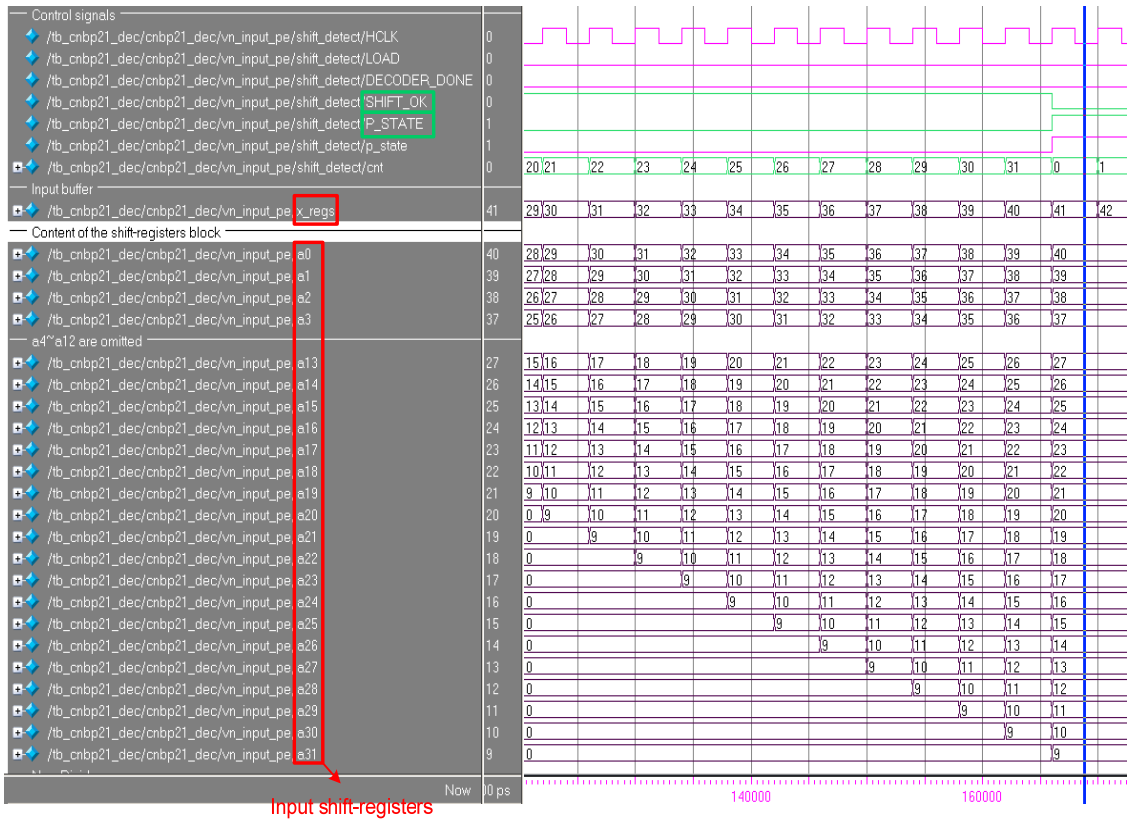


Fig. 4.11: Simulation waveforms of the Initialization mode.

In the Initialization mode, data previously stored at the input buffer (x_regs) are shifted into the shifted-registers block when SHIFT_OK is 1. See Fig. 4.11.

The simulation waveforms in Fig. 4.12 describe the MUX with registered-output block in the read operation mode and the optimized switch network in the switch operation mode. As seen above, the contents of the input shift-registers block are as follows:

Reg a0 = 40

Reg a1 = 39

Reg a2 = 38

Reg a3 = 37

Reg a4 = 36

... ...

... ...

Reg a30 = 10

Reg a29 = 9

We can check that the specific contents out of the input shift-registers are loaded into the registered-output of the MUX at time ITER = 1 and COUNT = 0. These messages (i.e., initial soft data), such as 37, 35, 30, 28 and 21, are used to compute the first layer of the base matrix $\mathbf{H_b}$. In Fig. 4.12, 15 different messages are used to illustrate the correct switching operation. During the Switch Operation mode, we can see that blue highlighted data (shifting by 0) are correctly switched by the control signals. Fig. 4.13 shows the C2V_MEM read operation and valid data at the MUX with registered-outputs. We generated 32 dual-port RAM modules by using the Xilinx CORE Generator™ block memory modules. By default, block RAM is initialized with all zeros during the device configuration sequence. For the functional verification, we initialized some values in 32 RAMs, which are shown as follows.

	RAM 1	RAM 2	RAM 3	...	RAM 31	RAM 32
Address 0	8	9	10		38	39
1	9	10	11		39	40
2	0	0	0		0	0
3	4	5	6		34	35

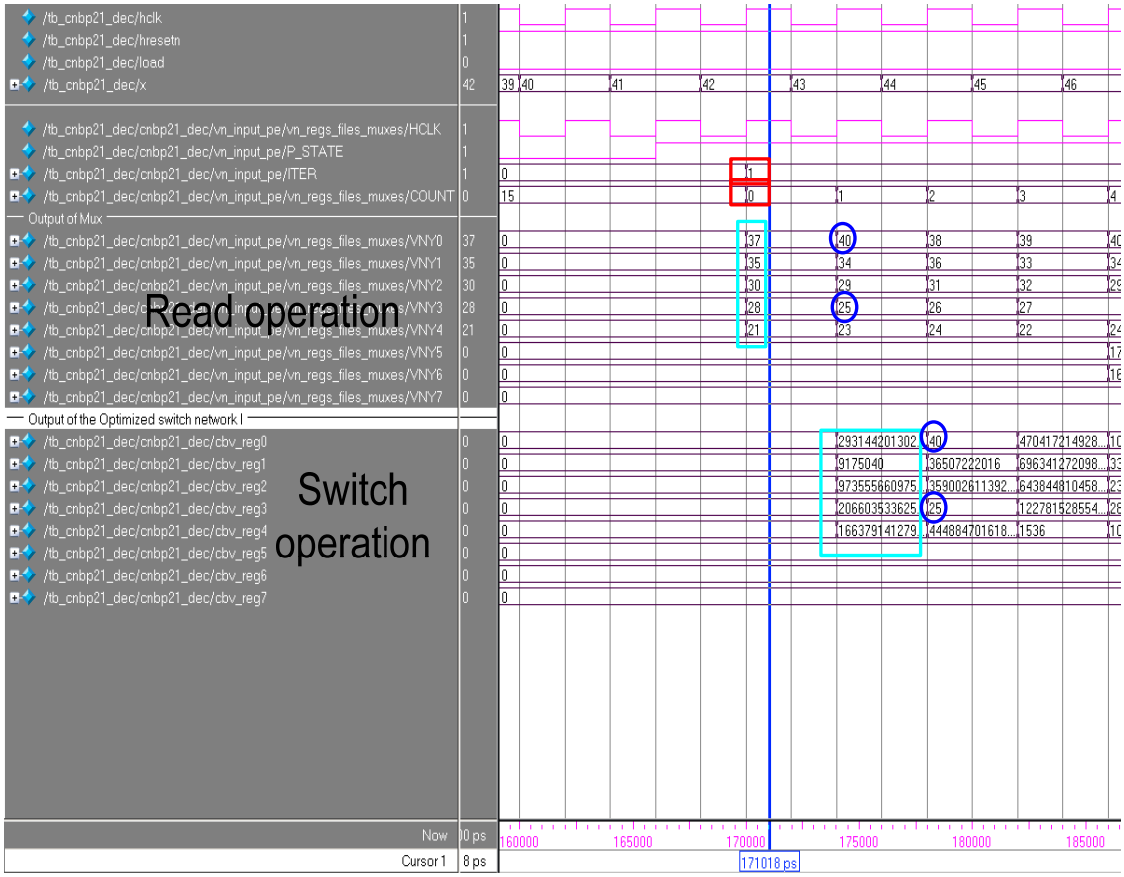


Fig. 4.12: Simulation waveforms of the Read/Switch Operation mode.

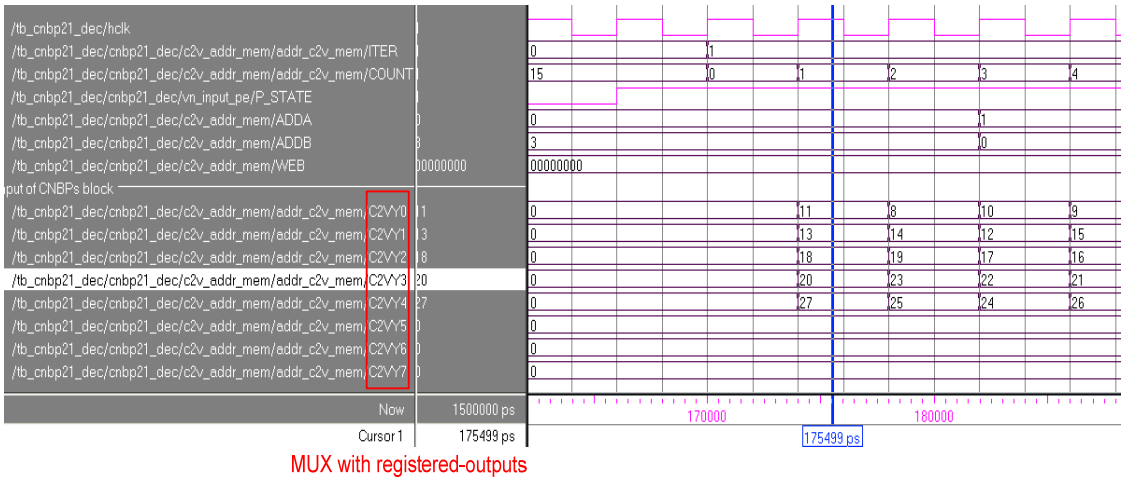


Fig. 4.13: Simulation waveforms of C2V_MEM and MUX with registered-outputs.

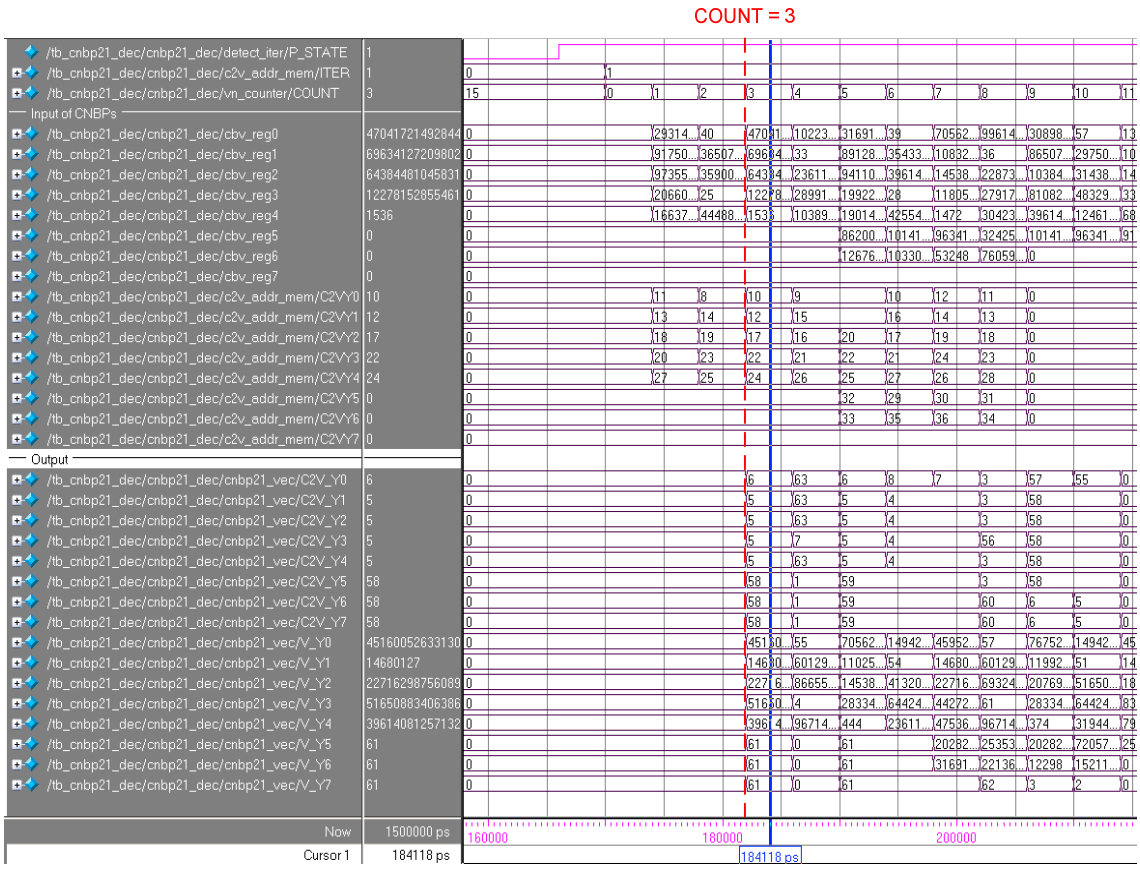


Fig. 4.14: Simulation waveforms of the CNBPs.

We inserted pipeline registers to decrease the critical path of the CNBP. After pipelining, the CNBP processing takes 2 clock cycles. The CNBP was verified using C++ at the algorithm level and Verilog at the architecture level in Task 1. Here, we will show the timing diagram of the CNBPs. As shown in Fig. 4.14, the output data becomes valid at time COUNT = 3. In the writing operation mode, Fig. 4.15 shows that 6 clock cycles are required to process the received soft vector. In other words, one sub-iteration takes 6 clock cycles. In a rate-1/2 LDPC code, one iteration consists of 16 sub-iterations. However, we used pipelining in the CNBPs, Mux, DeMux, and optimized switch network in order to reduce the number of clock cycles per decoding processing step and the critical path delay. Using the verification set-up shown in Fig 4.16, the rate-1/2 LDPC decoder was simulated in both C and Verilog.

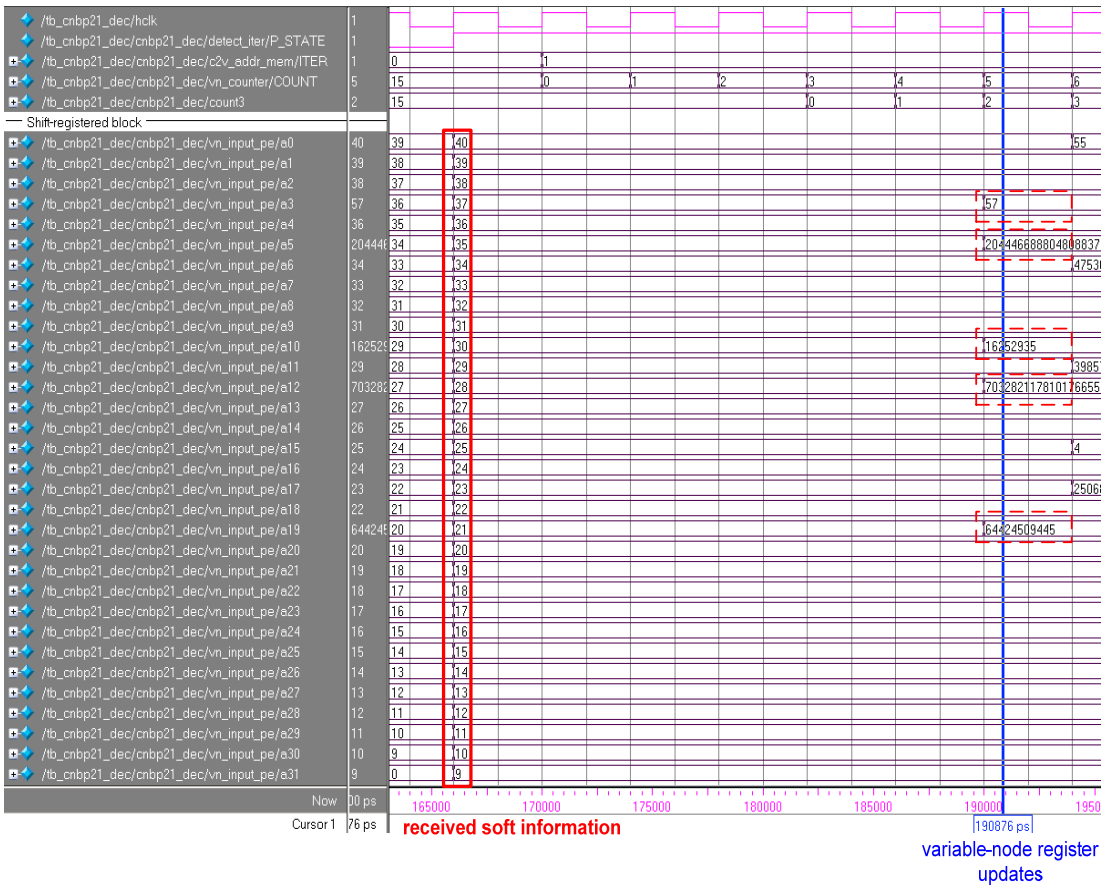


Fig. 4.15: Simulation waveforms of the input shift-registers.

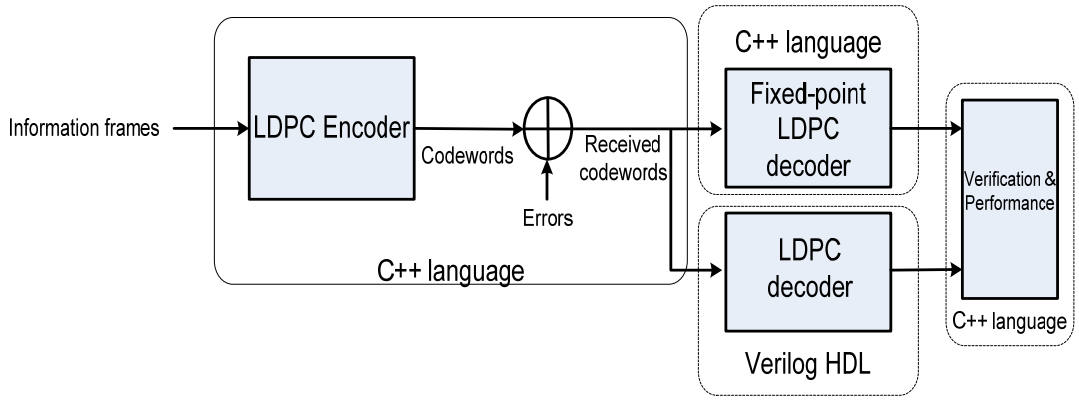


Fig. 4.16: Block diagram of the LDPC decoder verification.

4.7 Conclusion

We have proposed an efficient architecture for layered LDPC decoding by reducing the interconnection complexity without any degradation in the decoding throughput. Our design requires only a single shuffle network, rather than the two shuffle networks used in prior designs. The results show a 9.3% reduction in the number of required FPGA slices compared to a standard layered decoding architecture. Implementation of a 672-bit, rate-1/2 irregular LDPC code on a Xilinx Virtex-4 xc4vlx200 FPGA device achieves an information throughput of 822 Mb/s with a maximum of 8 iterations.

Chapter 5

Quantization of FFT-Based Belief Propagation for Nonbinary LDPC Codes

5.1 Introduction

LDPC codes defined over nonbinary fields GF(q) were introduced by Davey and MacKay [49] and the binary sum-product or belief propagation (BP) algorithm has been generalized to decode non-binary LDPC codes. A variation of the Min-Sum algorithm achieves a suboptimal iterative decoding at the cost of decoding performance loss by decreasing the complexity of the check node update [51] [52][53][57]. To reduce the loss in the decoding performance, the optimal iterative decoding, BP or sum-product, algorithm can be performed by using the fast Fourier transform (FFT), which was utilized by Richardson and Urbanke [50] for decoding binary LDPC codes, followed by an inverse FFT at the check node update in the probability domain [54] [55].

In order to reduce the complexity of BP used with the FFT/IFFT, which is referred to as the FFT-based BP algorithm, the authors in [56] [58][59][65] performed the calculations in the logarithm domain. However, the logarithm and exponential computations in the check node update may incur overflows in the soft information due to the finite word-length. Investigation of the

optimal word-length for non-binary LDPC decoders was introduced by selecting the proper word-length for FFT-based BP [65] or a log-domain version [61] of BP using the max*-operation, where $\max^*(\alpha, \beta) \equiv \ln(\exp(\alpha) + \exp(\beta))$. Subtractions in the log domain, instead of using a normalization step in the probability domain, was used in [58] and the max*-operation in [61], which increases the number of look-up tables (LUTs), was required to normalize the outputs of the variable nodes.

In this chapter, we focus on quantization effects in the nonlinear functions and investigate the dynamic range at various points in the check node update. In particular, we introduce an improved FFT-based BP decoding algorithm having a threshold factor. This enables us to reduce the size of the LUTs used in the check node update without sacrificing the decoding performance.

5.2 FFT-Based BP Algorithm in the Logarithm Domain

In an LDPC code over GF(q) = {0, 1,…, q – 1}, where q is a prime number or a power of a prime number, each entry $H_{i,j}$ in a sparse parity-check matrix **H** of size $M \times N$ is one of the q elements in GF(q). In particular, an LDPC code over GF($q = 2^p$), which is an extension field of GF(2), groups p bits into an element of this field. Let $\mathbf{C} = [c_1\ c_2\ \ldots\ c_N]^T$ denote the transmitted codeword, where c_n corresponds to a symbol defined over GF(2^p), for $1 \le n \le N$. Binary-phase-shift-keyed (BPSK) signaling (i.e., with the mapping $0 \rightarrow 1$ and $1 \rightarrow -1$) is used in mapping each symbol c_n to p-bits in the transmitted signal. We consider an additive white Gaussian noise (AWGN) channel in which the noise is a zero-mean Gaussian signal with variance σ^2. Suppose that $\mathbf{Y} = [y_1\ y_2\ \ldots\ y_{pN}]^T$ is a channel observation corresponding to the transmitted vector $\mathbf{X} = [x_1\ x_2\ \ldots\ x_{pN}]^T$, where **X** is the BPSK version of **C**.

As with other iterative algorithms, the FFT-based BP algorithm relies on the exchange between variable (symbol) nodes and check nodes to achieve correct symbol decisions. In the

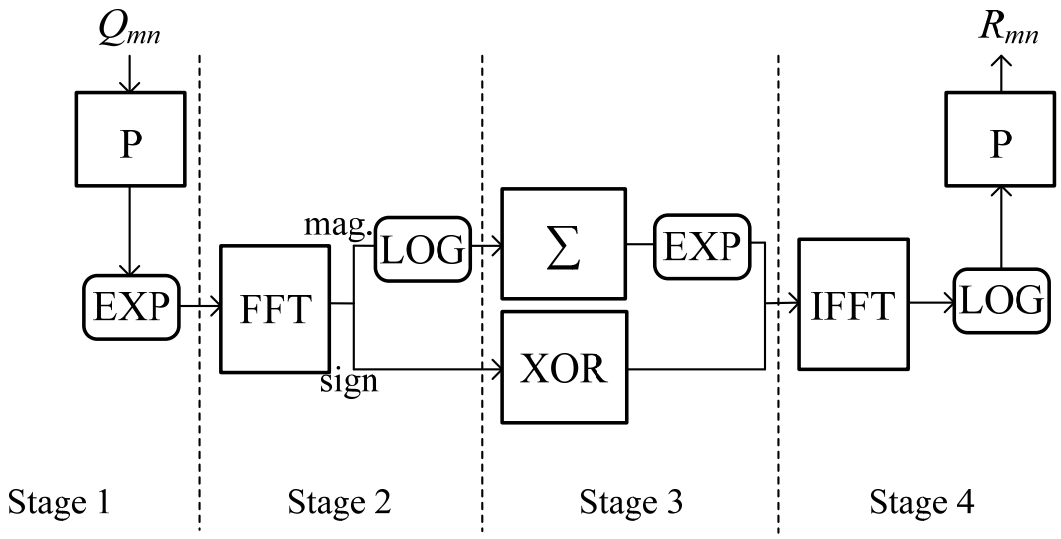


Fig. 5.1: Check node update block.

initial stage, the prior log likelihood value of each variable node c_n being equal to a, where $a \in \mathrm{GF}(2^p)$, using the channel output $\mathbf{Y}$ is as follows:

$$L_{\text{channel}}(c_n = a) = \sum_{k=1}^{p} \frac{y_{(n-1)\times p+k}}{\sigma^2}(-1)^{a_k}, \tag{5.1}$$

where a_k is the kth bit of the binary representation of a and σ^2 is the estimated noise variance. Therefore, all prior log likelihood values, which are intrinsic messages, at the variable node n can be represented as a vector $\mathbf{L}_{\text{channel}}(c_n = a) = [L_{\text{channel}}(c_n = 0)\ \ L_{\text{channel}}(c_n = \alpha^0)\ \ L_{\text{channel}}(c_n = \alpha^1)\ \ \ldots\ L_{\text{channel}}(c_n = \alpha^{q-2})]^T$, where a is a primitive element of GF(q). Given a set of prior log likelihood values on the symbols, each iteration computes the check-to-variable message R_{mn} and variable-to-check message Q_{mn} for all checks $\{m\}$ and variables $\{n\}$. For each $(m, n) \in \{(i, j) \mid H_{i,j} \neq 0\}$ and $a \in \mathrm{GF}(2^p)$, we initialize R_{mn} and Q_{mn}, which are referred to extrinsic messages, as follows:

$$R_{mn}(a) = 0, \tag{5.2}$$

$$Q_{mn}(a) = L_{\text{channel}}(c_n = a). \tag{5.3}$$

The iterative decoding stages between the check nodes and the variable nodes are similar to the conventional FFT-based BP algorithms [56] [58][59][65] in the logarithm domain. The check node update based on the permutation and FFT operation (see for example [58][65]) is illustrated in Fig. 5.1. It consists of two nonlinear functions, exponential (EXP) and natural logarithm (LOG), two permutations (P), one summation (∑), exclusive-or (XOR) gates, FFT and IFFT blocks. Compared with the above check node update, the operations performed at the variable nodes are quite simple. For each $(m, n) \in \{(i, j) \mid H_{i,j} \neq 0\}$ and $a \in \text{GF}(2^p)$, the update operation at the variable nodes is given by

$$Q_{mn}(a) = L_{\text{channel}}(c_n = a) + \sum_{i \in M(n), i \neq m} R_{in}(a), \tag{5.4}$$

where $M(n)$ denotes the set of check nodes connected to variable node n. The estimated codeword is determined by making a hard decision of the Q_n for variable node n computed as follows:

$$Q_n = \arg\max_a \left\{ L_{\text{channel}}(c_n = a) + \sum_{i \in M(n)} R_{in}(a) \right\}. \tag{5.5}$$

The decoding algorithm terminates if either the estimated codeword satisfies all the parity check equations or if the maximum number of iterations is reached.

5.3 Improved Quantization Scheme for FFT-Based BP Decoding

In a fixed-point environment, the limited precision used in calculating nonlinear functions results in propagation of errors and, consequently, leads to a performance degradation. Here, we introduce a procedure which reduces the performance loss without having to increase the precision of the system.

Table 5.1: Output Range of Exponential Function for Various Quantizations

	(7, 1) quantization	(7, 3) quantization	(7, 4) quantization	(7, 5) quantization
w	$\lvert w \rvert \leq 31.5$	$\lvert w \rvert \leq 7.875$	$\lvert w \rvert \leq 3.9375$	$\lvert w \rvert \leq 1.96875$
$\exp(w)$	$(0,\ 4.78\times10^{13})$	$(0.003,\ 2630)$	$(0.019,\ 51.29)$	$(0.13,\ 7.161)$

Specifically, we propose an FFT-based BP algorithm with a threshold factor to improve the performance of non-binary LDPC decoders. Implementing the EXP and LOG function as small LUTs leads us to consider the dynamic range of the nonlinear functions in order to take more precisely into account the effect of the finite precision on the internal data.

Let (t, f) be the quantization scheme, where t and f are the total bit size and the number of fractional bits, respectively. For example, the intrinsic message $L_{\text{channel}}(c_n = a)$, which is the input of the decoder, can be quantized to $(t, f) = (7, 1)$, where the quantization steps are uniform. The (t, f) quantization scheme has a quantization precision of 2^{-f} with a maximum value of $2^{t-f-1} - 2^{-f}$. For the EXP blocks at stages 1 and 3, the quantized samples Q_{mn} are normalized to the scaled version of Q_{mn}, and therefore, the appropriate quantization scheme is required to cover the full-scale range of the EXP function. Table 5.1 shows the dynamic range of the EXP function corresponding to various quantization schemes. Note that a finer quantization step gives a narrower dynamic range of the EXP function output. This shows that propagation of the quantization error will be increased significantly compared to a coarser quantization, thereby degrading the decoding performance.

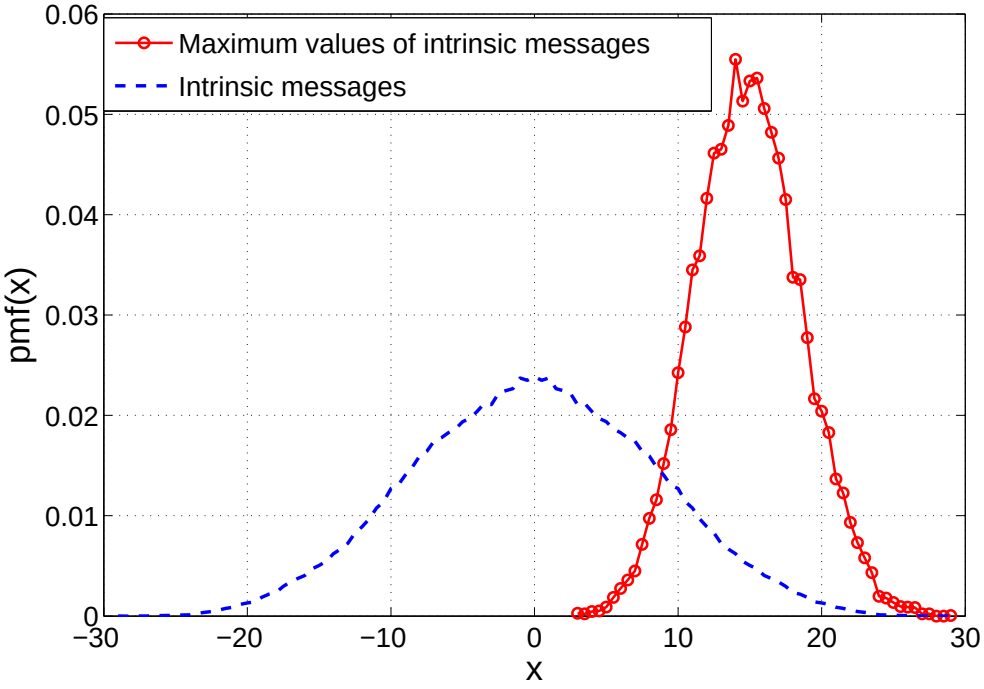


Fig. 5.2: Probability mass function of the maximum values for intrinsic messages when SNR = 4.0 and (7, 1) quantization.

Thus, maintaining the relative magnitudes of the message information in a given quantization, the output value z of the LUTs for the EXP functions at stages 1 and 3 can be pre-computed as follows:

$$z = \frac{z_{\max}}{\exp\left(w_{\max}\right)} \times \exp\left(w\right), \tag{5.6}$$

where $z_{\max}$ is the maximum value in a given quantization (at stages 2 and 4) and $w \in \Omega$, in which $\Omega = \{-w_{\max}, -w_{\max} + w_{\text{step}}, -w_{\max} + 2w_{\text{step}}, \ldots, w_{\max} - w_{\text{step}}, w_{\max}\}$, where $w_{\max}$ and w_{step} indicate the maximum value and the step size, respectively, of the quantized samples.

Fig. 5.2 shows the probability mass function (pmf) of the maximum values obtained when the (7, 1) quantization is used for the intrinsic message $\mathbf{L}_{\text{channel}}(c_n = a)$ at an SNR = 4.0. It is clear that

we need to provide some modifications in the check node update in order to propagate the relative strengths of the intrinsic messages without increasing the dynamic range of the extrinsic messages. It is necessary to use a more reliable value (e.g. larger magnitude) of each message during the exponential transformation. To do so, we use a threshold w_{th}, where w_{th} denotes the minimum of the maximum values of all the messages and is determined by a large number of samples. We denote the original and modified input messages of the EXP blocks as $U_{mn}(a)$ and $W_{mn}(a)$, respectively. Therefore, the updating operation with a threshold factor w_{th} is given by following offset –based scheme:

$$W_{mn}(a) = \begin{cases} U_{mn}(a) - \left(U_{mn}^{\max} - w_{th}\right), & U_{mn}^{\max} > w_{th} \\ U_{mn}(a), & \text{otherwise,} \end{cases} \tag{5.7}$$

where :

$$U_{mn}^{\max} = \max_{a \in GF(q)} U_{mn}(a) \tag{5.8}$$

denotes the maximum value over $U_{mn}(a)$ messages, and $U_{mn}(a) \in \Omega$. It is intuitive to propagate larger magnitude messages while message information in the low-value (i.e., less reliable) region is clipped to the lowest value. Using this offset approach, internal messages represented in the logarithm domain can be properly transformed into corresponding messages in the real domain. From (5.7) and (5.8), the size of LUTs for the EXP blocks is reduced from 2^t (number of input points) × t (word length) to only $l \times t$ bits, in which l represents the reduced number of input points determined by w_{th}. For example, in the case of (7, 1) quantization each LUT consists of 128×7 bits while each LUT with the proposed offset–based scheme using w_{th} = 3.5 consists of 9×7 bits (a 93% reduction). Based on the threshold w_{th}, the message distribution before the LOG blocks has a reduced dynamic compared to a standard FFT-based BP decoding algorithm. Therefore, the dynamic range determined by w_{th} scales the output of LOG blocks to be distributed within a given

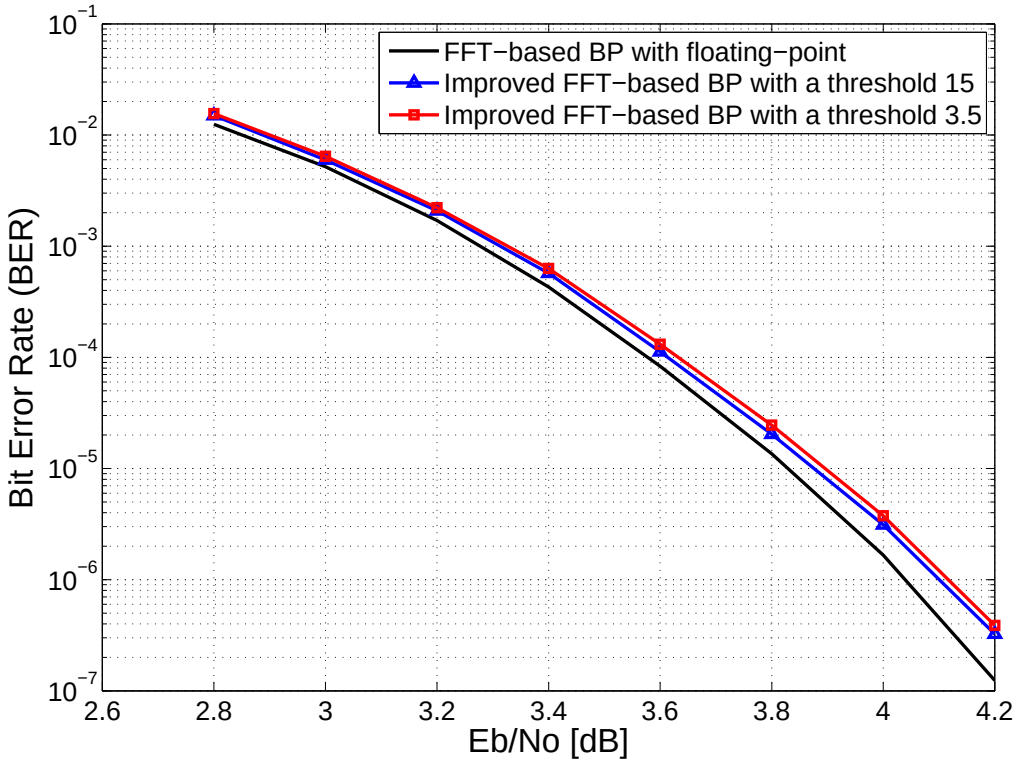


Fig. 5.3: Performance comparison of the FFT-based BP and the proposed method with the (7, 1) quantization (Maximum number of iterations = 50).

quantization.

5.4 Simulation Results

We use a nonbinary quasi-cyclic (QC) LDPC code which is constructed using the array dispersions method of [60]. We consider an (N, K) = (225, 165) QC LDPC code over GF(2^4), where N and K are the block length and the number of information symbols in the code, respectively.

Fig. 5.3 illustrates the simulated decoding performance, where the improved FFT-based BP decoding algorithm with the threshold w_{th} exhibits a negligible performance degradation of less

than 0.1 dB compared with a floating-point simulation of the conventional FFT-based decoding algorithm. The size of the LUTs can be reduced without degrading the performance of the decoder by setting w_{th} to be the minimum of the maximum values of all of the messages. Note that an extremely small w_{th} results in near-zero-forcing of all messages and no actual information is preserved.

5.5 Conclusion

In this chapter, we have investigated the quantization effects on decoding performance of a nonbinary (N, K) = (225, 165) QC LDPC code over GF(2^4). In particular, quantization effects of FFT-based decoding algorithm using nonlinear functions such as exponential and logarithm functions are considered. We have proposed an improved FFT-based BP decoding algorithm having a threshold factor to utilize the relative magnitude of the quantized data from these nonlinear functions. The proposed offset-based scheme reduces the size of the LUTs used in the check node updating without sacrificing the decoding performance.

Chapter 6

Efficient FFT-Based BP Decoder Architecture for Nonbinary LDPC Codes

6.1 Introduction

Long-length binary LDPC codes are known to have excellent performance for high-speed data transmission. However, for moderate or short code length, binary LDPC codes have been shown to have an early error floor and degraded decoding performance. In addition, it has been shown that binary LDPC codes have degraded decoding performance when burst errors occur in the channel [56]. To overcome these disadvantages of binary LDPC codes, Davey and MacKay [12][67][68] introduced a generalization of the belief propagation (BP) algorithm [10] by using a forward-backward procedure for decoding nonbinary LDPC codes [49].

In decoding nonbinary LDPC codes, there are a variety of available algorithms, such as BP using a forward-backward algorithm [49] or BP using two-point fast Fourier transforms (FFT) in the logarithm (log) domain [56][58], log-BP using a Jacobi logarithm [61], max-log-BP [61], which is a simplified version of log-BP using a simple $max(x, y)$ function, an extended min-sum (EMS) algorithm [51][62] and a selective min-max algorithm [63]. Among these various decoding

algorithms, the EMS and selective min-max decoding algorithms can reduce the hardware complexity compared to that of conventional BP algorithms at a cost of performance degradation, since complex computations at the check nodes can be implemented with simple summation and comparison operations. Recently, an FFT-based BP decoder based on the mixed (logarithm and probability) domain has been proposed to remove the complicated multiplications in the decoder [65]. However, it requires many look-up tables (LUTs) for implementing the nonlinear exponential and logarithm functions and its computational load is unbalanced between the processing units in the decoder.

Compared to finite precision effects in binary BP decoding algorithms, the finite precision effects in nonbinary LDPC decoding algorithms have been less extensively studied. Therefore, to achieve a practical implementation, we consider these finite precision effects, particularly for the FFT-based BP algorithm in the mixed domain. In this chapter, we show an improved decoding performance by using an offset-based scheme and appropriate scaling techniques in the main processing units. In addition, we propose novel FFT-based BP decoder architectures which balance the computational load between the main processing units.

The remainder of this chapter is organized as follows. In Section 6.2, we briefly describe the FFT-Based BP algorithms for nonbinary LDPC codes. In Section 6.3, we investigate quantization effects in the FFT-Based BP algorithm. We then propose the offset-based scheme and appropriate scaling techniques to achieve improved decoding performance for the FFT-based BP decoder. Implementation of the proposed decoder on a Xilinx Virtex-4 xc4vlx200 FPGA device is given in Section 6.4. Finally, our conclusions are presented in Section 6.5.

6.2 BP Algorithm for Nonbinary LDPC Codes

In a nonbinary LDPC code, all elements in the parity-check matrix are elements in a Galois Field (GF) of order q, which is denoted as a GF(q). Here, q denotes the order of the GF, and we restrict our discussion to $q = 2^p$, which is an extension field of GF(2). In particular, an LDPC code over GF($q = 2^p$) groups p bits into an element of this field. Therefore, the nonbinary LDPC code is a code defined by a parity check matrix in which most elements have a value of 0, and the remaining elements are nonzero elements of the field.

Let $\mathbf{C} = [c_1\ c_2\ \ldots\ c_N]^T$ denote the transmitted codeword, where c_n corresponds to a symbol defined over GF(2^p), for $1 \le n \le N$. Therefore, a codeword of the nonbinary LDPC code, $\mathbf{C}$, is a vector having a length of N of elements of GF(2^p), and satisfies the following parity-check equation:

$$\sum_{n=1}^{N} h_{mn} \otimes c_n = 0,\ \forall m \in \{1, 2, ..., M\} \tag{6.1}$$

where h_{mn} is an element of the parity-check matrix $\mathbf{H}$ of the nonbinary LDPC code and M (N) indicates the number of rows (columns) of the parity-check matrix $\mathbf{H}$. Note that additive and multiplicative ($\otimes$) operations in (6.1) are defined over GF($q = 2^p$). The main decoding problem is to decide the most probable vector $\mathbf{L}$ such that $\mathbf{H} \otimes \mathbf{L} = 0$, where $\mathbf{L} = [L_1\ L_2\ \ldots\ L_N]^T$ is the received vector through a channel.

The BP algorithms for nonbinary LDPC codes can be described in two message stages, i.e., check-to-variable message R_{mn} and variable-to-check message Q_{mn}. Let us consider an example of a nonbinary decoding algorithm such as the FFT-based BP algorithm in the probability domain (see [54] for more details). The check node updating (CNU) unit and the variable node updating (VNU) unit are illustrated in Fig 6.1. They consists of two multiplications ($\prod$), one permutation

(P), one inverse permutation (IP), an FFT and an inverse FFT (IFFT). The main disadvantages of FFT-based BP in the probability domain is that it needs a normalization constant for each R_{mn}, Q_{mn} and posterior (decoded soft output) message, as well as multiplication operations in the CNU and VNU units. To reduce hardware complexity, the conventional FFT-based BP algorithm in the mixed domain has been proposed using nonlinear functions [65].

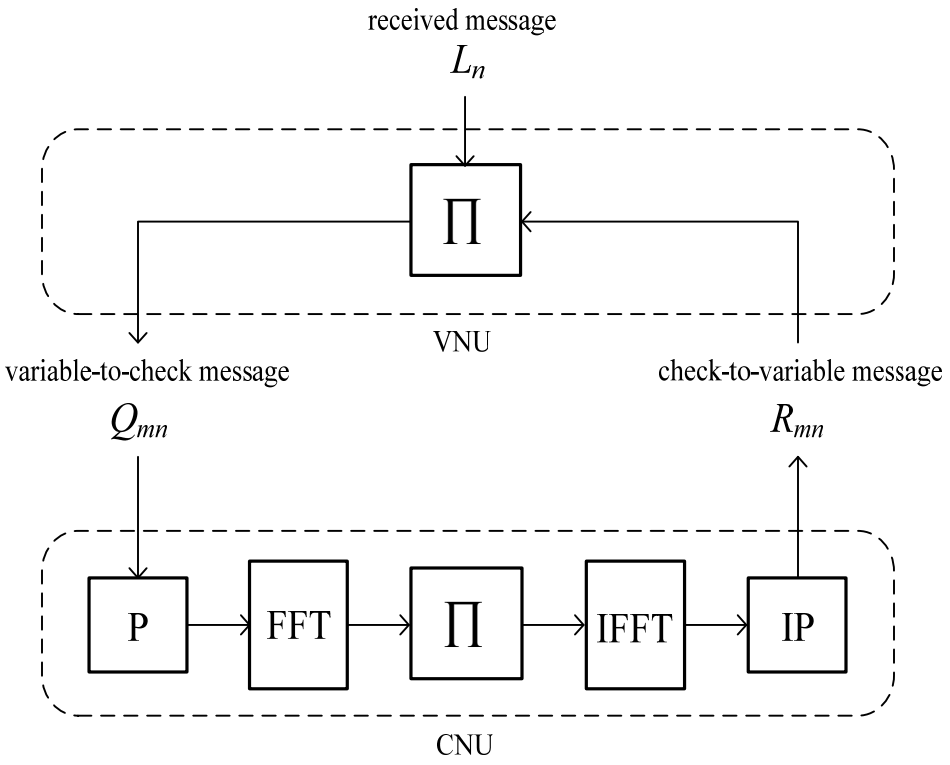


Fig. 6.1: Simplified diagram of a check node updating (CNU) unit and a variable node updating (VNU) unit.

6.3 Finite Word-Length Implementation of FFT-Based BP

Due to the high hardware complexity of nonbinary LDPC decoding algorithms, we must analyze the finite word-length effects on the performance of a nonbinary LDPC decoder. The finite word-length effects needs to be considered for the intrinsic information and the extrinsic information. In

terms of the hardware complexity, the finite word-length of intrinsic and extrinsic messages directly determines the memory sizes and ultimately determines the overall implementation complexity of a nonbinary LDPC decoder. In [61], the quantization effects of the intrinsic (received) data and the extrinsic data for a sum product algorithm using the max*-operation, where $\max^*(\alpha, \beta) \equiv \ln(\exp(\alpha) + \exp(\beta))$, in the logarithm domain were investigated. In [65], the finite word-length effects of the FFT-Based BP algorithm having the nonlinear functions of logarithm (LOG) and exponential (EXP) were presented. In that reference, it is mentioned that an 8-bit quantization scheme represents a good tradeoff between hardware complexity and decoding performance for a code length $N = 720$ with a nonbinary LDPC code over GF(2^3). However, the decoding behavior of the FFT-based BP algorithm with finite word-lengths was not investigated at various points such as in the LOG and EXP functions in the check node update. Consequently, no information about the dynamic range or scaling (quantization) effects at those points is available. In other words, an analysis of the LOG and EXP blocks in the FFT-based BP decoder has not been previously studied. In this section, we present an improved quantization scheme to achieve better decoding performance by considering the dynamic range of intrinsic and extrinsic values as well as efficient scaling operations in an FFT-based BP decoder.

6.3.1 Quantization Procedure

An LDPC decoder implementation of FFT-Based BP algorithms approximates the ideal operation of the iterative message-passing algorithm. This approximation is necessary for the following reasons: 1) we need to restrict the word-length used for representing messages in the decoder, which leads to the quantization effects in the decoding performance and 2) the number of iterations are usually limited to a predetermined maximum value.

We first consider the quantization of intrinsic (received) information in the FFT-based BP decoding algorithm over GF(2^4). The appropriate word-length in the decoder represents a tradeoff

between decoding performance and hardware complexity. This decision also affects the memory sizes for extrinsic as well as intrinsic data. In addition, we impose a maximum of 8 iterations on the FFT-based BP decoding algorithm in order to achieve a high-throughput decoder.

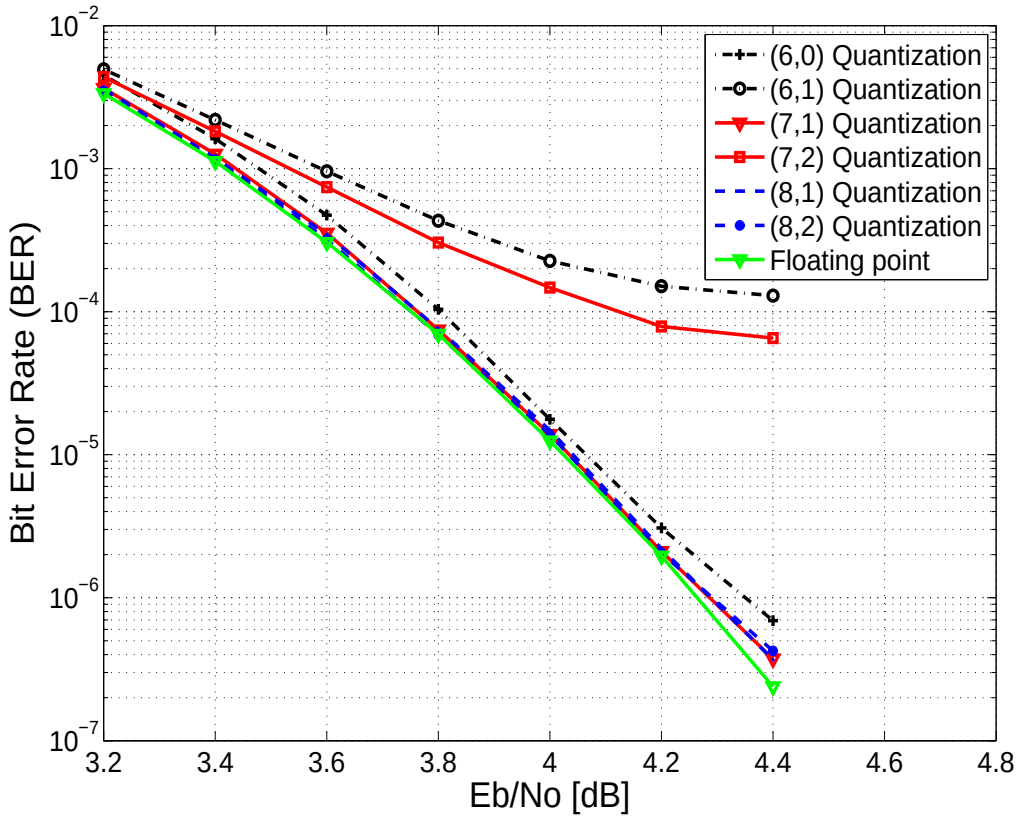


Fig. 6.2: Performance comparison of the FFT-Based BP for various quantization schemes of the received data. (Maximum number of iterations = 8)

Let (t, f) be the quantization scheme, where t and f are the total bit size and the number of fractional bits, respectively. For the quantization of the intrinsic information, we consider t = 6, 7 and 8 bits in the FFT-based BP decoding algorithm for an N =225 (data length K = 165), rate-11/15 nonbinary quasi-cyclic (QC) LDPC code [60] over GF(2^4). Various quantization schemes for the intrinsic data, i.e., (6, 0), (6, 1), (7, 1), (7, 2), (8, 1) and (8, 2), have been simulated and analyzed. Note that here extrinsic messages are computed with single precision floating point. The

simulation results are shown in Fig. 6.2. Note that both (6, 1) and (7, 2) quantization schemes, both of which use 5-bits for the integer part of the intrinsic data, suffer from serious performance degradation and early error floors. Thus, the dynamic ranges of those two schemes are insufficient to cover the message values at a high signal-to-noise ratio (SNR). The other quantization schemes, i.e. (7, 1), (8, 1) and (8, 2), have better performance. Based on these simulation results, we can see that the (7, 1) quantization scheme for intrinsic data is the best choice considering the tradeoff between hardware complexity and decoding performance.

Using the (7, 1) quantization for the intrinsic messages, we can further analyze the finite word-length effects on the extrinsic messages passing through the nonlinear (i.e., exponential and natural logarithm) functions. In the check node updating unit, intermediate extrinsic values are outputs of exponential (EXP) functions, which can be implemented as LUTs. From the EXP function graph, as shown in Fig. 6.3, it is observed that the output messages of the EXP function may become saturated in the (7, 1) quantization. Thus, the characteristics of this function can not be fully captured, since the updated extrinsic messages are truncated at values of –32 or 31.5. In chapter 5, we showed that a smaller quantization step at the input of the EXP function gives a narrower dynamic range of the EXP function output (see Section 5.3 for several quantization schemes with a larger value *f*). This result shows that the propagation of the quantization errors will be increased significantly, thereby degrading the decoding performance. In hardware implementations, a smaller quantization step can be simply realized by arithmetic or logical right shift operations.

To prevent the propagation of errors due to relatively decreased magnitudes of the messages at the output of the EXP function, the scaled EXP function in Fig. 6.3 can be used with shift operations at the input of the EXP function. In this case, the output values of the EXP are distributed within a given quantization, but the decoding performance is still degraded due to loss

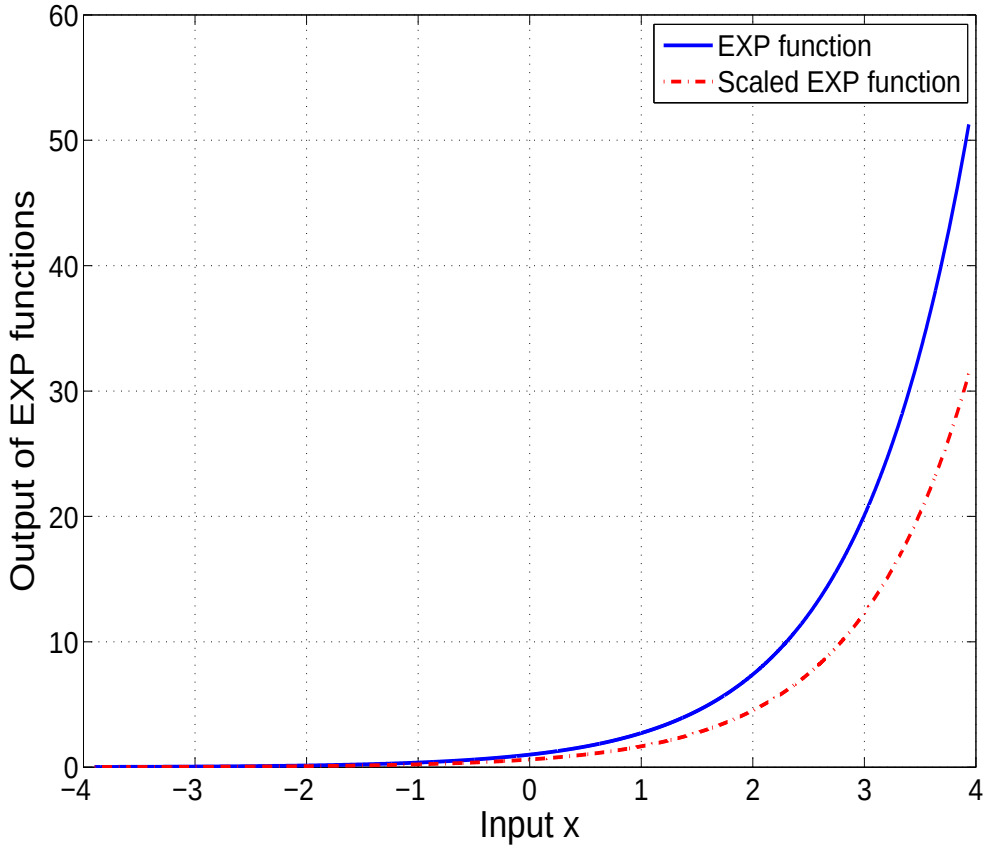


Fig. 6.3: The exponential (EXP) functions.

of precision for the input values.

From the above observation, scaling the EXP function or message values at the input of EXP function leads to the performance loss. In [69], we considered this problem and analyzed the quantization effects of the FFT-based BP algorithm based on the EXP and LOG functions for nonbinary LDPC codes. In Chapter 5, it was shown that an offset-based approach utilizes the relative magnitudes of the quantized data to reduce the dynamic range under a given quantization scheme without increasing a larger number of quantization bits. The performance loss between the

offset-based approach with the (7, 1) quantization scheme and the floating-point simulation is no more than 0.1 dB.

6.3.2 Finite Precision Analysis

In this section, we analyze the quantization effects of the extrinsic messages of an FFT-based BP decoder. Based on the simulation results in section 6.3.1, we use the (7, 1) quantization scheme for the received data and consider the dynamic range of the intermediate extrinsic messages in order to take more precisely into account the effect of finite precision on the EXP and LOG functions in the check node updating units.

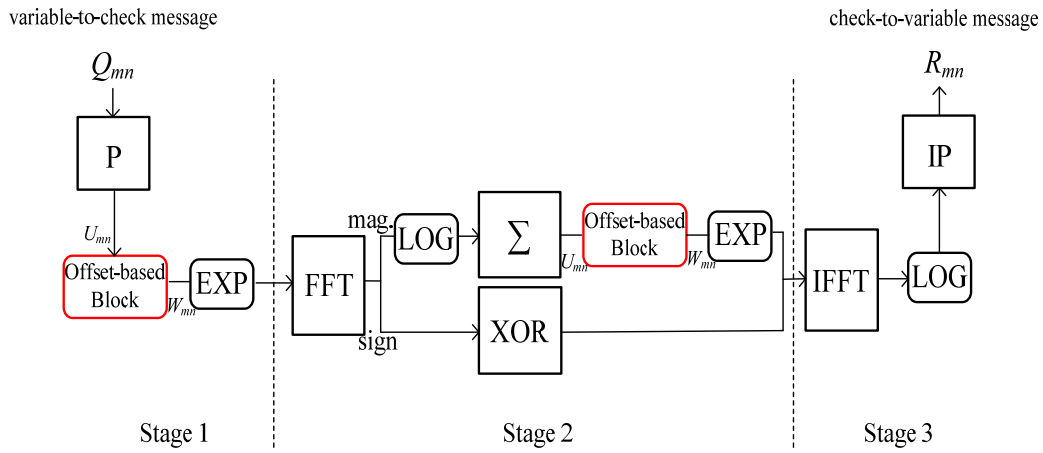


Fig. 6.4: Dataflow of a check node updating (CNU) unit.

The dataflow in a check node updating unit is shown in Fig. 6.4. It consists of two nonlinear functions, exponential (EXP) and natural logarithm (LOG), a permutation (P), an inverse permutation (IP), one summation (∑), exclusive-or (XOR) gates, two offset-based blocks, FFT and IFFT blocks. In Fig. 6.4, the offset-based blocks at stages 1 and 2, which are used with a threshold factor w_{th} in [69], can be expressed as:

$$W_{mn}(a) = \begin{cases} U_{mn}(a) - \left(U_{mn}^{\max} - w_{th}\right), & U_{mn}^{\max} > w_{th} \\ U_{mn}(a), & \text{otherwise,} \end{cases} \tag{6.3}$$

where:

$$U_{mn}^{\max} = \max_{a \in GF(q)} U_{mn}(a) \tag{6.4}$$

denotes the maximum value over $U_{mn}(a)$ messages. The threshold w_{th} indicates the minimum of the maximum values of all the extrinsic messages.

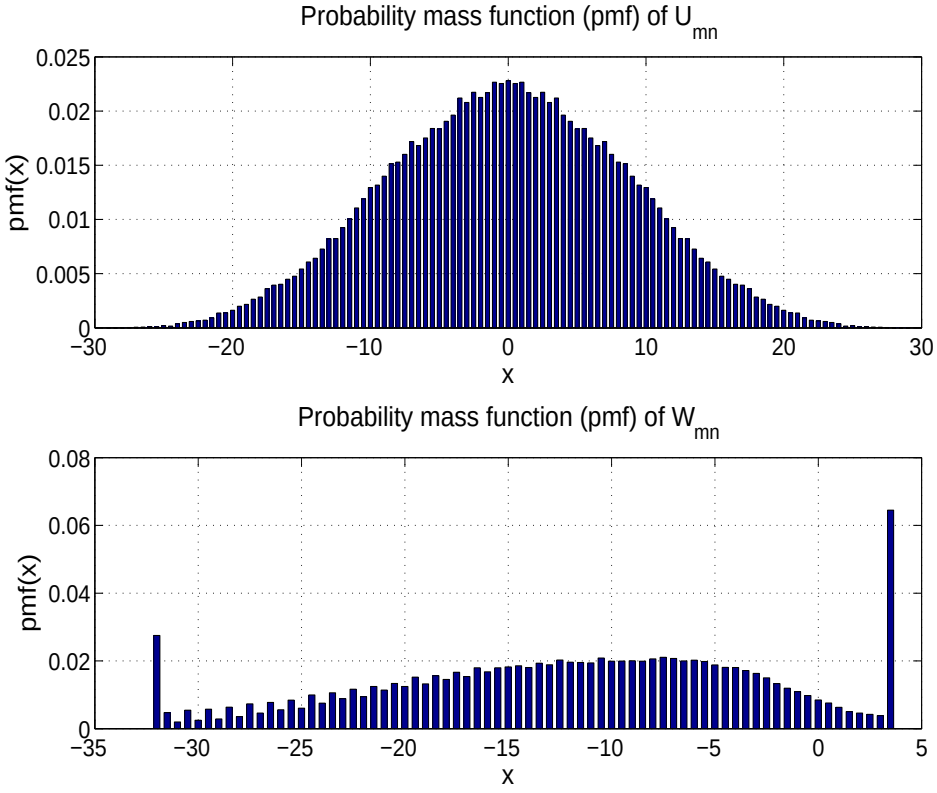


Fig. 6.5: Probability mass function of the extrinsic messages U_{mn} and W_{mn} at stage 1 when SNR = 4.2 and the (7, 1) quantization are used. Note that W_{mn} is simulated with threshold w_{th} = 3.5.

Fig. 6.5 shows the probability mass function (pmf) of the intermediate (extrinsic) messages U_{mn} and W_{mn} at stage 1 in Fig. 6.4 when the SNR = 4.2 and when the (7, 1) quantization scheme is

used. Note that the distribution of the U_{mn} messages is equal to the input distribution of the EXP block for a conventional FFT-Based BP. From the simulation results, the dynamic range can be reduced to the following range of values: [the minimum value in a given quantization, w_{th}]. We can consider U_{mn} and W_{mn} as the original and modified input messages of the EXP blocks, respectively. The size of the LUTs used for the EXP blocks can be reduced because messages with $x < 0$ in the EXP blocks do not affect the decoding performance. In other words, only using messages with $x \geq 0$ for the EXP blocks at stages 1 and 3 is sufficient to propagate the values of the messages without sacrificing the decoding performance.

Table 6.1: Look-up table (LUT) for EXP blocks using offset-based scheme with $w_{th} = 3.5$

LUT input for (7, 1) quantization with offset-based scheme	LUT output (7, 1) quantization	
Decimal value	Decimal value	Binary value
3.5 (w_{th})	31.5	0111111
3.0	19.0	0101000
2.5	11.5	0011000
2.0	7.5	0001111
1.5	4.5	0001001
1.0	2.5	0000101
0.5	1.5	0000011
0.0	1.0	0000010
− 0.5	0.0	0000000
− 1.0		
− 1.5		
…		
− 31.5	0.0	0000000
− 32.0		

For example, Table 6.1 presents the number of LUT entries used for the offset-based scheme. If the (7, 1) quantization scheme is used for the EXP function, each LUT in the conventional design [65] is equivalent to 128 entries of 7-bit data (i.e., 128 $\times$7 bits). However, using the offset-based scheme (with w_{th} = 3.5), each LUT consists of 9 entries of 7-bit data (i.e., 9$\times$7 bits). This results in a 93% reduction in the number of LUT entries with negligible performance degradation at a high SNR level, which will be further discussed in Section 6.4.

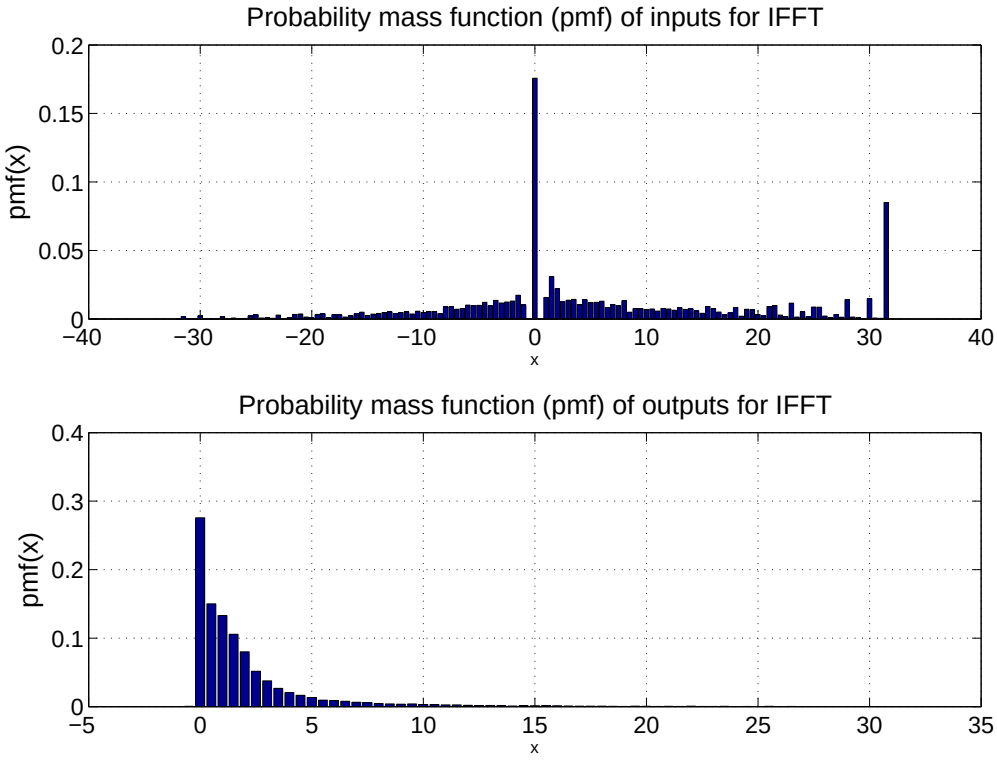


Fig. 6.6: Probability mass function of inputs and outputs for the IFFT at stage 3 in the (7, 1) quantization scheme.

The FFT and IFFT blocks in the CNU unit can be implemented using the Hadamard transform. The FFT/IFFT operation can be expressed in terms of its radix-2 butterfly diagram using only addition and subtraction operations. Scaling operations are required in the FFT/IFFT blocks to

reduce the dynamic range of extrinsic messages and the resulting hardware complexity. Fig. 6.6 shows the probability mass function of the input and output values when scaling operations are used in the IFFT block at stage 3. In general, the required number of bit positions to shift by in order to achieve the scaling operation depends on the magnitudes of intermediate messages, the size of FFT/IFFT blocks, and the SNR value. For implementation simplicity, we apply a fixed scaling operation for the FFT and IFFT blocks of 2^{-1} and 2^{-2}, respectively. From simulation results, a more reliable value of the extrinsic messages at the output of FFT and IFFT is distributed over large values of x (i.e., $x \geq 1$ in the (7, 1) quantization). Therefore, we can reduce the size of the LUTs for the LOG blocks with a negligible performance degradation compared to a floating-point computation of the FFT-Based BP decoding algorithm. To perform a simple comparison of the hardware complexity of the LUTs used for the LOG blocks, we use a uniform quantization method for these blocks. The estimated number of bits for each LUT using the proposed methods is 61 entries of 7-bit data (i.e., 61×7 bits). Note that the number of LUT entries for the LOG is considered for $x \geq 1$ in the (7, 1) quantization. This result shows a 52% reduction in the number of LUT by using the proposed offset-based scheme.

6.4 Low Complexity Architecture for Quantized FFT-Based BP Decoding

In the previous section, we gave a behavioral analysis of the FFT-based BP algorithm using both the offset-based scheme for EXP and the scaling operation for LOG. In this section, we present an improvement of the FFT-based BP decoding algorithm and a novel FFT-based BP decoder architecture with the reduced sizes of LUTs.

6.4.1 FFT-Based BP Decoding Performance

The decoding performance of the FFT-based BP algorithm under the offset-based scheme and with scaling operations in a fixed-point environment is discussed. The performance of the (N, K) = (225, 165) QC LDPC code over GF(2^4), where N and K are the block length and the number of information symbols in the code, respectively, with floating point and with our proposed quantization method of Section 6.3, are shown in Fig. 6.7.

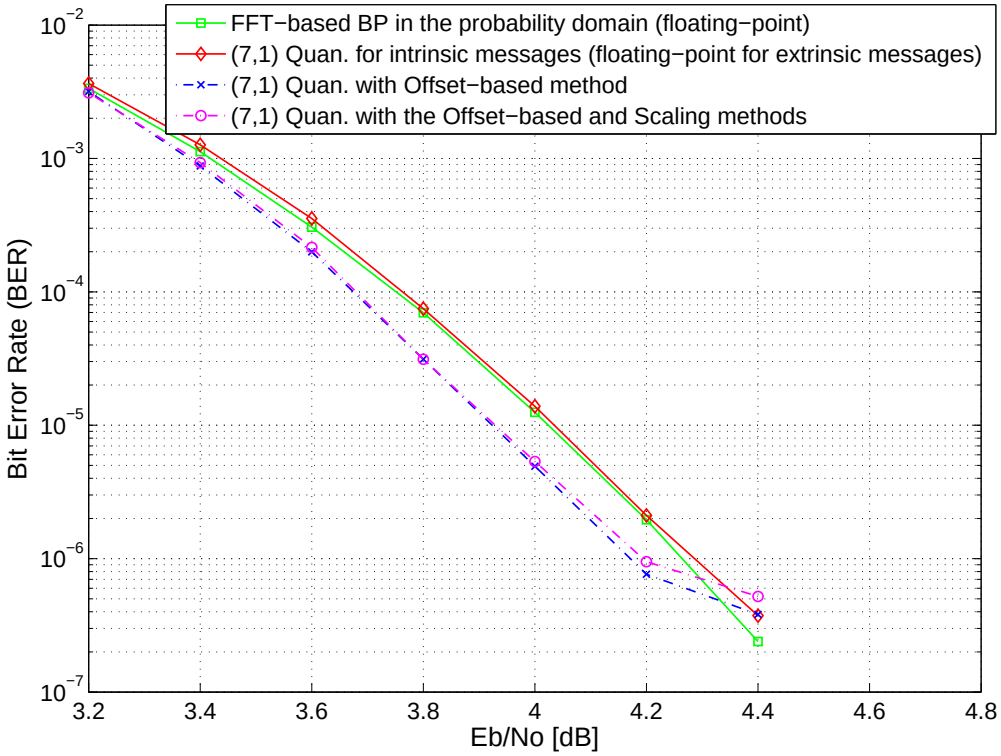


Fig. 6.7: Performance comparison of the conventional FFT-Base BP with the proposed methods (Threshold w_{th} = 3.5).

In Fig. 6.7, the performance of FFT-based BP in the probability domain with single precision floating point computation exhibits a performance degradation compared with a fixed-point simulation of the FFT-based BP using the proposed schemes in the mixed (i.e., logarithm and

probability) domains. In this case, the decoders using BP computations in the probability domain are very sensitive to the precision loss due to the normalization factors in both CNUs and VNUs. The FFT-based BP algorithm in the mixed domain [65] with the (7, 1) quantization scheme used only for intrinsic messages achieves almost the same performance, and its performance loss is negligible when compared with a floating-point simulation of the FFT-based decoding algorithm in the probability domain. In this case, the output values of the EXP blocks are increased and those values might propagate to LOG blocks in the CNU units. Therefore, a large portion of large message values have no effect on message values at the output of LOG blocks in a floating point simulation. Note that extrinsic messages for this case are computed as floating point numbers. Our proposed FFT-Based BP with the offset-based scheme for EXP blocks in the (7, 1) quantization provides much better BER performance than the conventional FFT-based BP [65]. However, we have seen the error floor at a high SNR of 4.4 because we fix the threshold $w_{th} = 3.5$ independent of the SNR level. In the previous chapter, we considered the decoding performance by setting the proper w_{th} to be the minimum of the maximum values of all of the messages. From the simulation results of Chapter 5, one expects to achieve performance gains using a larger w_{th} value at a high SNR level.

Moreover, FFT-based BP with the offset-based scheme and the scaling operation for EXP/LOG blocks exhibits a negligible performance degradation of less than 0.1 dB compared with the (7, 1) quantization with the offset-based method. From the simulation results in Fig. 6.6, it is shown that the FFT-based BP with the offset-based scheme and scaling operation provides significant performance gains over previous designs while reducing the sizes of LUTs.

6.4.2 Efficient FFT-Based BP Decoder

In this subsection, a novel FFT-based BP decoder architecture with reduced LUT sizes is discussed. Moreover, we present a modified decoding algorithm to balance the computational load between CNUs and VNUs compared to that of a standard FFT-based BP algorithm. In [65], CNU memory blocks and VNU memory blocks are inserted to store R_{mn} and Q_{mn} messages, as shown in Fig. 6.4. As a result, the computational load between a pair of CNU and VNU units is unbalanced. To avoid the complexity difference between these two units, we need to reformulate the computations.

Let $\mathbf{Q} = (Q_{mn})$ and $\mathbf{R} = (R_{mn})$, for $(m,n) \in \{(i,j) \mid h_{ij} \neq 0\}$, denote the variable-to-check messages and the check-to-variable messages, respectively. We denote two shifting values for the permutation (P) block and the inverse permutation (IP) block as $\mathbf{P} = (P_{mn})$ and $\mathbf{P^{-1}} = (P_{mn}^{-1})$, respectively, for $(m,n) \in \{(i,j) \mid h_{ij} \neq 0\}$. Let $\mathbf{L} = (L_n)$ and $\mathbf{Q} = (Q_n)$, for $n = 1, \ldots, N$, be the received and the decoded soft output messages, respectively. Note that $M(n) = \{m \mid h_{mn} \neq 0\}$ denotes the set of check nodes connected to variable node n, and $N(m) = \{n \mid h_{mn} \neq 0\}$ denotes the set of variable nodes connected to check node m. The modified FFT-based BP algorithm can be described as follows:

Modified FFT-based BP Algorithm

For each $(m,n) \in \{(i,j) \mid h_{ij} \neq 0\}$ and $a \in \mathrm{GF}(q)$

1) Initialization:

$$R_{mn}(a) = 0$$

2) Variable node unit (VNU) updating:

$$A_{mn} = P_{mn}^{-1}\left(R_{mn}\right)$$
$$B_{mn}(a) = L_n(a) + \sum_{i \in M(n),\, i \neq m} A_{in}(a)$$
$$Q_n = \underset{a}{\operatorname{argmax}} \left\{ L_n(a) + \sum_{i \in M(n)} A_{in}(a) \right\}, \text{ for } n = 1, \ldots, N$$
$$Q_{mn} = P_{mn}\left(B_{mn}\right)$$

where $P_{mn}(\bullet)$ is a permutation shifting P_{mn} and $P_{mn}^{-1}(\bullet)$ is the corresponding inverse permutation. If **HQ** = 0, then the iterative decoding is terminated as a valid codeword has been found.

3) Check node unit (CNU) updating:

$$C_{mn} = \mathrm{FHT}\left(Q_{mn}\right)$$
$$D_{mn}(a) = \sum_{j \in N(m),\, j \neq n} C_{mj}(a)$$
$$R_{mn} = \mathrm{IFHT}\left(D_{mn}\right)$$

where $\mathrm{FHT}(\bullet)$ is a fast Hadamard transform and $\mathrm{IFHT}(\bullet)$ is the corresponding inverse Hadamard transform.

Figs. 6.8 and 6.9 show the structure of the CNU and VNU units, respectively, based on the modified FFT-based BP algorithm. For simplicity, exclusive-or (XOR) gates and offset-based blocks are not shown in the proposed CNU or VNU unit. Note that the FHT and IFHT blocks can be constructed with simple addition and subtraction operations. In Fig. 6.9, the Max block selects a symbol $a \in \mathrm{GF}(q)$ from among the decoded soft output values Q_n. We give the estimated total number of hardware resources required for the conventional decoder [65] and for our proposed decoder in Table 6.2. Let d_c and d_v denote the check node degree and variable degree, respectively. In this case, d_c is equivalent to 6 and d_v is equivalent to 3. Note that the Adder tree (in the CNU and VNU units) and the Max block (in the VNU unit) are not considered in the table because the hardware resources are the same for those two designs. It is easily seen that the computational load in the proposed architecture is well balanced between the CNU and VNU units.

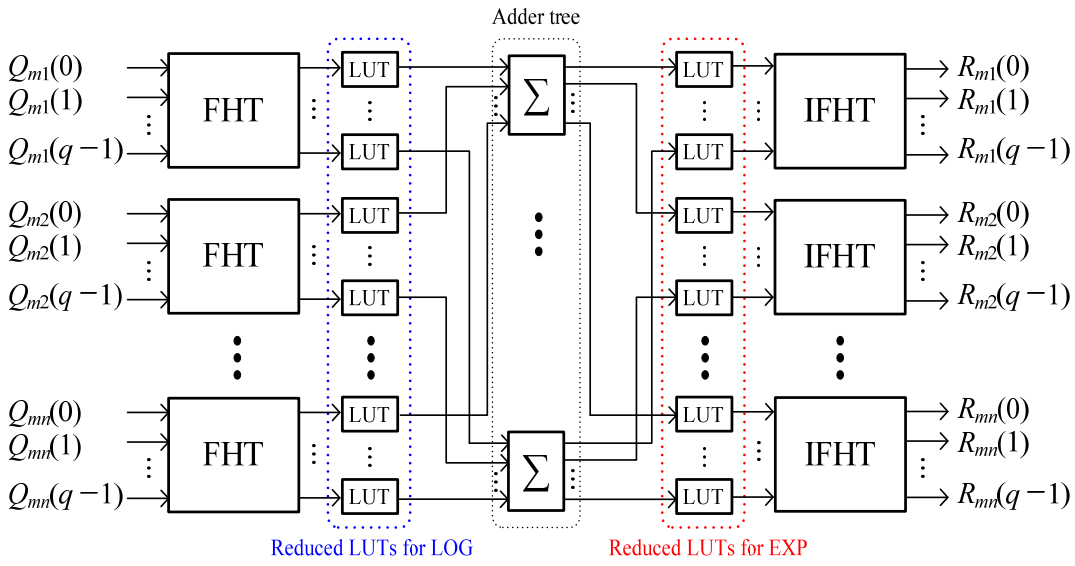


Fig. 6.8: Proposed architecture for check node updating (CNU) unit.

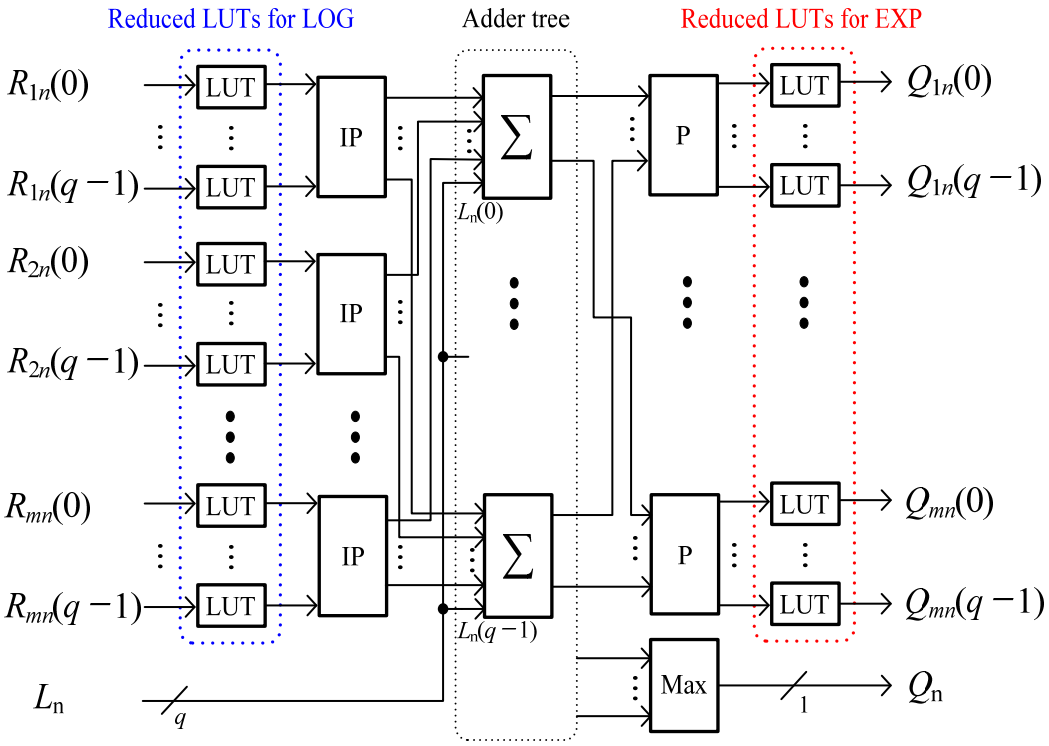


Fig. 6.9: Proposed architecture for variable node updating (VNU) unit.

Table 6.2: Estimated Key Hardware Resources of FFT-based BP Decoders over GF(q = 16)

	Conventional decoder (d_c = 6 and d_v = 3)		Proposed decoder (d_c = 6 and d_v = 3)	
	CNU unit	VNU unit	CNU unit	VNU unit
LUT (bits)	$4\times d_c\times q\times(128\times 7)$ = 344,064	-	$d_c\times q\times(9\times 7)+$ $d_c\times q\times(61\times 7)$ = 47,040	$d_v\times q\times(9\times 7)+$ $d_v\times q\times(61\times 7)$ = 23,520
	344,064		70,560	
q-input Permutation (P and IP)	$2\times d_c$	-	-	$2\times d_v$
	12		6	
2-to-1 Comparator in Offset-based block	-	-	$d_c\times(q-1)=90$	$d_v\times(q-1)=45$
	-		135	

Table 6.3: Xilinx Virtex4 Xc4vlx200 FPGA Synthesis Results

Resource	Conventional decoder (d_c = 6 and d_v = 3)		Proposed decoder (d_c = 6 and d_v = 3)	
	CNU unit	VNU unit	CNU unit	VNU unit
Slices	6,679	192	2,687	937
	6,871 (100%)		3,624 (53%)	
4 input LUTS	12,620	304	4,704	1,718
	12,924 (100%)		6,422 (50%)	
Delay (*nsec*)	40.67	9.38	42.29	14.07

To perform a fair comparison of the hardware complexity of the conventional and the proposed design, we implemented the CNU and VNU units on a Xilinx Virtex-4 xc4vlx200 FPGA. The synthesis results for both designs are given in Table 6.3. The proposed decoder, using the offset-based scheme and the scaling operation, results in a 53 % reduction in the number of slices with a negligible degradation in the decoding performance at high SNR. It is observed that the critical path of each CNU and VNU unit of the proposed decoder is slightly longer than that of the conventional one because the offset-based block for implementing the equation (6.3) is required.

6.5 Conclusion

In this chapter, we have investigated the quantization effects on the decoding performance of the FFT-based BP algorithm in the mixed domain for nonbinary LDPC codes. Then, we have proposed an offset-based scheme and appropriate scaling operation for the FFT-based BP

algorithm having the reduced sizes of the LUTs for the nonlinear functions. Simulation results show that the proposed offset-based and scaling techniques with a fixed w_{th} achieve much better performance than the conventional FFT-based BP algorithm.

Moreover, we have presented an efficient architecture for FFT-based BP decoding by using the proposed offset-based and scaling schemes and by balancing the computational load between CNU and VNU units. The results show a 53 % reduction in the number of required FPGA slices and a 50 % reduction in the number of required FPGA 4-input LUTs compared to a standard FFT-based BP architecture.

Chapter 7

Conclusions and Future Work

In this dissertation we studied efficient algorithm and architecture aspects for binary and nonbinary LDPC codes. Our main focus was to improve the performance of LDPC decoders by investigating quantization effects in several decoding algorithms and to achieve reduced-complexity LDPC decoder architectures without significant performance degradation.

We presented adaptive quantization schemes in the normalized min-sum decoding algorithm. We considered scaling effects in a fixed-point environment and extensively investigated the finite precision effects on the performance of an (N, K) = (1920, 1280) irregular LDPC code. The proposed adaptive quantization scheme achieves the optimal performance in selecting suitable LLR input values to the decoder. From the simulation results, the proposed quantization scheme achieves much better performance than the conventional quantization scheme.

For implementing a flexible high-throughput LDPC decoder architecture, we investigated several LDPC decoders based on block-serial scheduling of the decoding computations. The block-serial scheduling is useful for low-throughput and low-complexity applications. However, for high-throughput wireless applications such as IEEE 802.11n, IEEE 802.16e, and IEEE 802.5.3c standards, a block-parallel scheduling scheme is required. We presented a novel decoding processing unit based on the block-parallel scheme (using a layered decoding schedule). Moreover, system flexibility is achieved by allowing the decoding processing units and the memory banks to be configured according to the code rate and block size of the LDPC code of interest.

We have also investigated other aspects of the layered decoding schedule to reduce the interconnection complexity. To do this, we attempted to remove unnecessary operations in the two switch networks of conventional designs. The proposed offset permutation scheme in the switch network reduces the interconnection complexity with no degradation in the decoding throughput. The proposed architecture is realized for a 672-bit, rate-1/2 irregular LDPC code on a Xilinx Virtex-4 FPGA device. The results show a 9.3% reduction in the number of required FPGA slices compared to a standard decoding architecture.

In designing decoder architectures for high-throughput applications, we designed decoders based on LDPC codes for several standards. For future work, instead of using given LDPC codes, we will able to co-design LDPC codes and decoder architectures to maximize decoding performance and throughput. This can be helpful to make flexible decoders that support different code rates and codeword sizes in practical applications.

In the chapters 5 and 6, we studied decoding algorithms and decoder architectures for nonbinary LDPC codes. First, we investigated the dynamic range of inputs and outputs for the nonlinear functions of interest. Then we analyzed quantization effects in these nonlinear functions in order to reduce the word length used in the system and the resulting LUTs sizes needed for those functions. We proposed an offset-based approach by utilizing the relative magnitudes of the quantized data to reduce the dynamic range under a given quantization. With the offset-based technique, we were able to decrease the size of the LUTs in the check node update without sacrificing the decoding performance. We also proposed an efficient architecture for the FFT-based BP decoding algorithm by balancing the computational load between CNU and VNU units. The results show a 53 % reduction in the number of required FPGA slices compared to a standard FFT-based BP architecture.

For future work, LDPC decoding algorithm with higher-order modulation (i.e., other than BPSK) can be considered. Moreover, the effects of other noise conditions such as burst errors can be considered for nonbinary LDPC codes. As mentioned previously in the chapter 1, flash memory requires ECC to ensure data integrity. Current single-level cell (SLC) NAND flash memory devices require only single-bit ECC per 512 bytes. Therefore, simple Hamming codes are used on the market today. As NAND flash geometries shrink, more sophisticated ECCs like nonbinary LDPC codes are expected to used in multi-level cell (MLC) flash memory systems. Nonbinary LDPC codes are attractive for correcting burst errors in MLC flash memory because they can correct multi-bit errors.

Bibliography

[1] S. Lin and D. J. Costello, *Error Control Coding, 2nd Edition*. Prentice Hall, NJ, 2004.

[2] T. K. Moon, *Error Correction Coding, Mathemathical Methods and Algorithms, 1st Edition*, John Wiley & Sons Inc., NJ, 2005.

[3] G.hn, *a PHY For All Seasons*. Available: http://homepnablog.typepad.com/my_weblog/2008/12/ghn-a-phy-for-all-seasons.html

[4] "T.T.S.I. Digital Video Broadcasting (DVB) second generation framing structure for broadband satellite application," [Online]. Avaiable: http://www.dvb.org

[5] *IEEE Standard for Information Technology-Telecomumnications and Information Exchange Between Systems-Local and Metroplitan Area Networks-Specific Requirements Part3: Carrier Sense Multiple Access with Collision Detection* (*CSMA/CD*) *Access Method and Physical Layer Specifications*, IEEE Std 802.3an-2006 (Amendment to IEEE Std 802.3-2005), 2006 [Online]. Available: http://standards.ieee.org

[6] *Unapproved Draft Amendment to 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* (*Amendment and Corrigendum to IEEE Std. 802.16-2004*), IEEE Std P802.16e/D9, 2005 [Online]. Available: http://standards.ieee.org

[7] *Draft Standard for Information Technology-Telecommnunications and information exchange between systems-Local and metropolitan area networks-Specific requirements─Part II: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 4: Enhancements for Higher Throughput*, IEEE Unapproved Draft Std P802.11n_D3.00, Sep. 2007 [Online]. Avaialable: http://standards.ieee.org

[8] J. Lu and J. M. F. Moura, "Structured LDPC codes for high-density recording: large girth and low error floor," *IEEE Trans. Magnetics*, vol. 42, no. 2, pp. 208-213, February 2006.

[9] H. Zhong, W. Xu, N. Xie, and T. Zhang, "Area-efficient min-sum decoder design for high-rate quasic-cyclic low-density parity-check codes in magnentic recording," *IEEE Trans. Magnetics*, vol. 43, no. 12, pp. 4117-4122, Dec. 2007.

[10] R. G. Gallager, *Low-Density Parity-Check Codes*. MIT Press, Cambridge, MA, 1963.

[11] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching low-density parity check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 619-637, Feb. 2001.

[12] D. J. C. Mackay, “Good codes based on very sparse matrices,” *IEEE Trans. Inform. Theory*, vol. 45, no. 3, pp. 399-431, March 1999.

[13] S. Y. Chung G. D. Forney, T. Richardson and R. Urbanke, “On the design of low-density check codes within 0.0045dB of the Shannon limit,” *IEEE Trans. Commun. Lett*., vol. 5, pp. 58-60, Feb. 2001.

[14] K. Shimizu, T. Ishikawa, N. Togawa, T. Ikenaga, and S. Goto, “Partially-Parallel LDPC Decoder Achieving High-Efficiency Message-Passing Schedule,” *IEICE Trans. Fundamentals of Electronics, communications and computer science*, vol. E89-A, no. 4, April 2006.

[15] J. Zhang, M. Fossorier, and D. Gu, “Two-demensional correction for min-sum decoding of irregular LDPC codes,” *IEEE Transactions on Communications Letters*, vol. 10, no. 3, pp. 180-182, March 2006.

[16] T. Zhang, Z. Wang, and K. K. Parhi, “On finite precision implementation of low density partiy check codes decoder,” in *Proceedings of 2001 IEEE International Symposium on Circiuts and Systems* (*ISCAS 2001*), vol. 4, May 2001, pp. 202-205.

[17] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, “On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes,” *IEEE Transactions on Communications*, vol. 53, no. 4, pp. 549-554, April 2005.

[18] J. Chen, R. M. Tanner, C. Jones, and Y. Li, “Improved min-sum decoding algorithms for irregular LDPC codes,” in *Proceedings of 2005 IEEE International Symposium on Information Theory* (*ISIT 2005*), September 2005, pp. 449-453.

[19] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X. Hu, “Reduced-complexity decoding of LDPC codes,” *IEEE Trans. Commun.*, vol. 53, No. 8, pp. 1288-1299, Aug. 2005.

[20] J. Chen, R.M. Tanner, C. Jones, and Y. Li, “Improved min-sum decoding algorithms for irregular LDPC codes,” in *Proc. IEEE Int. Symp. on Inform. Theory* (*ISIT*), Sep. 2005, pp. 449-453.

[21] D. Oh and K.K. Parhi, "Performance of Quantized Min-Sum Decoding Algorithms for Irregular LDPC Codes," In *Proc. IEEE Int. Symp. on Circuits Sys.* (*ISCAS*), May 2007, pp. 2758-2761.

[22] Z. Zhang, L Dolecek, M Wainwright, V. Anantharam, and B. Nikolic, “Quantization Effects in Low-Density Parity-Check Decoders,” in *Proc. IEEE Int. Conf. Commun*., pp. 6231-6237, June 2007.

[23] IEEE 802.16e. Air interface for fixed and mobile braoadband wireless access systems. IEEE P802.16e/D12 Draft, Oct 2005.

[24] "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wirelss Personal Area Networks (WPANs): Amendment 2: Millimeter-wave based Alternative Physical Layer Extension," 2008. IEEE P802.15.3c/D00.

[25] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI Syst*., vol. 11, no. 6, pp. 976-996, Dec. 2003.

[26] H. Zhong and T. Zhang, "Block-LDPC: A Practical LDPC coding system design approach," *IEEE Trans. Circuits Syst. I*, vol. 52, no. 4, pp. 766-775, Apr. 2005.

[27] E. Sharon, S. Litsyn, and J. Goldberger, "An efficient message-passing schedule for LDPC decoding," in *Proc. The 23rd IEEE Convention of Electrical and Electronics Engineering*, Sept. 2004, pp. 223-226.

[28] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. IEEE Workshop Signal Process. Syst.* (*SIPS*), Oct. 2004, pp. 107-112.

[29] M. Rovini, G. Gentile, F. Rossi, L. Fanucci, "A minimum-latency block-serial architecture of a decoder for IEEE 802.11n LDPC codes," in *Proc. IEEE Int. Conf. Very Large Scale Integration* (*VLSI*) *and System-on-Chip* (*SoC*), Oct. 2007, pp.236-241.

[30] G. Gentile, M. Rovini, L. Fanucci, "Low-Complexity Architectures of a Decoder for IEEE 802.16e LDPC Codes," in *Proc. 13th Euromicro Conf. Digital System Design* (*DSD*), Aug. 2007, pp. 369-375.

[31] T. Bhatt, V. Sundaramurthy, V. Stolpman, D. McCain, "Pipelined Block-Serial Decoder Architecture for Structured LDPC Codes," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Process.* (*ICASSP*), May 2006, vol. 4, pp. 225-228.

[32] T. Richardson and V. Novichkov, "Methods and apparatus for decoding LDPC codes," US Patent App. US2003-0033575, Feb. 2003.

[33] V. E. Benes, "Optimal rearrangeable multistage connecting networks," Bell Syst. Tech. J., vol. 43, pp. 1641-1656, 1964.

[34] S. Kim, G. E. Sobelman, H. Lee, "Adaptive quantization in min-sum based irregular LDPC decoder," in *Proc. IEEE Int. Symp. on Circuits Syst*., May, 2009, pp. 536-539

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

[36] K. K. Parhi, *VLSI Digital Signal Processing Systems*: *Design and Implementation*. New York: Wiley, 1999.

[37] T. Zhang and K. K. Parhi, "Joint (3,k)-regular LDPC code and decoder/encoder design," *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp. 1065-1079, Apr. 2004.

[38] S. Kim, K. K. Parhi, and R. Liu, "System and method for designing RS-based LDPC code decoder," US Patent App. US2007-0033484, Feb. 8, 2007.

[39] L. Liu and C.-J. R. Shi, "Sliced Message Passing: High Throughput Overlapped Decoding of High-Rate Low-Density Parity-Check Codes," *IEEE Trans. Circuits Syst.—I: Reg. Papers*, vol. 55, no. 11, pp. 3697-3710, Dec. 2008.

[40] M. M. Mansour and N. R. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 684-698, Mar. 2006.

[41] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, Austin, TX, Oct. 2004, pp. 107-112.

[42] Z. Cui, Z. Wang, and Y. Liu, "High-Throughput Layered LDPC Decoding Architecture," *IEEE Trans. VLSI Systems*, vol. 17, no.4, pp. 582-587, Apr. 2009.

[43] K. K. Gunnam, G. S. Cho, and M. B. Yeary, "A Parallel VLSI Architecture for Layered Decoding for Array LDPC Codes," in *Proc. Int. Conf. on VLSI Design,* Bangalore, India, Jan. 2007, pp. 738-743.

[44] S. Kim, G. E. Sobelman, and H. Lee, "Flexible LDPC decoder architecture for high-throughput applications," in *Proc. IEEE Asia Pacific Conf. on Circuits Syst*. (*APCCAS*), Macao, China, Nov. 2008, pp. 45-48.

[45] *Draft Standard for Wireless Medium Access Control* (*MAC*) *and Physical Layer* (*PHY*) *Specifications for High Rate Wireless Personal Area Networks* (*WPANs*): *Amendment 2*: *Millimeter-wave based Alternative Physical Layer Extension*, IEEE P802.15.3c/D04, 2008 [Online]. Available:http://standards.ieee.org

[46] S. Olcer, "Decoder architecture for arrary-code-based LDPC codes," in *Proc. IEEE Global Telecommun.* (*GLOBECOM*) *Conf*., San Francisco, CA, Dec. 2003, pp. 2046-2050.

[47] F. Quaglio, F. Vacca, C. Castellano, A. Tarable, and G. Masera, "Inter-connection framework for high-throughput, flexible LDPC decoders," in *Proc. Design Automation and Test in Europe*, Mar. 2006, vol. 2, pp. 6-10.

[48] J. Tang, T. Bhatt, V. Sundaramurthy, and K. K. Parhi, "Reconfigurable shuffle network design in LDPC decoders," in *Proc. Application Specific Systems, Architecture and Processors* (*ASAP*), Steamboat Springs, CO, Sep. 2006, pp. 81-86.

[49] M. Davey and D.J.C. MacKay, "Low density parity check codes over GF(q)," *IEEE Commun. Lett.,* vol. 2, no. 6, pp. 165-167, June, 1998.

[50] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 599-618, Feb., 2001.

[51] D. Declercq and M. Fossorier, "Decoding Algorithms for Nonbinary LDPC Codes Over GF(q)," *IEEE Trans*. *Communications*, vol. 55, no. 4, pp. 633-643, Apr., 2007.

[52] D. Declercq and M. Fossorier, "Extended minsum algorithm for decoding LDPC codes over GF(q)," in *Proc. IEEE Int. Symp. on Inf. Theory*, pp. 464-468, Adelaide, Australia, Sept. 2005.

[53] Y. Liu, J. Ning and J. Yuan, "Modified min-sum algorithm with threshold filtering for nonbinary LDPC codes over GF(q)," in *Proc. IEEE Int. Symp. Inf. Theory and its Appl*, pp. 1-4, Auckland, New Zealand, Dec. 2008.

[54] L. Barnault and D. Declercq, "Fast decoding algorithm for LDPC over GF(2^q)," in *Proc. Workshop on Inf. Theory,* pp. 70-73, Paris, France, Mar., 2003.

[55] R. A. Carrasco and M. Johnston, *Non-binary Error Control Coding for Wireless Communication and Data Storage*, 1st ed., John Wiley & Sons, 2008.

[56] H. Song and J. R. Cruz, "Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording," *IEEE Trans. Magnetics*, vol. 39, no. 2, pp. 1081-1087, Mar., 2003.

[57] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Architecture of a low-complexity non-binary LDPC decoder for high order fields," *Proc. IEEE ISCIT*, pp. 273-278, Sydney, Australia, Oct. 2007.

[58] C. Spagnol, W. Marnane and E. Popovici, and " FPGA Implementations of LDPC over GF(2^m) Decoders," *in Proc. IEEE Workshop on Signal Processing Systems*, pp. 273-278, Shanghai, China, Oct. 2007.

[59] H. Wymeersch, H. Steendam and M. Moeneclaey, "Log-domain decoding of LDPC codes over GF(q)," in *Proc. IEEE Int. Conf. Commun.*, pp. 772-776, Paris, France, June, 2004.

[60] B. Zhou, L. Zhang, J. Kang, Q. Huang, S. Lin and K. Abdel-Ghaffar, "Array dispersions of matrices and constructions of quasi-cyclic LDPC codes over non-binary fields," in *Proc. IEEE Int. Symp. on Inf. Theory*, pp. 1158-1162, Toronto, Canada, July, 2008.

[61] H. Wymeersch, H. Steendam and M. Moeneclaey, "Computational complexity and quantization effects of decoding algorithms for non-binary LDPC codes," in *Proc. IEEE Intl. Conf. on Acoustics, Speech and Signal Processing*, pp. 669-672, Montreal, Canada, May, 2004.

[62] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Low-complexity, Low-memory EMS algorithm for non-binary LDPC codes," in *Proc. IEEE Int. Conf. Commun.*, Glasgow, Scotland, Jun. 2007, pp. 671-676.

[63] V. Savin, "Min-Max decoding for non binary LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, Canada, Jul. 2008, pp. 960-964.

[64] J. Lin, J. Sha, Z. Wang, and L. Li, "An Efficient VLSI Architecture for Nonbinary LDPC Decoders," *IEEE Trans. Circuits Sys. II*, vol. 57, no. 1, pp. 51-55, Jan. 2010.

[65] C. Spagnol, E. Popovici, and W. Marnane, "Hardware implementation of GF(2^m) LDPC decoders," *IEEE Trans. Circuits Sys. I*, vol. 56, no. 12, pp. 2609-2620, Dec. 2009.

[66] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*: *Network of Plausible Inference*, San Mateo, CA: Morgan Kaufmann, 1988.

[67] D. J. C. Mackay and R. M. Neal, "Good codes based on very sparse matrices," in *Proc. Cryptography and Coding. 5th IMA Conf.*, Colin Boyd, Ed., number 1025 in lecture notes in computer science. Berilin, Germany: Springer, 1995, pp. 100-111.

[68] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457-458, Mar. 1997.

[69] S. Kim and G. E. Sobelman, "Quantization of FFT-Based Belief Propagation Decoding for Nonbinary LDPC Codes," submitted to *IEEE Trans. Communications*, 2010.