AN ABSTRACT OF THE DISSERTATION OF

Zhiqiang Cui for the degree of Doctor of Philosophy in
Electrical and Computer Engineering presented on September 4, 2007.

Title:

Low-Complexity High-Speed VLSI Design of Low-Density Parity-Check Decoders

Abstract approved:

_____

Zhongfeng Wang

Low-Density Parity-check (LDPC) codes have attracted considerable attention due to their capacity approaching performance over AWGN channel and highly parallelizable decoding schemes. They have been considered in a variety of industry standards for the next generation communication systems. In general, LDPC codes achieve outstanding performance with large codeword lengths (e.g., N>1000 bits), which lead to a linear increase of the size of memory for storing all the soft messages in LDPC decoding. In the next generation communication systems, the target data rates range from a few hundred Mbit/sec to several Gbit/sec. To achieve those very high decoding throughput, a large amount of computation units are required, which will significantly increase the hardware cost and power consumption of LDPC decoders. LDPC codes are decoded using iterative decoding algorithms. The decoding latency and power consumption are linearly

proportional to the number of decoding iterations. A decoding approach with fast convergence speed is highly desired in practice.

This thesis considers various VLSI design issues of LDPC decoder and develops efficient approaches for reducing memory requirement, low complexity implementation, and high speed decoding of LDPC codes. We propose a memory efficient partially parallel decoder architecture suited for quasi-cyclic LDPC (QC-LDPC) codes using Min-Sum decoding algorithm. We develop an efficient architecture for general permutation matrix based LDPC codes. We have explored various approaches to linearly increase the decoding throughput with a small amount of hardware overhead. We develop a multi-Gbit/sec LDPC decoder architecture for QC-LDPC codes and prototype an enhanced partially parallel decoder architecture for a Euclidian geometry based LDPC code on FPGA. We propose an early stopping scheme and an extended layered decoding method to reduce the number of decoding iterations for undecodable and decodable sequence received from channel. We also propose a low-complexity optimized 2-bit decoding approach which requires comparable implementation complexity to weighted bit flipping based algorithms but has much better decoding performance and faster convergence speed.

Low-Complexity High-Speed VLSI Design of Low-Density Parity-Check Decoders

by

Zhiqiang Cui

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented September 4, 2007
Commencement June 2008

Doctor of Philosophy dissertation of Zhiqiang Cui presented on September 4, 2007.

APPROVED:

_____

Major Professor, representing Electrical and Computer Engineering

_____

Director of the School of Electrical Engineering and Computer Science

_____

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

_____

Zhiqiang Cui, Author

ACKNOWLEDGEMENTS

I am grateful to my advisor, Dr. Zhongfeng Wang for providing me an opportunity to conduct research under his supervision. I would like to express my sincere thanks for his continuing encouragement, guidance, and support throughout the course of my study at Oregon State University. Working with him has been an extremely valuable learning experience. I would like to thank the members of my program committee – Dr. Bella Bose, Dr. Huaping Liu, Dr. Thinh Nguyen, Dr. David McIntyre, and Dr. David Hackleman for their effort and time in supporting my work. Their comments and suggestions helped me improve my thesis.

I would also like to thank the members of our group, particularly, Qingwei Li, Lupin Chen, and Jinjin He for many useful discussions.

This thesis is dedicated to my wonderful family. I would like to express my deepest gratitude to my parents, my wife, my brother, and my sisters for their unconditional love, constant encouragement, and support.

TABLE OF CONTENTS

Page

1 INTRODUCTION 1

1.1 OVERVIEW 1

1.2 SUMMARY OF CONTRIBUTIONS 2

1.2.1 Memory Efficient Decoder for Quasi-Cyclic LDPC Codes 2

1.2.2 Efficient Design of High Speed LDPC Decoders 2

1.2.3 Low Complexity Decoding of LDPC Codes 3

1.2.4 Reducing Iterations for LDPC Codes 4

2 DECODING OF LDPC CODES 5

2.1 INTRODUCTION OF LDPC CODES 5

2.1.1 Representations of LDPC Codes 5

2.1.2 LDPC Code Construction and Encoding 7

2.2 BELIEF PROPAGATION DECODING ALGORITHM 7

2.3 MIN-SUM DECODING ALGORITHMS 10

2.4 BCJR ALGORITHM BASED DECODING APPROACH 13

2.5 BIT FLIPPING AND WEIGHTED BIT FLIPPING BASED ALGORITHMS 15

3 MEMORY-EFFICIENT DECODER ARCHITECTURE 17

3.1 THE PERFORMANCE OF HIGH RATE QC-LDPC CODE 17

3.2 THE REARRANGED (MODIFIED) MIN-SUM ALGORITHM 18

3.3 THE MEMORY EFFICIENT DECODER ARCHITECTURE 19

3.3.1 Parallel Decoder Architecture for H Matrix Consisting of Weight-1 Circulant and Possible Zero Submatrices 20

TABLE OF CONTENTS (Continued)

Page

3.3.2 Architecture for H Matrices with Weight-1, Weight-2 Circulant Matrices and Zero Matrices 25
3.4 Optimization on the Partially Parallel Decoder Architecture 27
3.4.1 The Optimized CNU 27
3.4.2 The Optimized Data Scheduling Unit 30
3.4.3 The Optimized Data Merge Unit 32
3.5 Summary 33
4 EFFICIENT VLSI DESIGN OF HIGH THROUGHPUT LDPC DECODERS 34
4.1 Efficient Message Passing Architecture 34
4.1.1 Efficient Message Passing Schemes with Min-Sum Algorithm 35
4.1.2 Architecture for Permuation Matrices Based LDPC Codes 39
4.1.3 Further Complexity Reduction with Non-uniform Quantization 42
4.2 Layered Decoding Architecture for Quasi-Cyclic Codes 44
4.2.1 Row Permutation of Parity Check Matrix of QC-LDPC Codes 45
4.2.2 Approximate Layered Decoding Approach 46
4.2.3 Decoder Architecture with Layered Decoding Approach 49
4.2.4 Hardware Requirement and Throughput Estimation 54
4.3 An FPGA Implementation of Quasi-Cyclic LDPC Decoder 56
4.3.1 The (8176, 7156) EG-based QC LDPC Code 57
4.3.2 Partially Parallel Decoder Architecture 58
4.3.3 Fixed-point implementation 66
4.3.4 FPGA Implementation 71

TABLE OF CONTENTS (Continued)

Page

4.4 SUMMARY 71

5 PRACTICAL LOW COMPLEXITY LDPC DECODERS 73

5.1 THE OPTIMIZED 2-BIT DECODING 73

5.1.1 Decoding Scheme 73

5.1.2 Decoding Performance Simulation 75

5.2 LOW COMPLEXITY 2-BIT DECODER DESIGN 78

5.2.1 Memory Reduction Scheme 78

5.2.2 Computation Units Design 80

5.3 SUMMARY 84

6 REDUCING ITERATIONS FOR LDPC CODES 85

6.1 EXTENDED LAYERED DECODING OF LDPC CODES 85

6.1.1 The Proposed Layered Decoding Approach 86

6.1.2 Overlapped Message Passing Decoding 87

6.1.3 Simulation Results 89

6.2 AN EFFICIENT EARLY STOPPING SCHEME FOR LDPC DECODING 92

6.3 THE FAST DECODING SCHEME FOR WBF-BASED ALGORITHMS 95

6.3.1 Multi-threshold Bit Flipping Scheme 95

6.3.2 Performance Simulation 96

6.4 SUMMARY 99

7 CONCLUSIONS AND FUTURE WORKS 100

7.1 CONCLUSIONS 100

TABLE OF CONTENTS (Continued)

Page

7.2 FUTURE WORK 101

8 BIBLIOGRAPH 103

LIST OF FIGURES

Figure Page

2.1 An example of Tanner graph. 6

2.2 The mesh and contour plot of $g(x, y)$ 12

2.3 The two-state trellies of a SPC code. 13

3.1 BER/FER performance of the (4608, 4096) QC-LDPC codes. 18

3.2 The partially parallel decoder architecture for ***H*** matrix containing weight-1 and possible zero submatrices. 21

3.3 The structure of data distributor 24

3.4 The structure of single-port memory supporting simultaneous read and write operation. 25

3.5 The decoder architecture for ***H*** matrix containing weight-2 circulant matrices 27

3.6 The structure of pseudo rank order filter 28

3.7 The architecture of the optimized CNU. 29

3.8 The data scheduling unit. (a) structure (b) data flow 31

3.9 The structure of the merge unit. 33

4.1 Computation units using reformulated Min-Sum algorithm.
(a) Variable node unit (b) Check node unit 37

4.2 Computation units using APP-based Min-Sum algorithm.
(a) Variable node unit (b) Check node unit 39

4.3 The structure of 8-input folded check node unit. 40

4.4 The ***H*** matrix of an LDPC code example. 41

4.5 Decoder architecture for the example code. 42

LIST OF FIGURES (Continued)

Figure Page

4.6 BER and FER of various decoding approaches (24 iterations) for the (2048, 1723) LDPC code. 43

4.7 An array of circulant submatrices. 46

4.8 Permuted matrix. 46

4.9 Performance of the approximate layered decoding approach. 49

4.10 Decoder architecture (*P*=2). 50

4.11 The computation path of the proposed architecture (*P*=2). 52

4.12 The optimization of the SM-to-2'S unit and the adder in segment-1. 53

4.13 The optimization of the SM-to-2'S unit and two adders in segment-3. 53

4.14 The structure of a data shifter. 54

4.15 A 15x15 circulant matrix. 57

4.16 Check node unit architecture. 59

4.17 Variable node unit architecture. 60

4.18 Enhanced partially parallel decoder architecture for QC-LDPC code. 61

4.19 Memory partitioning and data switching scheme for even shifting offset case. 63

4.20 Memory partitioning and data switching scheme for odd shifting offset case. 64

4.21 Memory partitioning and data switching scheme for identity matrix. 64

4.22 State transition diagram of controller. 65

4.23 Block diagram of controller. 66

LIST OF FIGURES (Continued)

Figure Page

4.24 Decoding performance of fixed-point quantization and double precision. 67

4.25 The uniform quantization of $\Psi(x)$. 68

4.26 Check node unit with non-uniform quantization. 70

4.27 Variable node unit with non-uniform quantization. 70

5.1 Performance of the (2048, 1723) rate-0.84 LDPC codes. 76

5.2 Average number of iterations for decoding the (2048, 1723) rate-0.84 LDPC codes. 77

5.3 Performance of the (1974, 987) rate-0.5 LDPC codes. 77

5.4 Average number of iterations for decoding the (1974, 987) rate -0.5 LDPC codes. 78

5.5 Structure of the check node unit for the optimized 2-bit decoding approach. 82

5.6 Structure of the variable node unit for the optimized 2-bit decoding approach. 82

5.7 The computation core needed in the bit flipping operation for a WBF-based decoder. 83

6.1 Serial computation of the smallest and the second smallest magnitude. 88

6.2 The data flow of overlapped message passing scheme. 89

6.3 Average number of iterations and bit error rate (BER) for the rate-0.84 code with standard layered decoding, proposed approach, and TPMP SPA decoding. 91

6.4 Average number of iterations and BER for the rate-0.5 code with standard layered decoding, proposed approach, and TPMP SPA decoding. 91

LIST OF FIGURES (Continued)

Figure Page

6.5 Performance of the (2048, 1723) rate-0.84 LDPC code. 97

6.6 Average number of iterations for decoding the (2048, 1723) rate-0.84 LDPC code. 97

6.7 Performance of the (1974,987) rate-0.5 LDPC code. 98

6.8 Average number of iterations for decoding the (1974,987) rate-0.5 LDPC code. 98

LIST OF TABLES

Table Page

3.1 Memory needed by the proposed architecture……..…… 24

3.2 Memory needed by the traditional approaches……..…… 24

4.1 3-bit quantization for received symbol……..……44

4.3 Gate count estimation for computing blocks……..……...56

4.4 Storage requirement estimate……..…….. 56

4.5. Uniform to non-uniform quantization conversion..... 69

4.6. Xilinx virtexII-6000 FPGA utilization statistics……..……... 71

5.1 Data conversion for the rate-0.84 code……..……... 75

5.2 Memory requirement of the 2-bit decoder……...……..80

5.3 Memory requirement of MWBF decoder……..…80

5.4 Complexity of computation units for the optimized 2-bit and WBF-based decoder……..84

6.1 The value of δ_i for the rate-0.84 code and the rate-0.5 code……..……96

LOW-COMPLEXITY HIGH-SPEED VLSI DESIGN OF LOW-DENSITY PARITY-CHECK DECODERS

1 INTRODUCTION

1.1 Overview

Reliable and efficient information transmission and storage has been increasingly demanded in recently years. Error correcting codes are widely used in digital communication and storage systems to protect data against transmission error cause by channel noise. From channel coding theory, for a channel with a capacity C, there exist codes of rate $R < C$ that have an arbitrarily small decoding error probability with Maximum Likelihood Decoding (MLD). The arbitrarily small error probabilities are achievable by increasing the codeword length for block code or the encoder memory order for convolutional code [17]. Because the implementation complexity of typical decoding algorithm such as MLD becomes very large as codeword length or encoder memory order increases, researchers have made significant amount of effort to develop new coding schemes which can be decoded using simpler approaches and have decoding performance close to what could be achieved using MLD.

Low-Density Parity-Check (LDPC) codes invented by Gallager in the early 1960s are a class of near Shannon limit error correcting codes and can be decoded using belief propagation algorithm. Because of the limitation of the computation capabilities at that time, they have largely been ignored for more than 30 years. With the innovation of VLSI and computer technology, the implementation cost of LDPC codec is reduced. Since the late 1990s, LDPC codes have attracted considerable attention due to their capacity approaching performance over AWGN channel and highly parallelizable decoding schemes.

This research is devoted to the efficient VLSI architecture design and implementation for LDPC codes. We have considered various VLSI design issues of LDPC decoder and developed efficient approaches for reducing memory requirement, low complexity implementation, and high speed decoding of LDPC codes.

1.2 Summary of Contributions

1.2.1 Memory Efficient Decoder for Quasi-Cyclic LDPC Codes

Quasi-cyclic LDPC (QC-LDPC) codes [4][13][14], being a special class of LDPC codes, are well suited for hardware implementation. The encoders of QC-LDPC codes can be built with shift-registers [16]. In addition, QC-LDPC codes also facilitate efficient high-speed decoding because of the regularity in their parity check matrices. LDPC codes achieve outstanding performance only with large codeword lengths (*e.g.,* $N \geq 1000$ *bits*), which lead to a linear increase of memory requirement for storing all the soft messages in LDPC decoding.

To reduce hardware cost of QC-LDPC decoder, we proposed a memory efficient partially parallel decoder architecture for high rate QC-LDPC codes, which stores soft messages in the Min-Sum decoding algorithm in a compressed form. In general, over 30% memory can be saved. To further reduce the implementation complexity, various optimization techniques were developed.

1.2.2 Efficient Design of High Speed LDPC Decoders

For high throughput applications, the decoding parallelism is usually very high. Hence a complex interconnect network is required which consumes a significant amount of silicon area and power. In a pioneer design of high throughput LDPC decoder [57], the

power dissipation of the decoder was largely determined by the switching activity of interconnect network. The utilization of chip area was only 50%.

To reduce complexity of interconnect network, we propose an efficient message passing decoder architecture using Min-Sum algorithm for permutation matrices based LDPC codes. QC-LDPC codes are a sub-class of permutation matrices based LDPC codes. The regularity in their parity check matrix can be further exploited. We develop a multi-Gbit/sec low-cost layered decoding architecture for generic QC-LDPC codes. To demonstrate the design of high speed LDPC decoder, we implement an enhanced partially parallel decoder architecture with FPGA for a (8176, 7156) Euclidian geometry based QC-LDPC code. A worst-case source information decoding throughput (at 15 iterations) over 170Mbps is achieved.

1.2.3 Low Complexity Decoding of LDPC Codes

The Sum-Product LDPC decoding algorithm (SPA) (also known Belief-Propagation algorithm) has the best decoding performance and the highest implementation complexity. On the other hand, various weighted Bit-Flipping (WBF) based decoding approaches were proposed in order to seek very low decoding complexity.

We analyzed the decoding complexity of state-of-the-art WBF-based algorithms from a VLSI implementation point of view. To maintain low decoding complexity while further narrowing the performance gap from the SPA, we present an optimized 2-bit soft decoding approach. The implementation complexity of the proposed method is comparable to WBF-based algorithms. However, the proposed approach achieves much better decoding performance and faster convergence speed.

1.2.4 Reducing Iterations for LDPC Codes

LDPC codes are decoded using iterative decoding algorithms. To increase decoding speed, it is highly desired to reduce the number of decoding iterations without significant performance loss. We propose an extended layered decoding approach which can be applied to any structure of parity check matrix. Simulations on both random and structured LDPC codes show that the proposed approach achieves faster convergence over conventional two phase message passing decoding algorithm. On the other hand, it happens frequently that a valid codeword can not be found even though a large number of decoding iterations are performed at low to medium signal-to-noise ratios. We propose an efficient early stopping scheme to detect such undecodable cases as early as possible in order to avoid unnecessary computation. In addition, we demonstrate that the decoding convergence of WBF-based algorithm can be significantly speeded up with a multi-threshold detection scheme.

2 DECODING OF LDPC CODES

2.1 Introduction of LDPC Codes

Low-Density Parity-Check (LDPC) codes [1] invented by Gallager are a class of error correcting codes and can be decoded using belief propagation algorithm. Since the late 1990s, LDPC codes have attracted considerable attention due to their capacity approaching performance over AWGN channel and highly parallelizable decoding schemes. In recent years, LDPC codes have been considered in a variety of industry standards for the next generation communication systems such as DVB-S2, WLAN (802.11.n), WiMAX (802.16e) and 10GBaseT (802.3an).

2.1.1 Representations of LDPC Codes

LDPC codes are a class of linear block codes whose parity-check matrices $\boldsymbol{H}$ are very spare binary matrices. Conventionally, LDPC codes are characterized in matrix representation and graphical representation [17]. In matrix representation, an LDPC code is described as a *k*-dimensional subspace *C* of the vector space $\boldsymbol{F}_2^n$ of all binary n-tuples over the Galois field GF(2). It is possible to find *k* linearly independent codewords, $\boldsymbol{g}_0, \boldsymbol{g}_1, \cdots, \boldsymbol{g}_{k-1}$, such that every codeword $\boldsymbol{c} \in C$ is a linear combination of these *k* codewords (*i.e.*, $\boldsymbol{c} = u_0\boldsymbol{g}_0 + u_1\boldsymbol{g}_1 + \cdots + u_{k-1}\boldsymbol{g}_{k-1}$). In matrix form, $\boldsymbol{c} = \boldsymbol{uG}$, where, $\boldsymbol{u} = [u_0\, u_1 \cdots u_{k-1}]$, is the information to be encoded and $\boldsymbol{G}$ is a $k \times n$ generator matrix whose rows, $\{\boldsymbol{g}_0, \boldsymbol{g}_1, \cdots, \boldsymbol{g}_{k-1}\}$, span the (*n*, *k*) LDPC code. For a generator matrix $\boldsymbol{G}$, there exists a $(n-k) \times k$ matrix $\boldsymbol{H}$ such that $\boldsymbol{GH}^T = \boldsymbol{0}$. Thus for every codeword $\boldsymbol{c} \in C$, $\boldsymbol{cH}^T = \boldsymbol{0}$. In graphical representation, an LDPC code is represented by a bipartite graph (also called Tanner graph). Fig. 2.1 shows a Tanner graph. Nodes in a Tanner graph are partitioned into

two disjoint classes, *i.e.*, variable nodes and check nodes. Variable nodes are associated with digits of the codeword and check nodes are associated with the set of parity-check constraints which define the code. The 1-componets in parity-check matrix are associated to edges in Tanner graph. An edge in Tanner graph may only connect two nodes of different classes.

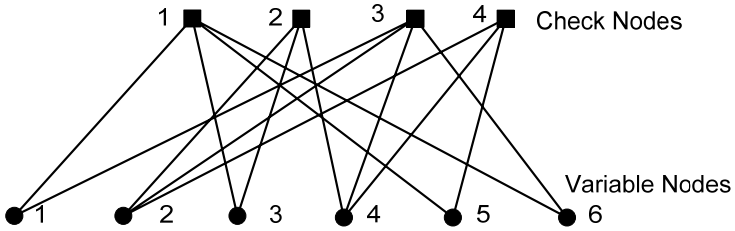


Figure 2.1 An example of Tanner graph.

An LDPC code is usually characterized by its check node and variable node degree distribution polynomials. The terminology, node degree, is defined as the number of edges connected to a node in graph. In LDPC matrix representation, it is equal to the number of 1-components in a row or column of the parity check matirx. The check node and variable node degree distribution polynomials ususlly denoted by $\rho(x)$ and $\lambda(x)$, respectively [6]. More specifically,

$$\rho(x) = \sum_{d=1}^{d_c} \rho_d x^{d-1} \text{ and } \lambda(x) = \sum_{d=1}^{d_v} \lambda_d x^{d-1},$$

where, ρ_d denotes the fraction of all degrees connected to degree-*d* check nodes and d_c denotes the maximum check node degree. Similarly, λ_d denotes the fraction of all degrees connected to degree-*d* variable nodes and d_v denotes the maximum variable node degree.

2.1.2 LDPC Code Construction and Encoding

In the literature, various LDPC code construction approaches have been proposed. Among them, progressive edge-growth graph (PEG) construction [15] [30] and finite field algebraic construction [18] [19][20] are widely used in practice. All these approach construct a low-density parity-check matrix $\boldsymbol{H}$. Although the parity check matrices of LDPC codes are sparse by code construction, the generator matrices are usually high density matrices. Therefore, the direct encoding approach, $\boldsymbol{c} = \boldsymbol{uG}$, has the encoding complexity of $O(N^2)$, where N is the block length of an LDPC code. To reduce the encoding complexity, various efficient encoding method has been proposed. Each one is usually only suitable for a specific class of LDPC codes. The encoding details for general LDPC codes, QC-LDPC codes and repeat-accumulate LDPC codes can be found in [8] **Error! Reference source not found.** [16] [68].

2.2 Belief Propagation Decoding Algorithm

The LDPC decoding algorithm was originally provided by Gallager in 1960s [1]. Since then, the decoding algorithm has been independently rediscovered by other researchers. Belief propagation algorithm (BPA) is also named as Sum-Product algorithm (SPA) in the literature. In general, it has the best decoding performance among all LDPC decoding algorithms. Let C be a binary (n, k) LDPC code specified by a parity-check matrix $\boldsymbol{H}$ with M rows and N columns. Using a notation similar to that in [4], let $N(m) = \{n : H_{mn} = 1\}$ denote the set of variable nodes that participate in check m. Similarly, let $M(n) = \{m : H_{mn} = 1\}$ denote the set of checks in which variable node n participates. Let $N(m) \setminus n$ represent the set $N(m)$ with variable node n excluded and $M(n) \setminus m$ represent

the set $M(n)$ with check *m* excluded. Let $\boldsymbol{c}=(c_1,c_2,\cdots,c_N)$ and $\boldsymbol{x}=(x_1,x_2,\cdots,x_N)$ denotes coded sequence and the transmitted vector. The received vector and the corresponding hard-decision vector are denoted by $\boldsymbol{y}=(y_1,y_2,\cdots,y_N)$ and $\boldsymbol{z}=(z_1,z_2,\cdots,z_N)$, respectively. Let $P_v=\Pr[c_v=1\,|\,y_v]$ be the probability that the transmitted digit in position *v* is a 1 conditional on the received digit in position *v*, and let P_{mn} be the same probability for the *n*'th digit in the *m*'th parity-check set. Let $\Pr[c_v=1\,|\,\boldsymbol{y},S]$ be the probability that the transmitted digit in position v is a 1 conditional on the set of received vector $\boldsymbol{y}$ and on the event *S* that the transmitted digits satisfy all *j* parity-check equations on digit *v*. Assume the digits be statistically independent of each other. Then

$$\frac{\Pr[c_v=0\,|\,\boldsymbol{y},S]}{\Pr[c_v=1\,|\,\boldsymbol{y},S]}=\frac{\Pr[c_v=0\,|\,\boldsymbol{y}]}{\Pr[c_v=1\,|\,\boldsymbol{y}]}\times\frac{\Pr[S\,|\,c_v=0,\boldsymbol{y}]}{\Pr[S\,|\,c_v=1,\boldsymbol{y}]} \tag{2.1}$$

Consider a sequence of d_c independent binary digits c_v in which $P_v=\Pr[c_v=1\,|\,y_v]$. Then the probability that the sequence contains an even number of 1's and an odd number of 1's are expressed in (2.2) and (2.3) respectively.

$$\frac{1+\prod_{v=1}^{d_c}(1-2P_v)}{2} \tag{2.2}$$

$$\frac{1-\prod_{v=1}^{d_c}(1-2P_v)}{2} \tag{2.3}$$

Then

$$\frac{\Pr[c_v=0\,|\,\boldsymbol{y},S]}{\Pr[c_v=1\,|\,\boldsymbol{y},S]}==\frac{1-P_v}{P_v}\prod_{m\in M(v)}\frac{1+\prod_{n\in N(m)\backslash v}(1-2P_{mn})}{1-\prod_{n\in N(m)\backslash v}(1-2P_{mn})} \tag{2.4}$$

For the actual computation, it is more convenient to use (2.4) in terms of log-likelihood ratios (LLR). Let

$$\ln\left(\frac{\Pr[c_v = 0 \mid y_v]}{\Pr[c_v = 1 \mid y_v]}\right) = \alpha_v \beta_v \quad ,$$

where α_v is the sign and β_v is the magnitude. Similarly, let

$$\ln\left(\frac{\Pr[c_v = 0 \mid \boldsymbol{y}, S]}{\Pr[c_v = 1 \mid \boldsymbol{y}, S]}\right) = \alpha_v' \beta_v' \quad .$$

We can rewrite (2.4) as

$$\alpha_v' \beta_v' = \alpha_v \beta_v + \sum_{m \in M(v)} \left\{ \left(\prod_{n \in N(m) \backslash v} \alpha_{mn} \right) \times \ln\left(\frac{1 + \prod_{n \in N(m) \backslash v} \left(\frac{e^{\beta_{mn}} - 1}{e^{\beta_{mn}} + 1}\right)}{1 - \prod_{n \in N(m) \backslash v} \left(\frac{e^{\beta_{mn}} - 1}{e^{\beta_{mn}} + 1}\right)} \right) \right\} , \tag{2.5}$$

$$= \alpha_v \beta_v + \sum_{m \in M(v)} \left\{ \left(\prod_{n \in N(m) \backslash v} \alpha_{mn} \right) \times \Psi\left(\sum_{n \in N(m) \backslash v} \Psi(\beta_{mn}) \right) \right\} \quad , \tag{2.6}$$

where $\Psi(\beta) = \ln\left(\frac{e^{\beta} + 1}{e^{\beta} - 1}\right)$.

The standard two phase message passing (TPMP) belief propagation (BP) iterative decoding approach is formulated as follows. Let $I_v = \ln\left(\frac{\Pr[c_v = 0 \mid y_v]}{\Pr[c_v = 1 \mid y_v]}\right)$ denote the intrinsic message. Assuming that $c_v = 0$ and $c_v = 1$ are equal likely. For binary input, AWGN channel, mapping the transmitted digit $x_v = 1 - 2c_v$, the intrinsic message can be obtained by

$$I_v = \ln\left(\frac{\Pr[c_v = 0 \mid y_v]}{\Pr[c_v = 1 \mid y_v]}\right) = \ln\left(\frac{p(y_v \mid x_v = +1)\Pr(x_v = +1) / p(y_v)}{p(y_v \mid x_v = -1)\Pr(x_v = -1) / p(y_v)}\right) = \frac{2y_v}{\sigma^2} .$$

Let R_{cv} represent the check-to-variable message conveyed from the check node *c* to the variable node *v*, and L_{cv} represent the variable-to-check message conveyed from the variable node *v* to the check node *c*.

1. Initialization:

$$L_{cv} = I_v \text{ for } v = 1, 2, \cdots, N \text{ and } c = 1, 2, \cdots, M \ . \tag{2.7}$$

2. Check-to-variable message passing:

Each check node *c* computes the check-to-variable message R_{cv} with variable-to-check message L_{cv}.

$$R_{cv} = \prod_{n \in N(c) \backslash v} sign(L_{cn}) \times \Psi\left\{\sum_{n \in N(c) \backslash v} \Psi(|L_{cn}|)\right\}, \tag{2.8}$$

3. Variable-to-check message passing:

Each variable node *v* computes the variable-to-check message L_{cv} with check-to-variable message R_{cv}.

$$L_{cv} = \sum_{m \in M(v) \backslash c} R_{mv} + I_v \tag{2.9}$$

4. Tentative decision and parity check:

Each variable node *v* computes the LLR message L_v and makes tentative decision.

$$L_v = \sum_{m \in M(v)} R_{mv} + I_v \tag{2.10}$$

$$z_v = 0 \text{ if } L_v \geq 0, \ z_v = 1 \text{ otherwise.} \tag{2.11}$$

If $\boldsymbol{z}\boldsymbol{H}^T = \boldsymbol{0}$, a valid codeword is found. The decoding is terminated if a valid codeword is found or the maximum decoding iteration is reached. Otherwise, go to step 2 for a new decoding iteration. To reduce the computation complexity, the *a posteriori probability* based decoding approach can be used by replacing (2.9) with (2.10).

2.3 Min-Sum Decoding Algorithms

In the literature, various approximate LLR belief propagation decoding algorithms were proposed to simplify the decoding complexity. The general approximate method can be summarized as follows. Let us rewrite (2.8) in the form of (2.12).

$$R_{mv} = \left(\prod_{n \in N(m) \backslash v} \alpha_{mn} \right) \times \ln \left(\frac{1 + \prod_{n \in N(m) \backslash v} \left(\frac{e^{\beta_{mn}} - 1}{e^{\beta_{mn}} + 1} \right)}{1 - \prod_{n \in N(m) \backslash v} \left(\frac{e^{\beta_{mn}} - 1}{e^{\beta_{mn}} + 1} \right)} \right) \tag{2.12}$$

Let a real number $x_v = \alpha_v \beta_v$, where α_v and β_v are the sign and the magnitude of x_v, respectively. Let us define

$$f(x_1, x_2) = \left(\prod_{v=1}^{2} \alpha_v \right) \times \ln \left(\frac{1 + \prod_{v=1}^{2} \left(\frac{e^{\beta_v} - 1}{e^{\beta_v} + 1} \right)}{1 - \prod_{v=1}^{2} \left(\frac{e^{\beta_v} - 1}{e^{\beta_v} + 1} \right)} \right) = \left(\prod_{v=1}^{2} \alpha_v \right) \times \ln \left(\frac{1 + e^{\beta_1 + \beta_2}}{e^{\beta_1} + e^{\beta_2}} \right) \tag{2.13}$$

It can be shown that

$$f(x1, \cdots, x_{v-1}, x_v) = \alpha_v \times sign(f(x_1, \cdots, x_{v-1})) \times \ln \left(\frac{1 + e^{|f(x_1, \cdots, x_{v-1})| + \beta_v}}{e^{|f(x_1, \cdots, x_{v-1})|} + e^{\beta_v}} \right)$$

$$= f(f(x_1, \cdots, x_{v-1}), x_v)$$

Therefore the computation of (2.12) can be recursively performed using the core computation expressed in (2.13) [2]. Using Jacobian logarithm twice [30], the core operation $f(x_1, x_2)$ becomes

$$f(x_1, x_2) = \prod_{i=1}^{2} sign(x_i) \times \left(\min(|x_1|, |x_2|) + g(|x_1|, |x_2|) \right), \tag{2.14}$$

where $g(x, y) = \ln \left(\frac{1 + e^{-|x+y|}}{1 + e^{-|x-y|}} \right)$, $x \geq 0$, and $y \geq 0$. It can be seen from Fig. 2.2 that $g(x, y) \leq 0$. Hence, $|f(x_1, x_2)| \leq \min(|x_1|, |x_2|)$ and $|f(x_1, \cdots, x_{v-1}, x_v)| \leq \min_{i=1}^{v} (|x_i|)$. Based on the observation, two widely used near optimum decoding algorithms, scaled Min-Sum algorithm (MSA) and offset Min-Sum algorithm [24][25][26], can be obtained. Because they are approximations of BP algorithm, the overall decoding procedure is similar to the

standard BP algorithm except that the check-to-variable message passing is replaced by (2.15) and (2.16), respectively. The near optimum decoding performance can be obtained with $\alpha = 0.75$ and $\beta = 0.15$ in most cases.

$$R_{cv} = \alpha \times \prod_{n \in N(c) \backslash v} \operatorname{sgn}(L_{cn}) \times \min_{n \in N(c) \backslash v} |L_{cn}|, \tag{2.15}$$

$$R_{cv} = \prod_{n \in N(c) \backslash v} \operatorname{sgn}(L_{cn}) \times \max\left(\min_{n \in N(c) \backslash v} |L_{cn}| - \beta, 0 \right). \tag{2.16}$$

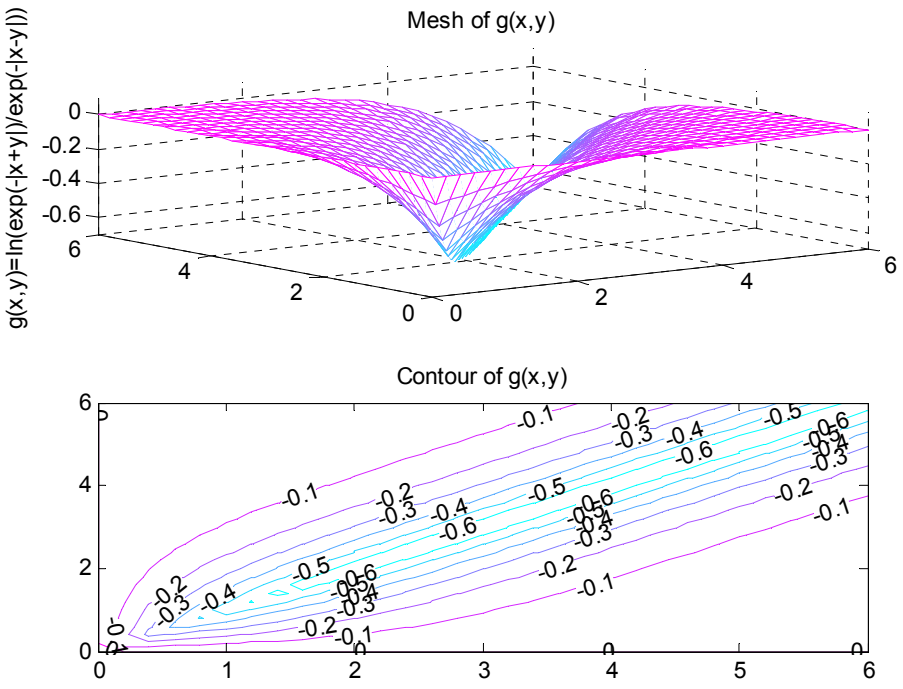


Figure 2.2 The mesh and contour plot of $g(x, y)$.

2.4 BCJR Algorithm Based Decoding Approach

As suggested by Mackay in [5], the check-to-variable message can also be computed by use the forward-backward algorithm [3]. The detailed computation approaches using the BCJR algorithm were elaborated by Zhang and Mansour [46][50][51]. The 1-components of a row in ***H*** matrix define a single parity-check (SPC) code. The LDPC code is defined by the concatenation and intersection of all SPC code. A two-state trellis can be drawn for a SPC code as shown in Fig. 2.3. The state of the trellis is the binary summation of corresponding digits. The state of the source at time *t* is denoted by S_t. The source starts in the initial state $S_t = 0$, and produces an output sequence $c_1^T = c_1, c_2, \cdots, c_T$. ending in the terminal state $S_T = 0$. The received sequence is $y_1^T = y_1, y_2, \cdots, y_T$. $\alpha_t(m)$ and $\beta_t(m)$ are forward and backward state metric, respectively. $\gamma_t(m', m)$ denotes the path metric at time *t* from $S_{t-1} = m'$ to $S_t = m$.

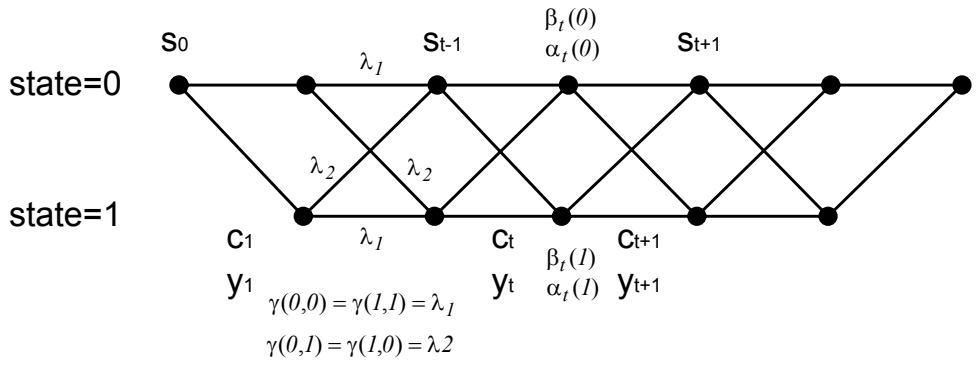


Figure 2.3 The two-state trellies of a SPC code.

Following the standard BCJR algorithm, it can be shown that,

$$\frac{\Pr[c_t = 0 \mid y_1^T]}{\Pr[c_t = 1 \mid y_1^T]} = \frac{p[c_t = 0, y_1^T]}{p[c_t = 1, y_1^T]} = \frac{\sum_{(m',m)\in\sum_t^0} p(S_{t-1} = m', S_t = m, y_1^T)}{\sum_{(m',m)\in\sum_t^1} p(S_{t-1} = m', S_t = m, y_1^T)} \tag{2.17}$$

where, Σ_t^0 is the set of all state paris that correspond to the input bit $c_t = 0$ at time *t*. Similarly, Σ_t^1 is the set of all state paris that correspond to the input bit $c_t = 1$. The equation (2.17) can be rewritten as (2.18).

$$\frac{\Pr[c_t = 0 \mid y_1^T]}{\Pr[c_t = 1 \mid y_1^T]} = \frac{\sum_{(m',m)\in\Sigma_t^0} \alpha_{t-1}(m')\gamma_t(m',m)\beta_t(m)}{\sum_{(m',m)\in\Sigma_t^1} \alpha_{t-1}(m')\gamma_t(m',m)\beta_t(m)}, \tag{2.18}$$

where

$$\alpha_t(m) = p(S_t = m, y_1^{t)} = \sum_{m'} \alpha_{t-1}(m')\gamma_t(m',m) \tag{2.19a}$$

$$\beta_t(m) = p(y_{t+1}^T \mid S_t = m) = \sum_{m'} \beta_{t+1}(m')\gamma_t(m,m') \tag{2.19b}$$

$$\gamma_t(m',m) = p(S_t = m, y_t \mid S_{t-1} = m') \tag{2.19c}$$

The boundary conditions are $\alpha_0(0) = \beta_T(0) = 1$ and $\alpha_0(1) = \beta_T(1) = 0$. For the convenience of implementation, equations (2.18) and (2.19) can be reformulated in log-domain. After some mathematical manipulation, equation (2.20) can be obtained.

$$\ln\left(\frac{\Pr[c_t = 0 \mid y_1^T]}{\Pr[c_t = 1 \mid y_1^T]}\right)$$

$$= \ln\left(\frac{\Pr(c_t = 0)}{\Pr(c_t = 1)}\right) + \ln\left(\frac{p(y_t \mid c_t = 0)}{p(y_t \mid c_t = 1)}\right) + \ln\left(\frac{\sum_{(m',m)\in\Sigma_t^0} e^{\alpha_{t-1}(m')+\beta_t(m)}}{\sum_{(m',m)\in\Sigma_t^1} e^{\alpha_{t-1}(m')+\beta_t(m)}}\right) \tag{2.20}$$

The extrinsic information expressed in the last portion of (2.20) is the check-to-variable information. The variable-to-check information is fed to the check node as the *a priori* information for computing the branch metric $\gamma_t(m',m)$. For VLSI implementation, the metric differences rather than the absolute metrics are used to reduce the memory requirement for state metrics and maximize the dynamic range of the metrics. The BCJR algorithm for check node computation can be viewed as a different data scheduling in

belief propagation algorithm. Therefore, the decoding approach discussed in the section is equivalent to the belief propagation algorithm presented in Section 2.2.

2.5 Bit Flipping and Weighted Bit Flipping Based Algorithms

In the early 1960s, the Bit-Flipping (BF) algorithm [1] was introduced. It has very low decoding complexity since only simple logical operations are needed. However, it suffers from significant performance loss from those soft decoding approaches such as the SPA and various MSAs. To narrow the performance gap, a weighted Bit-Flipping (WBF) algorithm [21] was proposed, in which reliability information was incorporated.

The decoding procedures of WBF-based algorithms include an initialization and a number of iteration steps. In the initialization of the original WBF algorithm, the reliability of each parity check equation is calculated using (2.18) and a binary hard-decision is made for each bit. In an iteration step, two computation steps, check-sum updating and bit flipping, are performed. In the check-sum updating step, the check-sum vector is given by (2.19). If $\boldsymbol{s} = \mathbf{0}$, the decoding procedure is terminated and $\boldsymbol{z}$ is taken as the estimated codeword. In the bit-flipping step, a weighted check sum is computed as (2.20) for each bit position n. The hard-decision bit z_l corresponding to the maximum E_n is flipped. The index value l is given by (2.21).

$$w_m = \min_{i \in N(m)} | y_i |, m \in [1, M] \tag{2.18}$$

$$\boldsymbol{s} = \boldsymbol{z}\boldsymbol{H}^T \tag{2.19}$$

$$E_n = \sum_{j \in M(n)} (2s_j - 1) w_n, \; n \in [1, N] \tag{2.20}$$

$$z_l = z_l \oplus 1, \quad where, \; l = \arg \max_{n \in [1,N]} E_n \tag{2.21}$$

To further reduce the performance loss, the modified WBF (MWBF) algorithm [27] was proposed by incorporating both the check constraint messages and the intrinsic

message for each bit. The weighted check sum is computed using (2.22) instead of (2.20) in the bit-flipping step. The optimal value of α varies for different codes and decreases slowly as the SNR increases.

$$E_n = \sum_{j \in M(n)} (2s_j - 1)w_n - \alpha \mid y_n \mid, \ n \in [1, N] \tag{2.22}$$

Further improvements were presented in [64] and [65]. The improved modified WBF (IMWBF) algorithm discussed in [65] has the best decoding performance in general and largest complexity among these modifications of WBF algorithm. The IMWBF algorithm adopts (2.23) to calculate the reliability of each parity check equation in the initialization and (2.24) to compute the weighted check sum for each bit in the bit flipping step.

$$w_{nm} = \min_{i \in N(m) \backslash n} \mid y_i \mid, m \in [1, M], n \in N(m) \tag{2.23}$$

$$E_n = \sum_{j \in M(n)} (2s_j - 1)w_{nj} - \alpha \mid y_n \mid, \ n \in [1, N] \tag{2.24}$$

3 MEMORY-EFFICIENT DECODER ARCHITECTURE

In general, LDPC codes achieve outstanding performance only with large code word lengths (e.g., $N \geq 1000$ *bits*). Thus, the memory part normally dominates the overall hardware of a LDPC codec. A memory efficient serial decoder was presented in [34]. The decoding throughput of each tile is less than 5.5Mbps. Partially parallel decoder architectures, which can achieve a good trade-off between hardware complexity and decoding throughput, are more appropriate for practical applications. This chapter depicts a memory efficient partially parallel decoder architecture for high rate QC-LDPC codes, which exploits the data redundancy of soft messages in the Min-Sum decoding algorithm. In general, over 30% memory can be saved. In addition, the proposed architecture can be extended to other block-based LDPC codes, e.g., (general) permutation matrix based LDPC codes.

3.1 The Performance of High Rate QC-LDPC Code

For LDPC codes with the same size of $\boldsymbol{H}$ matrix, the error floor is significantly lower when the variable node degree is increased. Fig. 3.1 shows the Bit Error Rate (BER) and Frame Error Rate (FER) of two (4608, 4096) rate-8/9 regular QC-LDPC codes with variable node degree 3 and 4, respectively. The scaled Min-Sum algorithm with a scaling factor 0.75 is used in the simulation. It can be seen from Fig. 3.1 that the BER and FER of the code with variable node degree 4 is more than an order of magnitude lower than that of the code with variable node degree 3 at SNR=4.5dB. In conventional designs, more memories are needed to store the extrinsic messages for a code with larger variable and check degree. We proposed a memory efficient partially parallel architecture that only causes negligible increase of memory size when the density of 1-entries in the $\boldsymbol{H}$ matrix is increased.

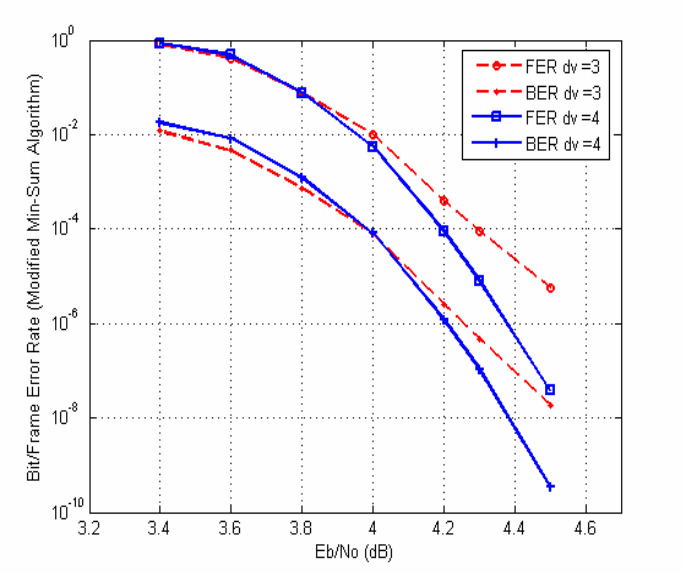


Figure 3.1 BER/FER performance of the (4608, 4096) QC-LDPC codes.

3.2 The Rearranged (Scaled) Min-Sum Algorithm

In the modified Min-Sum decoding algorithm, two classes of computation units, the variables nodes and the check nodes, iteratively exchange soft messages with each other through the edges in the Tanner graph. In the check-to-variable message updating phase, each check node c computes the check-to-variable messages R_{cv} with variable-to-check messages L_{cv} as (2.15) which is written as (3.1)

$$R_{cv}^{(k)} = \alpha \times \prod_{n \in N(c) \backslash v} \operatorname{sgn}(L_{cn}^{(k-1)}) \times \min_{n \in N(c) \backslash v} | L_{cn}^{(k-1)} |, \qquad (3.1)$$

where α is a scaling factor. The superscript (k) is used to indicate that the data is generated in the *k-th* iteration.

In the variable-to-check message updating phase, each variable node *v* computes the L_{cv} messages using the R_{cv} messages and the intrinsic messages I_v using (2.9). The equation (2.9) is rewritten in (3.2).

$$L_{cv}^{(k)} = \sum_{m \in M(v) \setminus c} R_{mv}^{(k)} + I_v . \tag{3.2}$$

In the conventional decoding procedure, two classes of extrinsic messages in their individual form are involved. On the other hand, it can be observed from (3.1) that the magnitudes of the R_{cv} messages corresponding to one row of $\boldsymbol{H}$ matrix have only two possible values. By extending the idea presented in [47] and [51], a rearranged decoding procedure of the Min-Sum algorithm is developed to reduce the memory requirement for extrinsic messages. The new decoding procedure is expressed as follows.

$$R_{cv}^{(1)} = \alpha \times \prod_{n \in N(c) \setminus v} \operatorname{sgn}(I_n) \times \min_{n \in N(c) \setminus v} | I_n |, \tag{3.3}$$

$$R_{cv}^{(k)} = \alpha \times \prod_{n \in N(c) \setminus v} \operatorname{sgn}(S_n^{(k-1)} - R_{cn}^{(k-1)}) \times \min_{m \in N(c) \setminus v} | S_n^{(k-1)} - R_{cn}^{(k-1)} |, \tag{3.4}$$

$$S_v^{(k)} = I_v + \sum_{m \in M(v)} R_{mv}^{(k)} . \tag{3.5}$$

In the $R_{cv}^{(k)}$ message updating phase, the required inputs $L_{cn}^{(k-1)}$, $n \in M(v) \setminus c$, are not directly retrieved from memory. Instead, they are computed using $S_n^{(k-1)} - R_{cn}^{(k-1)}$ as shown in (3.4). The regular variable-to-check message updating phase is replaced by the column sum updating phase. The column sum S_v is calculated using (3.5) and stored in memory. Based on the new decoding procedure, optimized low complexity decoding architectures are developed.

3.3 The Memory Efficient Decoder Architecture

In the proposed design, the check-to-variable messages R_{cv} and the column sum S_v are stored in separated memories. All the R_{cv} messages corresponding to one row of $\boldsymbol{H}$

matrix are stored in a compressed form to significantly reduce memory requirement. In general, the parity check matrices ***H*** of QC-LDPC codes contain cyclically shifted identity submatrices, zero sub-matrices and compound circulant submatrices. Each of the compound matrices consists of 2 superimposed cyclically shifted identity matrices. For simplicity, the cyclically shifted identity matrix and the compound circulant matrix are called weight-1 and weight-2 circulant matrix, respectively. We will deal with various cases in the following.

3.3.1 Parallel Decoder Architecture for H Matrix Consisting of Weight-1 Circulant and Possible Zero Submatrices

Fig. 3.2 shows the decoder architecture for QC-LDPC codes with ***H*** matrix consisting of weight-1 circulant matrices and possible zero matrices. The *R-memory* module is used to store the R_{cv} messages. The messages corresponding to one row of ***H*** matrix are stored into one entry of *R-memory* in a compressed form with four elements.

1) The smallest magnitude *m1*.

2) The second smallest magnitude *m2*.

3) The index of the smallest magnitude *index*.

4) The signs of all soft messages of the row.

To recover the individual R_{cv} messages from their compressed forms, data distributor is introduced. Each *S-memory* and *I-memory* modules are used to respectively store the column sum S_v and the intrinsic messages I_v corresponding to a block column of the ***H*** matrix. The Check-Node Unit (CNU) performs the computation expressed in (3.3) and (3.4) in the check-to-variable message updating phase. The Variable-Node Unit (VNU) works in both decoding phases. It performs the computation $S_n^{(k-1)} - R_{cn}^{(k-1)}$ in the $R_{cv}^{(k)}$ message updating phase and performs (3.4) in the column sum $S_v^{(k)}$ updating phase. The

column sum S_v is accumulated for each column of $\boldsymbol{H}$ matrix. The decision and parity equation check unit is introduced for tentative decision and parity check computation.

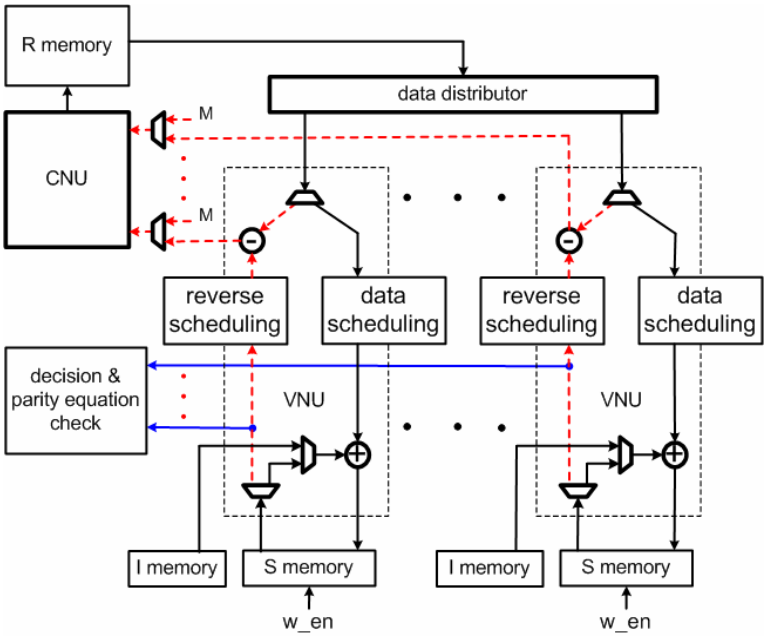


Figure 3.2 The partially parallel decoder architecture for $\boldsymbol{H}$ matrix containing weight-1 and possible zero submatrices.

To facilitate multiple data accesses per clock cycle in a *p*-parallel (p>1) architecture, the data in *p* adjacent rows or columns are stored into one memory entry. In the *R-memory*, check-to-variable messages R_{cv} in *p* adjacent rows are stored in one *R-memory* entry. Similarly, in one *S-memory* (*I-memory*) entry, column sums S_v (intrinsic messages I_v) in *p* adjacent columns are stored in one memory entry. Because the data stored in the *R-memory* are in the row order while the data stored in the *S-memory* are in the column order, data scheduling and reverse scheduling units are introduced to resolve the data access conflicts. In Fig. 3.2, for clarity, we use one symbol (*e.g.*, adder, MUX, and data distributor) to represent *p* components for parallel processing case.

Assuming that the $\boldsymbol{H}$ matrix is a $J \times K$ block matrix, each submatrix is a $t \times t$ weight-1 circulant matrix, and *p*-parallel processing is adopted, the message updating procedure is as follows. In initialization, the intrinsic messages in *I-memory* are dumped into *S-memory* for iterative decoding. It takes $\lceil t/p \rceil$ clock cycles. In the k^{th} iteration, 2 phases of decoding are performed.

In check-to-variable message updating phase, for each row c, two kinds of messages $R_{cv}^{(k-1)}$ and $S_v^{(k-1)}$, where $v \in N(c)$, are sent to K VNUs via the path indicated by the dashed line in Fig. 3.2. Then, all of the intermediate results $L_{cv}^{(k-1)}$ calculated by the K VNUs are sent to a CNU. Finally, the compressed $R_{cv}^{(k)}$ messages are generated and stored into one entry of the *R-memory*. It takes $J \times (\lceil t/p \rceil + 1)$ clock cycles.

In the column sum updating phase, the summation of I_v and $R_{cv}^{(k)}$ messages for each column *v* of $\boldsymbol{H}$ matrix is accumulated. For each row *c*, the compressed $R_{cv}^{(k)}$ messages are read out from *R-memory* module. Next, the individual $R_{cv}^{(k)}$ messages recovered by the data distributor are sent to *K* VNUs for accumulating $S_v^{(k)}$ messages via the path indicated by the solid line. If *c* is in the first block row, I_v and R_{cv} are added together and stored in *S-memory*. When *c* is in other block rows, R_{cv} is accumulated to S_v. At the end of this phase, the final column sum $S_v^{(k)}$ corresponding to all columns of the $\boldsymbol{H}$ matrix are accumulated in the K *S-memory* modules. $J \times (\lceil t/p \rceil + 1)$ clock cycles are needed for this phase.

If the maximum iteration number is set to n, totally, it takes $\lceil t/p \rceil + n \times 2 \times J \times (\lceil t/p \rceil + 1)$ clock cycles to complete the decoding for one code block.

If $\boldsymbol{H}$ matrix contains zero submatrices, data received by the CNU and VNUs corresponding to zero submatrices must be filtered out. Therefore, additional data path for

zero matrices are introduced as the following. When computing R_{cv} messages, the largest positive value *M* determined by the word length of soft message is sent to the CNU from the inserted MUXs. According to (3.4), the value *M* has no effect on the computation in the check-to-variable message updating phase. When accumulating column sum S_v, to avoid adding an undesired data to the summation, the write enable signal *w_en* for the associated *S-memory* must be disabled. Apparently, if the ***H*** matrix only consists of weight-1 circulant matrices, all the MUXs connected to the CNU can be removed.

Fig. 3.3 shows the structure of data distributor which is introduced to convert the R_{cv} messages corresponding to a row of ***H*** matrix into their true values. The data read from an *R-memory* entry is composed of four elements as mentioned before. On the output side, the relative location of *m2* is determined by the index decoder. *m1* is distributed to other locations. Because the order of the sign bits in *signs* part is the same as that of the R_{cv} messages corresponding to a row of ***H*** matrix, the sign bits can be simply distributed. It is convenient to use two's complement data representation in column sum computation. Therefore, sign-magnitude to two's-complement conversion unit is needed at input of VNU.

For a quantitative comparison of the memory size between the proposed architecture and the conventional approaches [38][43] [53] [55], let us consider a (4608, 4096) (4, 36) rate 8/9 regular QC-LDPC code designed for read channel. In the traditional approaches, the two classes of messages in their true values are alternately stored into the common extrinsic memory modules. Assuming that 6-bit quantization is used, the required memories are summarized in Table 3.1 and Table 3.2. It can be seen that the total memory for *R-memory*, *S-memory*, *I-memory* and estimated codeword is 163k bits with the proposed approach. The total required memory for the extrinsic messages, intrinsic messages and estimated codeword is 258k bits using conventional approaches. Thus,

nearly 37% of memory is reduced in this case. If the code rate is not so high, the memory savings in percentage would be relatively smaller. In general, for high rate LDPC codes, *e.g.,* rate > 2/3, more than 30% of memory reduction can be achieved with the proposed architecture.

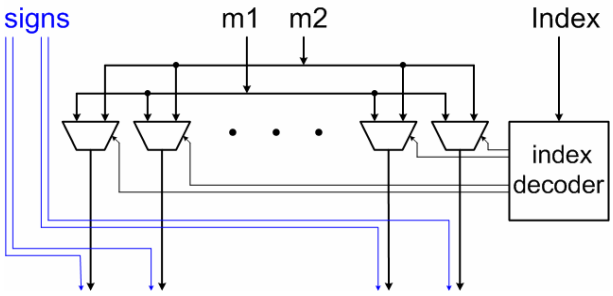


Figure 3.3 The structure of data distributor.

TABLE 3.1 MEMORY NEEDED BY THE PROPOSED ARCHITECTURE

Memory Component	Memory Size (bits)
The *R-memory* modules (DP)	128x4x2x(5*2+36+6)=53,248
The *S-memory* modules (DP)	128x36x2x(6+2)=73,728
The *I-memory* modules (SP)	128x36x1x6=27,648
The decoded bits (DP)	128x36x2x1=9,216
	Total=163,840

TABLE 3.2 MEMORY NEEDED BY THE TRADITIONAL APPROACHES

Memory Component	Memory Size (bits)
The extrinsic memory modules (DP)	128x36x4x2x6=221,184
The intrinsic memory modules (SP)	128x36x1x1x6=27,648
The decoded bits (DP)	128x36x2x1=9,216
	Total=258,048

In the above discussion, dual-port memories are used to support simultaneous memory read and write operations for each memory bank. Because adjacent memory entries are sequentially accessed (for read or write operation), we can also use a single-port memory and two buffers as shown in Fig. 3.4 to support simultaneous two memory accesses. The data width of the single-port memory needs to be doubled compared to that

of the *ReadData* port. The memory access data flow is as the follows. At cycle c_k, both port *ReadData* and buffer *D1* get data from the same entry of the single-port memory. In the same time, data is written from port *WriteData* to buffer *D2*. At next cycle c_{k+1}, port *ReadData* gets data from buffer *D1*. Simultaneously, data from port *WriteData* and buffer *D2* are written back into the same entry of the single-port memory. This procedure is repeated in following cycles. In this way, the needed hardware size for a memory bank is significantly reduced compared to that of using dual-port memories.

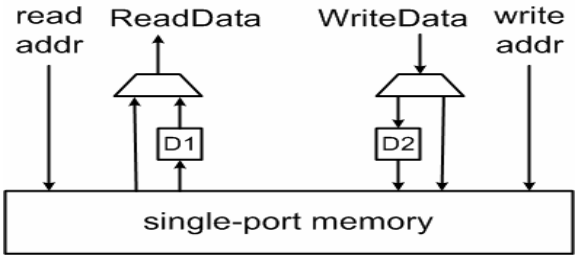


Figure 3.4 The structure of single-port memory supporting simultaneous read and write operation.

3.3.2 Architecture for H Matrices with Weight-1, Weight-2 Circulant Matrices and Zero Matrices

We next discuss a more general case in which ***H*** matrix may also contain a small portion of weight-2 submatrices. Similar to the method illustrated above, for *p*-parallel processing, messages corresponding to *p* adjacent rows (columns) are stored in one memory entry. For each weight-2 circulant matrix, *2p* messages stored in both *S-memory* and *R-memory* are accessed at each clock cycle in both decoding phases. The tricky issue of partially parallel processing is how to schedule the data order for the *2p* messages. Each weight-2 circulant matrix can be decomposed into two weight-1 submatrices. Consequently, the parallel processing techniques discussed before can be utilized.

The decoder architecture for LDPC codes with ***H*** matrices containing weight-1, weight-2 circulant matrices and zero matrices is shown in Fig. 3.5. Because the matrix decomposition method is used, a new block row with zero submatrices and a small portion of weight-1 circulant submatrices is introduced for each original block row in the ***H*** matrix. Thus, to generate check-to-variable messages R_{cv} from the data corresponding to the two decomposed rows, an additional processing unit, CNUb, is needed for the new block row. For the considered ***H*** matrix, the number of the block columns containing weight-2 circulant matrices is small. By block column permutation, these block columns can be arranged together. In this way, the size of CNUb is much smaller than CNUa. The final compressed R_{cv} messages corresponding to one row of ***H*** matrix are assembled by the merge unit using the outputs from the two CNUs. To facilitate distributing the R_{cv} messages from the compressed form to two decomposed rows, 6 elements in the compressed message for one row are needed, *i.e.*, 1) the smallest magnitude *m1*, 2) the second smallest magnitudes *m2*, 3) and 4) the index of the smallest magnitude for the two decomposed row *I_a*, and *I_b*, 5) and 6) the sign bits for the two decomposed row *sgn_a* and *sgn_b*. The second data distributor is introduced to distribute R_{cv} message onto the shorter decomposed block rows. If the index of the second smallest magnitude for a decomposed row is set to an invalid value by the merge unit, only the smallest magnitude can be distributed to that row.

For a block column containing weight-2 circulant matrix, to recover the two variable-to-check messages L_{cv} at each clock cycle, two VNUs are needed. Similarly, two message accumulation operations are performed at each clock cycle in the S_v message updating phase. Therefore, for *S-memory,* either two-port memory with the technique discussed in Section 3.3.1 or register array is required to support two read and two write operations in the same clock cycle. On the other hand, if most of block columns of the ***H***

matrix containing weight-2 circulant matrices, the presented approach is not suited in the sense of area-efficiency. Instead, the architecture proposed in [36] can be employed.

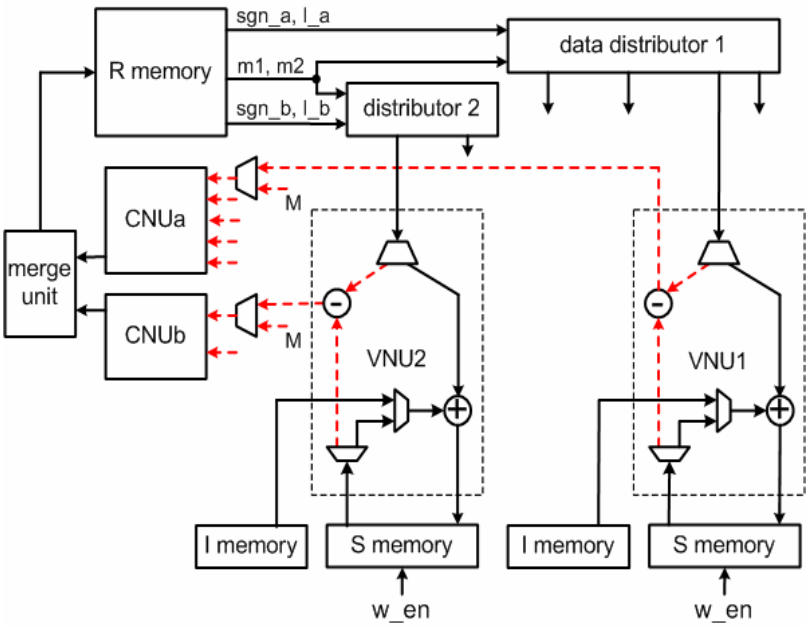


Figure 3.5 The decoder architecture for ***H*** matrix containing weight-2 circulant matrices

There is a special case for the *p*-parallel processing. For a weight-2 circulant matrix, if the difference between the two cyclic shifting offsets is less than *p/2*, the two R_{cv} message corresponding to two 1-entries in one column may need to be added together and accumulated into the summation in one clock cycle. Thus, a multiplex network is needed to select different structures of VNU.

3.4 Optimization on the Partially Parallel Decoder Architecture

3.4.1 The Optimized CNU

The critical task of CNU is to find the two smallest magnitudes from all input data and identify the relative position of the input data with the smallest magnitude. In this

section, an optimized 2×2 Pseudo Rank Order Filter (PROF) is proposed. Then, an efficient CNU based on the PROF is presented to minimize hardware complexity.

The 2×2 PROF sorts two presorted sequences and decides which sequence containing the smallest data with respect to their magnitudes. Only the smallest and the second smallest magnitudes are sent out. Fig. 3.6 shows the architecture of the PROF. Input [a2 a1] and [b2 b1] are two presorted sequences such that $a2 \geq a1$ and $b2 \geq b1$. All elements in the two input sequences are non-negative number. On the output side, *m1* and *m2* stand for the smallest and the second smallest input data, respectively. *index* is used to indicate which group contains the smallest input data. Its value is 0 if the smallest input is in the group [a2 a1], otherwise, it is 1. It can be observed that the smallest magnitude must be either a1 or b1. The second smallest magnitude can be selected from a2, b2, and the intermediate comparison result of a1 and b1. Therefore, only one stage compare-and-swap unit plus a very simple combinational logic is used to perform the task of the PROF.

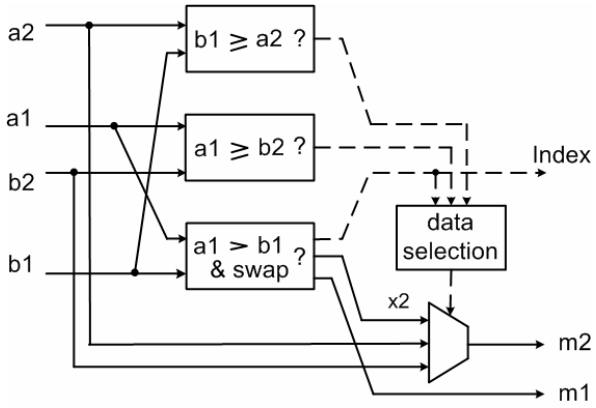


Figure 3.6 The structure of pseudo rank order filter.

The architecture of a CNU with eight inputs is shown in Fig. 3.7. Input data are variable-to check messages and represented in sign-magnitude format. On the output side, the scaled smallest and second smallest magnitude, the relative position of the second

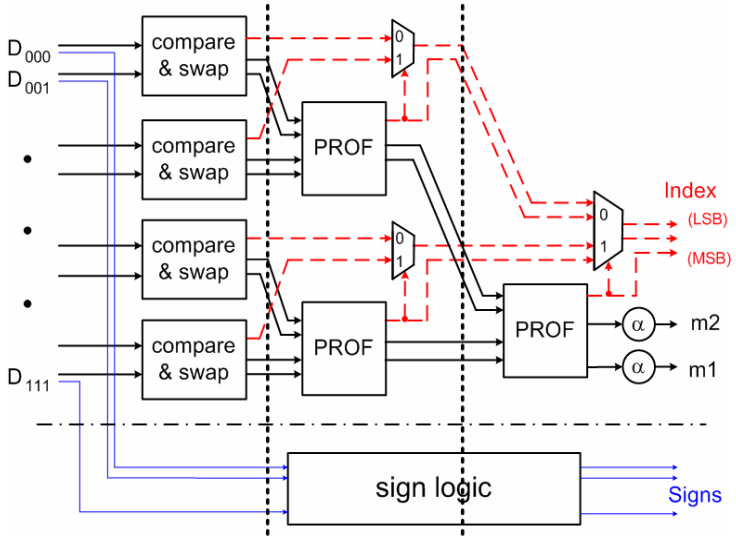


Figure 3.7 The architecture of the optimized CNU.

smallest magnitude, and the signs of all data computed in (3.4) are denoted as *m1*, *m2*, *index*, and *signs*, respectively. In Fig. 3.7, the part above the dash-dotted line performs the data sorting task. The compare-and-swap unit is used to compare two input data. If the larger magnitude is in the upper position, the comparison result indicated by the dashed line is 1, otherwise, it is 0. In the same time, the larger magnitude is placed at upper output position. In the optimized CNU, the bits of *index* are aggregated stage by stage. To illustrate the aggregation procedure, let us assume that the input data with the smallest magnitude has an index of 101. In the first stage, the comparison result of the third compare-and-swap unit (from top to bottom) is 1. In the second stage, the *index* bit of the second PROF is 0. In the same time, it is used to select the index bit generated by the previous stages. Therefore, the aggregation result after the second stage is "10". In the final stage, the index bit of the PROF is 1. This index bit is used to select the "10" generated from the previous stages. In this way, the final value of index is "101". The part below the dash-dotted line is for computing the sign bit of each R_{cv} message as in the

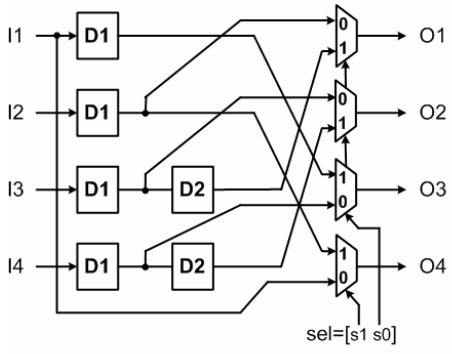
conventional approaches. The relative position of each computed sign bit is not changed when they are grouped together. To reduce the critical path of CNU, pipeline stages are inserted as indicated by the vertical dashed lines.

3.4.2 The Optimized Data Scheduling Unit

The data scheduling (reverse scheduling) unit plays the key role in the partially parallel processing architecture. In [47], a component performing a similar task was implemented with a combination of data concatenation unit and cyclic shifter, which significantly increases the hardware cost and computation latency. In this chapter, an efficient data scheduling unit is proposed by exploiting the structure of LDPC codes. For QC-LDPC codes, the number of non-zero matrices in one block column of $\boldsymbol{H}$ matrix is very limited, typically around four. It implies that the number of cyclic shift patterns in a block column is limited. Hence, a very simple switching block can be employed to resolve the data access conflict for each circulant matrix. If there are W non-zero matrices in a block column of $\boldsymbol{H}$ matrix, W switching blocks are needed for the block column. Thus, a $W:1$ multiplexer is used to select a switching block for a circulant permutation matrix.

To illustrate the design of data scheduling unit, let us use a 13×13 circulant matrix with shift offset 7 without loss of generality. Fig. 3.8(a) shows the structure of data scheduling unit for 4-parallel processing. It has four inputs indicated by I1…I4 and 4 outputs indicated by O1…O4. The corresponding data flow is shown in Fig. 3.8(b). At clock cycle 0, 4 messages corresponding to row 0, 1, 2, and 3 are sent to the input of the switch block. Their column indices are 7, 8, 9, and 10. At cycle 1, messages corresponding to row 4, 5, 6, and 7 are sent to the input. Their column indices are 11, 12, 0, and 1. The control signal for MUX array is set to “00”. The messages with column indices 8, 9, 10,

and 11 can be outputted. At cycle 2, the outputted message is for the last column 12. The other three values are discarded. Other entries in the figure can be read in a similar way. It can be seen that the input messages are from 4 adjacent rows and the output messages are for 4 adjacent columns. The scheduling unit introduces one clock cycle delay from input to output. The data reverse scheduling unit can be designed in a similar way. In the proposed decoder architecture, the two message updating phases are not overlapped. Due to the similarity in the structure of data scheduling and reverse scheduling units, the two components can be combined into one unit. It works alternately in the two message updating phases.



(a)

cycle	input	[s1 s0]	D1	D2	output
0	m7 m8 m9 m10				
1	m11 m12 m0 m1	[0 0]	m7 m8 m9 m10		m8 m9 m10 m11
2	m2 m3 m4 m5	[0 0]	m11 m12 m0 m1		m12 x x x
3	m6 x x x	[1 1]	m2 m3 m4 m5	 m0 m1	m0 m1 m2 m3
4	m7 m8 m9 m10	[0 1]	m6 x x x	 m4 m5	m4 m5 m6 m7

(b)

Figure 3.8 The data scheduling unit. (a) structure (b) data flow

It should be mentioned that overlapping the two decoding phases will nearly double the decoding throughput while introducing some extra hardware. In addition, the proposed

architecture can be extended to other block–based LDPC codes such as general permutation matrix based LDPC codes.

3.4.3 The Optimized Data Merge Unit

Fig. 3.9 shows the architecture of the optimized data merge unit to calculate the smallest magnitude and the second smallest magnitude from the outputs of the two CNUs. On the input side, the data set a_m1, a_m2, a_I, a_sgn and a_sgn_prod are the output of CNUa, which represent the smallest and the second smallest magnitude, the index of the second smallest magnitude, the sign bits and the product of sign bits. The data set b_m1, b_m2, b_I, b_sgn and b_sgn_prod are the output of CNUb in the same meaning. On the output side, the index I_a and I_b are used to control the data distribution for the two decomposed rows, respectively. If the smallest magnitude needs to be distributed to all the positions of a decomposed row, the index value for the decomposed row is set to a value z, where z is larger than the number of submatrices in this decomposed row. Otherwise, the index value shows the position to which the second smallest magnitude will be distributed.

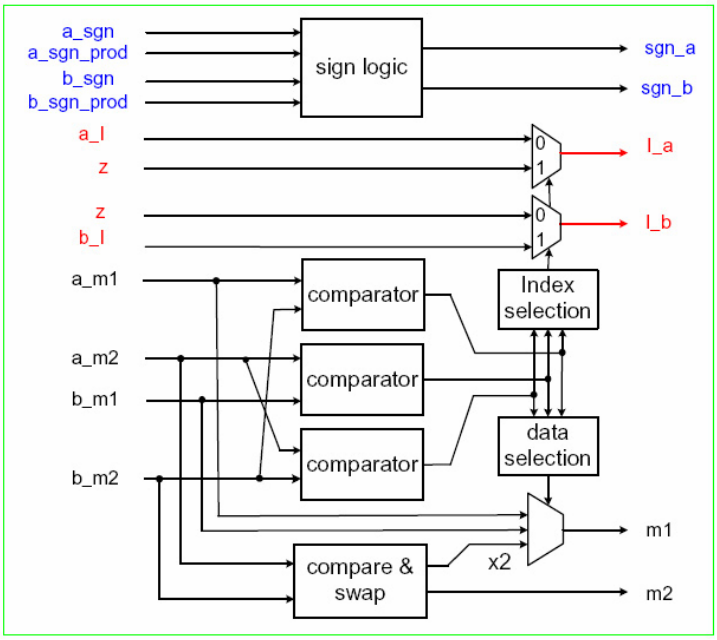


Figure 3.9 The structure of the merge unit.

3.5 Summary

A memory efficient partially parallel decoder architecture suited for (modified) Min-Sum decoding algorithm for QC-LDPC codes is proposed. By rearranging the decoding procedure of the Min-Sum algorithm and exploiting the data redundancy of extrinsic messages, generally over 30% memory reduction can be achieved over traditional designs. To minimize the computation delay, a low complexity CNU is developed. To facilitate parallel processing, an efficient data scheduling structure is proposed. The approach facilitates the applications of high variable degree and/or high rate LDPC codes in area/power sensitive high speed communication systems.

4 EFFICIENT VLSI DESIGN OF HIGH THROUGHPUT LDPC DECODERS

In the next generation communication systems, the target data rates range from a few hundred Mbit/sec to several Gbit/sec. To achieve those very high decoding throughput, a large amount of computation units are required. Because of the very high decoding parallelism, a complex interconnect network is required which consumes a significant amount of silicon area and power. In a pioneer design of high throughput LDPC decoder [57], the power dissipation of the decoder was largely determined by the switching activity of these wires. The utilization of chip area was only 50%.

In this chapter, design issues for high throughput LDPC decoders are discussed. Three LDPC decoder architectures which are accommodated to different types of LDPC codes and implementation technologies are presented. We propose an algorithmic transformation to facilitate the significant routing complexity reduction for LDPC decoders. Based on the algorithmic transformation, an efficient message passing decoder architecture for permutation matrices based LDPC code is proposed. Then, by exploiting the regularity in parity check matrices of QC-LDPC codes, we develop a high-throughput low-complexity decoder architecture for generic QC-LDPC codes. Finally, we demonstrate an FPGA implementation of a low complexity, high speed decoder for Euclidean geometry (EG) based QC-LDPC codes.

4.1 Efficient Message Passing Architecture

Recently, several techniques were introduced to reduce the total amount of interconnect wires in high throughput LDPC decoders. In [58], message passing of SPA was rescheduled. A check node only broadcasts a summation message to its neighboring

variable nodes. The needed check-to-variable messages for a variable node are recovered using check node summation messages and variable-to-check messages buffered in the variable node itself. From the hardware implementation point of view, a lot of wires are shared for message passing. In [59], SPA was further reformulated such that only summation messages are passed among check nodes and variable nodes. Separate variable-to-check messages are buffered in variable nodes. Similarly, separate check-to-variable messages are buffered in check nodes. This scheme can significantly mitigate the routing congestion in a high throughput LDPC decoder. However, because both computation units and memory for soft messages are duplicated, the area and power efficiency are largely sacrificed.

We propose an efficient message passing decoder architecture using MSA for permutation matrices based LDPC codes [18]. MSA is reformulated to facilitate significant reduction of routing complexity and memory usage. A high throughput decoder architecture for permutation matrices based LDPC code is presented. To further reduce hardware complexity, an optimized non-uniform quantization scheme using only 3 bit to represent each soft message is investigated.

4.1.1 Efficient Message Passing Schemes with Min-Sum Algorithm

Reformulated Min-Sum Decoding Algorithm

To reduce the interconnect complexity mentioned before, MSA can be reformulated for the following message passing scheme. In the variable-to-check message passing phase, a variable node *v* does not send separate variable-to-check messages L_{vc} to its neighboring check nodes. Instead, the column sum, L_v, is sent to its neighboring check nodes. In addition, only S_{vc} (i.e., the sign of L_{vc}) computed in the previous iteration is sent to the check node *c*, where, $c \in M(v)$.

In a check node *c*, the $R_{cv}^{(k-1)}$ messages that computed in the $(k-1)^{th}$ iteration are stored in a compressed format for recovering the needed input $L_{vc}^{(k)}$ for the k^{th} iteration using (4.1). The superscript *(k)* indicates that the data is generated in the k^{th} iteration.

$$L_{vc}^{(k)} = L_{v}^{(k)} - (S_{vc}^{(k-1)} \times S_{c}^{(k-1)} \times | R_{cv}^{(k-1)} |) \tag{4.1}$$

where, $S_c^{(k-1)}$ is the 1-bit product of S_{vc}. In the check-to-variable message passing phase, a check node *c* does not send out separate check-to-variable messages to its neighboring variable nodes. Instead, all R_{cv} messages are sent out in a compressed format, *i.e.*, the smallest magnitude, the second smallest magnitude, the index of the smallest magnitude, and the 1-bit product of all S_{vc} (denoted as $min1_c$, $min2_c$, $index_c$, and S_c, respectively). In a variable node *v*, the sign bits, $S_{vc}^{(k-1)}$, are stored for recovering the needed input $R_{cv}^{(k-1)}$ in the k^{th} iteration using (4.2) and (4.3).

$$| R_{cv}^{(k-1)} | = \begin{cases} min2_c^{(k-1)}, & if\ V = index_c \\ min1_c^{(k-1)} & otherwise \end{cases} \tag{4.2}$$

$$sign(R_{cv}^{(k-1)}) = S_c^{(k-1)} \times S_{vc}^{(k-1)} \tag{4.3}$$

where, *V* is the index of the block column that the variable node *v* belongs to.

Fig. 4.1(a) illustrates the structure of a variable node unit (VNU). The inputs are compressed check-to-variable messages from three CNUs. The needed $R_{cv}^{(k-1)}$ messages are recovered by equal-and-select (E&S) unit using (4.2). In the output, the sign bit of each $L_{vc}^{(k-1)}$ is sent to the check node *c* where, $c \in M(v)$. The magnitude of each $L_{vc}^{(k)}$ is not needed. Instead, only the column sum $L_v^{(k)}$ is broadcasted to its neighboring check node units (CNUs). Fig. 4.1(b) shows the structure of a CNU performing (4.1) and (2.15). The inputs are $L_v^{(k)}$ and $S_{vc}^{(k-1)}$ from six VNUs. The needed $L_{vc}^{(k)}$ messages are recovered using

(4.1). The compressed check-to-variable messages are broadcasted to its neighboring VNUs.

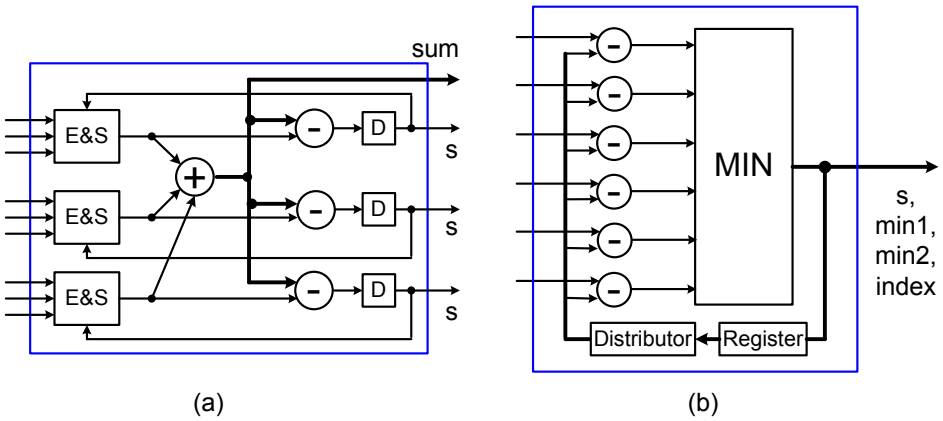


Figure 4.1 Computation units using reformulated Min-Sum algorithm.
(a) Variable node unit (b) Check node unit

The proposed approach significantly reduces the amount of outgoing wires of a computation unit. For example, the ***H*** matrix of the LDPC code discussed in Section 4.1.2 has row weight 32 and column weight 6. If using 4-bit quantization, the conventional method [44][56] needs 24 outgoing wires for a VNU and 128 outgoing wires for a CNU. Using the proposed approach, one VNU needs (5+6)=11 outgoing wires; and one CNU needs (5+3+3+1)=12 outgoing wires. Hence, 54% outgoing wires of each VNU and 90% outgoing wires of each CNU are reduced. Thus, significant reduction of routing complexity and memory usage can be obtained.

Reformulated APP-based Min-Sum Algorithm

To reduce decoding complexity, *a posteriori probability* (APP) based Min-Sum algorithm was presented [24]. The variable-to-check and check-to-variable message passing phases are formulated in (4.4) and (4.5), respectively.

$$L_{vc} = L_v = I_v + \alpha \times \sum_{m \in M(v)} R_{mv} \tag{4.4}$$

$$R_{cv} = \prod_{n \in N(c) \setminus v} sign(L_{nc}) \times \min_{n \in N(c) \setminus v} | L_{nc} | \tag{4.5}$$

To minimize the interconnect complexity, we can reformulate APP-based MSA for the following message passing scheme. In the check-to-variable message passing phase, a check node c only sends the smallest magnitude, the second smallest magnitude, and 1-bit product of all $sign(L_v)$ (denoted as $min1_c$, $min2_c$, and S_c, respectively) to its neighboring variable nodes. The index of *min1* is not needed.

In a variable node *v*, the $L_v^{(k-1)}$ message that computed in the $(k-1)^{th}$ iteration is stored for recovering the needed input $R_{cv}^{(k-1)}$ in the k^{th} iteration using (4.6).

$$| R_{cv}^{(k-1)} | = \begin{cases} min2_c^{(k-1)}, & if \ | L_v^{(k-1)} | = min1_c^{(k-1)} \\ min1_c^{(k-1)} & otherwise \end{cases}, \tag{4.6}$$

$$sign(R_{cv}^{(k-1)}) = S_c^{(k-1)} \times sign(L_v^{(k-1)}) \tag{4.7}$$

Fig. 4.2(a) illustrates the structure of a VNU. The inputs are compressed check-to-variable messages from three CNUs. The needed $R_{cv}^{(k-1)}$ messages are recovered by E&S unit using (4.6). In the output, the column sum $L_v^{(k)}$ is broadcasted to its neighboring CNUs. Fig. 4.2(b) shows the structure of a CNU performing (4.5). The inputs are $L_v^{(k)}$ from six VNUs. The outputs, *min1*, *min2*, and 1-bit *S*, are broadcasted to its neighboring VNUs.

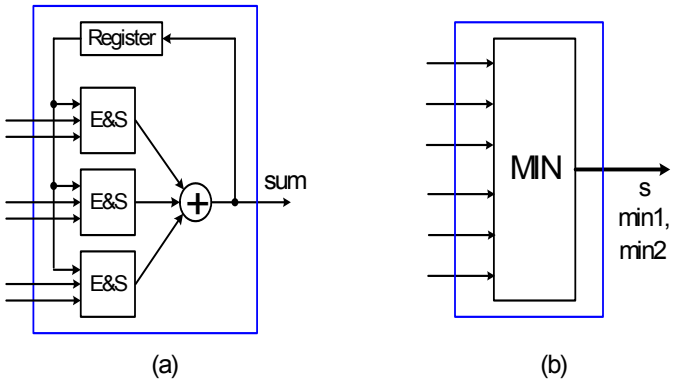


Figure 4.2 Computation units using APP-based Min-Sum algorithm. (a) Variable node unit (b) Check node unit

4.1.2 Architecture for Permuation Matrices Based LDPC Codes

We use a (2048, 1723) permutation matrices based LDPC code [12] to illustrate the efficient message passing architecture for LDPC decoder. The ***H*** matrix of the (2048, 1723) LDPC code is composed of 6×32 submatrices. Each submatrix is a 64×64 permutation matrix. To facilitate high throughput decoder design, we partition the ***H*** matrix into 4 block columns. Each variable node in a block column is mapped to a VNU. Hence, 4 variable nodes are mapped to one VNU. Each check node is mapped to an 8-input folded CNU. It takes 4 clock cycles to complete the computation shown in (4.5) for 32 input data. The decoding method discussed in Section 4.1.1 is employed.

The structure of the 8-input folded CNU is shown in Fig. 4.3. Input data are variable-to-check messages. Output data are *min1*, *min2*, and *S*. Each compare-and-swap unit compares the magnitude of two input data and swaps the larger magnitude to its upper output position. Each 2×2 pseudo rank order filter (PROF) compares 4 data from two presorted vectors in parallel and outputs the smallest and the second smallest magnitude to its lower and upper output position, respectively. The design details of PROF is provided

in Section 3.4.1. In the k^{th} iteration, the intermediate values of check-to-variable messages are stored in a scratch register *R1*. In the beginning of the $(k+1)^{th}$ iteration, the final check-to-variable messages given in the end of the k^{th} iteration is stored into register *R2*. To increase the clock speed of CNU, pipelining stages can be inserted. The structure of variable node can be straightforwardly designed with (4.6), (4.7) and (4.4).

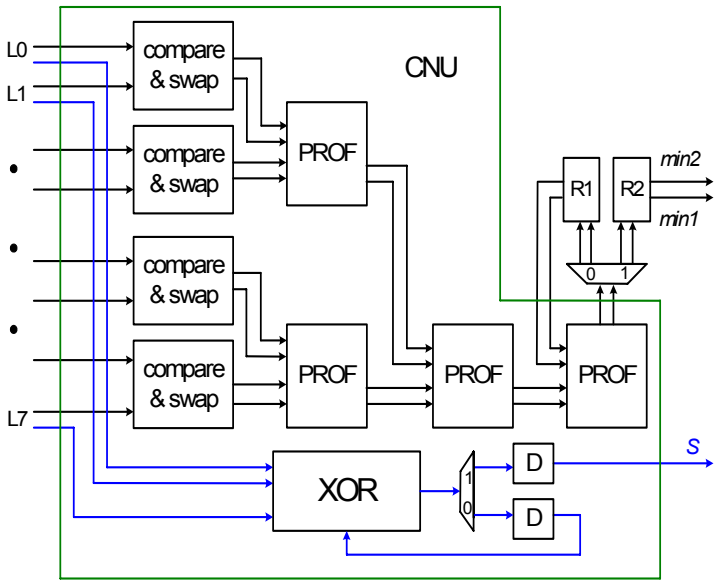


Figure 4.3 The structure of 8-input folded check node unit.

To elaborate the top-level decoder architecture, let us use a very short LDPC code as an example. Its ***H*** matrix is shown in Fig 4.4. It is composed of 2×4 submatrices. Each submatrix is a 3×3 permutation matrix. The ***H*** matrix is partitioned into 2 block columns.

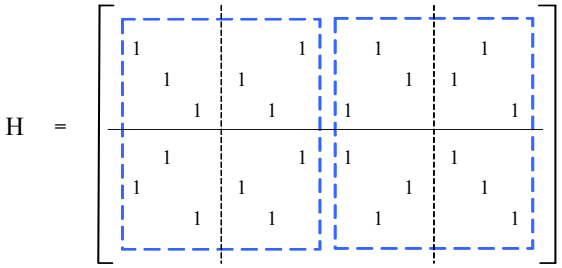


Figure 4.4 The ***H*** matrix of an LDPC code example.

Fig. 4.5 shows the corresponding decoder architecture. The message passing procedure in the k^{th} iteration is as follows. In the first clock cycle, the 6 column sums of the first block column are calculated by VNUs. Then, they are sent to 6 2-input folded CNUs. In the same time, the 6 column sums are stored in *L* register array for recovering R_{cv} messages needed in the next iteration. Each CNU processes 2 data for a check node. The intermediate computation results are stored in *R1* register array. In the second clock cycle, similar computations are performed using the data corresponding to the second block column. After the k^{th} iteration is completed, the final results of check-to-variable messages are sent to *R2* register array for the next iteration.

It can be observed from Fig. 4.5 that no separate variable-to-check and check-to-variable messages are transferred. Each VNU sends out only one summation data to multiple CNUs; and each CNU sends out only one compact data to multiple VNUs.

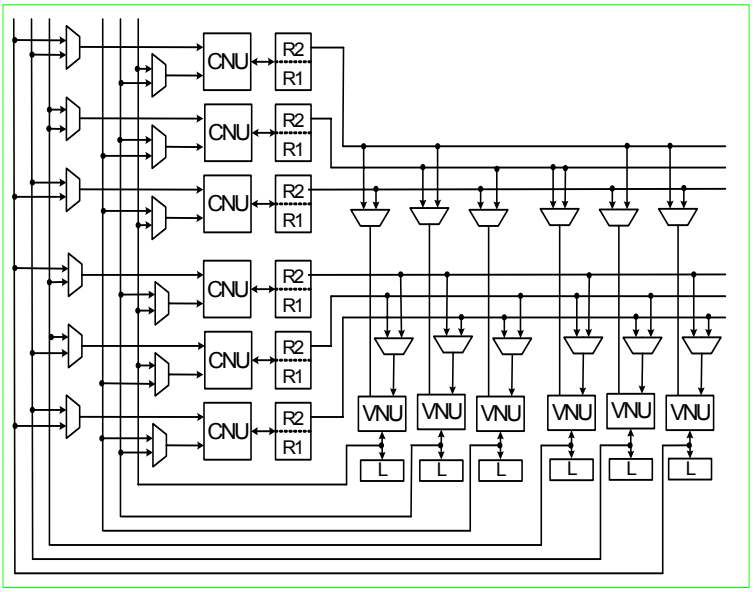


Figure 4.5 Decoder architecture for the example code.

4.1.3 Further Complexity Reduction with Non-uniform Quantization

To further reduce routing and computation complexity, we can consider reducing word length, which will directly lead to linear reduction in routing complexity and memory usage. However, using less quantization bits usually leads to performance loss. Non-uniform quantization schemes for SPA were studied [45] [60] to mitigate the performance loss of finite precision implementation. In this section, an optimized non-uniform quantization scheme for MSA using only 3 bits to represent each message is investigated.

The simulation result for the (2048, 1723) code is shown in Fig. 4.6. It can be observed that only 0.25dB performance loss from floating-point SPA is caused when the target BER is 10^{-7}. The maximum iteration number is 24. The details of the non-uniform quantization scheme using 3 bits for MSA decoding are as follows. 1) Each received soft message is quantized using non-uniform boundaries optimized for performance. In practice, the non-uniform quantizing boundaries can be obtained through simulation. Assuming that the binary bits of an LDPC codeword are transmitted over AWGN channel

with BPSK mapping from {0,1} to {1,-1}, the quantizing boundaries for the (2048, 1723) code are depicted in Table 4.1. It is assumed that each received symbol x from the front end of receiver is originally quantized using 5:4 quantization scheme [37], in which 5 bits are used to represent each data. 2) In the check-to-variable message passing phase, data in the non-uniform format are directly used in (4.5). 3) In the variable-to-check message passing phase, two kinds of look-up tables are needed. One is for converting 2-bit magnitude of an input data of (4.4) from non-uniform to two's complement format. The other is for converting the magnitude of the column sum from 2's complement to 2-bit non-uniform format. Table 4.2 shows the details of the data conversion for the (2048, 1723) code. Please note that sign bit of each data is not changed and all intrinsic and extrinsic soft messages stored in memory are in 3-bit. The scaling factor α of (4.4) is chosen to be 0.5.

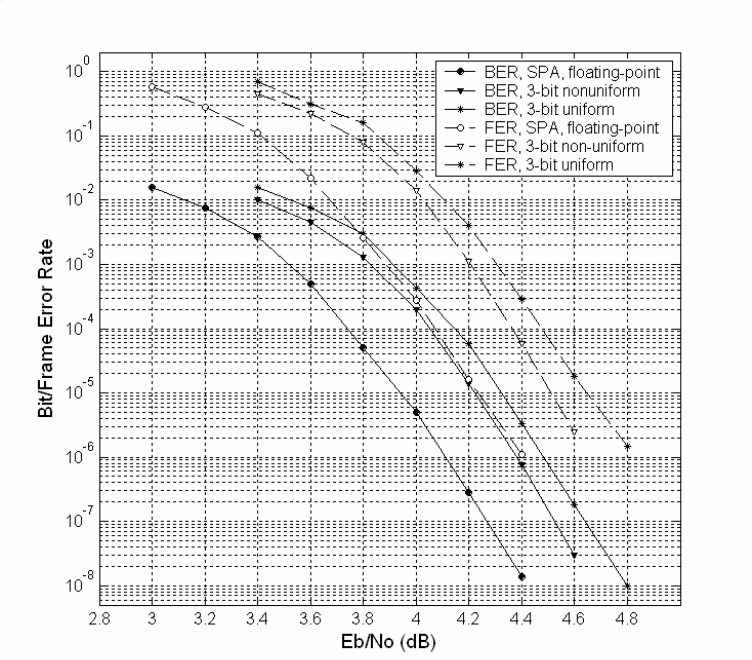


Figure 4.6 BER and FER of various decoding approaches (24 iterations) for the (2048, 1723) LDPC code.

TABLE 4.1 3-BIT QUANTIZATION FOR RECEIVED SYMBOL

The Range of Received Symbol	*Quantization Output*
$x \geq 6/8$	011
$6/8 > x \geq 3/8$	010
$3/8 > x \geq 3/16$	001
$3/16 > x \geq 0$	000
$0 > x \geq -3/16$	100
$-3/16 > x \geq -3/8$	101
$-3/8 > x \geq -6/8$	110
$-6/8 > x$	111

TABLE 4.2 DATA CONVERSION BETWEEN UNIFORM QUANTIZATION AND NON-UNIFORM QUANTIZATION.

Non-uniform to uniform			**Uniform to non-uniform**	
Input	***Output***		***Input***	***Output***
000	1		$2 \geq x \geq 0$	000
001	3		$6 > x > 2$	001
010	6		$10 > x \geq 6$	010
011	10		$x \geq 10$	011
100	-1		$-2 \leq x < 0$	100
101	-3		$-6 < x < -2$	101
110	-6		$-10 < x \leq -6$	110
111	-10		$x \leq -10$	111

4.2 Layered Decoding Architecture for Quasi-Cyclic Codes

In practice, QC-LDPC codes have been considered for many applications. We present a high-throughput low-cost layered decoding architecture for generic QC-LDPC codes. In this design, row permutation approach is proposed to significantly reduce the implementation complexity of interconnect network. An approximate layered decoding approach is explored to increase clock speed and hence to increase the decoding throughput. An efficient implementation technique which is based on Min-Sum algorithm is employed to minimize the hardware complexity. The computation core is further optimized to reduce the computation delay.

4.2.1 Row Permutation of Parity Check Matrix of QC-LDPC Codes

The parity check matrix of a QC-LDPC code is an array of circulant submatrices. To achieve very high decoding throughput, an array of cyclic shifters are needed to shuffle soft messages corresponding to multiple submatrices for check nodes and variable nodes. In order to reduce the VLSI implementation complexity for the shuffle network, the shifting structure in circulant submatrices is extensively exploited. Suppose the parity check matrix H of a QC-LDPC code is a $J \times C$ array of $p \times p$ circulant submatrices. With row permutation, it can be converted to a form as shown in (4.8).

$$H_P = \begin{bmatrix} A_1 & A_2 & A_3 & \cdots & A_C \\ A_1\sigma & A_2\sigma & A_3\sigma & \cdots & A_C\sigma \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_1\sigma^{\frac{p}{m}} & A_2\sigma^{\frac{p}{m}} & A_3\sigma^{\frac{p}{m}} & \cdots & A_C\sigma^{\frac{p}{m}} \end{bmatrix}, \tag{4.8}$$

where σ is a $p \times p$ permutation matrix representing a single left or right cyclic shift. The submatrix $A_i\sigma^j$ can be obtained by cyclically shifting the submatix $A_i\sigma^{(j-1)}$ for a single step. A_i is a $mJ \times p$ matrix determined by the shift offsets of the circulant matrices in block column i ($i=1,2,\ldots C$), m is an integer such that p can be divided by m.

For example, the matrix H_a shown in Fig. 4.7 is a 2×3 array of 8×8 cyclically shifted identity submatrices. With the row permutation described bellow, a new matrix H_b shown in Fig. 4.8 can be obtained, which has the form shown in (4.8). First, the first 4 rows of the first block row of H_a are distributed to 4 block rows of H_b in a round-robin fashion (*i.e.*, the row 1, 2, 3, 4 of H_a are distributed to row 1, 5, 9, 13 of H_b). Then the second 4 rows are distributed in the same way. The permutation can be continued until all rows in the first block row of matrix H_a are moved to matrix H_b . Then the second block row of H_a are distributed in the same way. It can be seen from Fig. 8 that H_b has the form shown in (4.8). In the above example, the row distribution is started from the first row of

each block row. In general, to distribute a block row to a new matrix, the distribution can be started from any row of the block row.

For an LDPC decoder which can process all messages corresponding to the 1-components in an entire block row of matrix H_P (e.g. H_b in Fig. 4.8), the shuffle network for LDPC decoding can be implemented with very simple data shifters.

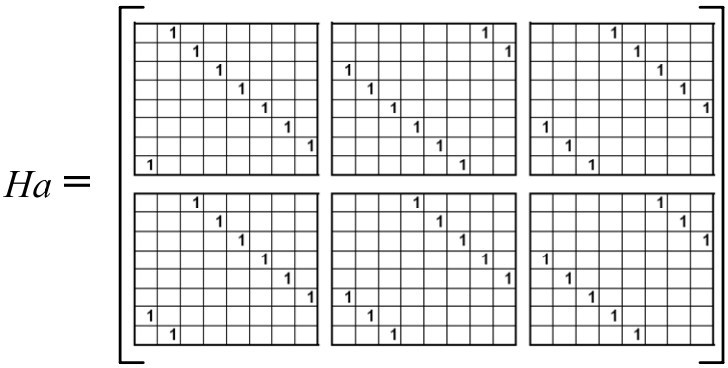


Figure 4.7 An array of circulant submatrices.

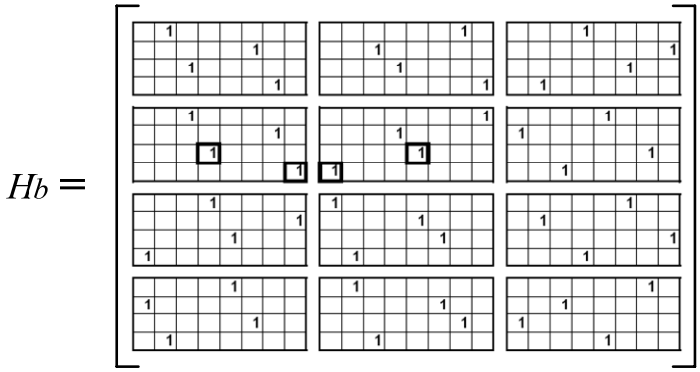


Figure 4.8 Permuted matrix.

4.2.2 Approximate Layered Decoding Approach

Recently, layered decoding approach [49] [50] [52] has been found to converge much faster than conventional TPMP decoding approach. With layered decoding approach,

the parity check matrix of an LDPC code is partitioned into L layers: $H^T = \left[H_1^T\ H_2^T \cdots H_L^T\right]$. The layer H_t defines a supercode C_t and the original LDPC code is the intersection of all supercodes: $C = C_1 \cap C_2 \cdots \cap C_L$. The column weight of each layer is at most 1.

In the k^{th} iteration, the log-likelihood ratio (LLR) message from layer t to the next layer for variable node v is represented by $L_v^{k,t}$, where $t=1,2,\ldots,L$. The layered message passing with Min-Sum algorithm can be formulated as (4.9)-(4.11).

$$L_{cv}^{k,t} = L_v^{k,(t-1)} - R_{cv}^{(k-1),t}, \tag{4.9}$$

$$R_{cv}^{k,t} = \alpha \times \prod_{n \in N(c)\backslash v} sign(L_{cn}^{k,t}) \times \min_{n \in N(c)\backslash v} |L_{cn}^{k,t}|, \tag{4.10}$$

$$L_v^{k,t} = L_{cv}^{k,t} + R_{cv}^{k,t}. \tag{4.11}$$

In a layered LDPC decoder, the check node unit (CNU) is for the computation shown in (4.10) and the variable node unit (VNU) performs (4.9) and (4.11). In the case that all soft messages corresponding to the 1-components in an entire block row of parity check matrix are processed in a clock period, the computations shown in (4.9)-(4.11) are sequentially performed. The long computation delay in the CNU inevitably limits the maximum achievable clock speed. Usually pipelining technique can be utilized to reduce the critical path in computing units. However, due to the data dependency between two consecutive layers in layered decoding, pipelining technique can not be applied directly.

For instance, suppose that one stage pipelining latch is introduced into every CNU. To compute $L_{cv}^{k,3}$ messages corresponding to the third block row of H_b, $L_v^{k,2}$ messages are needed, which can not be determined until $R_{cv}^{k,2}$ messages are computed with (4.10). Due to the one-clock delay caused by the pipelining stage in CNUs, $R_{cv}^{k,2}$ messages are not available in the required clock cycle. The data dependency between layer 3 and layer 2 occurs at column 4, 8, 9, and 13 as marked by bold squares in Fig. 4.8. To enable pipelined

decoding, we propose an approximation of layered decoding approach. Let us rewrite (4.10) as the following:

$$R_{cv}^{k,t} = f(\underbrace{L_{cn_1}^{k,t}, L_{cn_2}^{k,t}, L_{cn_3}^{k,t}, \cdots}_{n_i \in N_1 \bigcup N_2}),$$

where $N_1 \bigcup N_2$ is the variable node set $N(c) \setminus v$. The data dependency between layer *t* and *t+1* occurs in the column positions corresponding to the variable node set N_1. For the variable nodes *v* belonging to the variable node set N_2, the following equation is satisfied.

$$L_{cv}^{k,t} = L_v^{k,(t-1)} - R_{cv}^{(k-1),t} = L_v^{k,(t-2)} - R_{cv}^{(k-1),t}.$$

For $v \in N_1$,

$$\begin{aligned} L_{cv}^{k,t} &= L_v^{k,(t-1)} - R_{cv}^{(k-1),t} \\ &= L_v^{k,(t-2)} + (-R_{c'v}^{(k-1),(t-1)} + R_{c'v}^{k,(t-1)}) - R_{cv}^{(k-1),t} \\ &\cong L_v^{k,(t-2)} - R_{cv}^{(k-1),t}. \end{aligned}$$

Based on the above consideration, an approximate layered decoding approach is formulated as (4.12)-(4.14).

$$L_{cv}^{k,t} = L_v^{k,(t-1-P)} - R_{cv}^{(k-1),t}, \tag{4.12}$$

$$R_{cv}^{k,t} = \alpha \times \prod_{n \in N(c) \setminus v} sign(L_{cn}^{k,t}) \times \min_{n \in N(c) \setminus v} | L_{cn}^{k,t} |, \tag{4.13}$$

$$L_v^{k,(t-P)} = L_v^{k,(t-1-P)} - R_{cv}^{(k-1),(t-P)} + R_{cv}^{k,(t-P)}. \tag{4.14}$$

where, *P* is a small integer. In order to demonstrate the decoding performance of the proposed approach, a (3456, 1728), (3, 6) rate-0.5 QC-LDPC code constructed with progressive edge-growth (PEG) approach [30] is used. Its parity check matrix is permuted as discussed in Section 4.2.1. The number of rows in each layer is 144. The parameter *P* in (4.12) and (4.14) is set to 2 to enable two stage pipelines. The maximum iteration number is set to 15. It can be observed that the proposed approach has about 0.05 dB performance degradation compared with the standard layered decoding scheme. The conventional

TPMP approach has about 0.2 dB performance loss compared with layered decoding scheme because of its slow convergence speed. It should be noted that, by increasing the maximum iteration number, the performance gap among the three decoding schemes decrease. However, the achievable decoding throughput is reduced.

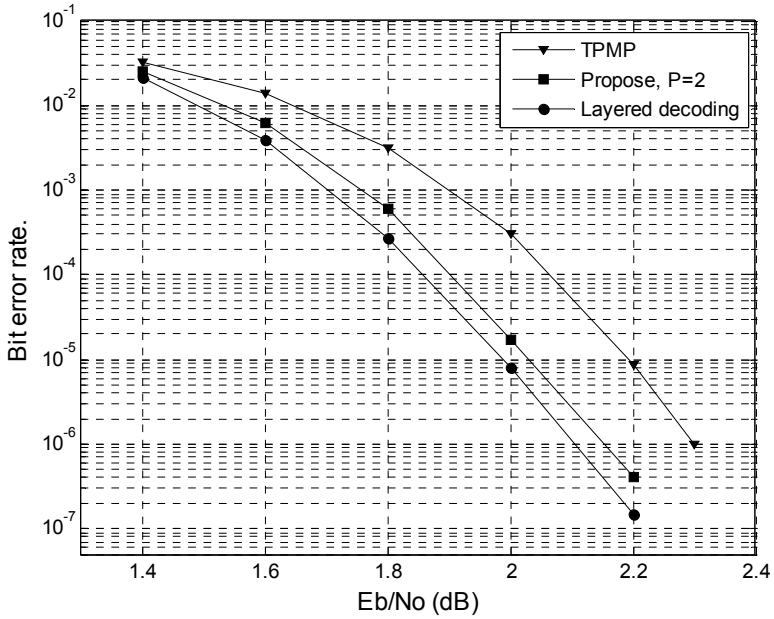


Figure 4.9 Performance of the approximate layered decoding approach.

4.2.3 Decoder Architecture with Layered Decoding Approach

The Overall Decoder Architecture

The proposed decoder computes the check-to-variable messages, variable-to-check messages, and LLR messages corresponding to an entire block row of H_P matrix in one clock cycle. The decoder architecture is shown in Fig. 4.10. It consists of five portions. 1) L layer R-register arrays. Each layer is used to store the check-to-variable messages R_{cv} corresponding to the 1-components in a block row of matrix H_P. At each clock cycle,

R_{cv} messages in one layer are vertically shifted down to the adjacent layer. 2) A check node unit (CNU) array for generating the R_{cv} messages for one layer of R-register array in a clock cycle. The dashed-lines in the CNU array denote 2 pipeline stages. 3) C LLR-register arrays. Each LLR-register array stores the L_v messages corresponding to a block column of matrix H_P. 4) C variable node unit (VNU) arrays. Each VNU array is used for computing the variable-to-check messages and LLR messages corresponding to a block column of matrix H_P. Each VNU is composed of two adders. 5) C data shifters. The L_v messages corresponding to a block column of matrix H_P is shifted one step by a data shifter array. In Fig. 4.10, each VNU, MUX, and data shifter is used to represent C computing unit arrays.

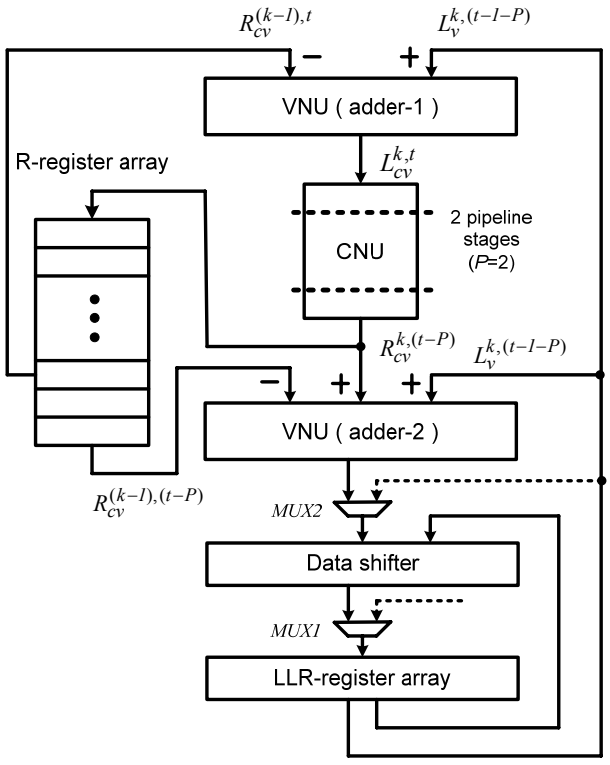


Figure 4.10 Decoder architecture (P=2).

In the decoding initialization, the intrinsic messages are transferred to LLR-register arrays via the *MUX1* arrays. At the first *P* clock cycles, R_{cv} messages are not available due to the *P* pipeline stages in the CNU array. Therefore, the *MUX2* arrays are needed to prevent LLR-registers from being updated. In one clock cycle, only a portion of LLR-messages are updated. The updated LLR-messages correspond to the 1-component in the layer of matrix H_P are sent to data shifter via computation path. The remained LLR-messages are directly sent to the data shifter from the LLR-register array.

The Critical Path of the Proposed Architecture

The computation path of the proposed architecture is shown in Fig. 4.11. The equations shown in (4.12)-(4.14) are sequentially performed. The computation results of (4.12) are represented in two's complement format. It is convenient to use the sign-magnitude representation for the computation expressed in (4.13). Thus, two's complement to sign-magnitude data conversion is needed before data are sent to CNU. The R_{cv} messages from CNU array and R-register arrays are in a compressed form to reduce memory requirement. More details are explained in the next paragraph. To recover the individual R_{cv} messages, a data distributor is needed. The R_{cv} messages sent out by the data distributor are in sign-magnitude representation. Correspondingly, sign-magnitude to two's complement conversion is need before data are sent to VNU.

In this design, the computation path can be divided into three segments as shown in Fig. 4.11. The implementation of the SM-to-2'S unit and the adder in segment-1 can be optimized by merging the adder into the SM-to-2'S unit to reduce computation delay. The optimization for segment-1 is shown in Fig. 4.12. With the Min-Sum algorithm, the critical task of a CNU is to find the two smallest magnitudes from all input data and identify the relative position of the input data with the smallest magnitude. The implementation of

CNU can be found in Section 3.4.1. The dataflow in a CNU is very briefly discussed in this section. Because the number of input data is six, four computation steps are needed in a CNU. The first step is compare-and-swap. Then, two pseudo rank order filter (PROF) stages are needed. In the last step, the two smallest magnitudes are corrected using a scaling factor α (usually, α is set as 3/4). In this way, the R_{cv} messages output by a CNU are in a compressed form with four elements, *i.e.*, the smallest magnitude, the second smallest magnitude, the index of the second smallest magnitude, and the signs of all R_{cv} messages. The optimized implementation of segment-3 is shown in Fig. 4.13. The adder in the last stage can be implemented with a [4:2] compressor and a fast adder. The data shifter can be implemented with one-level multiplexers.

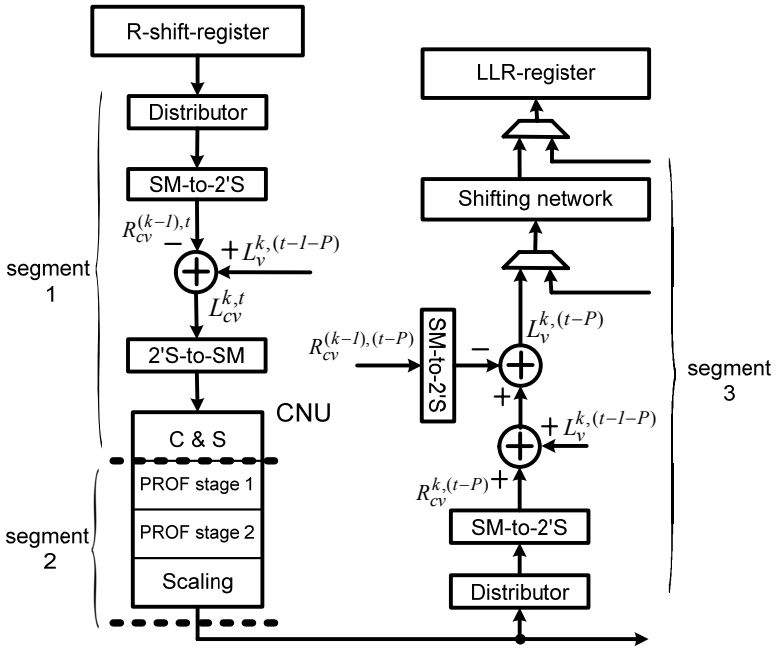


Figure 4.11 The computation path of the proposed architecture (P=2).

It can be observed that the critical path of segment-1 consists of three adders and four multiplexers. The longest logic path of segment-2 includes three adders and two multiplexers. The critical path of segment-3 has two adders and four multiplexers. By inserting two pipeline stages among the three segments, the critical path of the overall decoder architecture is reduced to three adders and four multiplexers.

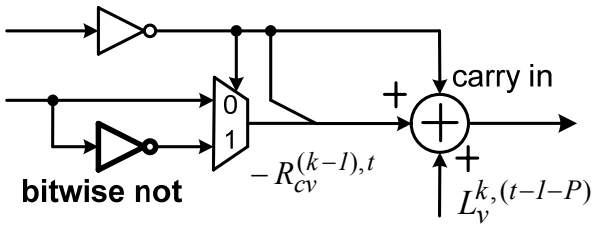


Figure 4.12 The optimization of the SM-to-2'S unit and the adder in segment-1.

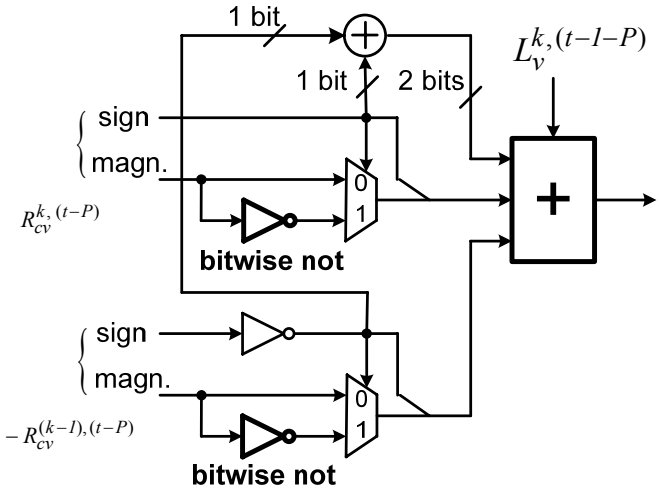


Figure 4.13 The optimization of the SM-to-2'S unit and two adders in segment-3.

Data shifter

It can be seen from Fig. 4.8 that by a single left cyclic shift, the block $H_P^{(t+1),i}$ is identical to $H_P^{t,i}$, for *i=1, 2, ... C* and *t=1, 2, ... L-1*. Therefore, repeated single-step left cyclic-shift operations can ensure the message alignment for all layers in a decoding iteration. After the messages corresponding to the last block row are processed, a reverse cyclic-shift operation is needed for the next decoding iteration. Based on the above observation, only the edges of the tanner graph for the first layer of matrix H_P are mapped to the fixed hardware interconnection in the proposed decoder. A very simple data shifter which is composed of one level two-input one-output multiplexers is utilized to perform the shifting operation for one block column of matrix H_P. Fig. 4.14 shows the structure of a data shifter for the matrix H_b. When the value of control signal, *S*, is 1, the shifting network performs a single-step left cyclic-shift. If *S* is set to 0, the reverse cyclic-shift is performed.

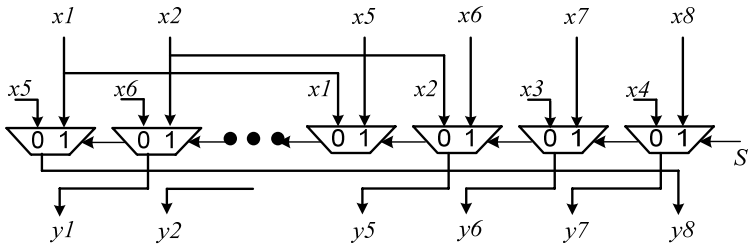


Figure 4.14 The structure of a data shifter.

4.2.4 Hardware Requirement and Throughput Estimation

The hardware requirement of the decoder for the example LDPC code is estimated except the control block and parity check block. In Table 4.3, the gate count for computing blocks is provided. Each MUX stands for a 1-bit 2-to-1 multiplexer. Each XOR represents a 1-bit two-input XOR logic unit. The register requirement is estimated in Table 4.4. In the

two tables, W_R and W_L represent the word length of each R_{cv} message and L_{cv} (or L_v) message, respectively. The critical path of the proposed decoder is three adders and four multiplexers. In the decoder architecture presented in [63], each soft message is represented as 4 bits. The critical path consists of an R-select unit, two adders, a CUN, a shifting unit and a MUX. The computation path of a CNU has a 2'S-SM unit, a two-least-minimum computation unit, an offset computation unit, an SM-to-2'S unit stage, and an R-selector unit. The overall critical path is longer than 10 4-bit adders and 7 multiplexers. The post routing frequency is 100MHz with 0.13*u* CMOS technology. Because the critical path of the proposed decoder architecture is about one-third of the architecture presented in [63], using 4-bit for each soft message, the clock speed for the proposed decoder architecture is estimated to be 250MHz with the same CMOS technology. In a decoding iteration, the required number of clock cycles is 12. To finish a decoding process of 15 iterations, we need $12 \times 15 + 3 = 183$ clock cycles. Among them, 1 cycle is needed for initialization and 2 cycles are for pipeline latency. Thus, the throughput of the layered decoding architecture is at least $3456 \times 250 \times 10^9 \div 183 \approx 4.7$ Gbit/sec. Because a real design using the proposed architecture has not been completed, we can only provide a rough comparison with other designs.

Lin, *et al*, [61], designed an LDPC decoder for a (1200, 720) code. The decoder achieves 3.33 Gbit/sec throughput with 8 iterations. Sha, *et al*, [62] proposed a 1.8Gbps decoder with 20 iterations. The decoder is targeted for a (8192,7168) LDPC code. The decoding throughput of the both decoders is less than the proposed architecture with 15 iterations. Gunnam, *et al,* [63], presented an LDPC decoder architecture for (2082, 1041) array LDPC codes. With 15 iterations, it can achieve 4.6 Gbit/sec decoding throughput. The number of CNUs and VNUs are 347 and 2082, respectively. It can be seen from Table 4.3 that much less computing units are needed in our pipelined architecture. The registers requirement in our design is more than that in [63] because an LDPC code with a larger

block length for a better decoding performance is considered in our design. The two pipeline stages in CNU array also require additional registers. The design in [63] is only suitable for array LDPC codes. The proposed decoder architecture is for generic QC-LDPC codes. We would like to mention that the proposed architecture is scalable. For example, the considered LDPC code can be partitioned into 8, 12, or 18 layers for different trade-offs between hardware cost and decoding throughput.

TABLE 4.3 GATE COUNT ESTIMATION FOR COMPUTING BLOCKS

Component	(Number) Count	Estimated gate count
CNU	144	$1584 \times W_R$ 1-bit adder + $2880 \times W_R$ MUX + 1584 XOR
Data distributor	$144 \times 6 \times 3 = 2592$	$2592 \times (W_R - 1)$ MUX
SM-to-2'S	$144 \times 6 \times 3 = 2592$	$2592 \times W_R$ MUX
VNU	$144 \times 6 = 864$	$864 \times 3 \times W_L$ 1-bit adder
2'S-to-SM	$144 \times 6 = 864$	$864 \times W_L$ MUX
Data shifter	6	$3456 \times W_L$ MUX

TABLE 4.4 STORAGE REQUIREMENT ESTIMATE

Component	Estimated register count
R-register array	$1728 \times (3 + 6 + 2 \times (W_R - 1))$
LLR-register array	$3456 \times W_L$
Pipeline register	$144 \times [6 \times W_R + 3 + 6 + 2 \times (W_L - 1)]$

4.3 An FPGA Implementation of Quasi-Cyclic LDPC Decoder

We implement an enhanced partially parallel LDPC decoder for a (8176, 7156) EG-based QC-LDPC code on FPGA to demonstrate the design of high throughput LDPC decoder. A worst-case source information decoding throughput (at 15 iterations) over

170Mbps is achieved. Optimizations at various levels are employed to increase the clock speed. More parallelism is introduced for the traditional partially parallel decoding architecture with small hardware overhead. An efficient non-uniform quantization scheme is proposed to reduce the size of soft message memories without sacrificing the decoding performance. The decoder architecture is suited for other QC LDPC codes as well.

4.3.1 The (8176, 7156) EG-based QC LDPC Code

The EG-based QC LDPC codes are a family of QC LDPC codes, which are constructed based on the decomposition of finite Euclidean geometries. The (8176, 7156) code (originally designed for NASA) is a regular QC LDPC code with a column weight of 4 and a row weight of 32 [23]. The parity-check matrix is a 2x16 array of thirty-two 511x511 submatrices as the following.

$$H = \begin{bmatrix} H_{1,1} & H_{1,2} & \cdots & H_{1,16} \\ H_{2,1} & H_{2,2} & \cdots & H_{2,16} \end{bmatrix}$$

Each submatrix $H_{i,j}$ is a circulant matrix with both column and row weight of 2. Fig. 4.15 shows a 15x15 matrix in the same form.

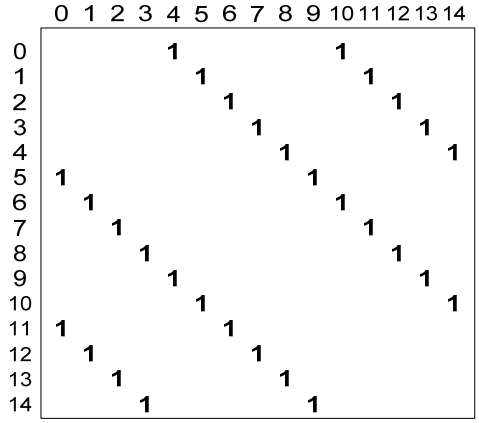


Figure 4.15 A 15x15 circulant matrix.

4.3.2 Partially Parallel Decoder Architecture

Balanced Computation Scheduling

The conventional SPA algorithm has unbalanced computation complexity between the variable-to-check and check-to-variable message updating phases. This leads to unbalanced datapaths between Variable node units (VNUs) and Check node units (CNUs).

To balance the computation load between the two decoding phases, a modified version based on algorithmic transformation was proposed in [38]. The check-to-variable and variable-to-check message passing are expressed in (4.15) and (4.16), respectively.

$$R_{cv} = -\prod_{n \in N(c) \setminus v} sign(L_{cn}) \sum_{n \in N(c) \setminus v} \Psi(L_{cn}) . \tag{4.15}$$

$$L_{cv} = \sum_{m \in M(v) \setminus c} (-sign(R_{mv}) \Psi(R_{mv})) - \frac{2r_v}{\sigma^2} . \tag{4.16}$$

where, $\Psi(x) = \log(\tanh(\left|\frac{x}{2}\right|))$, R_{cv} and L_{cv} stand for the check-to-variable message and the variable-to-check message, respectively.

The Check Node and Variable node units

Fig. 4.16 shows the architecture of a CNU, which performs check-to-variable message R_{cv} computation. Each CNU has 32 inputs and 32 outputs. The LUT-A is introduced to perform the function $\Psi(x) = \log(\tanh(\left|\frac{x}{2}\right|))$. The magnitude of the output is the sum of 31 out of 32 data values which come from LUT-A. The sign bit of the output is a product of 31 out of 32 sign bits which come from the inputs. In the last addition stage, each word of the two addends is separated into higher and lower parts. Two partial additions are performed in parallel to reduce the addition delay. To reduce the critical path

in the CNU, pipeline latches are inserted as indicated by the dashed lines. The data representations of the inputs of CNU, the outputs of LUT-As, and the final outputs of CNU are in two's complement, unsigned, and sign-magnitude, respectively.

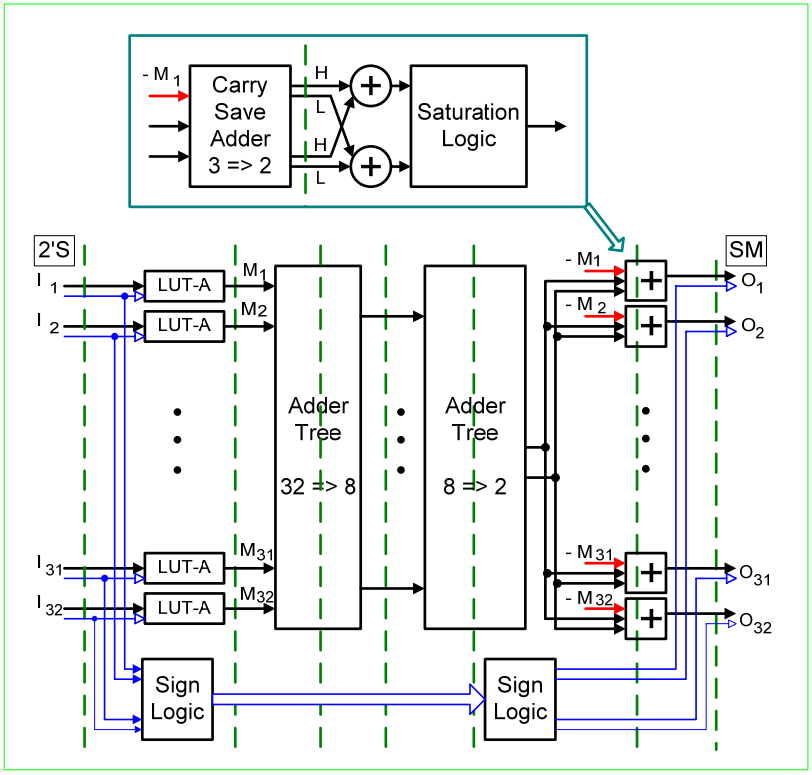


Figure 4.16 Check node unit architecture.

The architecture of a VNU is illustrated in Fig. 4.17, which performs variable-to-check message L_{cv} computation. Each VNU has 5 inputs and 5 outputs. Z and C stand for the intrinsic message and the tentative decoding bit, respectively. The LUT-B performs the function $-sign(R_{mv})\Psi(R_{mv})$. It is convenient to use the two's complement format in BPU computations. Thus, the data format of the intrinsic message Z, the outputs of LUT-Bs,

and the outputs of BPU are all in two's complement format. The inputs of LUT-Bs are in sign-magnitude format.

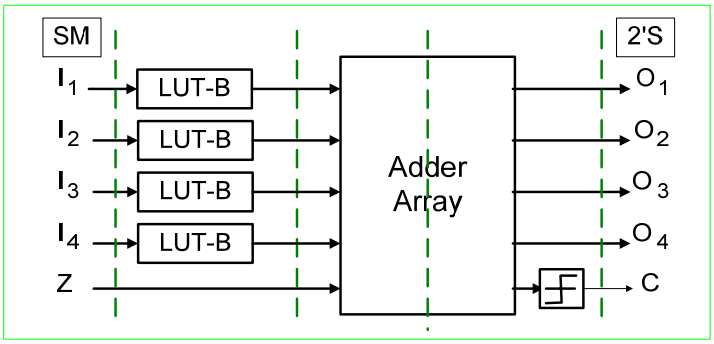


Figure 4.17 Variable node unit architecture.

Enhanced Partially Paralle Decoder Architecture

Conventionally, 1 CNU performs check-to-variable messages updating for 1 row of matrix ***H*** per clock cycle and is assigned for each block row of matrix ***H***. Similarly, 1 VNU performs variable-to-check messages updating for 1 column of matrix ***H*** per clock cycle and is assigned for each block column of matrix ***H***. To increase the parallelism, we propose an enhanced architecture that enables processing multiple rows/columns corresponding to each submatrix of ***H*** at the same time. In this design, only double parallelism is considered, though the proposed architecture can be easily extended to higher parallelism cases [35][36]. The key issue for this enhancement is how to access 4 soft messages corresponding to each submatrix at each clock cycle. For the considered EG-LDPC code, each submatrix consists of 2 cyclically shifted identity matrices. Thus, two memory modules are used for each submatrix. To facilitate 2 data accesses for each cyclically shifted matrix in both row and column updating phases, each memory module is partitioned into an even-addressed bank containing soft messages correspond to 1-components in the even rows of the submatrix and an odd-addressed bank containing data

corresponding to the odd rows. This approach works because any two soft messages corresponding to two adjacent 1-componentss of a cyclically shifted identity matrix must fall into the even-addressed memory bank and the odd-addressed memory bank, respectively. In addition, double VNU and CNU are required and data switching networks are needed to ensure the data is moved correctly between memory banks and CNU (VNU). It can be observed that the proposed architecture is also suited for the hardware implementation with Min-Sum algorithm. On the other hand, multiple adjacent soft messages can be stored at one memory entry to increase the parallelism. In this method, extra buffers and data switching networks are needed to ensure the correct data accesses in the variable-to-check messages updating phase [47].

The block diagram of the proposed architecture is shown in Fig. 4.18. Each memory block $M_{i,j}$, which consists of two memory modules, corresponds to a circulant matrix $H_{i,j}$ of the parity check matrix $\boldsymbol{H}$. They are used to store the extrinsic soft message conveyed at the both decoding phases. The memory modules Z_i and C_i and are used to store the intrinsic soft messages and the estimated codeword bits, respectively. As can be seen from the figure, the overall architecture has $16 \times 2 = 32$ VNUs and $2 \times 2 = 4$ CNUs.

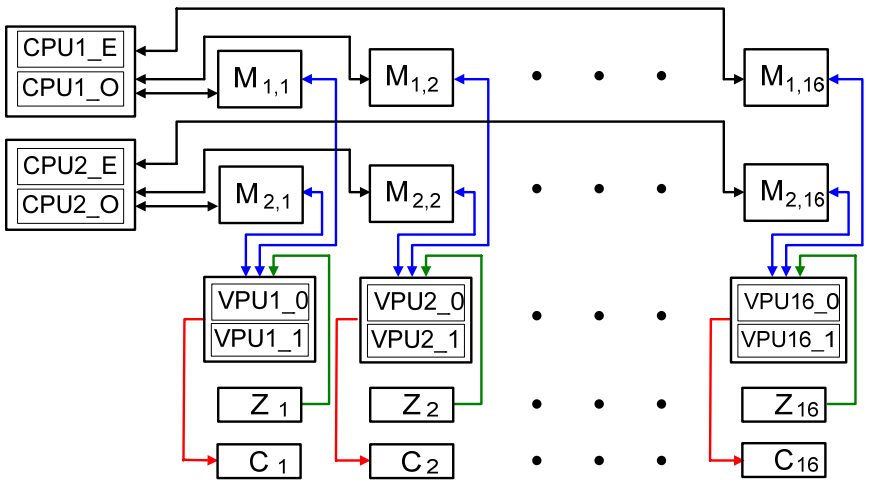


Figure 4.18 Enhanced partially parallel decoder architecture for QC-LDPC code.

To illustrate the details of dataflow, three cases, which correspond to even, odd, and zero shifting offsets that range from 0 to 510 for a cyclically shifted identity matrix, are considered in the following analysis.

Fig. 4.19 shows an example of the memory partitioning and data switching scheme applied to a 15x15 cyclically shifted identity matrix with an even (excluding 0) shifting offset of 6. In the check-to-variable message updating phase, the two data located in the even memory sub-bank MEM_E and the odd memory sub-bank MEM_O with the same index are sent to CNU_E and CUP_O in parallel, respectively, which are CNU components for even row and odd row message updating. In the variable-to-check message updating phase, the two data connected by an arrow are sent to VNU_0 and VNU_1 in the same clock cycle, respectively. Similarly as above, VNU_0 and VNU_1 are VNU components for even column and odd column data computation. A soft message $p(i, j)$ saved in a memory sub-bank corresponds to a 1-component located at row i and column j of a cyclically shifted identity matrix. In this example, the data located in the even columns 6, 8, 10, 12, and 14 of the matrix are stored in the even addressed memory sub-bank. However, the data located in the even columns 0, 2 and 4 are stored in the odd addressed memory sub-bank. Similar cases exist for the data located in the odd columns. Therefore, switching units are needed to route data between memories and VNUs in the variable-to-check message updating phase. Because the size of each circulant matrix associated with the EG-based QC LDPC code is an odd value, only the data in the last row (or column) of these matrices are accessed in the last clock cycle of the check-to-variable (or variable-to-checks) message updating phase. In this figure, symbol *Z*, *C*, and *I* represent an intrinsic soft message symbol, a decoding bit, and a fixed data value for initialization procedure, respectively.

A similar example for a cyclically shifted matrix with an odd shifting offset of 5 is shown in Fig. 4.20. Note that the last data in the even row is stored in the odd memory

bank. Without data displacement, data access confliction indicated by the dashed arrow occurs when the two data from column 4 and 5 are retrieved from the even memory sub-bank in the same clock cycle. Consequently, a pair of multiplexers is needed to steer the displaced data between the odd memory sub-bank and the CNU_E.

For the third case, *i.e.*, the shift value is 0, the cyclically shifted identity matrix becomes an identity matrix. The details of memory partitioning and data switching are shown in Fig. 4.21. This is in fact the simplest case.

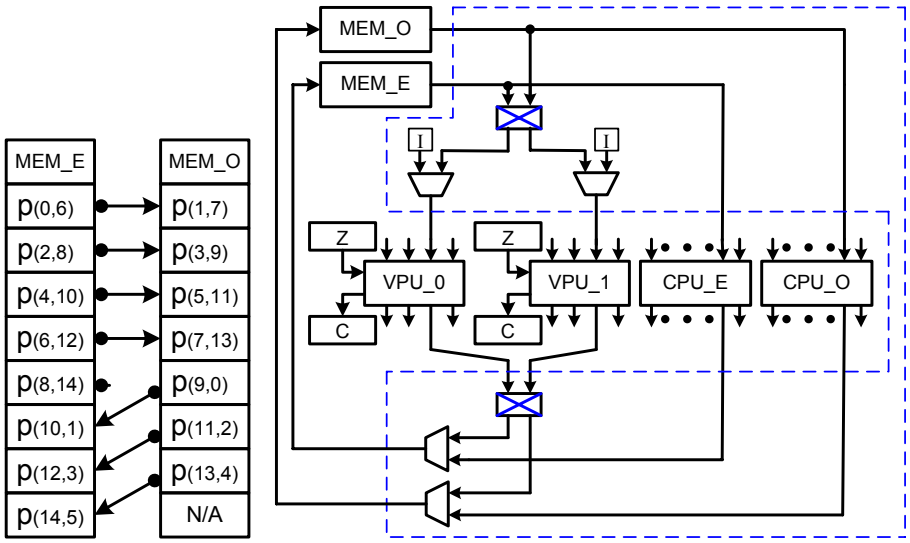


Figure 4.19 Memory partitioning and data switching scheme for even shifting offset case.

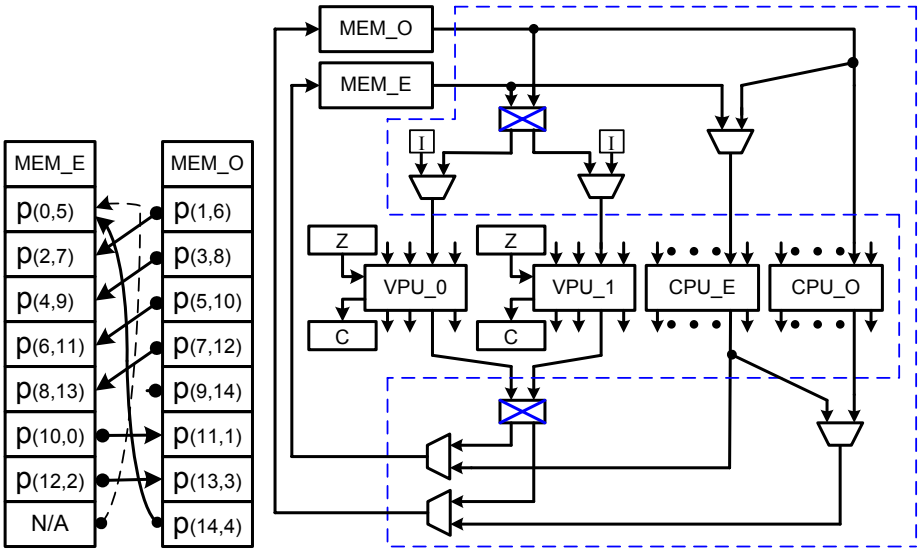


Figure 4.20 Memory partitioning and data switching scheme for odd shifting offset case.

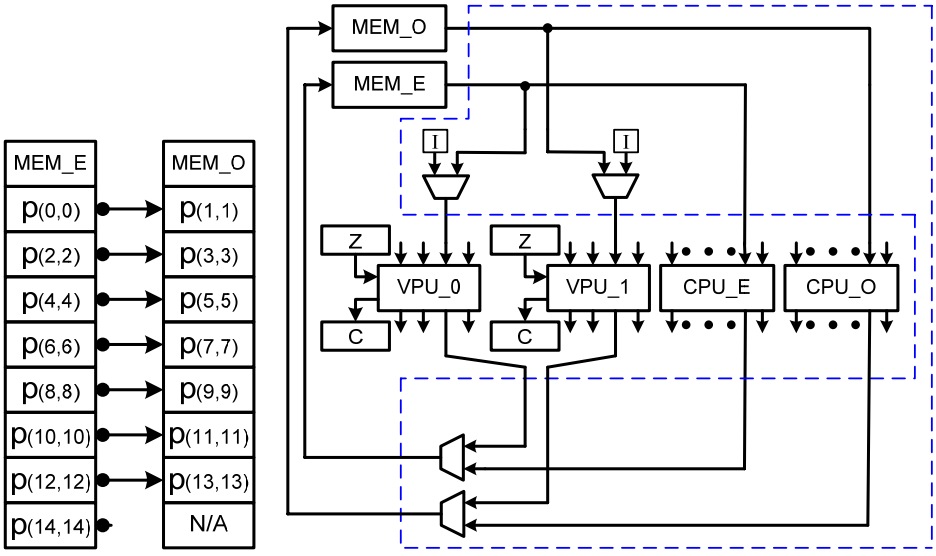


Figure 4.21 Memory partitioning and data switching scheme for identity matrix.

Architecture of the Controller

The controller, which generates the control signals of the data switch networks and memory addresses, is composed of a two-level finite state machine. Fig. 4.22 shows the state transition diagram of the finite state machine. In the initialization state, the intrinsic soft messages stored in the memory *Z* are transferred into the memory *M*. The anti-overlap state is introduced to avoid the data access confliction between the two decoding phases.

The block diagram of the controller is shown in Fig. 4.23. The memory write addresses and the data switching control signals for writing are the delayed versions of the memory read addresses and the control signals for reading, respectively. In order to increase the speed of the controller, memory addresses are generated such that retiming technique can be employed to reduce the critical path of the controller. By introducing one delay unit in the controller datapath, the critical path can be significantly reduced while introducing one cycle latency.

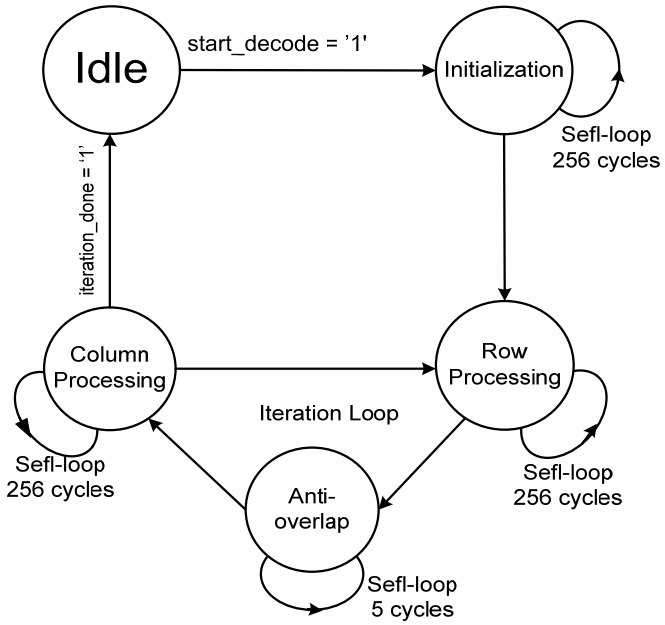


Figure 4.22 State transition diagram of controller.

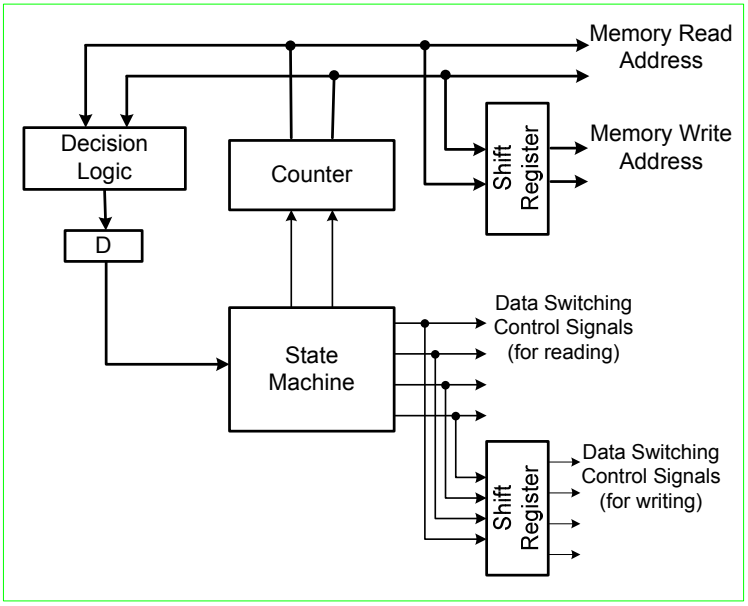


Figure 4.23 Block diagram of controller.

4.3.3 Fixed-point implementation

The word length of the soft messages determines the memory size, the computation unit size and the decoding performance of a LDPC codes decoder. The overall hardware of LDPC decoder is predominantly determined by the size of the memories holding intrinsic and extrinsic soft messages. Therefore it is very important to find an efficient quantization scheme for soft messages under the target decoding performance.

Uniform and Non-uniform Quantization Schemes

Using a similar notation as [37], let $q:f$ denote the uniform quantization scheme in which the finite word length is q bits, of which f bits are used for the fractional part of the value. If the target bit error rate (BER) is above 10^{-9}, 6:3 uniform quantization can be adopted with negligible performance loss. However, this design is targeted for BER below 10^{-10} considering potential applications of deep-space communications. Simulation results

reveal that 7:4 uniform quantization is needed to achieve this goal. Fig. 4.24 shows the performance comparison for 6:3 and 7:4 fixed-point quantization and double precision simulations.

A non-uniform quantization scheme which generally out-performs the uniform quantization under the same word length was proposed [45]. However, in this method, a q-bit non-uniform quantization scheme generally performs worse than the uniform quantization case with $(q+1)$-bit word length since less precision is maintained for large values. This section presents a new non-uniform quantization scheme that can achieve a decoding performance almost identical to that of the uniform quantization case with 1-bit longer word length.

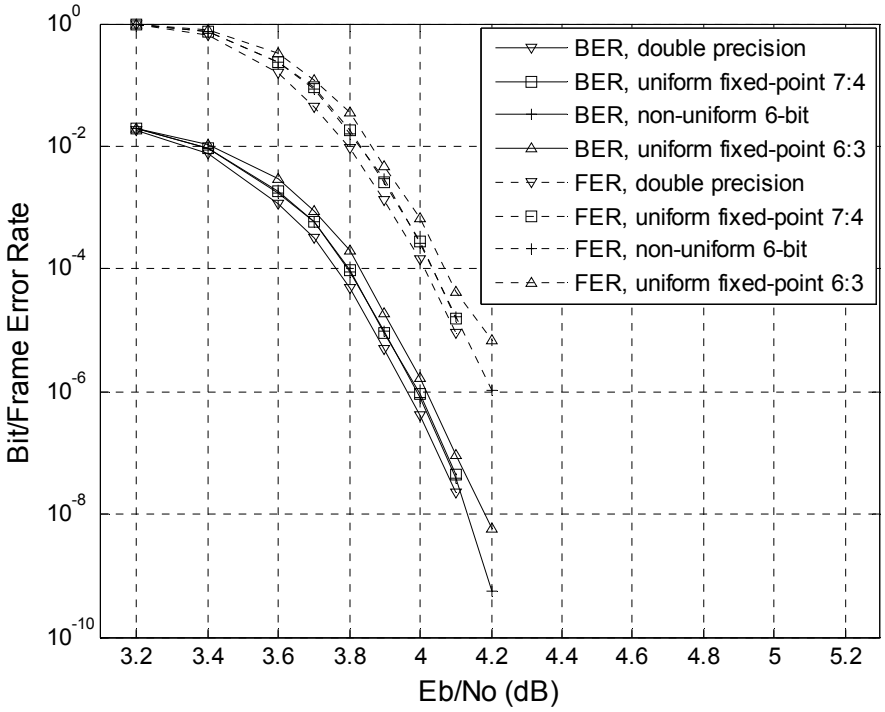


Figure 4.24 Decoding performance of fixed-point quantization and double precision.

In the both decoding phases, the extrinsic soft messages are sent to look-up tables, which perform the non-linear function $\Psi(x) = \log(\tanh(\left|\frac{x}{2}\right|))$. Fig.4.25 shows the 7:4 uniform quantization of $\Psi(x)$. It can be seen that the quantization result has many duplicated values. The improved non-uniform quantization scheme employs flexible non-uniform quantization steps for x to reduce the redundant elements in look-up table other than uses two fixed quantize steps as shown in [45] for the regions of $x < 1$ and $x \geq 1$, respectively. The uniform to non-uniform quantization conversion logic can be implemented with simple combination logic or look-up table, which depends on the complexity of the conversion mapping. In this method, the quantized value of $\Psi(x)$ is presented in uniform quantization.

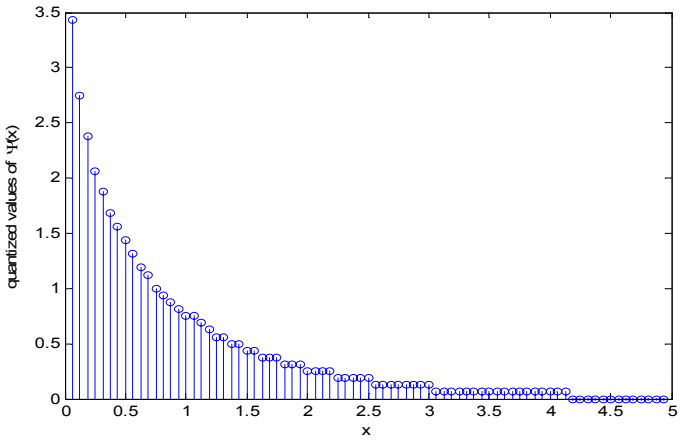


Figure 4.25 The uniform quantization of $\Psi(x)$.

TABLE 4.5. UNIFORM TO NON-UNIFORM QUANTIZATION CONVERSION

Data in 7-bit uniform quantization	Corresponding data in 6-bit non-uniform quantization	Data representation in 6-bit non-uniform quantization
000.f_1 f_2 f_3 f_4	00.f_1 f_2 f_3 f_4	0 0 f_1 f_2 f_3 f_4
001.f_1 f_2 f_3 f_4	01.f_1 f_2 f_3 f_4	0 1 f_1 f_2 f_3 f_4
010.f_1 f_2 f_3 f_4	10.f_1 f_2 f_3 f_4	1 0 f_1 f_2 f_3 f_4
011.f_1 f_2 f_3 f_4	011.f_1 f_2 f_3	1 1 0 f_1 f_2 f_3
100.f_1 f_2 f_3 f_4	100.f_1 f_2 f_3	1 1 1 f_1 f_2 f_3
others	111.111	1 1 1 1 1 1

A close study of the 7:4 uniform quantization results of $\Psi(x)$ reveals that no quantized values of $\Psi(x)$ are lost if x is non-linearly quantized as shown in the middle column of Table 4.5. Based on this observation, the new non-uniform quantization scheme for this specific case is as follows: the soft messages in uniform quantization computed from equations (1) and (2) are converted into non-uniform quantization as seen in Table 4.5 and are stored into memory blocks $M_{i,j}$. The input of the look-up table for $\Psi(x) = \log(\tanh(\left|\frac{x}{2}\right|))$ is in non-uniform quantization, and the values of $\Psi(x)$, which are stored in look-up table, are in uniform quantization. Fig. 4.24 shows that using the 6-bit non-uniform quantization scheme can achieve an almost identical decoding performance to that using the 7-bit uniform quantization scheme.

Processing Units with Non-uniform Quantization

The new architectures for CNU and VNU with the non-uniform quantization scheme are shown in Fig. 4.26 and Fig. 4.27, respectively. The uniform to non-uniform quantization converters, U2NUs, are employed as shown in the two figures. They are

implemented with simple combination logic. The look-up tables, LUTs, for both CNU and VNU are the same.

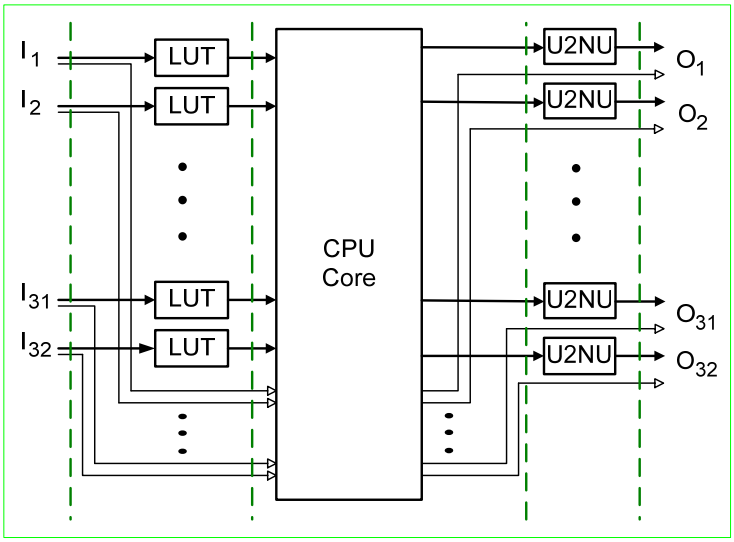


Figure 4.26 Check node unit with non-uniform quantization.

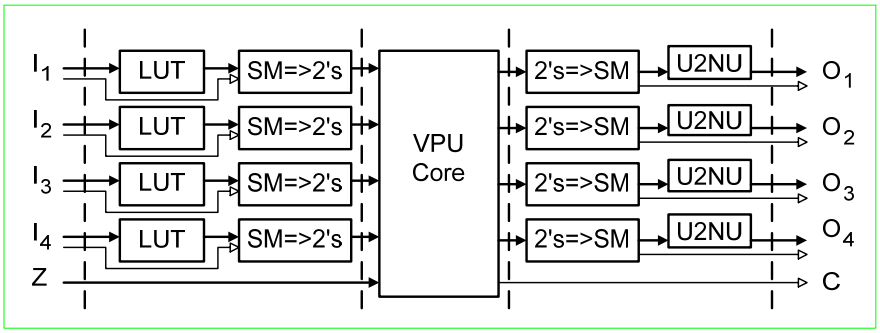


Figure 4.27 Variable node unit with non-uniform quantization.

4.3.4 FPGA Implementation

Based on the architectures described above, the (8176, 7156) EG-based LDPC code decoder was modeled in VHDL, simulated and synthesized targeting the Xilinx Virtex II-6000. Based on the Xilinx TRACE report, the maximum clock frequencies of the uniform and non-uniform quantization implementation are 193.4MHz and 192.6MHz, respectively. Table 4.6 shows the FPGA utilization statistics of both implementations.

The maximum iteration number is set to 15. It takes half of an iteration to transfer the intrinsic soft messages from memory Z into memory M of the decoder. It takes $2 \times 256 + 5 = 517$ clock cycles to perform one iteration in which 5 clock cycles are allotted to the anti-overlap state. Thus, the non-uniform quantization implementation can achieve a worst-case information decoding throughput of $(7154 \times 192.6)/[(15+0.5) \times 517] \approx 172$ Mbps. The achieved decoding throughput is more than twice faster (in terms of Mbps per iteration) than other published LDPC codec implementations based on similar platforms (e.g., [48][44]).

TABLE 4.6. XILINX VIRTEXII-6000 FPGA UTILIZATION STATISTICS

Resource	6-bit uniform quantization		6-bit non-uniform quantization	
	Used	Utilization ratio	Used	Utilization ratio
Slices	23052	68%	27,460	81%
Slice Flip Flops	26926	39%	38,266	56%
4-input LUTs	28229	41%	36848	54%
Block RAMs	128	88%	128	88%

4.4 Summary

In this chapter, design issues for high throughput LDPC decoders have been discussed. We have proposed an algorithmic transformation for significant reduction of

routing complexity in LDPC decoders. It has been shown that the proposed approach can reduce 54% outgoing wires of each VNU and 90% outgoing wires of each CNU if using 4-bit quantization for decoding a (2048, 1723) (6, 32) LDPC code. The detailed architecture for permutation matrices based LDPC codes have been illustrated. Furthermore, we have developed a high-throughput low-complexity decoder architecture for generic QC-LDPC codes by exploiting the regularity in parity check matrices of QC-LDPC codes. It has been estimated that 4.7 Gbit/sec decoding throughput for a (3456, 1728) (3, 6) QC-LDPC code can be achieved. Finally, we have demonstrated an FPGA implementation of a low complexity, high speed decoder for EG-based QC-LDPC codes. The FPGA implementation with Xilinx Virtex II 6000 achieves a maximum decoding throughout of over 170 Mbps at 15 iterations.

5 PRACTICAL LOW COMPLEXITY LDPC DECODERS

In this chapter, we briefly analyze the decoding complexity of WBF-based algorithms from the VLSI implementation point of view. To maintain low decoding complexity while further narrowing the performance gap from the SPA, we present an optimized 2-bit decoding approach. It is shown that the hardware implementation complexity of the proposed method is comparable to that of WBF-based algorithms. However, it has significantly better decoding performance and faster convergence speed.

5.1 The Optimized 2-bit Decoding

5.1.1 Decoding Scheme

In the optimized 2-bit decoding method, each message stored in the memory is represented as 2 bits, $b_s b_m$. Bit b_s is the hard-decision of a received soft message. Bit b_m indicates the hard-decision confidence. b_m=0 denotes a low confidence instead of a zero value. Similarly, b_m=1 represents a high confidence. The values of the two bits are given by (5.1) and (5.2), where, T_y represents a threshold. Its optimum value can be obtained through simulation.

$$b_s = sign(y) \tag{5.1}$$

$$b_m = \begin{cases} 1, & if\ |y| \rhd T_y \\ 0, & otherwise \end{cases} \tag{5.2}$$

The *a posteriori probability* (APP) based Min-Sum algorithm [24] is slightly modified in this section to maximally exploit the confidence bits in 2-bit intrinsic and extrinsic messages for a best decoding performance. The check-to-variable and variable-to-check message updating phases are formulated in (5.3) and (5.4), where $R_{mn}^{(k)}$, $L_{mn}^{(k)}$, and I_n stand for the check-to-variable, variable-to-check, and intrinsic messages, respectively.

$$R_{mn}^{(k)} = \prod_{i \in N(m) \backslash n} sign(L_{mi}^{(k-1)}) \times \min_{i \in N(m) \backslash n} | L_{mi}^{(k-1)} | \tag{5.3}$$

$$L_{mn}^{(k)} = L_{n}^{(k)} = g(f(I_n) + \alpha \sum_{j \in M(n)} f(R_{jn}^{(k)})) \tag{5.4}$$

Function $f(\cdot)$ converts a two-bit message to an integer. Correspondingly, function $g(\cdot)$ is for converting an integer to a two-bit message. With regard to the computation of (5.4), three steps are needed.

1) The input 2-bit data I_n or R_{jn} is converted to an integer number. Because b_m=0 indicates a low confidence, the converted integer is set to 1 (i.e., the smallest positive integer). Similarly, b_m=1 represents a high confidence. Therefore, a larger integer W is assigned for the converted result. The optimum value of W can be determined through simulation.

2) The summation is performed.

3) The integer summation is converted back to a 2-bit message before storing into memory. The L_{mn} conversion threshold, T_L, is determined through simulation. The sign bit is never changed in the above data conversion steps.

Next, we use one high rate code and one low rate code mentioned before to further explain the 2-bit decoding approach and illustrate its decoding performance. For both codes, α in (5.4) is set to 1/2 in our simulation. For the rate-0.84 code, we choose T_y, W, and T_L to be 3/8, 7, and 6, respectively, through simulation. Thus, each intrinsic soft message is assigned a value using (5.5).

$$b_s b_m = \begin{cases} 01, & if\ y > 3/8 \\ 00, & if\ 3/8 \geq y \geq 0 \\ 10, & if\ 0 > y \geq -3/8 \\ 11, & if\ y < -3/8 \end{cases} \tag{5.5}$$

For the computation of (5.3), 2-bit messages read from memory are directly used. Each 2-bit computation output is directly stored back to memory. For the computation

expressed in (5.4), each 2-bit message from memory must be converted to an integer number as shown in the left two columns of Table 5.1. The addition result of (5.4) must be converted to a 2-bit message as shown in the right two columns of Table 5.1 before storing back to memory. Similarly, we choose T_y, W, and T_L to be 1/2, 6, 5, respectively, for the rate-0.5 code. In general, to determine the values of the three thresholds for a given LDPC code with simulation, a coarse search precision can be used in the beginning to roughly identify a small search range. Then simulation can be performed with a fine search precision in the small search range for a best decoding performance.

TABLE 5.1 DATA CONVERSION FOR THE RATE-0.84 CODE

Message from memory	integer for addition in (11)	Addition result of (11)	Message to memory
00	*1*	$6 > x \geq 0$	*00*
01	*7*	$x \geq 6$	*01*
10	*-1*	$0 > x > -6$	*10*
11	*-7*	$x \leq -6$	*11*

5.1.2 Decoding Performance Simulation

For a comparison, WBF-based algorithms are simulated using double precision. For the (2048, 1723) (6,32) rate-0.84 code, the maximum iteration number is set to 48. The proposed 2-bit decoding method outperforms the IMWBF algorithm by 0.7dB when the target BER is 10^{-7}. For the (1974, 987) (5,10) rate-0.5 code, the maximum iteration number is set to 120. It can be observed that the 2-bit decoding approach significantly outperforms WBF-based algorithms. Its decoding performance is 2.4 dB better than that of IMWBF algorithm when the target BER is 10^{-7}. In WBF-based algorithms, the information delivered from one iteration to the next iteration is only one or a few flipped decision bits. Soft messages, which require much more memory than decision bits, are not well exploited. Consequently, it has large performance loss compared to belief propagation

decoding algorithm. On the contrary, the optimized 2-bit decoding method maximally exploits the confidence bits in 2-bit intrinsic and extrinsic messages of APP based Min-Sum algorithm for best decoding performance. It should be pointed out that both the WBF-based algorithms and the proposed 2-bit approach are not well suited for decoding LDPC code with only very low column weights (2 or 3) because of large performance loss from SPA. On the other hand, the performance gap between SPA and the low complexity decoding approaches mentioned before decreases as the column weight of LDPC code increases.

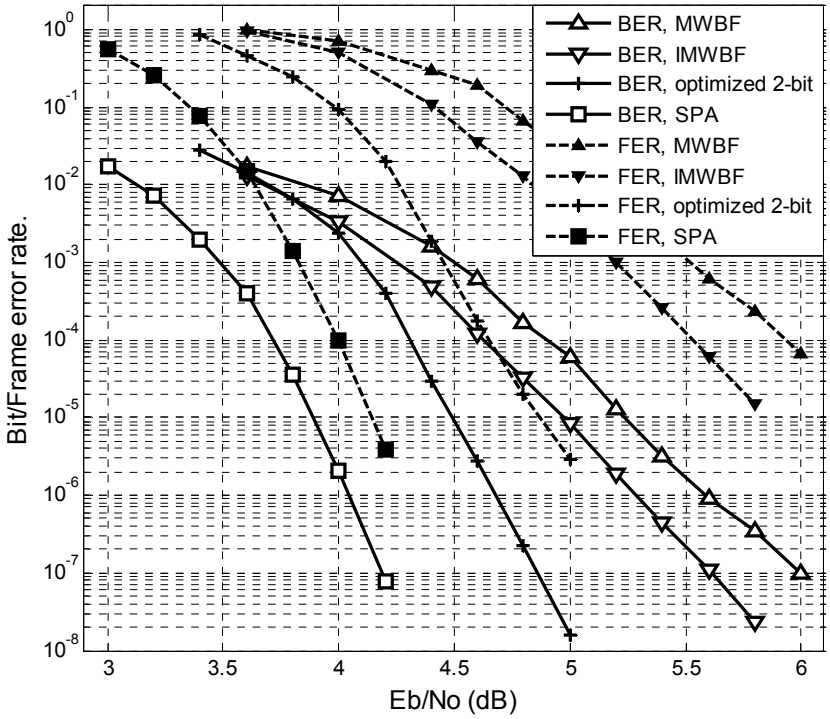


Figure 5.1 Performance of the (2048, 1723) rate-0.84 LDPC codes.

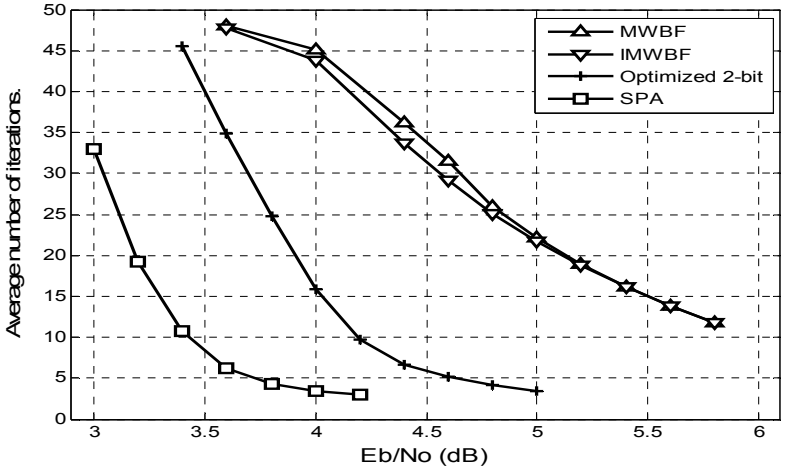


Figure 5.2 Average number of iterations for decoding the (2048, 1723) rate-0.84 LDPC codes.

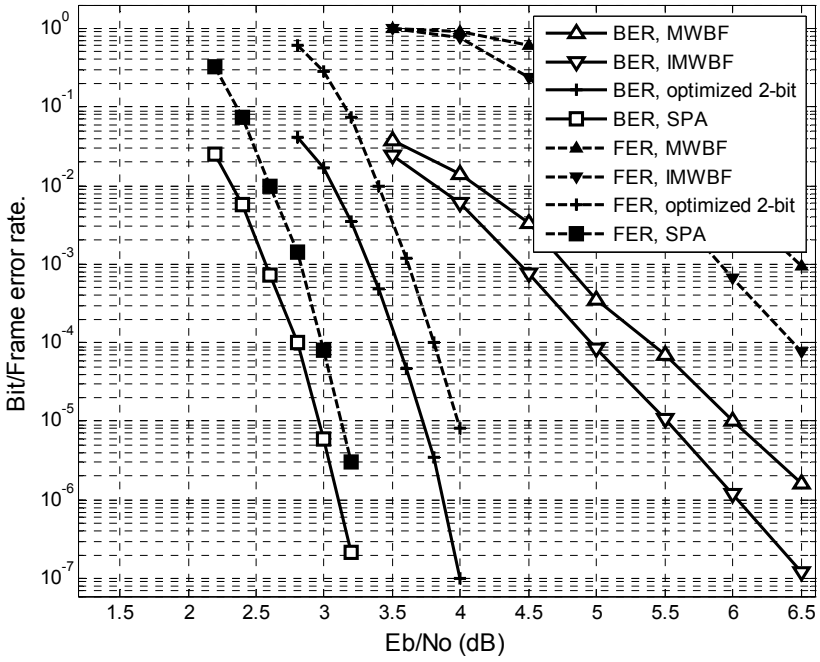


Figure 5.3 Performance of the (1974, 987) rate-0.5 LDPC codes

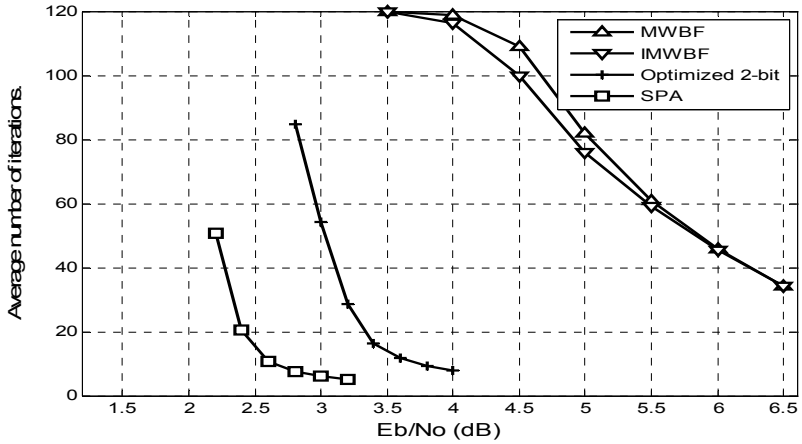


Figure 5.4 Average number of iterations for decoding the (1974, 987) rate -0.5 LDPC codes.

5.2 Low complexity 2-bit decoder design

5.2.1 Memory Reduction Scheme

To store the check-to-variable message R_{mn}, one simple method is to store all R_{mn} in their individual format. With this method, the 2-bit decoder needs much more memory than WBF-based decoders. One efficient method is to store the R_{mn} messages corresponding to one row of $\boldsymbol{H}$ matrix in a compressed format, i.e., the smallest magnitude (*min1*) and its index, the second smallest magnitude (*min2*), and all signs of the R_{mn} messages. An even more efficient method is employed in this design to reduce the memory requirement of the 2-bit decoder. For each row of $\boldsymbol{H}$ matrix, we only store *min1*, *min2*, and the product of all signs of the R_{mn} messages. The needed R_{jn} messages in (5.4) are completely recovered as (5.6) with the aid of L_n.

$$\begin{cases} |R_{jn}^{(k)}| = \begin{cases} min1_j^{(k)}, & if\ |L_n^{(k-1)}| > min1_j^{(k)} \\ min2_j^{(k)} & else \end{cases} \\ \\ \text{sgn}(R_{jn}^{(k)}) = S_j^{(k)} \times \text{sgn}(L_n^{(k-1)}) \end{cases} \tag{5.6}$$

Apparently, the computation in (5.6) increases the complexity of variable node unit. However, because only two bits are used for each message, the arithmetic computation in (5.6) and (5.4) are equivalent to very simple combinatorial logic computations. We will further discuss the hardware complexity of computation units later.

The estimated memory requirement for the proposed 2-bit decoder is shown in Table 5.2. *N* and *M* are the column and row dimensions of parity-check matrix of the considered LDPC code. Dual-port (DP) memory is assumed for appropriate memory modules to support simultaneous read and write operations. We also assume that the needed hardware resource of each dual-port memory bit is twice as that of each single-port (SP) memory bit. For a comparison, the memory requirement of MWBF decoder is estimated and listed in Table 5.3. We assume that three bits are used to quantize the magnitude of each received soft message for that method since 2-bit quantization is not acceptable. Our simulation shows that 2-bit quantization causes more than 0.5 dB performance loss compared to the floating-point simulation for either considered code. From our detailed analysis, we know that MWBF decoder requires the minimum hardware resource for both memory and computation unit among various modified WBF-based decoders. It can be concluded that the ratio of memory requirement for MWBF decoder and the 2-bit decoder is 1:1.2 for both of the rate-0.84 code and the rate-0.5 code.

TABLE 5.2 MEMORY REQUIREMENT OF THE 2-BIT DECODER

Message	Parameterized memory requirement	Memory for the rate-0.84 code	Memory for the rate-0.5 code
I_n (SP,2-bit)	$N \times 2$	4,096	3,948
min1, min2, and sign (DP, 3-bit)	$M \times (1+1+1) \times 2$	2,304	5,922
Hard-decision (DP, 1-bit)	$N \times 2$	4,096	3,948
$\lvert L_n \rvert$ (DP, 1-bit)	$N \times 2$	4,096	3,948
Total	$6N + 6M$	14,592	17,766

TABLE 5.3 MEMORY REQUIREMENT OF MWBF DECODER

Message	Parameterized memory requirement	Memory for the rate-0.84 code	Memory for the rate-0.5 code
$\lvert y_n \rvert$ (SP,3-bit)	$N \times 3$	6,144	5,922
w_{mn} (SP, 3-bit)	$M \times 3$	1152	2961
Hard-decision (DP, 1-bit)	$N \times 2$	4,096	3,948
Check-sum (DP, 1-bit)	$M \times 2$	768	1,974
E_n		0	0
Total	$5N + 5M$	12,160	14,805

5.2.2 Computation Units Design

Bcause only two bits are used for each message in the proposed 2-bit decoder, the arithmetic computation in (5.3), (5.6) and (5.4) are equivalent to very simple logic computations. Thus the hardware cost is very small. To illustrate its details, for simplicity and clarity, let us assume that the degree of variable node is 3 and the degree of check node is 6.

Fig. 5.5 shows the structure of the check node unit (CNU) for the optimized 2-bit decoding method. As we explained before, CNU is for calculating *min1*, *min2*, and the product of all signs of the R_{mn} messages. The portion below the dashed line is for generating the *min1* and *min2* from 6 1-bit inputs. The computation is performed in two stages. In the first stage, each *M1* unit is used to calculate the smallest and the second

smallest values from three 1-bit inputs. The logic function of the *M1* unit is shown in (5.7). In the second stage, an *M2* unit is used for generating the values of *min1* and *min2*. The logic equation of the *M2* unit is given in (5.8). The product of all sign bits is calculated using a XOR-tree. The output of a CNU is composed of 3 bits. The structure can be easily extended if the number of inputs is more than 6.

$$\begin{cases} a_1 = m_1 m_2 m_3 \\ a_2 = m_1 m_2 + m_3 (m_1 + m_2) \end{cases} \tag{5.7}$$

$$\begin{cases} min1 \ = a_1 b_1 \\ min2 \ = a_2 b_2 (a_1 + b_1) \end{cases} \tag{5.8}$$

Fig. 5.6 shows the structure of the variable node unit (VNU). Its task is performed in 4 steps: 1) Recovering three $R_{mn}^{(k)}$ messages in their individual format as expressed in (5.6). Each $R_{mn}^{(k)}$ is recovered by a compare-and-select (C&S) unit. 2) Converting 2-bit data to integer number. 3) Performing summation as shown in (5.4). 4) Converting the summation to 2-bit data before storing into memory. A VNU has 5 inputs: one 2-bit intrinsic message, I_n, one 2-bit extrinsic message, $L_n^{(k-1)}$, and three 3-bit intermediate messages. Each intermediate message is computed by a CNU. The logic function of C&S unit is given in (5.9) which is equivalent to (5.6). In (5.9), $|x|$ stands for the 1-bit magnitude of x. Step-2 and step-3 are performed with look-up table (LUT) and adder* unit, respectively. Because each 4-bit input of an adder* unit has only 4 possible values (see Table 5.1), the hardware cost of an adder* unit is much less than that of a 4-input 4-bit adder tree. Stpe-4 is completed by a 6-to-1 combinational logic unit. The sign bit of the output is identical to that of the summation value given by the adder* unit.

$$\begin{cases} | R_{mn}^{(k)} | = min1_m^{(k)} + min2_m^{(k)} \overline{| L_n^{(k-1)} |} \\ sign(R_{mn}^{(k)}) = S_m^{(k)} \oplus \mathrm{sgn}(L_n^{(k-1)}) \end{cases} \tag{5.9}$$

To quantitatively depict the hardware complexity of the discussed computation units, we used Verilog to model the check node unit and variable node unit addressed above. The two node units are synthesized using Leonardo Spectrum with TSMC 0.35*um* technology. All syntheses are optimized for area. The synthesis results are listed in Table 5.4.

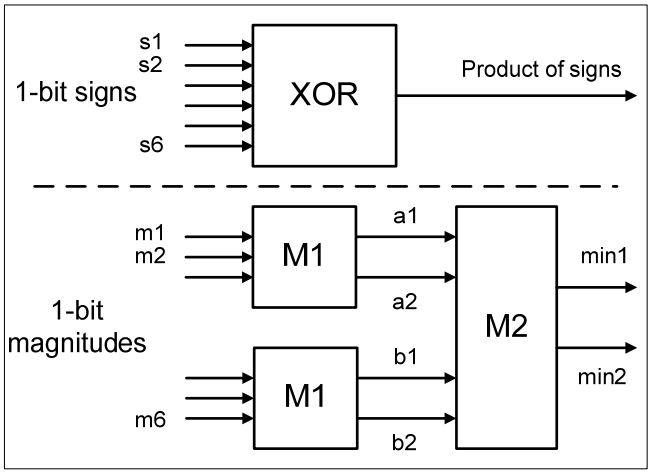


Figure 5.5 Structure of the check node unit for the optimized 2-bit decoding approach.

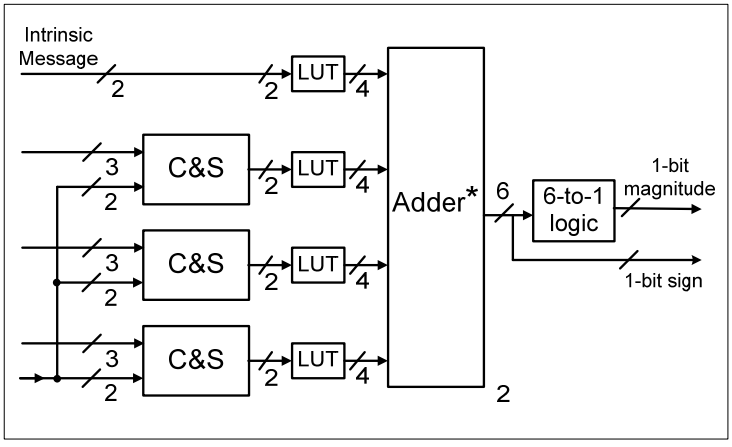


Figure 5.6 Structure of the variable node unit for the optimized 2-bit decoding approach.

For a comparison, the computing units for a WBF-based decoder are briefly discussed in the next. The computation core for the bit flipping operation is shown in Fig. 7. Block 1 is for (2.22). The maximum value of weighted check sum is computed by block 2. The sm=>2's unit is for sign-magnitude to 2'complement conversion. To be consistent with Section 5.2.1, 3-bit quantization is assumed for the magnitude of each soft message. We make the same assumption as above about node degrees. The hardware cost of the computing unit for (2.18) in WBF-based decoder is ignored. The synthesis result using the same technology and optimization constraints is shown in Table 5.4. For a fair comparison, we assume the same parallelism level is adopted for either decoder. Therefore, we need only compare the complexity of single copy of those computation units. It can be observed that CNU and VNU for 2-bit decoder need less logic gates than the computation units for WBF-based decoder.

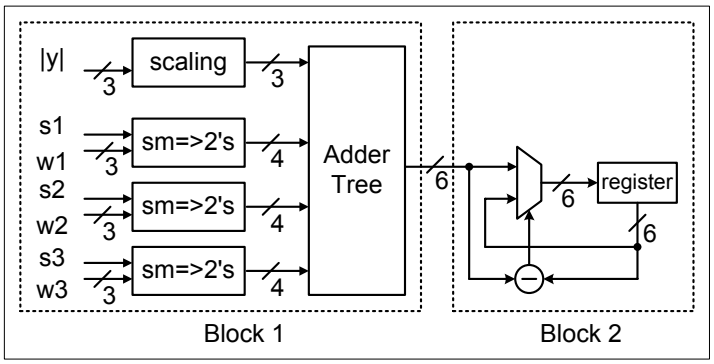


Figure 5.7 The computation core needed in the bit flipping operation for a WBF-based decoder.

For the considered two LDPC codes, the check node degree and variable node degree are respectively larger than those of the above example. Roughly speaking, the complexity of a computation unit linearly scales up as the number of inputs increases. Overall, the complexity ratio for computation units of LDPC decoders using different algorithms remain the same or similar for different rate codes. Therefore, we conclude that

the proposed 2-bit decoder has comparable, if not less, hardware in the computation core with that of WBF-based algorithms.

TABLE 5.4 COMPLEXITY OF COMPUTATION UNITS FOR THE OPTIMIZED 2-BIT AND WBF-BASED DECODER.

Computation units for the optimized 2-bit decoder	Gates
Check node unit.	19
Variable node unit.	55
Computation unit for WBF-based decoder	Gates
Computation core needed in the bit flipping operation. (Block 1 requires 90 gates and Block 2 requires 67 gates)	157

5.3 Summary

We have studied VLSI implementation issues for WBF-based LDPC decoding algorithms and presented an optimized 2-bit decoding approach. The proposed decoding approach significantly outperforms the state-of-the-art WBF-based decoding algorithms for the considered LDPC codes while maintaining comparable hardware complexity to WBF-based algorithms. Therefore, the proposed 2-bit decoding approach is more attractive than WBF-based decoding approaches in practical low complexity VLSI implementation of LDPC decoders.

6 REDUCING ITERATIONS FOR LDPC CODES

LDPC codes are decoded using iterative decoding algorithms. To increase decoding speed, it is highly desired to reduce the number of decoding iterations without significant performance loss. In this chapter, the decoding schemes which can reduce the number of decoding iterations for decodable and undecodable blocks are presented. In addition, we demonstrate that the decoding convergence of WBF-based algorithm can be significantly speeded up with a multi-threshold detection scheme.

6.1 Extended Layered Decoding of LDPC Codes

To improve decoding convergence, various rescheduled message passing schemes are presented. Sharon *et al.*, [52], proposed a message passing scheme based on a serial update of check nodes' messages with SPA. In [49], Hocevar developed a low complexity LDPC decoder using layered decoding approach, where the SPA algorithm is used for the computation of each layer. Mansour and Shanbhag, [50], proposed a turbo-decoding message passing (TDMP) decoding algorithm. All these approaches achieve significantly faster convergence speed and slightly better decoding performance over TPMP SPA. However, one common constraint in all these approaches is that the column weight of each layer is at most 1. More related works can be found in [28][66][67].

For many LDPC codes, such as irregular repeat accumulation (IRA) code [9], Euclidean geometry (EG) based code [23], and progressive edge growth (PEG) code [30], to satisfy the constraint of the standard layered decoding approach, the number of rows in each layer of parity check matrices could be very small. Because the computation has to be performed layer by layer in the layered decoding, the achievable decoding parallelism is thus limited, which is undesired for high throughput decoding.

To tackle the problem, we propose an extended layered decoding approach. Given any LDPC code, the parity check matrix can be partitioned into any number of horizontal layers and no constraint in the column weight of each layer is enforced. It enables more flexibility in high-throughput LDPC decoder design with layered decoding since many rows can be arranged in one layer and processing more rows per cycle over the standard approach becomes possible.

6.1.1 The Proposed Layered Decoding Approach

In the standard layered decoding approach, the parity check matrix of LDPC code is partitioned into L layers: $H^T = \left[H_1^T \; H_2^T \cdots H_L^T\right]$. Each layer defines a supercodes C_l and the original LDPC code is the intersection of all supercodes: $C = C_1 \cap C_2 \cdots \cap C_L$. The column weight of each layer is at most 1.

We propose an extended layered decoding approach which removes the constraint in column weight of each layer. The reliability message from layer l to l+1 for variable node v is represented by L_v^l. The message passing in the k^{th} iteration is as follows:

For the first layer, *i.e.*, l = 1, the extrinsic messages are updated using (6.1a), (6.2), and (63a). For other layers, *i.e.,* l = 2, 3, …, L, the updating of extrinsic message is expressed in (6.1b), (6.2), and (6.3b).

$$L_{cv}^{k,l} = L_v^{(k-1),L} - R_{cv}^{(k-1),l} \quad , \tag{6.1a}$$

$$L_{cv}^{k,l} = L_v^{k,(l-1)} - R_{cv}^{(k-1),l} \quad , \tag{6.11b}$$

$$R_{cv}^{k,l} = \prod_{n \in N(c) \backslash v} sign(L_{cn}^{k,l}) \times \Psi\left\{ \sum_{n \in N(c) \backslash v} \Psi(L_{cn}^{k,l}) \right\} \quad , \tag{6.22}$$

$$L_v^{k,l} = L_v^{(k-1),L} - \sum_{m \in M^l(v)} R_{mv}^{(k-1),l} + \sum_{m \in M^l(v)} R_{mv}^{k,l} \quad , \tag{6.3a}$$

$$L_v^{k,l} = L_v^{k,(l-1)} - \sum_{m \in M^l(v)} R_{mv}^{(k-1),l} + \sum_{m \in M^l(v)} R_{mv}^{k,l}, \quad (6.3b)$$

where, $M^l(v)$ denotes the set of check nodes in the l^{th} layer connected to the variable node v.

It can be observed that if the column weight of each layer is at most 1, the proposed approach coincides with the standard layered decoding approach with SPA. On the other hand, if we view the parity check matrix as one layer, the proposed approach becomes TPMP SPA. The extended layered decoding approach can be easily extended to the approximations of SPA such as MSA and A-min [32] by replacing (6.2) with the corresponding formula for check-to-variable message updating. Our simulation results show that, with the proposed approach, no performance is sacrificed compared to TPMP SPA. Similar to the standard layered decoding approaches, faster decoding convergence is achieved as well.

6.1.2 Overlapped Message Passing Decoding

For high–speed applications, it is desired to perform check-to-variable message updating and reliability message updating in parallel to maximize the decoding throughput. In this section, we propose a low complexity overlapped message passing scheme for block-serial decoders [54] [69]. In a block-serial decoder, the data corresponding to a layer are processed block column by block column in a serial fashion. A block column includes one or multiple blocks. Each CNU loads one input data per clock cycle and all check node units (CNUs) associated with a block column work in parallel. After the computation of check-to-variable messages corresponding to a layer is completed, all reliability messages are updated. In [54] [69], a mirror memory for storing all the reliability messages is introduced to enable overlapped message passing. A close study of the data flow in (6.1)-(6.3) shows that the mirror memory for L_v messages can be eliminated without sacrificing

the decoding throughput. In the new approach, MSA for check-to-variable message updating is adopted.

The key computation of a check node unit (CNU) is to generate the smallest magnitude, *min1*, and the second smallest magnitude, *min2*. Fig. 6.1 shows the computation core of a serial CNU. Before the computation of check-to-variable messages for one row is completed, the scratch registers, m1_reg and m2_reg, store the temporal value of *min1* and *min2*. After the last L_{cv} message is delivered to CNU, the final values of *min1* and *min2* can be sent out from m1_reg and m2_reg, respectively. The dashed-lines are for updating enable signals. If $m1 > x$, m1_reg is updated using the value of x Otherwise, update is disabled. If $m2 > x$, m2_reg is updated using the value of either x or m1. Otherwise, its content is unchanged.

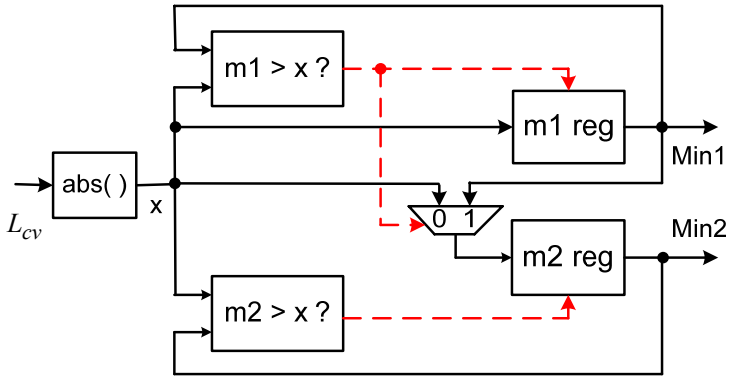


Figure 6.1 Serial computation of the smallest and the second smallest magnitude.

To enable overlapped message updating, in the k^{th} iteration, the data flow of a block-serial decoder is scheduled as shown in Fig. 6.2. In the first stage, (6.3) is performed. The reliability messages corresponding to layer (l-1) are computed and stored into memory. Then, the extrinsic messages for layer l are computed. In the last stage, the check-to-variable message computation is performed. It should be noted that the above

computations are performed block column to block column for every layer. In Fig. 6.2, $Min1^{k,l}$ and $Min2^{k,l}$ are the intermediate computation results. As soon as the data corresponding to all block columns in layer l are processed, the final values of $R_{cv}^{k,l}$ are generated immediately and stored into memory. It can be seen that no mirror memory L_v message is needed in the improved overlapped message passing scheme.

If the column weight of each layer is at most one, the data flow can be further simplified to avoid the duplicated subtraction occurred in stage 1 and stage 2 by storing extrinsic variable-to-check messages instead of reliability messages.

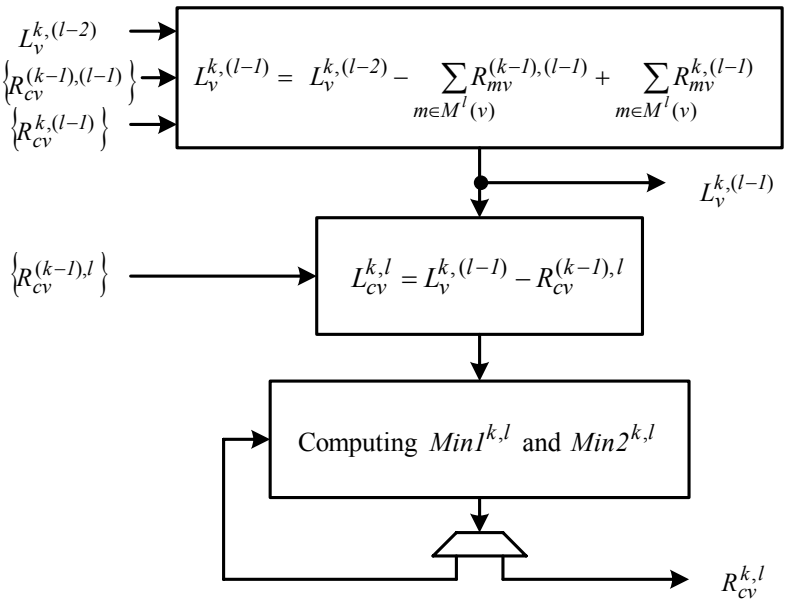


Figure 6.2 The data flow of overlapped message passing scheme.

6.1.3 Simulation Results

Two LDPC codes are considered in our simulations. One is (2038, 1723) rate-0.84 Reed-Solomon code based (6, 32) regular code [18]. Its parity check matrix is an array of

6×32 permutation matrices. The other is (1008, 504) rate-0.5 irregular code constructed with the progressive edge growth method [30]. It has variable and check node degree distribution as follows:

$$\lambda(x) = 0.477x^2 + 0.281x^3 + 0.035x^4 + 0.097x^5 + 0.009x^7 + 0.001x^{14} + 0.100x^{15},$$

$$\rho(x) = 0.01x^7 + 0.98x^9 + 0.01x^9.$$

In all simulations, the maximum number of iteration is set as 50. The $\boldsymbol{H}$ matrix of the rate-0.84 regular code is evenly partitioned into 6 layers, 3 layers, and 2 layers such that the column weight of each layer is 1, 2, and 3, respectively. For the first partitioning case, the proposed approach coincides with standard layered decoding approach. Fig. 6.3 shows the average number of iterations and bit error rate (BER) performance of different partitioning cases. We can see that the proposed approach converges faster than TPMP SPA. Meanwhile, it has better decoding performance than TPMP SPA. Without loss of generality, the $\boldsymbol{H}$ matrix of the rate-0.5 code is evenly partitioned into 504 layers, 6 layers, 3 layers, and 2 layers. Fig. 6.4 shows that the extended layered decoding approach converges much faster than TPMP SPA for the rate-0.5 PEG LDPC code. It can be observed that the proposed method outperforms the TPMP SPA in all partitioning cases.

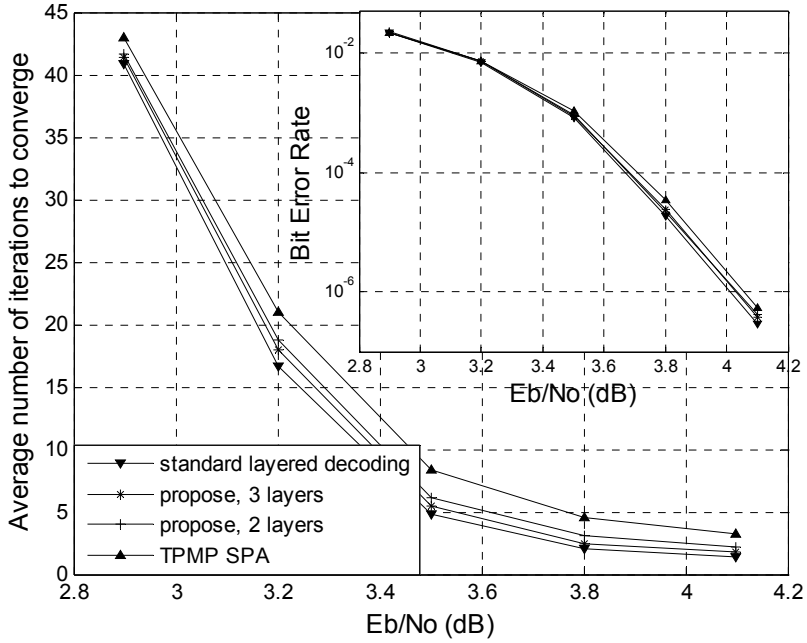


Figure 6.3 Average number of iterations and bit error rate (BER) for the rate-0.84 code with standard layered decoding, proposed approach, and TPMP SPA decoding.

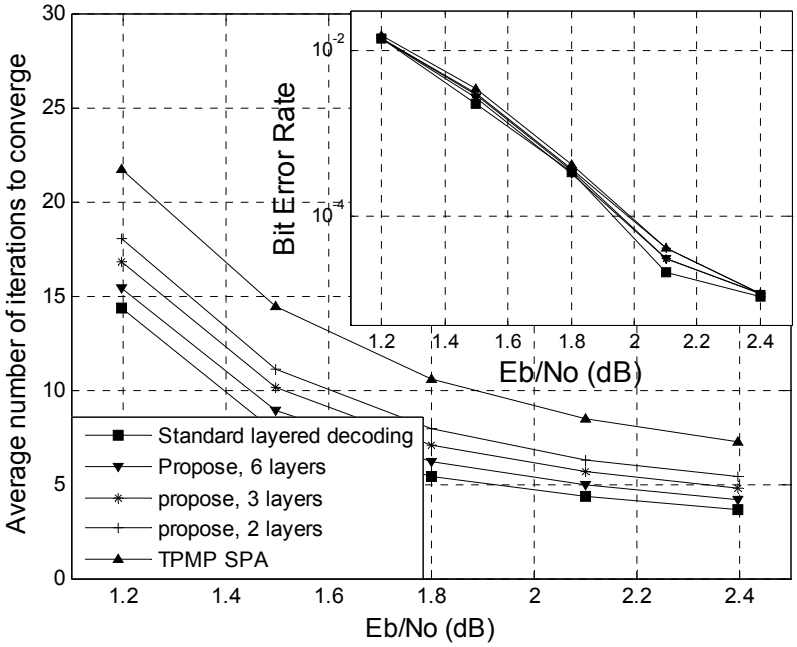


Figure 6.4 Average number of iterations and BER for the rate-0.5 code with standard layered decoding, proposed approach, and TPMP SPA decoding.

6.2 An Efficient Early Stopping Scheme for LDPC Decoding

It happens frequently at low to medium signal to noise ratios (SNRs) that a valid codeword can not be found even through a large number of decoding iterations are performed. An efficient scheme to detect such undecodable cases as early as possible and hence to avoid unnecessary computations is highly desired in practice. In the literature, various early stopping criteria [22][40][70] for turbo codes decoding have been proposed. A comprehensive overview is presented in [41]. Because of the similarity between the turbo decoding and LDPC decoding, some existing early stopping criteria for turbo decoding can be adapted for LDPC decoding. However, they may cause considerable performance loss at high SNRs. Recently, a convergence of mean magnitude (CMM) early stopping criterion [71] optimized for LDPC decoding was presented. This criterion is based on the evolution of the average magnitude of the log-likelihood ratio (LLR) messages in the decoding process. This approach can effectively detect undecodable cases. However, it has very high computation overhead because it involves the accumulation of the absolute values of all LLR messages and a large bit-width multiplication operation.

By exploring the statistic characteristics of extrinsic and reliability messages computed during the decoding process, we found that the sign of extrinsic messages and reliability messages can be utilized to predict whether the received block is decodable or not. For the convenience, the equations for check-to-variable node message passing of SPA are rewritten as (6.4) and (6.5).

$$S_c = \prod_{n \in N(c)} sign(L_{cn}), \tag{6.4}$$

$$R_{cv} = S_c \times sign(L_{cv}) \times \Psi\left\{\sum_{n \in N(c)} \Psi(L_{cn}) - \Psi(L_{cv})\right\}, \tag{6.5}$$

The check-sum P_c of parity equation $\boldsymbol{zH}^T$ corresponding to check node c is computed by (6.6).

$$P_c = \sum_{v \in N(c)} \oplus z_v \tag{6.6}$$

where $\sum\oplus$ represents binary addition and z represents the hard-decision vector.. If $P_c = 0$ for any check node c, a valid code is found and the decoding process can be terminated. In VLSI design, (6.4) is implemented in the same way as (6.6). Let S_S denote the summation of the binary mapping of every sign product computed in (6.4) (i.e., $S_S = \sum_{c=0}^{M-1} S_c$) and S_P as the summation of the check-sum of every parity equation computed in (6.6) (i.e., $S_P = \sum_{c=0}^{M-1} P_c$). In LDPC decoding, the value of S_P in the k^{th} iteration, S_P^k, decreases as k increases (even though a certain extent of fluctuation may occur) if the decoded block is decodable. S_P converges to zero when a valid code is found. It can be observed that the convergence of S_S^k is very similar to that of S_P^k during the decoding process. Both S_P^k and S_S^k can be utilized to detect undecodable blocks. In this design, S_S^k is exploited for the consideration of easy hardware implementation.

For undecodable blocks, usually S_S^k keeps a large value and fluctuates in a small dynamic range of magnitudes. If a received block is decodable, the variation duration of S_S^k generally is short and S_S^k goes to zero along a steep slope in most cases. Even if in the cases that S_S^k keeps fluctuating with a long duration, the fluctuant magnitude is much lager than that of undecodable cases. Therefore, the convergence of S_S^k can be exploited to predict the decoding convergence before the maximum number of iterations is reached or a valid codeword is found.

It should be pointed out that any individual detection trial may have three possible outcomes, *i.e.,* hit, miss detection, and false alarm. In LDPC decoding, a false alarm causes the performance loss. Thus, early stopping schemes should be optimized to minimize the false alarm rate at all SNRs. However, the block error rate is very small at high SNRs and

the computation power for undecodable blocks is very small, the early stopping scheme can be disenabled at high SNRs to avoid performance loss and save computation overhead. Based on the above discussion, an early stopping scheme for detecting the undecodable blocks is developed as follows:

Step 1:
Roughly check the SNR in the first iteration.
If it is at low to medium range, step 2a is performed, otherwise step 2b is performed.
Step 2a:
counter:=0
if (fluctuation occurred) **then**
$\Delta := S_S^{k-1} - S_S^k$;
if ($\Delta > 0$) **then**
if ($\Delta < \Delta_{TH}$) **then**
counter:=counter+1;
else
counter:=0;
endif
endif
if (counter>T) **then** stop decoding
else continue to the next iteration
endif
endif
Step 2b:
Continue to the next iteration.

In the step 2a, Δ_{TH} and T are two predetermined thresholds by simulation. S_S^k converges once if $\Delta > 0$ is satisfied. Under this condition, $\Delta < \Delta_{TH}$ indicates that a slow convergence occurs. T is for recording the duration of slow convergence. The proposed early stopping scheme can be implemented with a $\log_2 M$ -bit accumulator for counting the

number of 1s from the binary mapping of S_c and a few additional logic gates. Therefore, the hardware overhead is very small. Simulation results have demonstrated that the proposed scheme can significantly reduce the average number of decoding iterations at low to medium SNRs. The performance loss is very small at all SNRs. We have published more details in [42].

6.3 The Fast Decoding Scheme for WBF-based Algorithms

6.3.1 Multi-threshold Bit Flipping Scheme

For long codeword and/or low SNR channel, the hard-decision vector **z** given in the initialization step of WBF-based algorithms has a large number of errors. If only one bit is flipped per iteration, a large number of iterations are required, which leads to very long decoding latency. For the original bit-flipping algorithm, Gallager suggested to flip a decision bit which is contained in more than *b* unsatisfied parity-check equations [1]. The optimized integer *b* is a function of decoding iteration, check node degree and variable node degree. We extend the approach and employ multi-threshold scheme as the following to speed up the decoding process of WBF-based algorithms.

if k<4 **then** flip the bit n if $E_n^k > \delta_1$
else if k<8 **then** flip the bit n if $E_n^k > \delta_2$
else if k<12 **then** flip the bit n if $E_n^k > \delta_3$
else flip the bit corresponding the largest E_n^k

In WBF-based algorithms, the hard-decision bit z_l corresponding to the maximum value of E_n^k in (2.20), (2.22) or (2.24) is flipped. If the decoding process converges, the maximum value of E_n^k usually decreases as k increase. Thus, the condition of $\delta_1 > \delta_2 > \delta_3$

should be satisfied. For a specific LDPC code, the distribution of the maximum value of E_n^k at a given SNR can be easily found through simulation. It facilitates the setting of the initial values of δ_i. With additional performance simulation, the final value of δ_i can be determined. In practice, the values of δ_i optimized at a medium SNR value are also suitable for low and high SNRs. The simulations on two LDPC codes are presented. One is a (2048, 1723) (6, 32) rate-0.84 permutation matrix based LDPC code. The other is a (1974, 987) (5, 10) rate-0.5 quasi-cyclic LDPC code. For simplicity, the two codes are labeled as rate-0.84 code and rate-0.5 code, respectively. Table 6.1 lists the values of δ_i used in our simulations. It is shown that the proposed scheme can significantly speed up the WBF-based decoding algorithms.

TABLE 6.1 THE VALUE OF δ_i FOR THE RATE-0.84 CODE AND THE RATE-0.5 CODE

	δ_1	δ_2	δ_3
Rate-0.84 code	0.8	0.55	0.35
Rate-0.5 code	1.5	1.0	0.6

6.3.2 Performance Simulation

We simulated the IMWBF algorithm using double precision. For the (2048, 1723) (6,32) rate-0.84 code, the maximum iteration number is set to 48. We can see from Fig. 6.12 and Fig. 6.13 that the average number of iterations of IMWBF algorithm is significantly reduced by the fast decoding scheme with negligible bit error rate (BER) and frame error rate (FER) performance loss. The proposed 2-bit decoding method outperforms the IMWBF algorithm by 0.7dB when the target BER is 10^{-7}. For the (1974, 987) (5,10) rate-0.5 code, the maximum iteration number is set to 120. We can see from Fig. 6.14 and Fig. 6.15 that the proposed fast decoding scheme can reduce the average number of

iterations needed by IMWBF algorithm to one third at the SNR of 5.5 dB. The introduced performance loss is negligible.

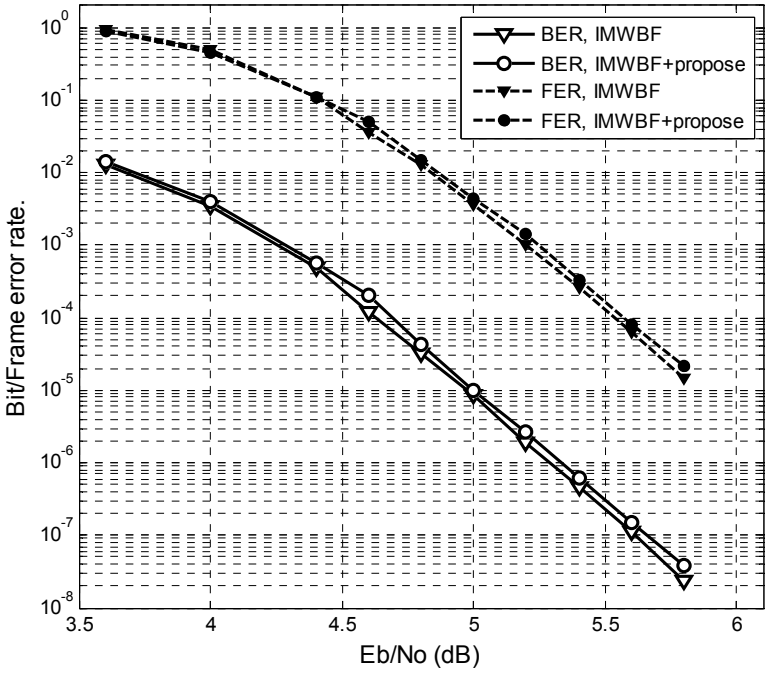


Figure 6.5 Performance of the (2048, 1723) rate-0.84 LDPC code.

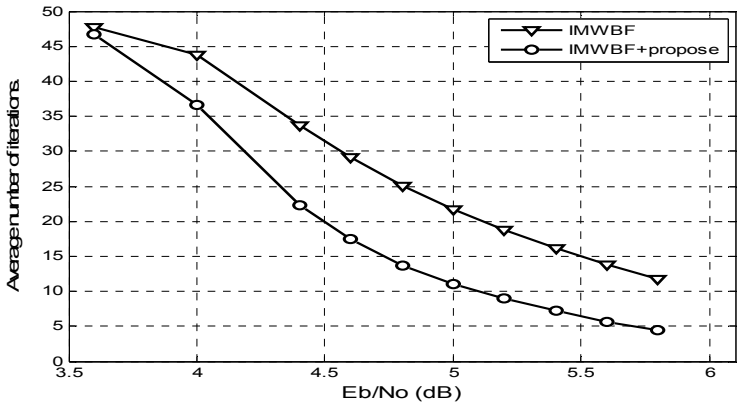


Figure 6.6 Average number of iterations for decoding the (2048, 1723) rate-0.84 LDPC code.

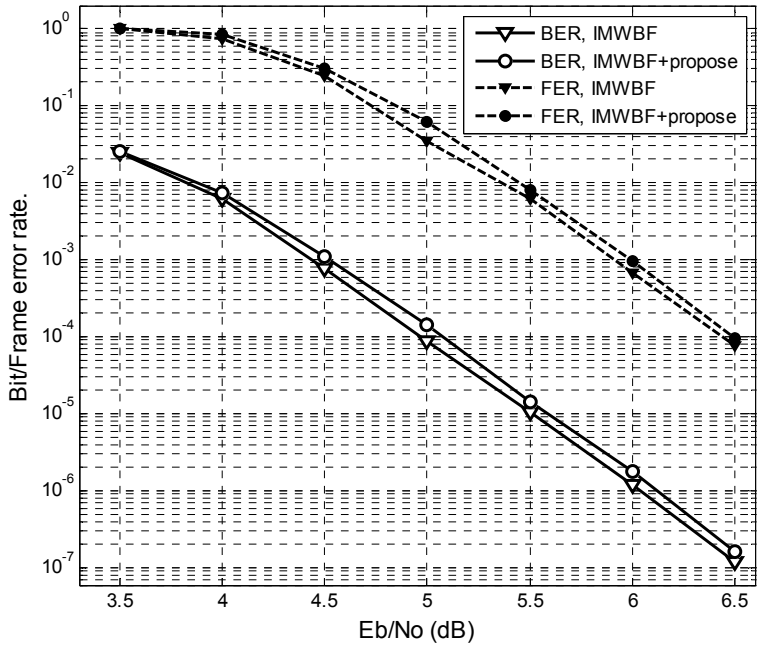


Figure 6.7 Performance of the (1974,987) rate-0.5 LDPC code.

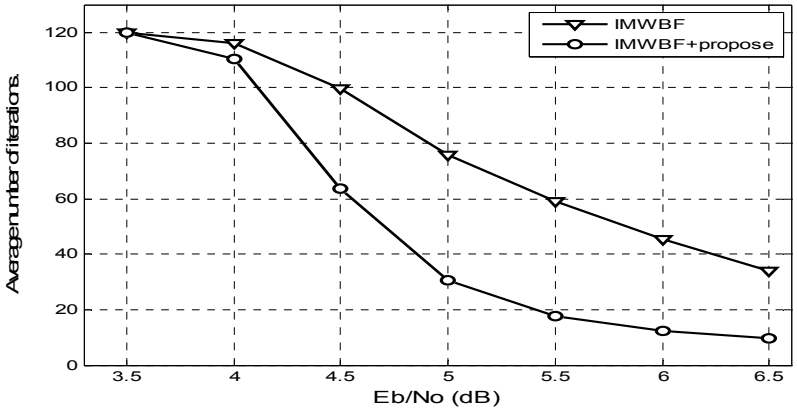


Figure 6.8 Average number of iterations for decoding the (1974,987) rate-0.5 LDPC code.

6.4 Summary

We have discussed the approaches to reduce the number of decoding iterations and hence to increase decoding speed. First, an extended layered decoding approach has been presented. It has been shown that it converges faster and has better error correction capability than the conventional TPMP LDPC decoding algorithm. The approach is suitable for both random and structured LDPC codes. Then, an efficient early stopping scheme has been proposed to detect undecodable blocks as early as possible in order to avoid unnecessary computation. The two approaches can be combined with SPA and its various near optimum approximate algorithms to speed up LDPC decoding. Finally, we demonstrate that the decoding convergence of WBF-based algorithm can be significantly speeded up with a multi-threshold detection scheme.

7 CONCLUSIONS AND FUTURE WORKS

7.1 Conclusions

This research has investigated various VLSI design issues of LDPC decoders and has proposed low-complexity high-speed decoder architectures to reduce VLSI implementation complexity and improve decoding throughput.

To reduce hardware implementation complexity of LDPC decoder, we have proposed a memory efficient partially parallel decoder architecture, which stores soft messages in the Min-Sum decoding algorithm in a compressed form. In general, over 30% memory can be saved. Various optimization methods have been presented to further reduce the implementation complexity and minimize the critical path.

We have investigated various design approaches for high throughput LDPC decoders. We have proposed an efficient message passing decoder architecture to reduce interconnect complexity. If using 4-bit quantization for decoding a (2048, 1723) (6, 32) LDPC code, the approach can reduce 54% outgoing wires per variable node unit and 90% outgoing wires per check node unit. Then, by exploiting the regularity in parity check matrices of QC-LDPC codes, we have developed a high throughput decoder architecture for QC-LDPC codes. It has been estimated that 4.7Gbit/sec decoding throughput for a (3456, 1728) (3, 6) QC-LDPC code can be achieved. In addition, we have also implemented an enhanced partially parallel decoder architecture with FPGA for a (8176, 7156) Euclidian geometry based QC-LDPC code. A worst-case source information decoding throughput (at 15 iterations) over 170Mbps is achieved.

For cost sensitive applications, we have proposed an optimized 2-bit soft decoding approach. The implementation complexity of the proposed method is comparable to WBF-

based algorithms. However, the proposed approach achieves much better decoding performance and faster convergence speed.

LDPC codes are decoded using iterative decoding algorithms. We have discussed the approaches to reduce the number of decoding iterations and hence to increase the decoding speed. First, we have proposed an extended layered decoding approach. Simulations on both random and structured LDPC codes have shown that the proposed approach converges faster than conventional TPMP decoding algorithm. Second, it happens frequently that a valid codeword can not be found even though a large number of decoding iterations are performed at low to medium signal-to-noise ratios. We have proposed an efficient early stopping scheme to detect such undecodable cases as early as possible in order to avoid unnecessary computation. Finally, we have demonstrated that the decoding convergence of WBF-based algorithm can be significantly speeded up with a multi-threshold detection scheme.

7.2 Future Work

This research has assumed that a binary codeword is BPSK modulated and transmitted through an AWGN channel. It has been found that non-binary LDPC codes have better performance than binary LDPC codes if the block length in binary bits are the same [6][32]. Unfortunately, the message passing decoding algorithm for non-binary LDPC code is more complex than that for binary LDPC codes. Recently, a few researchers have explored reduced complexity decoding algorithms for non-binary LDPC codes [29][72]. However, little effort has been made for investigating the VLSI implementation issues. To facilitate the applications of LDPC codes designed in high order Galois fields, the decoding complexity has to be significantly reduced. Our research work can be

extended to low complexity decoding of non-binary LDPC codes in both algorithm and architecture level. Extensive efforts are needed in efficient VLSI design for non-binary LDPC decoders.

8 BIBLIOGRAPH

[1] R. G. Gallager, “Low-density parity-check codes,” IRE Transactions on Information Theory, vol. IT-8, pp. 21-28, Jan. 1962.

[2] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” IEEE Trans. Inform. Theory, vol. 42, pp. 429-445, Mar. 1996.

[3] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rate”, IEEE Trans. Inform. Theory, vol. IT-20, pp. 284–287, March 1974.

[4] R. M. Tanner, D. Sridhara, A. Sridharan, T.E. Fuja, and D. J. Costello, “LDPC block and convolutional codes based on circulant matrices”, IEEE Trans. Inform. Theory, vol. 50, pp. 2966-2984, Dec. 2004.

[5] D. J. C. MacKay, “Good error-correcting codes based on very sparse matrices,” IEEE Trans. Inform. Theory, vol. 45, pp. 399-431, Mar. 1999.

[6] M. Davey and D. J. C. MacKay, “Low Density Parity Check Codes over GF(q),” IEEE Commun. Lett., vol. 2, pp. 165-167, June 1998.

[7] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, “Practical loss-resilient codes,” in Proc. 29th Annu. Symp. Theory of Computing, 1997, pp. 150–159.

[8] T. Richardson and R. Urbanke, “Efficient encoding of low-density parity-check codes”, *IEEE Trans. on Inform. Theory*, vol 47, pp. 638–656, Feb. 2001.

[9] H. Jin, A. Khandekar, and R. McEliece, “Irregular repeat-accumulate codes,”, Int. Confe. on Turbo codes, Sept. 2000.

[10] Y. Zhang, W. E. Ryan, and Y. Li, “Structured eIRA codes with low floors,” in Proc. International Symposium on Information Theory, 2005. pp. 174-178, Sept. 2005.

[11] M. Yang, W. E. Ryan and Y. Li, “Design of efficiently encodable moderate-length high-rate irregular LDPC codes,” *IEEE Trans. on Communications,* vol. 52, pp. 564-571, April 2004.

[12] Djurdjevic, Jun Xu, K. Abdel-Ghaffar, Shu Lin, "A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols," IEEE Communications Letters, vol 7, pp. 317-319, July 2003.

[13] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," IEEE Trans. on Info. Theory, vol. 50, pp. 1788-1793, Aug. 2004.

[14] J. L. Fan, "Array codes as low-density parity-check codes," in *Proc. 2nd Int. Symp. Turbo Codes*, Brest, France, Sept. 2000, pp. 545–546.

[15] Z. Li and B. V. K. V. Kumar, "A class of good quasi-cyclic low-density parity check codes based on progressive edge growth graph," Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 1990-1994, 2004.

[16] Z. Li, L. Chen, S. Lin, W. Fong and P. Yeh, "Efficient encoding of quasi-cyclic low-density parity-check codes", to appear in IEEE trans. on communications.

[17] S, Lin and D. J. Costello, Error Control Coding: Fundamentals and Applications,2nd edition, Prentice Hall, 2004.

[18] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols," IEEE Commun. Lett., vol. 7, pp. 317–319, July 2003.

[19] L. Chen, I. Djurdjevic, X. Jun,S. Lin and K. Abdel-Ghaffar, "Construction of quasi-cyclic LDPC codes based on the minimum weight codewords of reed-solomon codes," in proc. ISIT'04, pp. 239, June 2004.

[20] S. Song, L. Lan, S. Lin, and K. Abdel-Ghaffar, "Construction of Quasi-Cyclic LDPC Codes Based on the Primitive Elements of Finite Fields," in proc. CISS'06, pp. 835-838, March 2006.

[21] Y. Kou, S. Lin, and M. Fossorier, "Low density parity check codes based on finite geometries: a rediscovery and more," IEEE Trans. Inform.Theory, vol. 47, pp. 2711–2736, Nov. 2001.

[22] R. Y. Shao, S. Lin, and M.P.C.Fossorier, "Two simple stopping criteria for turbo decoding," *IEEE Trans. Comm.*, vol. 47, no. 8, pp. 1117 – 1120, Aug. 1999.

[23] L. Chen, J. Xu, I. Djurdjevic and S. Lin, "Near-Shannon-Limit Quasi-Cyclic Low-Density Parity-Check Codes," IEEE Transactions on Communications, vol. 52, pp. 1038-1042, Jul. 2004.

[24] J. Chen and M. P. C. Fossorier, "Decoding low-density parity check codes with normalized APP-based algorithm," GLOBECOM '01, vol. 2, pp.1026 – 1030, Nov. 2001.

[25] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," IEEE Trans. Commun., vol. 50, pp. 406-414, Mar. 2002.

[26] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, X. Hu, "Reduced-Complexity Decoding of LDPC Codes," IEEE Trans. on Commun., vol 53, pp. 1288-1299, Aug. 2005.

[27] J. Zhang and M. Fossorier, "A modified weighted bit-flipping decoding of low density parity-check codes," IEEE Commun. Lett., vol. 8, pp. 165–167, Mar. 2004.

[28] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," IEEE Asilomar Conf. on Signals, Sys. And Computers, pp. 8-15, Nov. 2002.

[29] D. Declercq and M. Fossorier, "Decoding Algorithms for Nonbinary LDPC Codes over GF(q)", IEEE Trans. on Commun., vol. 55(4), pp. 633-643, April 2007.

[30] Xiao-Yu Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementation of the sum-product algorithm for decoding LDPC codes," in Proc. IEEE Globecom, San Antonio, TX, Nov. 2001, pp. 1036–1036E.

[31] Xiao-Yu Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," IEEE Transactions on Information Theory, vol. 51, issue 1, pp. 386-398, Jan. 2005.

[32] Xiao.-Yu. Hu and E. Eleftheriou, "Binary Representation of Cycle Tanner-Graph GF(2^b) Codes," The Proc. IEEE Intern. Conf. on Commun., Paris, France, pp. 528-532, June 2004.

[33] E. Jones, M. Valles, M. Smith, and J. Villasenor, "Approximate Min* constraint node updating for LDPC code decoding," IEEE MILCOM conference, Oct. 2003.

[34] M. Cocco, J. Dielissen, M. Heijligers, A. Hekstra, J. Huisken, "A scalable architecture for LDPC decoding," Automation and Test in Europe Conference and Exhibition, vol. 3, pp. 88-93, Feb. 2004.

[35] Z. Wang, Y. Tan, and Y. Wang, "Low Hardware Complexity Parallel Turbo Decoder Architecture", in proc. of IEEE ISCAS'03, pp: II-53-56. May 2003.

[36] Z. Wang and Q. Jia, "Low complexity, high speed decoder architecture for quasi-cyclic LDPC codes," in proc. of IEEE ISCAS'05, Japan, May, 2005.

[37] Z. Wang, H. Suzuki and K. Parhi, "VLSI Implementation Issues of Turbo Decoder Design for Wireless Applications," SiPS 1999, pp. 503-512, Oct. 1999.

[38] Z. Wang, Y Chen and K Parhi, **"**Area-efficient decoding of quasi-cyclic low density parity check codes", *ICASSP 2004***,** vol. 5, pp.49-52, May 2004.

[39] Z. Wang and Q. Jia, "Low complexity, high speed decoder architecture for quasi-cyclic LDPC codes," to apprear in 2005 *IEEE International Symposium on Circuits and Systems,* Japan, May, 2005.

[40] Z.Wang and K. K. Parhi, "Decoding metrics and their applications in VLSI turbo decoders," in *Proc. ICASSP*, 2000, pp. 3370– 3373.

[41] Z. Wang, Y. Zhang, and K. K. Parhi, "Study of early stopping criteria for Turbo decoding and their applications in WCDMA systems," in Proc of ICASSP'06, pp. III-1016-1019, May 2006.

[42] Z. Cui, L. Chen, and Z. Wang, "An efficient early stopping scheme for LDPC decoding," in proc 13th NASA Symposium on VLSI design.

[43] T. Zhang and K. K. Parhi, "An FPGA implementation of (3,6)-regular low-density parity-check code decoder," EURASIP Journal on Applied Signal Processing, special issue on Rapid Prototyping of DSP Systems vol. 2003, no. 6, pp. 530-542, May, 2003.

[44] T. Zhang and K. K. Parhi, "A 54 Mbps (3,6)-regular FPGA LDPC decoder," in *Proc. IEEE SiPS'2002*, pp 127-132, 2002.

[45] T. Zhang, Z. Wang, and K. K. Parhi, "On finite precision implementation of low density parity check codes decoder," ISCAS 2001, vol.4, pp. 202-205, May 2001.

[46] T. Zhang and K. K. Parhi, "High-performance, low-complexity decoding of generalized low-density parity-check codes," IEEE GLOBECOM '01, vol. 1, pp. 181-185, Nov. 2001.

[47] D. E. Hocevar, "LDPC code construction with flexible hardware implementation", IEEE ICC '03, vol. 4, pp. 2708 – 2712.

[48] Y. Chen and D. E. Hocevar, "A FPGA and ASIC implementation of rate-1/2, 8088-b irregular low density parity check decoder," IEEE GLOBECOM '03, vol. 1, pp. 113-117, Dec. 2003.

[49] D. E. Hocevar**,** "A reduced complexity decoder architecture via layered decoding of LDPC codes," IEEE Workshop on Signal Processing Systems, pp. 107 - 112 , 2004.

[50] M. M. Mansour and N. R. Shanbhag, "Turbo decoder architectures for low-density parity-check codes," IEEE Global Telecommunications Conference, vol. 2, pp.1383-1388, Nov., 2002.

[51] M. M. Mansour, N. R. Shanbhag, "High-throughput LDPC decoders," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 11, pp. 976-996, Dec. 2003.

[52] 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 Engineers in Israel, pp. 223-226, Sept., 2004.

[53] M. Karkooti and J. R. Cavallaro, "Semi-parallel reconfigurable architectures for real-time LDPC decoding," *ITCC'2004*, vol. 1, pp. 579-585, Apr. 2004.

[54] P. Radosavljevic, A. de Baynast, M. Karkooti, and J. R. Cavallaro, , "Multi-Rate High-Throughput LDPC Decoder: Tradeoff Analysis Between Decoding Throughput and Area," IEEE PIMRC'06, Sept. 2006.

[55] Y. Li, M. Elassal, M. Bayoumi, "Power efficient architecture for (3,6)-regular low-density parity-check code decoder," in proc. ISCAS '04, vol. 4, pp. 81-84, May 2004.

[56] F. Kienle, T. Brack, and N. Wehn, "A synthesizable IP core for DVB-S2 LDPC code decoding," in proc. Design, Automation and Test in Europe, 2005, vol. 3, pp. 100 – 105, 2005.

[57] J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity check code decoder," IEEE Journal of Solid-State Circuits, vol. 37, pp. 404-412, March 2002.

[58] Darabiha, A. C. Carusone, F. R. Kschischang, "Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity," ISCAS 2005, vol. 5, pp. 5194-5197, May 2005.

[59] Se-Hyeon Kang and In-Cheol Park, "Loosely coupled memory-based decoding architecture for low density parity check codes," IEEE Trans. on Circuits and Systems I, vol. 53, pp. 1045 – 1056, May 2006.

[60] J. K. –S. Lee, J. Thorpe, "Memory-efficient decoding of LDPC codes," ISIT'05, pp. 456-463, Sept. 2005.

[61] Lin, K. Lin, H. Chang; and C. Lee, "A 3.33Gb/s (1200,720) low-density parity check code decoder," in Proc. of ESSCIRC'05, pp. 211-214, Sept. 2005.

[62] J. Sha, M. Gao, Z. Zhang, L. Li, and Z. Wang, "Efficient Decoder Implementation for QC-LDPC Codes," ICCCAS'06, vol. 4, pp. 2498-2502, June 2006.

[63] K. K. Gunnam, G. S. Choi, and M. B. Yeary, "A Parallel VLSI Architecture for Layered Decoding for Array LDPC Codes," VLSID'07, pp. 738-73**,** Jan. 2007.

[64] Guo, L. Hanzo, "Reliability ratio based weighted bit-flipping decoding for low-density parity-check codes," Electronics Letters, vol 40, pp. 1356-1358, Oct. 2004.

[65] M. Jiang, C. Zhao, Z. Shi, Yu Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," IEEE Communications Letters, vol. 9, pp. 814-816, Sep. 2005.

[66] H. Sankar and K. R. Narayanan, "Memory-efficient sum-product decoding of LDPC codes," IEEE Transactions on Communications, vol. 52, issue 8, pp. 1225-1230, Aug. 2004.

[67] Y. Dai, Z. Yan, and N. Chen, "High-Throughput Turbo-Sum-Product Decoding of QC LDPC Codes," in Proc. Confe. on Information Sciences and Systems, pp. 839-844, March, 2006.

[68] D. –U. Lee, W. Luk, C. Wang, C. Jones, "A flexible hardware encoder for low-density parity-check codes", *IEEE Symp. on FCCM'04*, pp. 101-111.

[69] T. Bhatt., V. Sundaramurthy, V. Stolpman, and D. McCain, "Pipelined Block-Serial Decoder Architecture for Structured LDPC Codes," IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 4, pp. IV-225 - IV-228, 2006.

[70] A. Matache, S. Dolinar, and F. Pollara, "Stopping rules for turbo decoders," Tech. Rep., Jet Propulsion Laboratory, Pasadena, California, Aug. 2000.

[71] J. Li, X. H. You and J. Li, "Early stopping for LDPC secoding: convergence of mean magnitude (CMM)," IEEE Comm. Letters, vol. 10, no. 9, Sept. 2006.

[72] H. Song and J.R. Cruz, "Reduced-Complexity Decoding of Q-ary LDPC Codes for Magnetic Recording," IEEE Trans.Magn., vol. 39, pp. 1081-1087, Mar. 2003.