Implementation of Iterative Decoding Algorithms on Digital VLSI Platforms

by

Farhad Zarkeshvari, B.Eng.

A thesis submitted to the

Faculty of Graduate Studies and Research

In partial fulfillment of the requirements for the degree of

Master of Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

Ottawa, Ontario

October, 2002

© Copyright

Farhad Zarkeshvari, 2002

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-79773-2

Canada

The undersigned hereby recommend to the faculty of Graduate Studies and Research acceptance of the thesis, "Implementation of Iterative Decoding Algorithms on Digital VLSI Platform", submitted by Farhad Zarkeshvari, in partial fulfillment of the requirements for the degree of Master of Engineering.

Chair, Department of Systems and
Computer Engineering

Professor Amir H. Banihashemi,
Thesis Supervisor

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
Ottawa, Ontario
October, 2002

Abstract

The complexity-performance trade-off of different iterative algorithms for decoding low-density parity-check (LDPC) codes is investigated and it is shown that min-sum algorithm is a good choice for digital implementation. The effects of clipping and the number of quantization bits on the performance of min-sum algorithm at short and intermediate block lengths are studied. It is shown that min-sum is robust against quantization effects, and in many cases, only four quantization bits suffices to obtain close to unclipped min-sum performance.

We also propose two modifications to min-sum algorithm that improve the performance by a few tenths of a dB with just a small increase in the complexity of decoding. These modifications can close the performance gap between min-sum and sum-product and even outperform the sum-product in some cases. The VLSI implementation of the decoder is investigated, two architectures are discussed, and the main blocks of a parallel decoder are designed.

Acknowledgements

I would like to appreciate all who helped me in this research specially my friends and colleagues Pirouz Zarrinkhat, Reza Yazdani and Arash Shokrani for their support and encouragements. I also thanks thesis supervisor Dr. Amir H. Banihashemi for his advices and comments. In addition, I thankfully acknowledge the financial support provided by Zarlink Semiconductor Corp. (formerly, Mitel Semiconductor Corp.) and Carleton University.

I would also like to sincerely thank professor John Knight for his valuable advice in the electronic part of this research.

Finally, I would like to thank my family, specially my wife, for their support and understanding and their sacrifices through the years of this research.

Table of Contents

Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Figures	vii
List of Symbols	xii
Introduction and motivation	1
LDPC Codes.....	3
1-1 Forward Error-Correcting Concept.....	4
1-2 Linear Block Codes.....	5
1-3 LDPC Codes and their Graph Presentation	8
1-4 LDPC Codes Characteristics.....	9
1-4-1 Comparison of turbo-codes and LDPC codes	10
1-4-2 LDPC codes and their implementation.....	10
Iterative Decoding Algorithms.....	12
2-1 Optimal and MAP decoding algorithms	13
2-2 Iterative decoding algorithms	14
2-2-1 Conventional belief propagation (sum-product) algorithm	2
2-2-2 Other optimal algorithms.....	5
2-2-3 Log-domain algorithms	10
2-2-3 Other iterative algorithms.....	14
2-3 Comparison of the algorithms.....	22
2-4 Implementation issues of iterative decoding algorithms	24
Quantization Effects on Min-sum Algorithm.....	30
3-1 Overview.....	31
3-2 Quantization Scheme	32
3-3 Clipping Threshold Effect.....	32
3-4 Quantization Effects.....	35
3-5 Complexity of Quantized Versions of Min-sum algorithm	39
3-6 Density Evolution	40
3-7 Density evolution for min-sum algorithm.....	41
Modifications to Min-sum Algorithm	47
4-1 Comparison of Min-sum and Log-ratio SP.....	48
4-2 Simplified Correction Function	49
4-3 Min-sum Modification with Conditional Correction Step.....	52
4-4 Min-sum Modification with Unconditional Correction Step.....	61
4-5 Min-sum Modifications and Their Complexity	65
Digital Implementation of Min-sum Algorithm.....	66
5-1 Digital Implementation of Iterative Decoding Algorithms	67
5-2 Main Issues in Digital Implementation	67
5-3 Different Design Approaches	68
5-3-1 Serial Approach: Optimization for Area	69
5-3-2 Parallel Decoder Architecture	71

5-3-3 Comparison with Turbo Code Implementations	72
5-4 Parallel Decoder Design and Optimization.....	73
5-4-1 Bit-node Design.....	74
5-4-2 Check-node Design.....	77
5-4-3 I/O Design.....	78
5-4-4 Overview of Bottom-up Design	78
5-4-5 Design characteristics and comparison.....	79
5-5 Decoder Interconnections	80
5-6 Special Consideration for High Degree Check-nodes	80
Conclusion.....	86
Future research	87
References	88
Appendix 1. Number of iterations Histogram.....	94
Appendix 2. Clipping and quantization results for other codes	96
Appendix 3. Message error rate	106
Appendix 4. Conditional correction step results for other codes	108
Appendix 5. Conditional correction step results for 5 and 6 quantization bits	112
Appendix 6. Unconditional correction step results for 5 and 6 quantization bits	116
Appendix 7. Proof of associativity for applying correction factor.....	119
Appendix 8. Average number of iterations for other codes	121

List of Figures

Figure 1. Parity check matrix for a $(10,6)<3,5>$ regular code.....	9
Figure 2. A sample Tanner graph for $(10, 6) <3, 5>$ regular code	9
Figure 3. Extrinsic messages pass to check-node for calculating outgoing messages	15
Figure 4. Messages pass to bit-node for hard decision	2
Figure 5. Messages pass to bit-node for calculating the outgoing message.....	2
Figure 6. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(1268,456)$ irregular code at $E_b/N_0=1$ dB.	34
Figure 7. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(1268,456)$ irregular code at $E_b/N_0=1.5$ dB.	34
Figure 8. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(1268,456)$ irregular code at $E_b/N_0=2$ dB.	34
Figure 9. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(1268,456)$ irregular code at $E_b/N_0=2.5$ dB.	34
Figure 10. BER curves for the $(1268,456)$ code, decoded with unclipped min-sum and unquantized min-sum with optimum clipping $cth=2$	35
Figure 11. The effect of clipping threshold cth on performance of quantized min-sum for $(1268,456)$ code at $E_b/N_0=1$ dB.	36
Figure 12. The effect of clipping threshold cth on performance of quantized min-sum for $(1268,456)$ code at $E_b/N_0=1.5$ dB.	36
Figure 13. The effect of clipping threshold cth on performance of quantized min-sum for $(1268,456)$ code at $E_b/N_0=2$ dB.	37
Figure 14. The effect of clipping threshold cth on performance of quantized min-sum for $(1268,456)$ code at $E_b/N_0=2.5$ dB.	37
Figure 15. BER curves for the $(1268,456)$ LDPC code decoded by unclipped min-sum and quantized min-sum for different quantization bits ($cth=2$).	39
Figure 16. Density evolution and simulation results for $(1200,900)<3,4>$ regular code with minimum girth 6 after 2 iteration.	45
Figure 17. Density evolution and simulation results for $(1200,900)<3,4>$ regular code with minimum girth 6 after 3 iteration.	45
Figure 18. Density evolution and simulation results for $(1200,900)<3,4>$ regular code with minimum girth 6 after 4 and 5 iteration	46
Figure 19. The difference between check-node degree three outputs in log-ratio SP and min-sum; x and y are extrinsic inputs and z is the difference between the outputs. .	49
Figure 20. The simplified correction function $f(d)$ for min-sum algorithm.	51
Figure 21. BER curves for $(1268,456)$ code, decoded by unclipped min-sum, modified unclipped min-sum with exact simplified correction factor and sum-product.....	51
Figure 22. An example of step-wise approximation of correction function which leads to conditional correction step; d_{step} and y_{step} are shown.	53
Figure 23. BER results for $(1268,456)$ code, decoded by unclipped min-sum with conditional correction step for different y_{step} and d_{step} at $E_b/N_0=1.5$ dB.	55
Figure 24. BER curves for $(1268,456)$ code, decoded by unclipped min-sum, modified unclipped min-sum with conditional correction step and sum-product.	56

Figure 25. BER results for (1268,456) code, decoded by unquantized min-sum with clipping and with conditional correction step, with different clipping threshold and $y_{\text{step}}=d_{\text{step}}=0.5$ at $E_b/N_0=1.5$ dB.....	56
Figure 26. BER results for (1268,456) code, decoded by unquantized min-sum with clipping and with conditional correction step for different y_{step} and d_{step} and optimum clipping $c_{\text{th}}=2.5$ at $E_b/N_0=1.5$ dB.....	57
Figure 27. BER results for (1268,456) code, decoded by unquantized min-sum with clipping and with conditional correction step, with different clipping threshold and $y_{\text{step}}=0.3$ and $d_{\text{step}}=0.2$ at $E_b/N_0=1.5$ dB.....	59
Figure 28. BER results for (1268,456) code, decoded by unquantized min-sum with clipping and with conditional correction step for different y_{step} and d_{step} and clipping $c_{\text{th}}=2$ at $E_b/N_0=1.5$ dB.....	59
Figure 29. BER curves for (1268,456) code, decoded by 4-bit modified min-sum with conditional correction step with different $d=d_{\text{step}}$ and $y=y_{\text{step}}$ ($c_{\text{th}}=2$).	60
Figure 30. BER curves for (1268,456) code, decoded by 4-bit min-sum, unclipped min-sum, unclipped min-sum with conditional correction $d_{\text{step}}=y_{\text{step}}=0.5$, 4-bit min-sum with conditional correction $m=n=1$, and sum-product.....	61
Figure 31. BER results for (1268,456) code, decoded by unclipped min-sum with conditional correction step at $E_b/N_0=1.5$ dB.	62
Figure 32. BER curves for (1268,456) code, decoded by unclipped min-sum with unconditional correction step.	63
Figure 33. BER curves for (1268,456) code decoded by sum-product, 4-bit quantized min-sum, and 4-bit quantized min-sum with the conditional and unconditional correction steps.....	64
Figure 34. BER curves for codes (273,191)(dashed lines) and (8000,4000)(solid lines), decoded by sum-product, 4-bit quantized min-sum, and 4-bit quantized min-sum with the conditional correction step.	64
Figure 35. Block diagram for serial decoder with hardware sharing architecture	70
Figure 36. Block diagram for parallel decoder architecture.....	72
Figure 37. Data path for fully parallel decoder	75
Figure 38. Synthesized circuit for bit-node degree 2	82
Figure 39. Synthesized circuit for bit-node degree 3	83
Figure 40. Synthesized circuit for check-node degree 4	84
Figure 41. Synthesized circuit for check-node degree 5	85
Figure 42. Histogram of the number of iterations for 4-bit min-sum at $E_b/N_0=1$ dB for (1268,456) irregular code.....	94
Figure 43. Histogram of the number of iterations for 4-bit min-sum at $E_b/N_0=1.5$ dB for (1268,456) irregular code.....	94
Figure 44. Histogram of the number of iterations for 4-bit min-sum at $E_b/N_0=2$ dB for (1268,456) irregular code.....	95
Figure 45. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (8000,4000)<3,6> regular code at $E_b/N_0=1.7$ dB.	96
Figure 46. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (8000,4000)<3,6> regular code at $E_b/N_0=1.8$ dB.	96
Figure 47. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (8000,4000)<3,6> regular code at $E_b/N_0=1.9$ dB.	96

Figure 48. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (8000,4000)<3,6> regular code at $E_b/N_o=2$ dB	96
Figure 49. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (273,191)<3,10> regular code at $E_b/N_o=2$ dB.	97
Figure 50. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (273,191)<3,10> regular code at $E_b/N_o=2.5$ dB.....	97
Figure 51. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (273,191)<3,10> regular code at $E_b/N_o=3$ dB.	97
Figure 52. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (273,191)<3,10> regular code at $E_b/N_o=3.5$ dB.....	97
Figure 53. The effect of clipping threshold c_{th} on error performance of unquantized min-sum with clipping for (273,191)<3,10> regular code at $E_b/N_o=4$ dB.....	97
Figure 54. BER curves for the (8000,4000) code, decoded with unclipped min-sum and unquantized min-sum with optimum clipping $c_{th} = 1.25$	98
Figure 55. BER curves for the (273,191) code, decoded with unclipped min-sum and unquantized min-sum with optimum clipping for each E_b/N_o . ($c_{th}=1.5, 1.5, 2.5, 3.25, 3, 2.5$ are used for $E_b/N_o=2, 2.5, 3, 3.5, 4, 4.5$, respectively.	98
Figure 56. The effect of clipping threshold c_{th} on performance of quantized min-sum for (8000,4000) code at $E_b/N_o=1.7$ dB.	99
Figure 57. The effect of clipping threshold c_{th} on performance of quantized min-sum for (8000,4000) code at $E_b/N_o=1.8$ dB.	99
Figure 58. The effect of clipping threshold c_{th} on performance of quantized min-sum for (8000,4000) code at $E_b/N_o=1.9$ dB.	100
Figure 59. The effect of clipping threshold c_{th} on performance of quantized min-sum for (8000,4000) code at $E_b/N_o=2$ dB.	100
Figure 60. The effect of clipping threshold c_{th} on performance of quantized min-sum for (273,191) code at $E_b/N_o=2$ dB.	101
Figure 61. The effect of clipping threshold c_{th} on performance of quantized min-sum for (273,191) code at $E_b/N_o=2.5$ dB.	101
Figure 62. The effect of clipping threshold c_{th} on performance of quantized min-sum for (273,191) code at $E_b/N_o=3$ dB.	102
Figure 63. The effect of clipping threshold c_{th} on performance of quantized min-sum for (273,191) code at $E_b/N_o=3.5$ dB.	102
Figure 64. The effect of clipping threshold c_{th} on performance of quantized min-sum for (273,191) code at $E_b/N_o=4$ dB.	103
Figure 65. BER curves for the (8000,4000) LDPC code decoded by unclipped min-sum, unquantized min-sum with clipping and quantized min-sum for different quantization bits ($c_{th}=1.25$).	103
Figure 66. BER curves for the (273,191) LDPC code decoded by unclipped min-sum, unquantized min-sum with clipping and quantized min-sum for different quantization bits ($c_{th}=1.5$).	104
Figure 67. BER curves for (8000,4000) code, decoded by unclipped min-sum, modified unclipped min-sum with exact simplified correction factor and sum-product.....	104
Figure 68. BER curves for (273,191) code, decoded by unclipped min-sum, modified unclipped min-sum with exact simplified correction factor and sum-product.....	105

Figure 69. MER curves for (8000,4000) code decoded by unclipped min-sum, unquantized min-sum with clipping, unclipped min-sum with conditional and unconditional correction step, 4-bit min-sum, 4-bit min-sum with conditional and unconditional correction step and sum-product.	106
Figure 70. MER curves for (273,191) code decoded by unclipped min-sum, unquantized min-sum with clipping, 4-bit min-sum, 4-bit min-sum with conditional and unconditional correction step and sum-product.	106
Figure 71. MER curves for (1286,456) code decoded by unclipped min-sum, unquantized min-sum with clipping, 4-bit min-sum, 4-bit min-sum with conditional and unconditional correction step and sum-product.	107
Figure 72. BER for (8000,4000) regular code at $E_b/N_0=1.7\text{dB}$ decoded by unclipped min-sum with conditional correction step for different $d_{\text{step}}=y_{\text{step}}$	108
Figure 73. BER for (273,191) regular code at $E_b/N_0=3.5\text{dB}$ decoded by unclipped min-sum with conditional correction step for different $d_{\text{step}}=y_{\text{step}}$	108
Figure 74. BER curves for (273,191) code, decoded by unclipped min-sum, modified unclipped min-sum with conditional correction step $d_{\text{step}}=0.2$ and $y_{\text{step}}=0.2$ and sum-product.....	109
Figure 75. BER curves for (8000,4000) code, decoded by unclipped min-sum, modified unclipped min-sum with conditional correction step $d_{\text{step}}=0.3$ and $y_{\text{step}}=0.3$ and sum-product.....	109
Figure 76. BER curves for (8000,4000) code, decoded by 4-bit modified min-sum without correction and with conditional correction step with different $d=d_{\text{step}}$ and $y=y_{\text{step}}$. The sum-product curve is also given for comparison.	110
Figure 77. BER curves for (273,191) code, decoded by 4-bit modified min-sum without correction and with conditional correction step with different $d=d_{\text{step}}$ and $y=y_{\text{step}}$. The sum-product curve is also given for comparison.	110
Figure 78. BER curves for (8000,4000) code, decoded by unclipped min-sum and 4-bit min-sum with and without conditional correction step . ($d_{\text{step}}=0.3$ and $y_{\text{step}}=0.3$ for unclipped and $d_{\text{step}}=y_{\text{step}}=2$ for 4-bit version are chosen) The sum-product curve is also given for comparison.	111
Figure 79. BER curves for (273,191) code, decoded by unclipped min-sum, 4-bit min-sum, modified unclipped min-sum and modified 4-bit min-sum. The sum-product result is also given for comparison.....	111
Figure 80. BER curves for (8000,4000) code, decoded by 5-bit modified min-sum without correction and with conditional correction step with different $d=d_{\text{step}}$ and $y=y_{\text{step}}$. The sum-product curve is also given for comparison.	112
Figure 81. BER curves for (8000,4000) code, decoded by 6-bit modified min-sum without correction and with conditional correction step with different $d=d_{\text{step}}$ and $y=y_{\text{step}}$. The sum-product curve is also given for comparison.	113
Figure 82. BER curves for (273,191) code, decoded by unclipped min-sum, modified unclipped min-sum, 5-bit min-sum and 5-bit min-sum with optimum conditional correction step $d_{\text{step}}=3, 3, 4, 3, 2, 3$ and $y_{\text{step}}=1, 1, 1, 2, 2, 1, 2$, for $E_b/N_0=2, 2.5, 3, 3.5, 4, 4.5$, respectively. The sum-product result is also given for comparison.....	113
Figure 83. BER curves for (273,191) code, decoded by unclipped min-sum, modified unclipped min-sum, 6-bit min-sum and 6-bit min-sum with optimum conditional	

correction step $d_{\text{step}}=6, 5, 8, 4, 4, 4$ and $y_{\text{step}}=1, 1, 1, 2, 3, 4, 2$, for $E_b/N_0=2, 2.5, 3, 3.5, 4, 4.5$, respectively. The sum-product result is also given for comparison.	114
Figure 84. BER curves for (1268,456) code, decoded by unclipped min-sum, modified unclipped min-sum, 5-bit min-sum and 5-bit min-sum with optimum conditional correction step for each E_b/N_0 ($d_{\text{step}}=3, 3, 4, 4$ and $y_{\text{step}}=3, 2, 2, 2$ for $E_b/N_0=1, 1.5, 2, 2.5$, respectively). The sum-product is also given for comparison.	114
Figure 85. BER curves for (1268,456) code, decoded by unclipped min-sum, modified unclipped min-sum, 6-bit min-sum and 6-bit min-sum with optimum conditional correction step for each E_b/N_0 ($d_{\text{step}}=8, 8, 8, 8$ and $y_{\text{step}}=3, 5, 3, 2$ for $E_b/N_0=1, 1.5, 2, 2.5$, respectively). The sum-product is also given for comparison.	115
Figure 86. BER curves for (273,191) code, decoded by 5-bit min-sum and 5-bit min-sum with different unconditional correction step. The sum-product curve is also given for comparison.	116
Figure 87. BER curves for (273,191) code, decoded by 6-bit min-sum and 6-bit min-sum with different unconditional correction step. The sum-product curve is also given for comparison.	116
Figure 88. BER curves for (8000,4000) code, decoded by 5-bit min-sum and 5-bit min-sum with different sizes of unconditional correction step. The sum-product curve is also given for comparison.	117
Figure 89. BER curves for (8000,4000) code, decoded by 6-bit min-sum and 6-bit min-sum with different sizes of unconditional correction step. The sum-product curve is also given for comparison.	117
Figure 90. BER curves for (1268,456) code, decoded by and 5-bit min-sum with different sizes of unconditional correction step. The sum-product curve is also given for comparison.	118
Figure 91. BER curves for (1268,456) code, decoded by 6-bit min-sum with different sizes of unconditional correction step. The sum-product curve is also given for comparison.	118

List of Symbols

R	rate of the code
n	length of the code
k	number of information bits
$\bar{v}$	codeword vector
g_i	i_{th} independent codeword
M	number of codewords of a code
C	binary block code
q	alphabet size of a code
G	generator matrix
H	parity check matrix
$d(., .)$	hamming distance
d	minimum hamming distance
$P(s_t s_r)$	conditional probability of transmitted signal s_t knowing received signal s_r
s_t	transmitted signal
s_r	received signal
$P(s_t)$	probability that signal s_t is transmitted
$p(s_r)$	probability density function of received signal s_r
$p(s_r s_t)$	conditional probability density function of received signal s_r knowing transmitted signal s_t
$D(., .)$	Euclidean distance
N_o	one sided noise spectral density
$M_{c \rightarrow s}(x)$	check-node to bit-node message corresponding to symbol x

$M_{s \rightarrow c}(x)$	bit-node to check-node message corresponding to symbol x
$W(i)$	initial weight for symbol i_{th}
σ	noise variance
r	received value
$p(r/0)$	conditional probability density function of receiving value r given zero is sent
$p(r/1)$	conditional probability density function of receiving value r given one is sent
X_e	set of incoming extrinsic messages to a node
$N(i)$	set of neighboring nodes of bit-node i
$N(i) \setminus j$	set of neighboring nodes of bit-node i but check-node j
$M(i)$	set of neighboring nodes of check-node i
$M(i) \setminus j$	set of neighboring nodes of check-node i but bit-node j
$\hat{s}$	estimate of received vector
$\hat{s}_k$	estimate of k_{th} bit of received vector
f_n^x	local weight for symbol x when expressed as probability
$m_{s \rightarrow c}$	message sent from bit-node s to check-node c
$m_{c \rightarrow s}$	message sent from check-node c to bit-node s
α_i	required coefficient to convert weight i to probability
$\mathcal{K}$	the operation defined in check-node of ratio SP algorithm
f_n	initial weight in difference SP algorithm
$d_{s \rightarrow c}$	difference of weights as the message from bit-node to check-node

$d_{c \rightarrow s}$	difference of weights as the message from check-node to bit-node
r_i	received value for the i^{th} symbol
$r_{c \rightarrow s}$	ratio weights as message from check-node to bit-node
lw_i	local weight for min-sum
d_c	check-node degree
d_s	bit-node degree
S	number of bit-nodes
C	number of check-nodes
U_s	physical bit-nodes
U_c	physical check-nodes
t_s	physical bit-node delay
t_c	physical check-node delay
H_s	physical bit-node hardware complexity
H_c	physical check-node hardware complexity
c_{th}	clipping threshold
q	number of quantization bits
it	iteration number
S_l	l^{th} quantization symbol
$b_l \text{ and } b_{l-1}$	borders of the l^{th} quantization symbol
l_q	quantization interval for specified uniform quantization scheme with q bits
d_{step}	difference step in modification with conditional correction step
y_{step}	correction step in modification with conditional correction step
$m_{e,\min}^{(ex)}$	extrinsic incoming message with the smallest absolute value

N_{it}	number of decoder iterations
MBW	message bandwidth
E	number of edges in a graph
T	throughput
E_b	bit energy
M_{max}	the maximum number that can be processed by decoder

Introduction and motivation

Introduction of the concept of channel capacity and the proof of existence of coding schemes capable of achieving this capacity by Shannon in the late forties [35], was a landmark in coding history. From then on, the problem of reliable communication in the presence of noise has been changed to finding practical coding schemes to get as close as possible to this capacity. A part of the answer was again given by Shannon: never discard information prematurely that may be useful in making a decision until after all decisions related to that information have been completed [38]. This is an important aspect in all soft-decision capacity-approaching coding schemes like turbo-codes and low-density parity-check codes.

Gallager introduced low-density parity-check (LDPC) codes in 1960 [18], but they were soon forgotten due to limited computational resources available at the time, which made the corresponding decoding algorithms look impractical. From coding theory, it is known that by increasing the codeword length or the encoder memory, greater coding gain can be achieved. At the same time the complexity of maximum likelihood decoding algorithms (MLDA) increases exponentially with the codeword length or the encoder memory and the algorithms become difficult to implement. The increased error correction capability of long codes requires a very high computational effort at the decoder. This has led to search for new coding schemes which could replace the MLDA with simpler decoding strategies. A solution can be using iterative decoding algorithms. Although iterative decoding algorithms were introduced by Gallager [18] in the sixties, it was not until the invention of turbo codes [7] in the nineties that researchers realized the phenomenal strength of iterative coding schemes, and the attractive performance/complexity tradeoff

that they offer. Since then, there has been a great amount of research devoted to LDPC codes and the performance and the complexity of associated iterative decoding algorithms [1], [2], [3], [4]. In the coding community, it is now widely accepted that in many applications, LDPC codes are serious competitors for turbo-codes and more research is underway to analyze, design and efficiently implement LDPC encoders and decoders.

Parallel to this trend in the area of information theory and coding theory, Moore [31] made his famous observation in 1965 (Moore's law). He observed an exponential growth in the number of transistors per integrated circuit and predicted that this trend would continue. The number of transistors on a chip, and so the computational capabilities of VLSI circuits (which can be used for decoding) have doubled every couple of years in the last few decades and this trend still goes on.

On the other hand, the nature of iterative decoding algorithms when they are applied to LDPC codes has an inherent parallelism which makes them suitable for VLSI implementation.

These facts motivate us to investigate the practical implementation of iterative decoding algorithms for LDPC codes on VLSI platform. We started by searching among the existing iterative decoding algorithms to find the one that most applies to digital implementation. We developed simulation programs to explore the behavior of decoding algorithms on various codes. In the next step, the quantization and clipping effects on the selected algorithms and codes were investigated. We then modify the algorithm to increase its coding gain. Finally we discuss the implementation issues and design fundamental blocks of a decoder.

Chapter 1

LDPC Codes

1-1 Forward Error-Correcting Concept

At present, error-correcting codes are an integral part of all modern digital communication systems. They are used widely for digital data communication or in storage systems.

Shannon's noisy channel coding theorem guarantees that there exist error-control codes such that information can be transmitted across the channel at rates less than channel capacity with arbitrary low bit error rate, but this theorem doesn't say how to build such codes. Researchers in the coding theory have been trying to find the best answer to this question for the last few decades. Many error-control coding schemes have been introduced, all involving the addition of redundancy to the transmitted data to provide the means for detecting and correcting errors that inevitably occur in any real communication process. Through adding redundancy to transmitted data, the bit-error rate can be reduced considerably without an increase in the transmission power (in case of power-limited channels) or bandwidth (in case of bandwidth-limited channels). Alternatively, keeping the bit-error rate constant, a reduced transmission power or less bandwidth is required.

There are many error-correcting codes, such as Reed-Solomon codes, cyclic codes, convolutional codes, turbo-codes, etc. The focus of this research is however, on low-density parity-check codes (LDPC codes), which are a subset of block codes. These codes show very good complexity-performance tradeoffs, and have been able to beat all known codes at long block lengths.

In this chapter we first explain block codes and LDPC codes and then discuss the advantages of LDPC codes. Afterwards, we explain the reason we have chosen these codes to investigate their VLSI implementation.

1-2 Linear Block Codes

In block error-correcting codes, encoding and decoding process is done on a block of data. Assume that $A = \{a_1, a_2, \dots, a_q\}$ is a finite set of alphabets. Any non-empty subset C of A^n with M elements is called a block code of length n with q alphabets and M codewords. The rate R of such a code is defined as

$$R = \frac{k}{n}$$

where

$$k = \log_q M$$

If $A = \{0,1\}$, then the code is binary. In practice k binary digits of information are encoded into one of $M = 2^k$ codewords with length n to reduce the probability of bit error. This comes at the cost of more bandwidth requirement. For a system with orthogonal signaling bandwidth increases exponentially with k .

A binary block code C is linear if C is a k dimensional subspace of $GF(2)^n$. In this case C is called a *binary linear block code* (n,k) . In our research we focus on binary linear block codes.

For a linear code, there are k linearly independent codewords $\vec{g}_0, \vec{g}_1, \vec{g}_2, \dots, \vec{g}_{k-1}$ such that any other codeword v in C can be obtained by a linear combination of those k codewords.

$$\vec{v} = u_0 \vec{g}_0 + u_1 \vec{g}_1 + \dots + u_{k-1} \vec{g}_{k-1}$$

where

$$u_i \in [0,1]$$

A generator matrix of the code is built by any such set of k linearly independent codewords:

$$\mathbf{G} = \begin{bmatrix} \bar{g}_0 \\ \bar{g}_1 \\ \vdots \\ \bar{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,k-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,k-1} \\ \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,k-1} \end{bmatrix}$$

The corresponding codeword of an information block can be obtained by multiplying it by the generator matrix.

$$\bar{v} = \bar{u} \cdot \mathbf{G} = (u_0, u_1, \dots, u_{k-1}) \begin{bmatrix} \bar{g}_0 \\ \bar{g}_1 \\ \vdots \\ \bar{g}_{k-1} \end{bmatrix} = u_0 \bar{g}_0 + u_1 \bar{g}_1 + \cdots + u_{k-1} \bar{g}_{k-1}$$

Note that any k linearly independent codewords can make a generator matrix, so the generator matrix of a code is not unique. For any $k \times n$ generator matrix $\mathbf{G}$ with k linearly independent rows, there exists a $(n-k) \times n$ matrix $\mathbf{H}$ with $(n-k)$ linearly independent rows called parity-check matrix such that any vector in the row space of $\mathbf{G}$ is orthogonal to any row of $\mathbf{H}$ and any vector that is orthogonal to all the rows of $\mathbf{H}$ is in the row space of $\mathbf{G}$. Thus for any codeword v which was produced by $\mathbf{G}$ we can write:

$$\mathbf{H} \cdot \bar{v} = 0$$

The 2^{n-k} codewords generated by the rows of parity-check matrix $\mathbf{H}$, make an $(n, n-k)$ code which is called the dual of the code C and is shown by $C^\perp$. A code is called self dual if $C = C^\perp$.

The number of non-zero elements of a codeword is called the Hamming weight of that codeword. The Hamming distance $d(\bar{v}, \bar{w})$ is the number of bit positions by which codewords $\bar{v}$ and $\bar{w}$ of a particular code are different. The minimum distance of the code is defined as the minimum Hamming distance between any two non-equal codewords:

$$d_{\min} = \min\{d(\bar{v}, \bar{w}) : \bar{v}, \bar{w} \in C, \bar{v} \neq \bar{w}\}$$

It can be shown that minimum distance of a linear code is equal to minimum non-zero Hamming weight of its codewords.

A linear block code is called systematic if the codewords can be divided into two parts: message part with k bits and redundancy parity-check part of $(n - k)$ bits. Systematic code generator matrix can be shown by $G = [P \mid I]$, where I is identity matrix and P is the parity generator part. In practice, at the transmitter side of a communication system, the encoder uses generator matrix to map each k -bit data block into an n -bit codeword. If the code is systematic, $n-k$ parity bits would be appended to a block of data in this process. At the receiver side incoming signal is demodulated and passed through a decoder. The decoder uses a parity-check matrix H and a decoding algorithm to decode the incoming block. The following results may happen:

1. No error is present, which means that the codeword produced by the decoder matches the original (transmitted) codeword,
2. Decoder detects but cannot correct bit errors and reports uncorrectable error,
3. Decoder detects no error, though errors are present.

The overall performance of the system would be better than uncoded case for the desirable range of SNRs. The coding gain is defined as the amount of additional SNR or

E_b/N_o that would be required to provide the same bit or message error rate (BER or MER) performance for an uncoded signal.

1-3 LDPC Codes and their Graph Presentation

An LDPC code C , like any other linear block code, can be fully described by its parity-check matrix H , through the parity-check equations $\bar{c} \cdot H^T = 0$, $\bar{c} \in C$. For LDPC codes however, the parity-check matrix happens to be sparse i.e. only a small fraction of elements are nonzero.

Iterative decoding algorithms are naturally described using a graph of the code constructed based on matrix H . Such a graph is called a Tanner graph (TG) [37]. To describe TGs and the associated decoding algorithms, we assume that the codes are binary. A TG is a bipartite graph that contains two types of nodes, bit (symbol) nodes and check nodes. The bit-nodes correspond to the columns and the check-nodes correspond to the rows of the parity-check matrix H . A '1' located at position (i, j) of H corresponds to an edge between bit-node j and check-node i . An example of a parity-check matrix and its corresponding Tanner graph is shown in Figure 1 and 2. Each type of node is just connected to the nodes of other type. The degree of a node is defined as the number of the edges connected to that node. The code is called *regular* if all the nodes of one type have the same degree. The code is *irregular* if nodes of one type have different degrees.

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

Figure 1. Parity check matrix for a (10,6)<3,5> regular code

Figure 2. A sample Tanner graph for (10, 6) <3, 5> regular code

1-4 LDPC Codes Characteristics

LDPC codes were first discovered by Gallager [18] and then rediscovered by Spielman et al. [36] and Mackay et al. [26], [27]. They exhibit a threshold phenomenon in very large block lengths when they decoded by iterative decoding algorithms. As the block length tends to infinity, for noise level smaller than a threshold, the code can perform with an arbitrary small bit error probability. Gallager first observed this phenomenon for regular codes and then Luby et al. [24] generalized this idea to irregular codes and showed that irregular codes perform better than regular ones. Richardson et al. [33] gen-

eralized the idea and proved that the decoder performance on random graphs converges to its expected value as the length of the code increases. They also proved concentration theorems which show that in the limit large random graphs can be assumed to be effectively cycle free. This allows the calculation of precise thresholds and optimization the code parameters. It also helps to find very long LDPC codes with performance extremely close to the capacity.

1-4-1 Comparison of turbo-codes and LDPC codes

Recent excitement in the coding community first was ignited by excellent performance of turbo codes in early 1990's [7] and then by the rediscovery of LDPC codes [26], [27], [36]. LDPC codes have several distinct advantages over turbo codes. The belief propagation decoding for LDPC codes is fully parallelizable and can be accomplished at significantly higher speeds. Many low complexity decoding schemes have been designed that closely approximate belief propagation performance. Decoding process is verifiable in LDPC codes in the sense that decoding to a correct code word is a detectable event. They can outperform turbo codes at large block lengths and perform really close to channel capacity [12]. For binary input additive white Gaussian noise (AWGN) channel, the performance of a code with rate one half and length one million fall to within 0.0045dB of channel capacity, surpassing the best known turbo-code performance [12].

1-4-2 LDPC codes and their implementation

It is known that turbo codes and LDPC codes are the best known coding techniques for achieving data rates that are very close to capacity on binary symmetric and AWGN

channels. In pervious section we discussed the advantages of LDPC codes over turbo codes. Iterative decoding (will be explained in detail in the next chapter) offers a practical way for decoding these codes with a reasonable complexity and makes them interesting for implementation. They have recently received significant attention and various results have been reported [27], [21], [29]. The requirements of interleaving and serial recursion [12] increase the inherent latency of turbo decoding and limits its usage in latency-sensitive applications like voice communication. On the other hand, LDPC decoding exhibits less inherent latency limitations due to inherent parallelism and simpler processing requirements [23]. Using the iterative message passing algorithms, LDPC decoders require an order of magnitude less arithmetic computations than equivalent turbo decoders [42]. However, the implementations of decoders of these codes suffer from complex interconnection design process and large memory requirements. LDPC decoding has a large amount of inherent parallelism, which if exploited efficiently, could make LDPC codes a potentially attractive coding technique for many applications, including those that are latency-sensitive. Parallel implementations naturally become more attractive as VLSI integration density continues to increase and cost per transistor continues to decrease.

Chapter 2

Iterative Decoding Algorithms

2-1 Optimal and MAP decoding algorithms

The output of the demodulator, either a correlation demodulator or a matched filter, contains sufficient statistics of received signal waveform required for detection. A decoder has to make a decision on the transmitted signal based on observation of the received signal in each interval such that the probability of correct decision is maximized. If transmitted signal is shown by s_t and received signal by s_r , then the *a posteriori* probability of s_t given s_r is defined as

$$P(s_t | s_r) = P(\text{signal } s_t \text{ was transmitted} | \text{signal } s_r \text{ is received}) \quad (2-1.1)$$

If the decision rule is based on the computation of *a posteriori* probabilities and finding the transmitted signal that maximizes it, the decoding is called maximum *a posteriori* probability (MAP) decoding.

Using Bayes' rule, *a posteriori* probabilities may be expressed with *a priori* probabilities:

$$P(s_t | s_r) = \frac{p(s_r | s_t) \cdot P(s_t)}{p(s_r)} \quad (2-1.2)$$

So the computation of *a posteriori* probabilities requires knowledge of the *a priori* probabilities $P(s_t)$ and the conditional probability density function (pdf) $p(s_r | s_t)$ for all possible transmitted signals. If all transmitted signals (codewords in block codes case) are equally probable, then $P(s_t)$ in (2-1.2) is independent of the transmitted signal. Consequently, the decision rule based on the signal that maximizes $p(s_r | s_t)$ is equivalent to finding the signal that maximizes likelihood function $p(s_r | s_t)$. This decoding is called maximum likelihood (ML) decoding. For the binary AWGN channel with noise power spectral density N_0 and BPSK modulation we can write

$$p(s_r | s_t) = \frac{1}{(\pi N_o)^{n/2}} \exp\left(-\sum_{k=1}^n \frac{(s_{rk} - s_{tk})^2}{N_o}\right) \quad (2-1.3)$$

It can be shown that ML decoding is equivalent to finding the signal s_t that minimizes the Euclidean distance

$$D(s_r, s_t) = \sum_{k=1}^n (s_{rk} - s_{tk})^2 \quad (2-1.4)$$

In case of block codes, the n real values from demodulator corresponding to n bits of received block are passed to the decoder, which compares the received block with all possible transmitted codewords and chooses the closest one as the received codeword. This optimum decoding rule results in a minimum probability of codeword error.

The perfect implementation of optimal decoding is very complicated. So other decoding algorithms are used which are less complex and more practical to implement. These algorithms work by updating the bits or corresponding messages based on constraints iteratively. Soft iterative decoding algorithms apply local constraints to the real (analog) messages while hard iterative decoding algorithms apply these constraints to the bits. With cycle-free Tanner graphs, the resulting decoding algorithms are maximum-likelihood for sequence (max-sum, min-sum) or for bit (sum-product) [16]. Iterative decoding occurs when the Tanner graph has cycles, as is the case for all good codes [14], and the resulting algorithms are generally suboptimal, but significant complexity reduction is possible. The Tanner graph of good codes generally contains cycles [14].

2-2 Iterative decoding algorithms

Iterative decoding algorithms, also called “message-passing algorithms,” are performed by exchanging messages between check-nodes and bit-nodes through the edges of the Tanner graph, in both directions and iteratively. The messages could be “hard” and/or

“soft” information, and the operations performed in bit and check nodes to generate messages depend on the nature of the decoding algorithm and the message type(s). Soft decoding algorithms have much better performance and are more complex than hard decoding algorithms. In this research, we concentrate on soft iterative algorithms.

In its conventional form, a soft iterative decoding algorithm starts by creating initial messages at bit nodes from the observations at the output of the channel and the channel characteristics. It then passes these messages to check nodes through the edges of the graph. Iterations then start, by each iteration consisting of two steps: 1) check nodes processing the initial information and passing new messages to the bit nodes. These messages usually reflect the estimate of check nodes from the value and reliability of bit nodes. (Figure 3) 2) bit nodes processing the messages sent by check nodes and sending back updated information about their value and the associated reliability to check nodes. In each iteration, hard decision on the value of each bit is made at bit nodes and the algorithm stops if the hard-decision assignments for bits satisfy all the check equations, or a maximum number of iterations is reached (Figure 4 and 5).

Figure 3. Extrinsic messages pass to check-node for calculating outgoing messages

$$M_{s_k \rightarrow c_j}(1) = W(1) + \sum_{i \in N(k) \setminus j} M_{c_i \rightarrow s_k}(1)$$

where

$N(k) \setminus l$ is set of all of all the neighbors of bit-node k excluding the one coming from check-node l ,

Hard decision: First two new weights for each bit corresponding to “zero” and “one” are calculated as follow:

$$M_{s_k}(0) = W(0) + \sum_{i \in N(k)} M_{c_i \rightarrow s_k}(0)$$

$$M_{s_k}(1) = W(1) + \sum_{i \in N(k)} M_{c_i \rightarrow s_k}(1)$$

Then hard decision results $\hat{s}_k$, a new estimate for bit k , by comparison of these two messages such as:

$$\text{If } M_{s_k}(0) \geq M_{s_k}(1) \quad \hat{s}_k = 0$$

$$\text{Otherwise} \quad \hat{s}_k = 1$$

2-2-3-2 Min-sum algorithm

Here, another version of max-sum is explained which uses the difference of the log-likelihoods as messages. This simplified version is called Min-Sum algorithm, implying that algorithm is performed just by finding minimum and summation. Min-Sum performs exactly the same as max-sum under perfect arithmetic. The local weights in max-sum are log likelihoods:

$$W_i(x_m) = \log\{(1/\sigma(2\pi)^{1/2}).e^{-(r_i+s_m)/2\pi\sigma^2}\}$$

where x_m is a possible symbol, r_i is the received value, and s_m can be normalized to -1 and 1 for a BPSK modulated signal. Then the difference of log likelihoods for binary case can be written as:

$$\begin{aligned} lw_i &= w_i(0) - w_i(1) = \log\{(1/\sigma(2\pi)^{1/2}).e^{-(r_i-1)/2\pi\sigma^2}\} - \log\{(1/\sigma(2\pi)^{1/2}).e^{-(r_i+1)/2\pi\sigma^2}\} \\ &= \log\left\{\frac{(1/\sigma(2\pi)^{1/2}).e^{-(r_i+1)/2\pi\sigma^2}}{(1/\sigma(2\pi)^{1/2}).e^{-(r_i-1)/2\pi\sigma^2}}\right\} = -4r_i \end{aligned}$$

Therefore, min-sum local weights are log-likelihood ratio just like local weights in log-ratio SP algorithm. We will explain later that min-sum can be considered as an approximation of log-ratio SP and will elaborate the differences. In fact, we use this comparison to introduce two modifications to min-sum algorithm in chapter four.

The advantage of the min-sum and log-ratio SP algorithms is that their local weights are independent of channel noise and thus these algorithms can skip the tedious processes of noise estimation and weight computations. Without any loss of performance lw_i can be normalized to r_i . Min-Sum algorithm can be summarized as follows:

Initialization:

$$m_{s_i \rightarrow c_j} = lw_i = r_i$$

Check-node step:

$$m_{c_i \rightarrow s_j} = \prod_{k \in M(i) \setminus j} \text{sgn}(m_{s_k \rightarrow c_i}) \cdot \text{Min}_{k \in M(i) \setminus j} |m_{s_k \rightarrow c_i}|$$

where $\text{sgn}(x) = 1$ or -1 , depending on $x \geq 0$ or < 0 , respectively.

Bit-node step:

$$m_{s_i \rightarrow c_j} = r_i + \sum_{k \in N(i) \setminus j} m_{c_k \rightarrow s_i}$$

Hard decision:

$$w_i = r_i + \sum_{k \in N(i)} m_{c_k \rightarrow s_i}$$

Create $\hat{s} = [\hat{s}_1, \hat{s}_2, \dots, \hat{s}_n]$ such that $\hat{s}_i = 0$ if $w_i > 1$ and $\hat{s}_i = 1$ if $w_i \leq 1$.

The bit-node operation in log-ratio SP is the same as min-sum algorithm but the check-node operations are different. Here we show the relation of their check-nodes operations.

Consider a degree three check-node with extrinsic incoming messages m_1 and m_2 , the outgoing message in log-ratio SP algorithm is:

$$\begin{aligned} \log(e^{m_1} \mathbb{K} e^{m_2}) &= \log \frac{1 + e^{m_1 + m_2}}{e^{m_1} + e^{m_2}} = \text{sgn}(m_1) \text{sgn}(m_2) \log \frac{1 + e^{|m_1| + |m_2|}}{e^{|m_1|} + e^{|m_2|}} \\ &= \text{sgn}(m_1) \text{sgn}(m_2) \log \frac{1 + e^{2 \min(|m_1|, |m_2|) + d}}{e^{\min(|m_1|, |m_2|)} + e^{\min(|m_1|, |m_2|) + d}} \\ &= \text{sgn}(m_1) \text{sgn}(m_2) \left[\log(e^{\min(|m_1|, |m_2|)}) + \log \frac{(e^{-2 \min(|m_1|, |m_2|)} + e^d)}{(1 + e^d)} \right] \\ &= \text{sgn}(m_1) \text{sgn}(m_2) \left[\min(|m_1|, |m_2|) + \log \frac{(e^{-2 \min(|m_1|, |m_2|)} + e^d)}{(1 + e^d)} \right] \end{aligned}$$

where $d = \left| |m_1| - |m_2| \right|$.

It can be observed that the difference of check-node operations in log-ratio SP and min-sum is the following term:

$$\text{Correction factor} = \log \frac{(e^{-2 \min(|m_1|, |m_2|)} + e^d)}{(1 + e^d)}$$

and because of this term log-ratio SP performs better than min-sum. We called this term the correction factor because by adding this term to the min-sum check-node operation it can perform as well as the optimal algorithm (log-ratio SP). Min-sum can be assumed as

log-ratio SP with the correction factor equal to zero. Note that $e^{-2\min(|m_1|, |m_2|)} \leq 1$ so

$$\frac{(e^{-2\min(|m_1|, |m_2|)} + e^d)}{(1 + e^d)} \leq 1 \text{ and thus correction factor is always non-positive. We will use differ-}$$

ent estimations of this correction factor to modify min-sum in chapter 4.

Min-sum involves addition and compare and select operation where in a VLSI implementation are much simpler and faster than multiplication or look up tables.

2-2-3-3 Two sub-optimal algorithms: APP-based and BP-based decoding algorithms

Sub-optimal algorithms are algorithms that are developed by approximating the belief propagation algorithm. The purpose of sub-optimal decoding algorithms is to reduce complexity.

Two of the suboptimal algorithms have been developed by Fossorier et al. [17], and are discussed here. They are called APP-based and BP-based decoding algorithms. Other sub-optimal algorithms have been introduced in [14],[30], and [43] but because their implementation involves look-up tables, their complexity is higher than min-sum. They will not be discussed in this section.

The APP-based algorithm is based on bit flipping algorithm in which the flipping rule uses soft messages coming from check-nodes. The check-node operation is similar to the min-sum algorithm but the bit-node operation is approximated. All outgoing bit-node messages are the same and equal to sum of incoming messages. The mathematical representation of algorithm is as follow:

Initialization:

Hard decision is made based on r_i

$$\hat{s}_i = \begin{cases} 0 & \text{if } lw_i = r_i > 0 \\ 1 & \text{if } lw_i = r_i < 0 \end{cases}$$

$$M_{s_i \rightarrow c_j} = M_{s_i} = r_i$$

Check-node operation at check-node c_j :

$$\sigma_j = \sum_{i \in N(j)} \hat{s}_i \pmod{2}$$

$$M_{c_j \rightarrow s_i} = (\bar{\sigma}_j - \sigma_j) \underset{k \in N(j)/i}{Min} |M_{s_k}|$$

where σ_j is the result of the check-node j evaluation and $\bar{\sigma}_j$ is its module 2 complement.

Symbol-node operation:

$$M_{s_n} = lw_i + \sum_{i \in N(n)} M_{c_i \rightarrow s_n}$$

If $M_{s_n} < 0$ then flip the bit: $\hat{s}_i = \hat{s}_i \oplus 1$

Like min-sum, this algorithm does not depend on noise and therefore does not require any *a priori* information about the AWGN channel. Comparing with SP, APP-based decoding algorithm has around 1 dB of performance degradation [17].

The mathematical representation of BP-based algorithm is as follow:

Initialization:

Hard decision is made based on r_i

$$\hat{s}_i = \begin{cases} 0 & \text{if } lw_i = r_i > 0 \\ 1 & \text{if } lw_i = r_i < 0 \end{cases}$$

$$\hat{s}_{i \rightarrow j} = \hat{s}_i$$

$$M_{s_i \rightarrow c_j} = M_{s_i} = r_i$$

where $\hat{s}_{i \rightarrow j}$ is the hard decision of bit-node i outgoing message to the check-node j .

Check-node operation at check-node c_j :

$$\sigma_{ji} = \hat{s}_i \oplus \sum_{k \in N(j) \setminus i} \hat{s}_{k \rightarrow j} \pmod{2}$$

$$\bar{\sigma}_{ji} = \sigma_{ji} \oplus 1$$

$$M_{c_j \rightarrow s_i} = (\bar{\sigma}_{ji} - \sigma_{ji}) \underset{k \in N(j) \setminus i}{Min} | M_{s_k \rightarrow c_j} |$$

Symbol-node operation:

$$M_{s_i \rightarrow c_j} = lw_i + \sum_{k \in N(i) \setminus j} M_{c_k \rightarrow s_i}$$

$$M_{s_i} = lw_i + \sum_{k \in N(i)} M_{c_k \rightarrow s_i}$$

If $M_{s_i} < 0$ then flip the bit estimation $\hat{s}_i = \hat{s}_i \oplus 1$ and if $M_{s_i \rightarrow c_j} < 0$ then flip the message hard estimation $\hat{s}_{i \rightarrow j} = \hat{s}_{i \rightarrow j} \oplus 1$.

This algorithm is similar to min-sum with a different process for finding the sign of the messages and has the same complexity and performance of min-sum decoding algorithm.

A comprehensive comparison of algorithms complexity is done in the next section. BP-based algorithm is not included because its complexity is exactly the same as min-

sum. In [9], this algorithm is modified by applying a normalization factor to the check-node outgoing messages. The normalized BP-based algorithm complexity is given in table (2-3.1) for comparison.

2-3 Comparison of the algorithms

Table (2-3.1) shows the number of operations required in each node for all discussed algorithms in terms of node degree. It also lists the number of operations required for weight calculation, initialization, hard decision, and codeword checking. The bit (hard) operations are not considered because in a soft decoding algorithm, the added complexity by bit operations is negligible. The check-node degree and bit-node degree are shown by d_c and d_s , respectively. To describe how these numbers are determined we drive some of them here.

Number of check-node operations in the conventional SP:

There is $2^{(d_c-2)}$ set of incoming messages which satisfy a check-node when the corresponding bit-node is zero, and in each set there is $d_c - 1$ numbers that have to be multiplied. Thus $d_c - 2$ multiplications are required in each set and $2^{(d_c-2)}(d_c - 2)$ for all sets. To add all sets together $(2^{(d_c-2)} - 1)$ addition is required. The same amount of multiplication and addition are required when corresponding bit-node is one. Thus total $2^{(d_c-1)}(d_c - 2)$ multiplication and $(2^{(d_c-1)} - 2)$ addition are required to calculate two weights of one outgoing messages. Then total $d_c(2^{(d_c-1)}(d_c - 2))$ multiplication and $d_c \times 2^{(d_c-1)}$ addition are required for a check-node.

Number of bit-node operation in the ratio SP:

For each outgoing message, the $d_s - 1$ extrinsic messages have to be multiplied which requires $d_s - 2$ multiplications and the result has to be multiplied by local weight which increases the number of multiplication to $d_s - 1$. Thus for d_s outgoing messages $d_s(d_s - 1)$ multiplications are required. Other numbers of table have been obtained based on similar reasoning.

As we can see in the table (2-3.1) conventional SP has the highest complexity among these algorithms. It uses two weights in each message and needs more than twice operation in each node compared to one-weighted SP like ratio SP and mixed SP. For implementation purposes in digital VLSI domain this algorithm requires two times memory compared with all other one-weighted algorithms.

Although the check-node operation in difference SP uses one weight for each message and is simpler than the complex check-node operation in conventional SP, but bit-node operation is still complex and needs two weights for each message. Converting one-weight messages to two-weight messages in each iteration also increases the complexity.

It can be observed that ratio SP and mix SP have less complexity than standard sum-product.

Ratio SP and mix SP use one weight for each message. Simpler node operations in these algorithms make them more proper for implementation. Most of real implementations have been reported for log-ratio SP [8], which doesn't involve multiplication.

Mix SP has less operation compared to ratio SP and seems to be the best choice for implementation in likelihood domain.

The min-sum algorithm has much lower complexity than all versions of SP algorithms. This algorithm does not involve multiplication and just performs addition and

compare and select. Although it is possible to avoid multiplication in log-ratio SP and mix SP in the logarithmic domain, but it makes check-node operation in log-ratio SP and weight conversion in mixed SP very complicated in a perfect arithmetic. For quantized version of these algorithms, these parts can be implemented with lookup tables which have high complexity especially for high number of quantization bits.

The APP-based algorithm has the lowest number of operations. This is due to simpler bit-node operation in this algorithm with respect to min-sum. This simplification causes around 0.5 dB performance loss with respect to min-sum algorithm. Some modifications were proposed in [9] and [10] to compensate for this performance gap. The modified versions of APP-based and BP-based algorithms also require multiplication.

2-4 Implementation issues of iterative decoding algorithms

The inherent parallelism in decoding of LDPC codes with iterative decoding algorithms, gives enormous flexibility in their implementation on VLSI platforms. This becomes more attractive when the density of transistors on the chip and their speed continue to increase while the cost per transistor continues to decrease.

For an efficient implementation many factors have to be considered. Some of these important factors will be discussed in this section. Encoding process in LDPC codes involves hard operations in order to multiply the data vector with the code generator matrix, and efficient methods have been devised for this purpose [34]. In implementing these coding schemes, a greater effort is made towards optimizing the decoder. In a practical approach, the kind of architecture, the way of partitioning the design, the memory management style, and the type of modules have to be chosen.

Table 2-3.1 Operations number for different iterative decoding algorithms

Algorithm	Check-node operation (per node per iteration)	Bit-node operation (per node per iteration)	Hard decision and codeword checking (per codeword per iteration)	Initialization and weight calculation (per codeword)	Performance loss compare to BP
Conventional sum-product	$2^{(d_c-1)} d_c (d_c - 2)$ (M) $d_c (2^{(d_c-1)} - 2)$ (S)	$2d_s (d_s - 1)$ (M)	n (CS) $2n$ (M)	$2n$ (LUT)	0
Standard sum-product	$d_c (d_c - 1)$ (M)	$d_s (d_s + 1)$ (M) d_s (S)	n (S) $2n$ (M)	$2n$ (LUT) n (S)	0
Ratio SP	$2d_c (d_c - 2)$ (M) $2d_c (d_c - 2)$ (S)	$d_s (d_s + 1)$ (M)	n (M)	$2n$ (LUT) n (M)	0
Log-ratio SP	$d_c (d_c - 2)$ (LUT)	$d_s (d_s + 1)$ (S)	n (S)	0	0
MBP	$d_c (d_c - 1)$ (M) $2d_c$ (S)	$d_s (d_s + 1) + d_s$ (M) $2d_s$ (S)	n (M)	$2n$ (LUT) n (S)+n (M)	0
Max-sum	$2^{(d_c-1)} d_c (d_c - 2)$ (S) $d_c (2^{(d_c-1)} - 2)$ (CS)	$2d_s (d_s + 1)$ (S)	n (S)	0	≈ 0.5 dB
Min-sum	$d_c (d_c - 2)$ (CS)	$2d_s (d_s + 1)$ (S)	n (S)	0	≈ 0.5 dB
BP-based Algorithm (with message normalization)	$d_c (d_c - 2)$ (CS) d_c (M)	$2d_s (d_s + 1)$ (S)	n($d_s + 1$) (S)	0	≈ 0
APP-based Algorithm	$d_c (d_c - 2)$ (CS)	d_s (S)	n (S)	0	≈ 1 dB
APP-based (with message normalization)	$d_c (d_c - 2)$ (CS) d_c (M)	d_s (S)	n (S)	0	≈ 0

1) Only soft operations are considered.

2) M stands for multiplication, S stands for summation, CS stands for compare and select, LUT stands for lookup table and n is block length.

Although other implementation methods, such as analog or optical, are possible, our focus in this work is digital implementation, where the weights and messages are represented by a finite number of bits and operations are performed in fixed-point arithmetic, and as a result, the issue of quantization effects on soft iterative decoding algorithms arises, which will be discussed in chapter 3.

In this section, we will define the important parameters in the digital implementation of LDPC codes and explain the relation ship between these parameters.

2-4-1 Throughput

The number of bits that are decoded in a time unit by a decoder is called decoder throughput. The decoder throughput has to be matched to the bit rate of the communication system.

2-4-2 Latency

The time it takes to decode a block is called latency. A maximum tolerable latency is defined for each system based on the application. This maximum tolerable latency can be long for data communication and short for real time applications. The following relation is between throughput and average latency.

$$\text{Throughput} = \text{block length} / \text{average latency}$$

2-4-3 Data throughput

From system design point of view, a better scale for decoding speed is data throughput. Data throughput is the number of information bits decoded per second.

$$\text{Data throughput} = (\text{block length} \times \text{code rate}) / \text{average latency}$$

Despite the relation between latency and throughput, the nature of block codes forces the designer to check both of these characteristics to match the decoding scheme to the communication system requirements. For example consider a LDPC code with length 2×10^7 and rate $\frac{1}{2}$. If the average latency for decoding each block is 10 seconds the data throughput would be 10^6 bits/sec. Although throughput is enough for transmitting many voice channels, this scheme cannot be used for real time two-way voice communications due to intolerable latency.

2-4-4 Performance

Performance is defined by the BER that the decoding algorithm offers in a certain range of E_b/N_o . Three regions can be distinguished in code performance behavior. For low E_b/N_o the coding scheme introduces more error than uncoded scheme. In second region, called water-fall region, BER drops rapidly with E_b/N_o . At higher E_b/N_o , BER reduction rate may decrease considerably or even stop. This region is called error floor region. LDPC codes have typically a lower error floor compared to turbo codes. In practice, all the codes are used in water fall region.

2-4-5 Complexity

The definition of complexity for iterative decoding algorithms is a bit fuzzy due to inherent parallelism in these algorithms when they apply to LDPC codes. There are three different definitions for iterative decoding algorithms complexity:

1. The amount of hardware required to implement their decoder.

2. The average number of iterations required to decode a block.
3. Total number of operations required to decode a block.

These definitions are related to each other and depending on how algorithm is implemented (parallel or serial) the tradeoff between hardware and decoding time will change. As we have seen in the previous sections, iterative decoding algorithms are performed in four steps:

1. initialization performed for each block
2. check-node operations performed in each iteration
3. bit-node operations performed in each iteration
4. hard decision and codeword checking

Steps 2 to 4 are repeated in each iteration consequently, but in each of these steps, operation in each node is performed independent of other nodes. This causes an inherent parallelism in iterative decoding algorithms for LDPC codes. In practice, the required hardware for implementation of the iterative decoding algorithms is related to the way these algorithms are implemented. The node operations can be performed in serial or parallel, or a combination of both.

Assume decoder has U_s and U_c physical nodes, with t_s and t_c being the delay for bit-node and check-node operations, respectively. Then the time needed for an iteration (in case of flooding scheduling) is:

$$t_{iteration} = \frac{S}{U_s} t_s + \frac{C}{U_c} t_c$$

where S and C are the number of bit and check nodes, respectively. Then the hardware complexity can be defined as:

$$\text{Hardware complexity} = H_s \cdot U_s + H_c \cdot U_c$$

where H_s and H_c are hardware required for bit and check nodes, respectively. In a (fully) parallel implementation, the iteration time is reduced to its minimum and the average numbers of iterations determine the overall throughput and average latency. In such cases, it seems that the number of iteration is a good measure for complexity of decoding algorithm.

It is worth to note that for a fair comparison of two algorithms, in addition to the number of iterations the complexity and delay of physical nodes have to be considered.

The third definition combines the complexity of physical node and the number of iterations but does not consider the parallelism factor.

Chapter 3

Quantization Effects on Min-sum Algorithm

3-1 Overview

As discussed in Chapter 2, the effect of quantization on iterative decoding algorithms is an important consideration in their implementation. Although the soft messages are analog and so they need an infinite number of bits for representation, they have to be represented by a finite number of bits in a digital implementation.

Recent research papers have addressed the quantization issues related to BP algorithms. Ping and Leung [32] have shown that the log-ratio SP version of BP has less sensitivity to the quantization effect than the difference SP. Log-ratio SP can perform close to the unclipped version performance with 6 bits compared to 8 bits required for difference SP. Richardson et al. [33] showed that a quantized BP algorithm can perform close to the unclipped and unquantized version performance by applying a two stage mapping to the 3 bit quantization scheme. In their approach however, just the messages can be stored using three bits, and the mappings cannot be performed with 3 bits in a hardware implementation.

In the comparison of the iterative decoding algorithms in section 2-3, it was shown that the min-sum algorithm has a good complexity performance tradeoff under perfect arithmetic. The following discussion describes how quantization will affect min-sum performance. It contains simulation results for three codes. In all simulation scenarios, the maximum number of iterations has been set to 200, which is at the end of the tail of the histogram of the number of iterations required for convergence. The histogram of iterations for (1268,456) code and 4-bit min-sum algorithm at $E_b/N_0=1, 1.5$ and 2 dB are shown in Appendix 1. Also, for each SNR, up to 10^6 codewords are simulated to have at least 30 (and much more than 100, on average) codeword errors.

3-2 Quantization Scheme

The received values are clipped symmetrically at a threshold c_{th} , and then uniformly quantized in the range $[-c_{th}, c_{th}]$. There are $2^q - 1$ quantization intervals, each represented by q quantization bits. The central quantization interval is symmetric with respect to the origin and is represented by 0. The rest of the intervals are pair-wise symmetric with respect to the origin and are represented by integer numbers with opposite signs.

Operations in both bit and check nodes are performed in fixed-point arithmetic. Note that overflow can not happen in check-nodes and the additions in bit nodes are performed in such an order that the possibility of overflow is minimized. It is easy to see that this is equivalent to sequentially adding values with opposite signs, when possible, or to have the adder size big enough, or by using the overflow tracking system described in Chapter 5. In the case of overflow in bit node computations, the result is mapped to $2^{q-1} - 1$ or $-(2^{q-1} - 1)$ depending on the sign being positive or negative, respectively.

3-3 Clipping Threshold Effect

It is clear that clipping is mandatory in a quantization scheme. At first glance, it seems that quantization reduces the performance due to the introduction of quantization noise into the system. Quantization noise can be divided to two parts: noise due to quantization error in each quantization interval; and noise due to the clipping at the edges of the quantization range. In fact these two elements of the quantization noise have different effects on the performance of the min-sum algorithm. It is shown in section 2-2-3-2 that check-node outgoing messages are larger in min-sum compared to those of log-ratio SP,

which has a better performance in terms of bit error rate. Clipping improves min-sum performance by reducing the message values and compensating for the oversized check-nodes outgoing messages. Although clipping improves performance of the min-sum algorithm, there is still a considerable gap between the performance of unquantized min-sum with clipping and log-ratio SP. It seems that more correction on oversized messages is required to close this performance gap. We will look at the size of the messages and their required correction in more details in Chapter 4.

We performed some simulations on three LDPC codes; two regular and one irregular. The regular codes, taken from [44], have parameters $(n, k) \langle d_s, d_c \rangle = (273, 191) \langle 3, 10 \rangle$ and $(n, k) \langle d_s, d_c \rangle = (8000, 4000) \langle 3, 6 \rangle$, where n and k are the block length and the dimension of the code while d_s and d_c are the bit-node and check-node degrees, respectively. The irregular code has a lower rate with parameters $(1268, 456)$, and is constructed by construction 2A [26] in [28]. The results for $(1268, 456)$ code are presented in the main body of the thesis and the other code results are given in the appendices.

Figure 6-9 demonstrate the performance of the decoding with clipping compared to unclipped min-sum algorithm for different clipping thresholds and for a given E_b/N_o . It is observed that the optimum clipping threshold is not very sensitive to SNR. The optimum clipping threshold for this particular irregular $(1268, 456)$ code is 2 and the BER asymptotically approaches the unclipped min-sum performance as the clipping threshold gets larger. The same behavior is observed for the $(8000, 4000)$ regular code except that the optimum threshold is 1.25. For the short block length regular $(273, 191)$ code however, the optimum threshold is not as distinguishable as it is for the medium and long block

Figure 6. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for (1268,456) irregular code at $E_b/N_o=1$ dB.

Figure 8. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for (1268,456) irregular code at $E_b/N_o=2$ dB.

Figure 7. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for (1268,456) irregular code at $E_b/N_o=1.5$ dB.

Figure 9. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for (1268,456) irregular code at $E_b/N_o=2.5$ dB.

length codes. The results for these codes are given in Appendix 2.

Figure 10 shows that for a range of SNR, the clipped min-sum algorithm has better performance than unclipped version when all the messages are clipped at a proper threshold.

Figure 10. BER curves for the (1268,456) code, decoded with unclipped min-sum and unquantized min-sum with optimum clipping $cth = 2$.

3-4 Quantization Effects

It was shown in the previous section that clipping could improve min-sum performance. In this section, the quantization effects on the min-sum algorithm are examined. The quantization scheme described in section 3.1 is considered and the effects of clipping and quantization are studied by simulating the performance of the three previously mentioned LDPC codes. At each SNR, and for each given number of quantization bits, the effect of the clipping threshold on BER was plotted for the irregular, long regular and short regular codes, in Figure 11-14, and Figure 56-64 in Appendix 2, at SNR = 2dB, 2dB and 4dB, respectively.

Figure 11. The effect of clipping threshold cth on performance of quantized min-sum for (1268,456) code at $E_b/N_0 = 1$ dB.

Figure 12. The effect of clipping threshold cth on performance of quantized min-sum for (1268,456) code at $E_b/N_0 = 1.5$ dB.

Figure 13. The effect of clipping threshold cth on performance of quantized min-sum for (1268,456) code at $E_b/N_0 = 2$ dB.

Figure 14. The effect of clipping threshold cth on performance of quantized min-sum for (1268,456) code at $E_b/N_0 = 2.5$ dB.

It can be seen that the optimum clipping threshold is almost the same for different numbers of quantization bits, and is approximately equal to 2 and 1.25 for the irregular and long regular codes, respectively. It also can be observed that the optimal clipping threshold is the same for unquantized min-sum with clipping and quantized min-sum. For the short block length regular codes however, the best threshold is not as distinguishable as it is for the medium and long block length codes.

Quantization degrades the performance with respect to unquantized min-sum with clipping but still the quantized version of the algorithm can demonstrate better performance than unclipped min-sum if a proper threshold has been chosen. This improvement is due to the fact that the overall performance is the mutual result of the clipping effect and the quantization effect on min-sum algorithm. The results show that performance with 4 quantization bits is close to or even better than unclipped, unquantized min-sum. The quantized min-sum performance improves slightly by increasing the number of quantization bits but a saturation can be reached (near unquantized min-sum with clipping) with around 6 bits, where increasing the number of quantization bits no longer improves the performance. These results are comparable with the result reported in [32] for log-ratio SP, that can perform like unclipped SP with 6 bits.

For all codes, bit error rate curves of unclipped min-sum and quantized min-sum with different numbers of quantization bits are plotted in Figure 15, and Figure 65,66. For each code, the clipping threshold is fixed and is chosen to be 2, 1.25 and 1.5, for the irregular, long regular and short regular codes, respectively. These figures show that by using 4 bits to represent the received values and messages, one can obtain near unclipped min-sum performance. It is noted that very little can be gained by increasing the number

Figure 15. BER curves for the (1268,456) LDPC code decoded by unclipped min-sum and quantized min-sum for different quantization bits ($c_{th}=2$).

of bits beyond 4. In fact, for the number of bits larger than 6, the results practically coincide with those for 6 bits. It is also interesting to note that in some cases, quantized min-sum outperforms unclipped min-sum. This is due to the clipping threshold, which improves min-sum performance.

3-5 Complexity of Quantized Versions of Min-sum algorithm

In this section, the complexity of the algorithms in terms of the average number of iterations is investigated. Table 3.5-1 gives the average number of iterations for quantized min-sum with different number of quantization bits. It can be observed that increasing the quantization levels decreases the average number of iterations. The results for the other two codes are given in Appendix 8.

E_b/N_0 (dB)	6-bit MS	5-bit MS	4-bit MS	3-bit MS	UN- MS	UN- LRSP
1	35.3	44.2	45.8	52.2	34.3	27.1
1.5	23.1	28.4	27.1	33.7	23.8	17.8
2	13.5	15.3	14.5	18.1	13.9	10.7
2.5	9.1	9.9	9.5	11.3	9.3	8

Table 3.5-1. Average number of iteration for (1268,456) code decoded by min-sum (MS),unclipped min-sum (UN-MS) and unclipped log-ratio SP (UN-LRSP), without quantization and with different number of quantization bits.

3-6 Density Evolution

This section describes part of the effort undertaken to describe the behavior of the code under clipping and quantization analytically. It will also be shown where and under what circumstances the density evolution technique can be applied to compute probability of error for a code.

Density evolution [33] is a technique to obtain expressions for the probability density functions of the messages exchanged in each iteration of an iterative decoding algorithm. This technique is applied to an ensemble of codes and a concentration theorem [33] guarantees that most of the codes in the ensemble demonstrate a performance close to what was predicted by density evolution. The underlying assumption for this technique is that the graph of the code is effectively cycle free. This technique is greatly simplified if the iterative decoding algorithm satisfies some symmetry conditions [33]. The results of this analysis are applicable asymptotically to large block length codes.

Most of the research work related to density evolution is concentrated on sum-product, but some work has also been done on min-sum [5], [10]. In the next section, we briefly explain this technique.

In these works, density evolution is used as a tool to calculate the threshold of the code. Threshold is defined as the minimum signal to noise ratio for which the probability of decoding error approaches zero as the length of the code and the number of iteration approaches infinity.

Although density evolution is usually applied to the codes with cycle-free graphs, it can be used to predict the probability of error for a practical code where the graph contains cycles, in the event that the number of the iterations is less than half of the minimum girth of the graph. In a Tanner graph, the girth at a node is defined as the length of the shortest cycle that passes through that node. If the number of iterations is less than half of the minimum girth of the graph, all the messages remain independent, because no message that has passed through an edge, has had enough time to return to that edge.

3-7 Density evolution for min-sum algorithm

In min-sum algorithm, the initial messages are received values for bits. Under BPSK modulation and AWGN channel assumptions, received values are independent and identically distributed (i.i.d.), and are summation of two independent random variables; transmitted values and noise. When number of iterations is smaller than the minimum girth of the graph the messages remain i.i.d., as they are linear combinations of the initial messages.

In the following, we describe the density evolution for a regular $\langle d_s, d_c \rangle$ code.

For bit-node operations, any outgoing message is obtained by adding the extrinsic incoming messages. Then, the probability density function (pdf) of the outgoing message is the convolution of the pdf of incoming messages:

$$P(M_{s_i \rightarrow c_j}) = \text{Conv}_{k \in N(i) \setminus j} \{P(M_{c_k \rightarrow s_i})\}$$

The above expression is valid as long as the messages $M_{c_k \rightarrow s_i}$ are independent. This is true if the Tanner graph is cycle free or the number of iterations is smaller than half of the minimum girth of the graph.

The corresponding expression for check-node operation is complicated, and since it was desired to apply density evolution to quantized version of min-sum therefore, the exact probability mass functions (pmf) of quantized messages were directly calculated.

Assuming that all zero codeword has been sent, the probability of the incoming messages to a check-node for an AWGN channel in the first iteration is:

$$P\{M_{s_i \rightarrow c_j}^1(S_l)\} = \int_{b_{l-1}}^{b_l} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-1)^2}{2\sigma^2}} dx$$

where

S_l is the l_{th} quantization symbol,

b_l and b_{l-1} are the borders of l_{th} quantization symbol,

σ is the variance of the noise.

and $P\{M_{s_i \rightarrow c_j}^1(S_l)\}$ is the probability of sending l_{th} quantization symbol as message of bit-node i to check-node j in first iteration

The pdf of check-node outgoing messages is then:

$$P\{M_{c_j \rightarrow s_i}^1(S_l)\} = \sum_{k=1}^{d_c-1} \binom{d_c-1}{k} P\{M_{s_i \rightarrow c_j}^1(S_l)\} \cdot \left[\sum_{m=(-|l|)}^{-(2^{q-1})+1} P\{M_{s_i \rightarrow c_j}^1(S_l)\} + \sum_{m=|l|}^{2^{q-1}-1} P\{M_{s_i \rightarrow c_j}^1(S_l)\} \right]$$

Unless the number of iterations is larger than half of the minimum girth, the information sent by a node does not return to it and all the incoming messages to a node have the same pdf. Therefore, the formula can be simplified as:

$$P\{M_{c \rightarrow s}^{it}(S_l)\} = \sum_{k=1}^{d_c-1} \binom{d_c-1}{k} P\{M_{s \rightarrow c}^{it}(S_l)\} \cdot \left[\sum_{m=(-|l|)}^{-(2^{q-1})+1} P\{M_{s \rightarrow c}^{it}(S_l)\} + \sum_{m=|l|}^{2^{q-1}-1} P\{M_{s \rightarrow c}^{it}(S_l)\} \right]$$

where

it is the iteration number,

d_c is check-node degree,

q is number of quantization bits,

l is the label of the quantization interval and $-(2^{q-1})+1 < l < 2^{q-1}-1$,

S_l is the l_{th} quantization symbol and we assume that the labels of quantization interval are signed numbers, symmetric with respect to origin, and include zero label for central interval.

The bit-node outgoing messages pdf can be calculated as follows:

$$P\{M_{s \rightarrow c}^{it}(S_l)\} = \sum_{\substack{\text{the_set_of} \\ \text{messages } S_{ji} \text{ that:} \\ S_{j1} + S_{j2} + \dots + S_{jd_c} = S_l}} \left[\prod_{\substack{i=1 \\ j \in A}}^{d_c-1} P\{M_{c \rightarrow s}^{it-1}(S_{ji})\} \right]$$

where

$A = \{-2^q+1, -2^{q-1}+2, \dots, -1, 0, 1, \dots, 2^q-1\}$ is the set of quantization interval labels,

and S_{ji} is the j_{th} quantization message passing by i_{th} check-node.

The density evolution results are compared with simulation results for a regular (1200, 300) code with $d_c = 4$ and $d_s = 3$, and with minimum girth equal to 6, in Figure 16, Figure 17 and Figure 18. Note that in the third iteration, for the nodes with girth 6, there is a feedback of primary information, and the incoming messages are no longer independent. As can be seen in these figures, density evolution results match to the simulation results for 2 iterations (the slight difference is due to the confidence interval of simulation results). This shows that density evolution can be used to calculate probability of error when the number of iterations is smaller than half of the minimum girth of the graph. As the number of iterations goes higher, deviation between density evolution and simulation results appear. In our example, for 3 iterations the deviation is small because only in the nodes with girth 6 feed back appears and messages are not i.i.d. any more. For more number of iteration, more nodes involve in feedback and more deviation from density evolution results is observed.

There is no connection between the simulation results and the density evolution results, when the number of iterations exceeds half of the minimum girth. This is simply because the dynamic of the algorithm changes dramatically after feedback appears, due to the cycles in the code's graph.

It may seem that density evolution technique can be applied to find the error floor of a practical code. Such an attempt could fail however, if the average number of iterations in error floor region is greater than half of the minimum girth.

Figure 16. Density evolution and simulation results for $(1200,900)$ $\langle 3,4 \rangle$ regular code with minimum girth 6 after 2 iteration.

Figure 17. Density evolution and simulation results for $(1200,900)$ $\langle 3,4 \rangle$ regular code with minimum girth 6 after 3 iteration.

Figure 18. Density evolution and simulation results for (1200,900) <3,4> regular code with minimum girth 6 after 4 and 5 iteration

Chapter 4

Modifications to Min-sum Algorithm

4-1 Comparison of Min-sum and Log-ratio SP

In the previous chapters, the min-sum algorithm was explained and the effects of clipping and the number of quantization bits on its performance were studied in detail. It was shown that although quantization (in general) degrades the performance of min-sum, clipping can provide some improvement, and that in many cases the overall effect is such that only 4 bits of quantization along with a proper clipping threshold result in a performance close to or even better than that of unclipped and un-quantized min-sum. This compares to 6 bits of quantization that is required in a similar implementation of log-ratio SP algorithm [32].

The min-sum algorithm has less complexity and lower sensitivity to quantization compared to different versions of SP algorithm. These advantages however, come at a cost in performance. For a given code and a given channel model, and for a prescribed error rate, the min-sum algorithm requires up to a few tenths of a dB more transmitted power than log-ratio SP.

There has been some recent works to partially close this gap and come up with new versions of iterative decoding algorithms which provide performance/complexity trade-offs that are between those of min-sum and log-ratio SP [9], [10].

In this chapter, the reason of existence of this performance gap will be investigated and two simple modifications are proposed that can considerably improve the performance of min-sum with negligible cost in complexity. As the main concern is practical implementation of decoding algorithms, emphasis is placed on modifications that can be incorporated in the quantized version of the min-sum algorithm with minimal efforts.

4-2 Simplified Correction Function

It was shown in Chapter 2 that the only difference between min-sum and log-ratio SP algorithms is their check-node operation. Suppose that there is a check-node p with degree 3, and let m_1 and m_2 be the input messages to this check-node. The difference between the check-node output messages in min-sum and log-ratio SP algorithms was given as the correction factor in section 2-2-3-2. It was shown that the correction factor is always non-positive so check-node output messages are larger in min-sum compared to log-ratio SP. Figure 19 depicts the difference of check-node output messages in these two algorithms. The correction factor is repeated here:

$$\text{Correction factor} = \log \frac{(e^{-2 \min(|m_1|, |m_2|)} + e^d)}{(1 + e^d)}$$

Figure 19. The difference between check-node degree three outputs in log-ratio SP and min-sum; x and y are extrinsic inputs and z is the difference between the outputs.

Where $d = \|m_1\| - \|m_2\|$. For large values of $\min(\|m_1\|, \|m_2\|)$, the term $e^{-2 \min(\|m_1\|, \|m_2\|)}$ is very small and we can write:

$$\log \frac{(e^{-2 \min(\|m_1\|, \|m_2\|)} + e^d)}{(1 + e^d)} \approx \log \frac{e^d}{1 + e^d} = \log \frac{e^d}{e^d(e^{-d} + 1)} = \log \frac{1}{1 + e^{-d}} = -\log(1 + e^{-d})$$

The simplified correction function, denoted by $f(d)$, is defined as

$$f(d) = -\log(1 + e^{-d})$$

and is shown in Figure 20.

To compare the effects of simplified correction with those of perfect correction, we applied it to the min-sum algorithm in the following order:

For a check-node p , let e be the incident edge with the smallest incoming message in absolute value. Let us denote this message by m_e . In min-sum algorithm, the absolute value of all the outgoing messages from p is equal to $|m_e|$, except for the message sent out along the edge e . The absolute value of this message is equal to $|m_{e^}|$, where m_{e^*} is the second smallest incoming message in size. For all outgoing messages except that of the edge e , $f(d)$ is first applied to $|m_e|$ and one of the other extrinsic messages chosen randomly from the set of remaining extrinsic messages. The result is shown by $|m_e|_1$. Then $f(d)$ is applied to $|m_e|_1$ and the next extrinsic message which also chosen randomly. This procedure continues until $f(d)$ is applied $(d_c - 2)$ times for all the extrinsic messages. Note that this is done completely randomly except that it is started with $|m_e|$. For the edge e , the same process is repeated, starting with $|m_{e^*}|$.*

It should be mentioned that applying the simplified correction factor to the outputs of a check-node is associative. The proof can be found in Appendix 7.

Based on our simulations, by applying the correction factor $f(d)$ to the outputs of the check-nodes, more than 90% of the performance gap between unclipped min-sum (floating-point arithmetic with no clipping or quantization) and log-ratio SP is closed (Figure 21).

Figure 20. The simplified correction function $f(d)$ for min-sum algorithm.

Figure 21. BER curves for (1268,456) code, decoded by unclipped min-sum, modified unclipped min-sum with exact simplified correction factor and sum-product.

The precise implementation of $f(d)$ is however complex. To simplify the implementation, different approaches, such as using a look-up table, approximating $f(d)$ with a piece-wise linear function, or simply a stepwise function, can be adopted. Clearly, each approach will result in a different performance-complexity tradeoff.

We devised two simple correction steps for min-sum, which are applied to the size of the outgoing messages from check-nodes (signs remain the same as those in conventional min-sum). These correction steps are named “conditional correction step” and “unconditional correction step”.

4-3 Min-sum Modification with Conditional Correction Step

A closer look at the simplified correction function curve in Figure 20 shows that two regions are distinguishable; a region with large amount of correction for small message differences and a region with small amount of correction for large message differences. This leads to a step-wise approximation of the simplified correction function. This step-wise approximation is shown in Figure 22. Here, d_{step} is a threshold; if the difference between the messages is smaller than d_{step} , a correction of $(-y_{step})$ is applied, and if the difference is greater than d_{step} , no correction is applied and the same outgoing message is sent. Now a conditional correction step is introduced:

Conditional correction: *For each edge e , let $m_{e,min}^{(ex)}$ denote the extrinsic incoming message with the smallest absolute value and $m_{e,out}$ denote the outgoing message. At the beginning, we set $m_{e,out} = m_{e,min}^{(ex)}$. Then, we select an incoming extrinsic message m , other*

Figure 22. An example of step-wise approximation of correction function which leads to conditional correction step; d_{step} and y_{step} are shown.

than $m_{e,min}^{(ex)}$, randomly, and update the size of the outgoing message along the edge e , reducing it to $\max(|m_{e,out}| - y_{step}, 0)$, if $|m| \in [|m_{e,out}|, |m_{e,out}| + d_{step}]$ for some predetermined positive numbers d_{step} and y_{step} , respectively. Starting with the updated $m_{e,out}$, we repeat this procedure until no incoming extrinsic message (other than $m_{e,min}^{(ex)}$) remains.

The correction step is not associative when $y_{step} < d_{step}$. Consider a case that $y_{step}=0.1$ and $d_{step}=0.2$ and messages with absolute values $|m_1|=0.8$, $|m_2|=0.89$ and $|m_3|=0.95$ are passed to a check node of degree 4. The absolute value of the outgoing message would be 0.6 or 0.7 depending on the order that conditional correction has been applied.

Conditional correction step is associative and commutative when $y_{step} \geq d_{step}$, and therefore the order of applying this modification to the incoming messages doesn't affect the final outgoing message. The proof follows:

Let us denote extrinsic incoming messages other than $m_{e,\min}^{(ex)}$ with $|m_i|$, $i = 1, 2, 3, \dots, n$. If one of these messages is in the interval $[|m_{e,\min}^{(ex)}|, |m_{e,\min}^{(ex)}| + d_{step}]$, the correction y_{step} is applied and the outgoing message changes to either $|m_{e,\min}^{(ex)}| - y_{step}$ or zero. If $y_{step} \geq d_{step}$ then

$$|m_i| - (|m_{e,\min}^{(ex)}| - y_{step}) \geq d_{step} \quad \text{for } i = 1, 2, 3, \dots, n.$$

It means than none of the other messages can satisfy the condition and trigger another correction. It is obvious that if the result of the first correction is zero, no further correction can be applied.

Note that when $y_{step} \geq d_{step}$, the conditional correction step is applied at most once for each outgoing message. This fact can be used to reduce the operation time in practical implementation. First, because no ordering of extrinsic messages is required and second, because it is possible to skip checking other extrinsic messages after the first correction was applied. It is noteworthy that the second reason is especially important in asynchronous digital implementation.

We studied the effects of this correction method on the irregular (1268, 456) LDPC code. At $E_b/N_o = 1.5$ dB, the above-mentioned conditional correction step was applied to unclipped min-sum for different values of d_{step} and y_{step} . For all cases the conditional correction was applied to randomly selected extrinsic messages. The results are shown in Figure 23. As it can be observed, the best result is achieved when $d_{step} = 0.5$ and $y_{step} = 0.5$. The advantage of the point corresponding to $d_{step} = y_{step} = 0.5$ is that the conditional correction step is associative at this point and it can be implemented randomly.

Figure 23. BER results for (1268,456) code, decoded by unclipped min-sum with conditional correction step for different y_{step} and d_{step} at $E_b/N_0=1.5$ dB.

The simulation results for unclipped min-sum when the conditional correction step with $d_{step} = y_{step} = 0.5$ is applied, are quite satisfactory and have been shown in Figure 24. Similar results for two other codes have been shown in Appendix 4. It can be seen that by applying the conditional correction step to the check-node outputs, more than 85% of the performance gap between min-sum (floating-point arithmetic with no clipping or quantization) and log-ratio SP is closed. At high SNR values, a result better than the result of applying simplified correction factor is achieved.

Figure 25 shows the bit error rate results of unquantized min-sum with clipping for different clipping threshold values, conditionally corrected with $d_{step} = y_{step} = 0.5$ at $E_b/N_0=1.5$ dB. Note that the unquantized min-sum with clipping and with conditional

Figure 24. BER curves for (1268,456) code, decoded by unclipped min-sum, modified unclipped min-sum with conditional correction step and sum-product.

Figure 25. BER results for (1268,456) code, decoded by unquantized min-sum with clipping and with conditional correction step, with different clipping threshold and $y_{\text{step}} = d_{\text{step}} = 0.5$ at $E_b/N_0 = 1.5$ dB.

correction is not that sensitive to the clipping point. More specifically, it can be said that clipping threshold does not much affect decoding performance, provided that $c_{th} > 2$.

This is because the conditional correction step reduces the size of the output messages of the check-nodes considerably. In this situation clipping can not much affect the performance by reducing the oversized message as it does for unmodified cases.

We also applied conditional correction with different values of d_{step} and y_{step} , to unquantized min-sum with clipping and with optimum clipping threshold $c_{th} = 2.5$, at $E_b/N_o = 1.5$ dB. The results are shown in Figure 26. Note that unlike unclipped min-sum, unquantized min-sum with clipping is not very sensitive to the value of d_{step} . This observation suggests that a correction step, which is independent of d_{step} , may work as well as conditional correction in these cases. We will discuss this matter in 4-4.

Figure 26. BER results for (1268,456) code, decoded by unquantized min-sum with clipping and with conditional correction step for different y_{step} and d_{step} and optimum clipping $c_{th}=2.5$ at $E_b/N_o=1.5$ dB.

The best result is achieved with $d_{step} = 0.2$ and $y_{step} = 0.3$. The results for unquantized min-sum with clipping and with the optimum conditional correction step ($d_{step} = 0.2$ and $y_{step} = 0.3$) and for different clipping threshold values are given in Figure 27.

It can be observed that the unquantized min-sum with clipping and with optimum conditional correction is also not that sensitive to the clipping threshold. It is again because optimum conditional correction reduces the magnitude of oversized messages and the resulting algorithm is closer to the optimal one (log-ratio SP).

Figure 28 shows the bit error rate performance of quantized min-sum with clipping threshold $c_{th} = 2$ for different d_{step} and y_{step} . The best result is still achieved at $d_{step} = 0.2$ and $y_{step} = 0.3$. Note that this BER is not much different from the one obtained based on $c_{th} = 2.5$, $d_{step} = y_{step} = 0.5$, i.e., BER = 0.002778. In general the optimal values of c_{th} , d_{step} , and y_{step} depend on the code and the SNR. However, our results for this code at $E_b/N_0 = 1.5$ dB indicate that optimizing d_{step} and y_{step} for the clipping threshold c_{th} optimized for min-sum without modification, results in more or less the same bit error rate.

For quantized min-sum, optimized values of d_{step} , y_{step} , and c_{th} depend on the number of quantization bits and SNR for each code. But since such an optimization is a very tedious task, we used the optimized c_{th} for the quantized min-sum without modification and only optimized d_{step} and y_{step} . The method of applying conditional correction step is exactly the same as unquantized case except that the parameters d_{step} and y_{step} are integer multiples of the quantization interval l_q , i.e.,

$$\begin{aligned} y_{step} &= n.l_q \\ d_{step} &= m.l_q \end{aligned}$$

Also the discussions on associativity of unquantized case are valid here.

Figure 27. BER results for (1268,456) code, decoded by unquantized min-sum with clipping and with conditional correction step, with different clipping threshold and ystep=0.3 and dstep=0.2 at $E_b/N_0=1.5$ dB.

Figure 28. BER results for (1268,456) code, decoded by unquantized min-sum with clipping and with conditional correction step for different ystep and dstep and clipping $c_{th}=2$ at $E_b/N_0=1.5$ dB.

The parameters m and n are optimized for quantized min-sum with 4, 5 and 6 quantization bits. For 4-bit quantized min-sum, we did this optimization and it turned out that the optimum value of m and n , as it has been shown in Figure 29, is 1. The conditional correction step with $m = n = 1$ was applied to min-sum (4-bit quantized version) and the results are compared to unclipped min-sum with and without conditional correction, 4-bit quantized min-sum, and log-ratio SP in Figure 30. As it can be seen, the modified versions considerably outperform their un-modified counterparts. Similar results for quantized min-sum with 5 and 6 quantization bits are given in Appendix 5. The results for other two codes are also shown in appendices 4 and 5. Clipping threshold are $c_{th}=2$, $c_{th}=1.25$ and $c_{th}=1.5$ for (1268,456), (8000, 4000) and (273, 191) codes, respectively.

Figure 29. BER curves for (1268,456) code, decoded by 4-bit modified min-sum with conditional correction step with different $d=dstep$ and $y=ystep$ ($c_{th}=2$).

Figure 30. BER curves for (1268,456) code, decoded by 4-bit min-sum, unclipped min-sum, unclipped min-sum with conditional correction $d_{step}=y_{step}=0.5$, 4-bit min-sum with conditional correction $m=n=1$, and sum-product.

With a more careful look at the simulation results, it can be said that for small values of y_{step} , better performance is achieved by increasing d_{step} . The results for a wider range of d_{step} is given in Figure 31. It can be seen that unclipped min-sum with conditional correction has very good performance for large values of d_{step} . Thus, a good performance is expected when $d_{step} \rightarrow \infty$. In this case, correction y_{step} is applied *unconditionally* to the check-node outputs so it is called unconditional correction.

4-4 Min-sum Modification with Unconditional Correction Step

More precisely, the unconditional correction step is defined as follows:

Unconditional correction¹ step: For each edge e , the size of the outgoing message is $\max(|m_{e,\min}^{(ex)}| - k, 0)$, for some predetermined positive number k .

Figure 32 shows the effect of unconditional correction step on unclipped min-sum for different corrections. Figure 33 and Figure 34 contain min-sum results with unconditional correction steps for 4 quantization bits. Results of quantized min-sum with 4 quantization bits and sum-product are also given for reference. For all the simulations, $k=1$. This value appears to result in the best performance. As expected, the performance of the unconditional modification of min-sum is superior to that of the conditional one, and very close to BP.

Figure 31. BER results for (1268,456) code, decoded by unclipped min-sum with conditional correction step at $E_b/N_0=1.5$ dB.

¹ A similar algorithm has been independently developed by J. Chen and M. Fossorier [10][11]. They have also analyzed the algorithm using density evolution. It is shown in [10] that the continuous version of this algorithm is only slightly inferior to the normalized BP-based algorithm of [9].

Surprisingly, for both codes in Figure 34, 4-bit MS with the unconditional correction step outperforms BP, especially at high SNRs (note that the two decoders are set to operate in parallel on the same set of received vectors. Also, both decoders have the same maximum number of iterations). This indicates that algorithms which are optimal on cycle-free graphs do not necessarily deliver the best performance on graphs with cycles. The results for 5 and 6 bits are given in Appendix 5.

The message error rate (MER) results are presented in Appendix 3. The MER curves for unclipped min-sum and sum-product have the same trend as bit error rate (BER) curves. But for modified versions of min-sum algorithm some differences exist between BER and MER curves. It can be observed that for (8000,4000) code, unconditional correction improves BER more than MER. Also, unclipped min-sum with conditional and unconditional correction have a better MER compared to quantized version of the algorithm.

Figure 32. BER curves for (1268,456) code, decoded by unclipped min-sum with unconditional correction step.

Figure 33. BER curves for (1268,456) code decoded by sum-product, 4-bit quantized min-sum, and 4-bit quantized min-sum with the conditional and unconditional correction steps.

Figure 34. BER curves for codes (273,191)(dashed lines) and (8000,4000)(solid lines), decoded by sum-product, 4-bit quantized min-sum, and 4-bit quantized min-sum with the conditional correction step.

For the (1268,456) code, MER curves with unconditional and conditional correction step do not follow the trend of BER curves at greater E_b/N_o values. For the (273,191) code, there is not much difference between trend of BER and MER curves.

4-5 Min-sum Modifications and Their Complexity

In this section the complexity of the algorithms in terms of average number of iterations is investigated. Table 4.4-1 gives the average number of iterations for different algorithms. It also contains the average number of iterations for different number of quantization bits. For quantized versions of min-sum algorithms, the average numbers of iterations are given for the optimal cases (optimal c_{th} for non modified versions, and optimal m , n and k for modified versions). It can be observed that number of iterations reduces when more quantization bits are used. Both modifications decrease the average number of iterations. This has direct effect on the average decoding time and can compensate the delay due to additional operations in conditional and unconditional correction steps.

E_b/N_o (dB)	6-bit MS	5-bit MS	4-bit MS	3-bit MS	6-bit MS- CCS	5-bit MS- CCS	4-bit MS- CCS	6-bit MS- UCS	5-bit MS- UCS	4-bit MS- UCS	UN- MS	UN- LRSP
1	35.3	44.2	45.8	52.2	29.6	34.2	35.2	35.8	34.3	36.2	34.3	27.1
1.5	23.1	28.4	27.1	33.7	20.4	23.6	24	17.3	18.8	21.1	23.8	17.8
2	13.5	15.3	14.5	18.1	12.4	13	13.6	11.9	11.16	12.1	13.9	10.7
2.5	9.1	9.9	9.5	11.3	9.1	9.2	9.1	8.2	8.1	9.2	9.3	8

Table 4.4-1. Average number of iteration for (1268,456) code decoded by min-sum (MS), min-sum with conditional correction step (MS-CCS) and with unconditional correction step (MS-UCS), unclipped min-sum (UN-MS) and unclipped log-ratio SP (UN-LRSP), without quantization and with different number of quantization bits.

Chapter 5

Digital Implementation of Min-sum Algorithm

5-1 Digital Implementation of Iterative Decoding Algorithms

This chapter addresses the implementation issues of iterative decoding algorithms for low-density parity-check (LDPC) codes. With the maturation of deep submicron technology, the possibility exists to implement those applications once thought too complex to fit into a single silicon die. The selected decoding algorithm, shown to have the best complexity-performance trade off among the iterative decoding algorithms, is a modified min-sum with an unconditional correction step.

In this chapter, the architectural and implementation issues related to building iterative decoders in VLSI will be investigated. The issues discussed herein include: computational hardware; memory requirements; the efficient mapping of this algorithm into architectures and its VLSI implementations.

5-2 Main Issues in Digital Implementation

With the introduction of smaller and faster transistors to the market, the computational capability of the digital VLSI implementation is increased. Digital implementation is robust and reliable, cheap and fast, and appears to be the most practical way to implement iterative decoding algorithms. Implementation of iterative decoding algorithms is also suggested in the analogue [24][19],[41] and optical [6] domains to increase speed and/ or decrease the power consumption, however these implementations face serious problems which make real implementation a particular challenge. The analogue approach must be viewed with concern as one faces problems in the areas of convergence, stability and robustness against various noise and temperature variations. It is not clear that this approach allows for application of codes with realistic sizes. The optical approach is fairly

immature and has a long way to progress before it is practical. Both these approaches lack flexibility and programmability, unlike digital VLSI which offers the advantages of both flexibility and programmability.

The main challenge faced when implementing the message passing algorithm for decoding of LDPC codes in a digital VLSI platform is the actual management of physical message passing. The issue which must be given special consideration is that of the bandwidth requirements for passing messages between the check-nodes and the bit nodes. The message bandwidth (MBW), measured in bits/sec of an LDPC code with length n and total number of edges E can be computed according to the following formula:

$$MBW = \frac{2q.N_{it}.ET}{n} \quad (5.1.1)$$

where q is the number of bits used to represent each message, N_{it} is the number of decoder iterations, T is the target coded throughput in bits/sec, and the factor of 2 is to count both variable and check messages. Difficult and varying challenges are faced, dependent upon the approach selected, to achieve the message passing bandwidth whether a serial decoder with hardware sharing approach is pursued or a parallel decoder architecture.

5-3 Different Design Approaches

Dependent upon the application, some constraints are imposed on the decoder design. For example, for high speed and real time applications, latency may be the main concern, whereas for mobile applications power consumption would be more important. There are two main approaches which can be followed in the design of LDPC codes decoders, one which optimizes for area and the other which optimizes for speed.

The inherent parallelism in iterative decoding algorithms can be used to reduce latency however at the cost of using more hardware. The relationship between latency and hardware is not straightforward, and is depend on the different design options chosen. The two main approaches for designing the LDPC decoder are that of use of a serial decoder with hardware sharing or parallel decoder architecture. A practical implementation can combine these two approaches.

5-3-1 Serial Approach: Optimization for Area

A hardware sharing LDPC decoder architecture consists of a small amount of hardware implementing either the check or bit node functionality, and a memory fabric in which to store the messages and to realize the graph connectivity. The shared memory fabric allows for concurrent transfers between functional units. This approach has been proposed in [41] and a generalized form of the hardware-sharing architecture is illustrated in Figure 35. The advantages of the hardware-sharing architecture are that the area of the decoder is minimized and it is readily configurable to support multiple block sizes and code rates. However, the throughput of the hardware-sharing architecture is limited by the need for the reuse of the computational units, and the requirement for multiple memory accesses required to perform each decoder iteration.

For example, using (5.1.1), to decode a code with a length of around one (1) Kbit and 3300 number of edges with 64 maximum iterations using 4-bit messages, and a throughput of one (1) Mb/s, requires a memory fabric with a reasonable bandwidth of 1.7 Gb/s. While it is possible to achieve this bandwidth with a single memory, higher throughputs are more problematic. For example, to achieve a coded throughput of one (1) Gb/s re-

quires 1.7 Tb/s of memory bandwidth, and this cannot be realized by making use of a single memory. Making use of multiple memories to achieve the required memory bandwidth is difficult because the essentially random or unstructured nature of the irregular LDPC graph resists a memory architecture which would allow both the bit-node and check-node messages to be efficiently addressed. High memory bandwidth requirements are also likely to translate into significant power dissipation. Another major issue associated with the hardware-sharing decoder architecture is the complexity of the control logic required for the representation of the graph connectivity and the corresponding address generation needed for fetching and storing of messages.

The hardware sharing decoder architecture seems best suited to low-throughput applications where area is the major concern.

Figure 35. Block diagram for serial decoder with hardware sharing architecture

5-3-2 Parallel Decoder Architecture

The iterative decoding algorithm maps extremely well to a parallel architecture in which the graph is directly instantiated in hardware. Figure 36 shows a parallel decoder architecture in which each of the bit and check nodes are realized in hardware and then routed together as defined by the LDPC code graph. To perform an iteration, the physical bit and check nodes are used once and messages are exchanged between them along the routed message wires. Very little control logic is required for the parallel architecture when compared to that required for serial hardware-sharing architecture, as the LDPC code graph is directly instantiated by the interconnection of the physical nodes. Higher throughput with a parallel decoder can be achieved by implementing a code with a larger block size, while maintaining the same clock frequency. The main challenge faced when implementing a parallel decoder architecture for LDPC codes is the interconnection of the physical nodes. However, by careful management of the physical design process, it is possible to solve routing congestion and its related timing problems.

In [8], a PLRA decoder for an irregular LDPC code with a block size of 1024 bits, rate $\frac{1}{2}$ was designed based on a graph with an average column weight of 3.25 and an average row weight of 6.5, with 3328 graph edges. The column weights used were 3, 6, 7 and 8, and the row weights used were 256 weight 6, and 256 weight 7 rows. It was shown that as the message passing algorithm iterates, the percentage of messages changing rapidly converges to a very small value which is determined by the input E_b/N_o . While the parallel decoder is necessarily more complex than the serial hardware-sharing architecture, the activity factor for the parallel architecture is very small, resulting in extremely low power dissipation.

Figure 36. Block diagram for parallel decoder architecture

The major drawbacks to the fully parallel decoder architecture are the relatively large area coupled with the inability to support multiple block sizes and code rates on the same core. However, for those applications which require high throughput and low power dissipation and which can tolerate a fixed code format, the parallel architecture proves very suitable.

5-3-3 Comparison with Turbo Code Implementations

The iterative nature of the decoding algorithms for turbo codes and turbo product codes presents significant implementation challenges. Each pass through the data block to perform one iteration requires the fetching, computation and storage of large amounts of

state information. Performing multiple iterations to achieve high coding gain, reduces throughput and increases power dissipation [20]. On the other hand, the graph representation of an LDPC code demonstrates that the computational dependencies for any node are dependent only on nodes of the opposing type. This permits the updating of all bit or check nodes in a block in a parallel manner which enables performing the iteration during shorter period, while also allowing a very high throughput. This is in stark contrast to the block-serial trellis dependencies inherent in turbo decoding.

In the next section we focus on the min-sum implementation for the (1268,456) irregular code.

5-4 Parallel Decoder Design and Optimization

Through simulations, it was found that 100 decoder iterations proved sufficient for good coding performance. It was also found that min-sum algorithm messages could be represented using only 4 bits, one sign bit to represent the parity and three magnitude bits to represent the reliability, without much penalty in coding performance. It was also shown that modified min-sum with an unconditional correction step can perform as well as sum-product. This section investigates the issues of min-sum implementation on a VLSI platform, by presenting a parallel architecture for decoding LDPC codes with a modified version of this algorithm. The architecture of the LDPC parallel decoders are modular, with each module containing physical bit-nodes and check-nodes as well as the I/O portion.

Every iteration can be considered as a data path, which starts from the bit-node edges, followed by check-node operation then bit-node operation and assignment checking, and

again bit-node at which point the outgoing message is ready on the edges for the next iteration. As shown in Figure 37, this data loop requires only a register for storage of the bit-nodes outgoing messages and the remainder of the path can be implemented using combinational logic. Thus, a reduction in decoder memory requirement to half is achieved compared to that required for a serial hardware sharing decoder.

The modular nature of the parallel architecture encourages us to first design the optimized physical check-node and bit-node modules. The bit-node module is designed to perform both bit-node operation and assignment checking.

5-4-1 Bit-node Design

The bit-node operation in the min-sum algorithm is performed by a series of addition. Overflow in the intermediate stages is an important issue which must be considered as it can produce serious errors. Consider a case in which three messages $-M_{\max}$, $M_{\max}$ and $M_{\max}$ have to add in a bit-node degree three to make an outgoing message. The order of addition can change the result significantly. If $M_{\max}$ is first added to $M_{\max}$ the result will be clipped to $M_{\max}$; then with addition of $-M_{\max}$, the overall results would be zero, significantly different than the correct result which would be $M_{\max}$.

There are different approaches which can be followed to overcome this problem. One is to use bigger adders however this increases complexity considerably due to the fact that in a fully parallel design each bit-node has a physical hardware and each addition has a physical adder. Use of bigger adders will also reduce the speed because of the serial nature of carry calculation. Using faster adders such as carry look ahead adder, Brent-Kung Adder, carry skip adder or carry select adder will also increase complexity.

Figure 37. Data path for fully parallel decoder

The other technique is a smart summation, means, each message will add to the message with the opposite sign if there is any. When considering the three messages $-M_{\max}$, $M_{\max}$ and $M_{\max}$ of the above example, if the messages with opposite signs are added first, the final result would be correct. It is noted that use of these techniques is not intended to prevent the overall overflow. Clipping due to overall overflow will improve the performance.

In min-sum, messages can be negative or positive. To add two messages with different signs, like subtraction, the smallest in absolute value has to be changed to its 1's complement; i.e., the bits are all inverted and one is added to the result. This meets the requirement to avoid the messages absolute values comparison before each addition. It does however increase complexity and adds delay.

Another approach was followed to design bit-node module using 2's complement scheme. In a 2's complement scheme, the positive numbers start with "0" and the negative numbers start with "1". A 3-bit example follows:

+3	011
+2	010
+1	001
0	000
-1	111
-2	110
-3	101
-4	100

A normal positive-number adder can be used to directly add numbers with opposite signs in the same way as one would be applied to numbers with the same sign. If the correct output would not overflow n -bits, then internal overflows will not cause an error. The overflow in the intermediate stages can be recovered automatically if the final result is in the range $[-M_{\max}, M_{\max}]$. For example, let x be a large number such that adding 5 to x will cause the overflow. If in the next immediate stage -6 is added to the result then another negative overflow will occur which will take the result back to $x-1$. The result is identical to that which would have ensued with no overflow. A Two's complement overflow will only occur when the absolute value of the final outgoing message is bigger than $|M_{\max}|$. In the case of overflow, the maximum positive becomes maximum negative or vice versa. Serious errors in bit-node operation will be caused and will change the sign of the outgoing message. A simple pure combinational overflow tracking system is designed to map the output to $-M_{\max}$ or $+M_{\max}$ in the case of negative or positive overflow in the final result. This approach results in a simple and fast bit-node module. All the incoming messages will add together and the sign of the results will be sent to check-nodes as the new estimate of that bit. The code for the bit-node degree two and

three was written in Verilog and synthesized with Synopsys Design Compiler. The synthesized circuit including scan test circuitry is shown in Figure 38 and Figure 39.

5-4-2 Check-node Design

The check-node operation in modified min-sum with unconditional correction step can be divided into three parts. The three parts are:

- a. find the minimum absolute value of the extrinsic message;
- b. apply the correction by reducing the absolute value of the outgoing message by one; and
- c. calculate the sign of the outgoing messages by XOR-ing signs of extrinsic incoming messages.

In addition, assignment checking is performed in the check-node by XOR-ing all new estimates (sign bits) coming from the bit-nodes. The result will show whether a check-node has been satisfied or not and the result will be sent to the I/O section.

All messages passed to the check-nodes are 2's complement but the comparison process in the 2's complement scheme is complex. Due to this complexity, the incoming values are converted to a sign-magnitude representation. After the check node operation, the outgoing messages are converted back to a 2's complement presentation and will be sent to the bit-nodes.

The process of finding the minimum of the extrinsic messages contains a series of compare and select operations. Check-nodes with the degree four and five were designed using Verilog HDL language and synthesized by Synopsys tools. The synthesized circuit

including scan test circuitry is shown in Figure 40 and Figure 41. In these designs, the minimum of extrinsic messages is computed separately for each of the outgoing messages. The other approach used for check-nodes with higher degrees is explained in section 5-5.

5-4-3 I/O Design

The parallel decoder can be considered as a three-block pipeline. While one block is being iteratively decoded, the next block is loaded into the decoder, and the previous block is read-out from the decoder. The number of registers depends on the application and how parallel data is loaded or unloaded. If w shift registers are used for loading or unloading, the length of the register has to be $l=n/w$, where n is the code block length.

The assignment checking for i_{th} iteration is done at the same time that check-node and bit-node operations for iteration $(i+1)_{th}$ are performed. The registers in the bit-nodes output would be updated with results of iteration $(i+1)_{th}$ if the result bit vector of iteration i_{th} was not a codeword.

5-4-4 Overview of Bottom-up Design

The parallel decoder structure is modular. The main modules are physical bit-nodes and check-nodes and the I/O. The I/O module has to be modified for each code although its main architecture remains the same. The physical bit-node and check-node modules are repeated n and $n-k$ times in the design for a (n,k) regular code, respectively. Consider the (1268,456) irregular LDPC code. The weights are 2 and 3 for 406 and 862 bit-nodes, and 4 and 5 for 662 and 150 check-nodes, respectively. So for example, module degree

two designed in the previous section is repeated 406 times in the decoder of this code. When the modules are ready, only the design of the modules connections remains.

5-4-5 Design characteristics and comparison

Table 5-4-5.1 shows the main characteristics of the design. Area, timing and power for each module are given by Synopsys design analyzer tool. Modules were designed using TSMC 0.18 μ technology libraries. The power calculated considering %50 activity for each net and they are not expressing real decoder power consumption. The 10ns clock interval and 20pF output capacitance is considered for each physical node.

With proposed clock, an iteration takes 20ns to complete, this means decoder can perform 5×10^7 iteration per second. Consider average iteration 50 requires for convergence, decoder can decode 10^6 codeword per second. Because design is modular, theoretically decoder can be built for any length of codeword. In such a case, I/O part would be the bottleneck of the design. For (1268,456) code throughput 1.2 Gbits/sec can be achieved. Consider the slack time, it might be possible to reduce the clock to 7ns. Then the throughput increases to 1.8 Gbits/sec. This is 80% more than sum-product implemented in [8].

	check-node degree five	check-node degree four	bit-node degree two	bit-node degree three
Total cell area	11912 μ^2	8541.8 μ^2	4325 μ^2	7021 μ^2
Calculation time	7.86ns	5.05ns	3.81ns	5.24ns
Total dynamic power	21.63mW	21.7 mW	11.95 mW	17.17 mW

Table 5-4-5.1 Area, timing and power characteristics of main physical modules.

5-5 Decoder Interconnections

In a fully parallel architecture, the modules connections is the bottle neck of the physical design which reduces the utilization factor considerably. In [8], the parallel architecture decoder for an irregular LDPC code with block size 1024 bits, rate-1/2, was designed based on a graph with an average column weight of 3.25 and an average row weight of 6.5. Using software optimization and five level metal, utilization of 50 % on a 7.5mm × 7.0mm die was achieved (0.16 CMOS technology). The power dissipation of the decoder is determined by the switching activity of the large number of very long message nets. The activity of these nets also limits the clock frequency of the decoder to 64 MHz.

5-6 Special Consideration for High Degree Check-nodes

The high degree nodes appear in the structure of all high rate codes and can even exist in the graph of some irregular low or medium rate LDPC codes. There are two architectures for implementation of check-nodes. In the first architecture, called direct architecture, each check-node outgoing message's absolute value is calculated by finding the minimum of the extrinsic messages absolute values. A second architecture for high degree check-nodes has been proposed and named two stages architecture. In this architecture, the two smallest messages' absolute values are found, and then the smallest value is sent as the absolute value of all outgoing messages with the exception of the edge of the smallest message which is sent the second smallest message. Table 5-5.1 shows the number of compare and select operations and the time of calculation for these architectures.

The direct architecture is almost twice faster than two stages architecture. For high degree check-nodes, this higher speed comes with the cost of much higher area and power consumption. On the other hand in direct architecture more fanout is required for bit-node output which increases the power consumption and/or the delay.

check-node degree	Direct architecture compare and select number	Two stages architecture compare and select number	Direct architecture calculation time	Two stages architecture calculation time
d_c	$d_c(d_c - 2)$ (c-s)	$2d_c - 3$ (c-s)	$\lceil \sqrt[3]{d_c - 1} \rceil (t_c)$	$\lceil \sqrt[3]{d_c} \rceil + \lceil \sqrt[3]{d_c - 1} \rceil (t_c)$
3	3	3	1	2
4	8	5	2	4
5	15	7	2	5
6	24	9	3	6
7	35	11	3	6
8	48	13	3	6
9	63	15	3	7
10	80	17	4	8

Table 5-6.1. Number of compare and selects and the longest path calculation time required for two check-node architectures; calculation time is expressed based on time of one q-bit word comparison t_c

Figure 38. Synthesized circuit for bit-node degree 2

Figure 39. Synthesized circuit for bit-node degree 3

Figure 40. Synthesized circuit for check-node degree 4

Figure 41. Synthesized circuit for check-node degree 5

Conclusion

In the search of proper iterative decoding algorithms for digital implementation, min-sum shows good complexity-performance trade-off. This algorithm also shows less sensitivity to quantization error and can perform close to unclipped min-sum with four quantization bits. The proper choice of clipping threshold will improve min-sum performance and this balance part of the degradation due to quantization error.

The performance gap between min-sum and sum-product is due the check-node operation dissimilarity. The min-sum check-node output error can be expressed as a function of inputs differences. This function was used to correct the check-node output. As the VLSI implementation of this correction function is complex two approximations of correction function were introduced in this thesis. Applying these correction factors improve min-sum performance significantly. Min-sum can perform as well as sum-product with less complexity when the check-node operation is modified with unconditional correction step. This modified version of min-sum is believed to be the best iterative decoding algorithm for implementation on digital VLSI platform.

Parallel architecture is suggested for decoder implementation in order to achieve higher throughput. To avoid unwanted overflow errors two's complement numbering scheme is chosen to represent negative numbers in bit-nodes. An alternative configuration is introduced for high degree check-nodes which decrease hardware requirement considerably.

Future research

In this research only uniform quantization scheme was considered and the effect of non-uniform quantization on dynamic and performance of the algorithms has been left for future research.

A mix-signal solution may reduce the area and/or iteration time. The first step can be replacing the check-node digital block with its mix-signal alternative. The new block contains two multiplex chains for routing and an analog comparator. Only simple digital to analog converter is required and comparators just determine the two smallest messages.

The optimum floor planning also requires a comprehensive research. The question is how divide the nodes into predetermined number of physical blocks to minimize the interconnection on the chip. Reducing the number and length of interconnections have a great impact on decoder size and speed.

Power reduction is another complex issue that needs a comprehensive study specially for mobile applications. The main component of decoder total power is nets switching power. Determining the activity of all the nets is required for computing decoder nets switching power. The power consumption of different blocks can be compared when nets activities for each design is determined.

References

- [1] "Concatenated coding techniques and iterative decoding: sailing toward channel capacity," special issue of *IEEE J. Selected Areas Comm.*, vol. 16, no. 2, Feb. 1998.
- [2] "Codes on graphs and iterative algorithms," special issue of *IEEE Trans. Inform. Theory*, vol. 47, no. 2, Feb. 2001.
- [3] "The turbo principle: from theory to practice I," special issue of *IEEE J. Selected Areas Comm.*, vol. 19, no. 5, May 2001.
- [4] "The turbo principle: from theory to practice II," special issue of *IEEE J. Selected Areas Comm.*, vol. 19, no. 9, Sept. 2001.
- [5] A. Anastasopoulos, "A Comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proc. IEEE Glob. Comm. Conf. (Globcom 01)*, vol. 2, pp. 1021-1025, 2001.
- [6] A.H. Banihashemi and S. Hemati, "Decoding in Optics," accepted for presentation at *2002 IEEE International Symposium on Information Theory*, Lausanne, Switzerland, June 30 - July 5, 2002.
- [7] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding," in *Proc. IEEE Int. Conf. Comm. (ICC)*, pp. 1067-1070, 1993.
- [8] A. J. Blanksby, C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, pp. 404-412, March 2002

- [9] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity-check codes," *IEEE Tran. Comm.*, vol. 50, no. 3, pp. 406-414, March 2002.
- [10] J. Chen and M. Fossorier, "Density evolution for two improved BP-based decoding algorithms of LDPC codes," *IEEE Comm. Lett.*, vol. 6, no. 5, pp. 208-210, May 2002.
- [11] J. Chen and M. Fossorier, "Decoding low-density parity-check codes with normalized APP-based algorithm," in Proc. *IEEE Glob. Comm. Conf. (Globcom 01)*, vol. 2, pp. 1026-1030, 2001.
- [12] S.Y Chung, G.D. Forney, Jr., T.J. Richardson, R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Comm. Lett.*, vol. 5, no. 2, pp. 58-60, Feb 2001.
- [13] H. Dawid and H. Meyr, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding," *IEEE Sympos. OnPIMRC'95*, Germany, pp. 193-197, Sep. 1995.
- [14] E. Eleftheriou, T. Mittelholzer, A. Dholakia, "Reduced-complexity decoding algorithm for low-density parity-check codes," *Electron. Lett.*, vol. 37 no. 2, pp 102-104, Jan. 2001
- [15] T. Etzion, A. Trachtenbeg, A. Vardy, "Which codes have cycle-free Tanner graphs?," *IEEE Trans. Inform. Theory*, vol 45, no. 6, pp. 2173 -2181, Sep. 1999.
- [16] G. D. Forney, Jr., "On iterative decoding and the two-way algorithm," in Proc. *Int. Symp. on Turbo Codes and Related Topics*, Brest, France, pp. 12-25, Sept. 1997.

- [17] M. Fossorier and M. Mihaljevic Imai, "H. Reduced complexity iterative decoding of low-density parity-check codes based on belief propagation Communications," *IEEE Trans.* vol. 47, pp. 673 -680, May. 1999.
- [18] R. G. Gallager, "Low density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, pp. 21-28, Jan. 1962.
- [19] J. Hagenauer, M. Winklhofer, "The analog decoder," *IEEE Int. Symp. on Inform. Theory*, pp. 145, 1998.
- [20] S. Hong and W. Stark, "Design and implementation of a low complexity VLSI turbo-code decoder architecture for low energy mobile wireless communications," *J. VLSI Signal Processing*, vol. 24, pp. 43-57, 2000
- [21] F. Kschischang and B. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 219-230, Feb. 1998.
- [22] F. Kschischang, B. Frey, A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no.2 pp. 498-519, Feb. 2001.
- [23] B. Levine, R. Taylor, and H. Schmit, "Implementation of near Shannon limit error-correcting codes using reconfigurable hardware", *IEEE Symp. On FCCM*, Los Alamitos, pp. 217-226, Apr. 2000.
- [24] A. H. Loeliger, F. Lustenberger, M. Helfenstein, F. Tarkoy, "Probability propagation and decoding in analog VLSI," *IEEE Trans. Inform. Theory*, vol. 47, pp. 837-843, Feb. 2001.

- [25] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *Proc. 30th Annu. ACM Symp. Theory of Computing*, 1998, pp. 249–258.
- [26] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity-check codes," *Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, 1996.
- [27] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.
- [28] Y. Mao, A. H. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," *Proc. IEEE Int. Conf. on Comm. (ICC)*, vol. 1, pp. 41–44, 2001.
- [29] R. McEliece, D. MacKay, and J. Cheng, "Turbo decoding as an instance of Pearl's 'Belief Propagation' algorithm," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 140–152, Feb. 1998.
- [30] T. Mittelholzer, A. Dholakia, E. Eleftheriou, "Reduced-complexity decoding of low density parity-check codes for generalized partial response channels," *IEEE Trans. Magnetics*, vol. 37 no. 2, pp. 721–728, March 2001.
- [31] G. Moore, "Cramming more components onto integrated circuits" *Electronics*, vol. 38, no. 8, 1965.
- [32] L. Ping, W. K. Leung, "Decoding low-density parity-check codes with finite quantization bits," *IEEE Comm. Lett.*, vol. 4, pp. 62–64, No. 2, Feb. 2000.
- [33] T. J. Richardson and R. Urbanke, "The capacity of low-density paritycheck codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb. 2001.

- [34] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 638–656, Feb. 2001.
- [35] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, July and October, 1948.
- [36] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1710–1722, Nov. 1996.
- [37] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533–547, Sept. 1981.
- [38] A. J. Viterbi, "Wireless Digital Communication: A View Based on Three Lessons Learned," *IEEE Commun. Mag.*, vol. 29, no. 9, pp. 33–36, Sept. 1991.
- [39] Y. Weiss and W. T. Freeman, "On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 736–744, Feb. 2001.
- [40] E. J. Weldon, "Difference-set cyclic codes," *Bell Syst. Tech. J.*, vol. 45, pp. 1045–1055, Sept. 1966.
- [41] C. Winstead, C. Myers, C. Schlegel, R. Harrison, "Analog decoding of product codes," *Proc. Inform. Theory Work shop*, pp. 131–133, 2001.
- [42] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magnetics*, vol. 37, no. 2, pp. 748–55, March 2001.
- [43] X. Yu Hu, E. Eleftheriou, D. M. Arnold, A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *Proc. IEEE Glob. Comm. Conf. (Globcom 01)*, vol. 2, pp. 1036–1036E, 2001.

[44] <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html#s14>

Appendix 1. Number of iterations Histogram

Figure 42. Histogram of the number of iterations for 4-bit min-sum at $E_b/N_0=1\text{dB}$ for (1268,456) irregular code.

Figure 43. Histogram of the number of iterations for 4-bit min-sum at $E_b/N_0=1.5\text{dB}$ for (1268,456) irregular code.

Figure 44. Histogram of the number of iterations for 4-bit min-sum at $E_b/N_o=2$ dB for (1268,456) irregular code.

Appendix 2. Clipping and quantization results for other codes

Figure 45. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for (8000,4000)<3,6> regular code at $E_b/N_o=1.7$ dB.

Figure 47. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for (8000,4000)<3,6> regular code at $E_b/N_o=1.9$ dB.

Figure 46. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for (8000,4000)<3,6> regular code at $E_b/N_o=1.8$ dB.

Figure 48. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for (8000,4000)<3,6> regular code at $E_b/N_o=2$ dB

Figure 49. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(273,191)<3,10>$ regular code at $E_b/N_0=2$ dB.

Figure 51. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(273,191)<3,10>$ regular code at $E_b/N_0=3$ dB.

Figure 50. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(273,191)<3,10>$ regular code at $E_b/N_0=2.5$ dB.

Figure 52. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(273,191)<3,10>$ regular code at $E_b/N_0=3.5$ dB.

Figure 53. The effect of clipping threshold cth on error performance of unquantized min-sum with clipping for $(273,191)<3,10>$ regular code at $E_b/N_0=4$ dB.

Figure 54. BER curves for the (8000,4000) code, decoded with unclipped min-sum and unquantized min-sum with optimum clipping $c_{th} = 1.25$

Figure 55. BER curves for the (273,191) code, decoded with unclipped min-sum and unquantized min-sum with optimum clipping for each E_b/N_0 . ($c_{th}=1.5, 1.5, 2.5, 3.25, 3, 2.5$ are used for $E_b/N_0=2, 2.5, 3, 3.5, 4, 4.5$, respectively).

Figure 56. The effect of clipping threshold cth on performance of quantized min-sum for (8000,4000) code at $E_b/N_0=1.7$ dB.

Figure 57. The effect of clipping threshold cth on performance of quantized min-sum for (8000,4000) code at $E_b/N_0=1.8$ dB.

Figure 58. The effect of clipping threshold cth on performance of quantized min-sum for (8000,4000) code at $E_b/N_0=1.9$ dB.

Figure 59. The effect of clipping threshold cth on performance of quantized min-sum for (8000,4000) code at $E_b/N_0=2$ dB.

Figure 60. The effect of clipping threshold cth on performance of quantized min-sum for (273,191) code at $E_b/N_0=2$ dB.

Figure 61. The effect of clipping threshold cth on performance of quantized min-sum for (273,191) code at $E_b/N_0=2.5$ dB.

Figure 62. The effect of clipping threshold cth on performance of quantized min-sum for (273,191) code at $E_b/N_o=3$ dB.

Figure 63. The effect of clipping threshold cth on performance of quantized min-sum for (273,191) code at $E_b/N_o=3.5$ dB.

Figure 64. The effect of clipping threshold cth on performance of quantized min-sum for (273,191) code at $E_b/N_o=4$ dB.

Figure 65. BER curves for the (8000,4000) LDPC code decoded by unclipped min-sum, unquantized min-sum with clipping and quantized min-sum for different quantization bits ($cth=1.25$).

Figure 66. BER curves for the (273,191) LDPC code decoded by unclipped min-sum, unquantized min-sum with clipping and quantized min-sum for different quantization bits ($cth=1.5$).

Figure 67. BER curves for (8000,4000) code, decoded by unclipped min-sum, modified unclipped min-sum with exact simplified correction factor and sum-product.

Figure 68. BER curves for (273,191) code, decoded by unclipped min-sum, modified unclipped min-sum with exact simplified correction factor and sum-product.

Appendix 3. Message error rate

In this appendix the message error rate (MER) is presented.

Figure 69. MER curves for (8000,4000) code decoded by unclipped min-sum, unquantized min-sum with clipping, unclipped min-sum with conditional and unconditional correction step, 4-bit min-sum, 4-bit min-sum with conditional and unconditional correction step and sum-product.

Figure 70. MER curves for (273,191) code decoded by unclipped min-sum, unquantized min-sum with clipping, 4-bit min-sum, 4-bit min-sum with conditional and unconditional correction step and sum-product.

Figure 71. MER curves for (1286,456) code decoded by unclipped min-sum, unquantized min-sum with clipping, 4-bit min-sum, 4-bit min-sum with conditional and unconditional correction step and sum-product.

Appendix 4. Conditional correction step results for other codes

As it can be observed conditional correction step improves min-sum performance on the other codes too. The optimum step size however is different.

Figure 72. BER for (8000,4000) regular code at $E_b/N_0=1.7\text{dB}$ decoded by unclipped min-sum with conditional correction step for different $d_{\text{step}}=y_{\text{step}}$

Figure 73. BER for (273,191) regular code at $E_b/N_0=3.5\text{dB}$ decoded by unclipped min-sum with conditional correction step for different $d_{\text{step}}=y_{\text{step}}$

Figure 74. BER curves for (273,191) code, decoded by unclipped min-sum, modified unclipped min-sum with conditional correction step $d_{\text{step}}=0.2$ and $y_{\text{step}}=0.2$ and sum-product.

Figure 75. BER curves for (8000,4000) code, decoded by unclipped min-sum, modified unclipped min-sum with conditional correction step $d_{\text{step}}=0.3$ and $y_{\text{step}}=0.3$ and sum-product.

Figure 76. BER curves for (8000,4000) code, decoded by 4-bit modified min-sum without correction and with conditional correction step with different $d=d_{\text{step}}$ and $y=y_{\text{step}}$. The sum-product curve is also given for comparison.

Figure 77. BER curves for (273,191) code, decoded by 4-bit modified min-sum without correction and with conditional correction step with different $d=d_{\text{step}}$ and $y=y_{\text{step}}$. The sum-product curve is also given for comparison.

Figure 78. BER curves for (8000,4000) code, decoded by unclipped min-sum and 4-bit min-sum with and without conditional correction step . ($d_{\text{step}}=0.3$ and $y_{\text{step}}=0.3$ for unclipped and $d_{\text{step}}=y_{\text{step}}=2$ for 4-bit version are chosen) The sum-product curve is also given for comparison.

Figure 79. BER curves for (273,191) code, decoded by unclipped min-sum, 4-bit min-sum, modified unclipped min-sum and modified 4-bit min-sum. The sum-product result is also given for comparison.

Appendix 5. Conditional correction step results for 5 and 6 quantization bits

In this appendix the effect of conditional correction step on quantized min-sum algorithm with 5 and 6 quantization bits is shown. The results similar to 4-bit are achieved but more quantization bits let more resolution on choosing step sizes.

Figure 80. BER curves for (8000,4000) code, decoded by 5-bit modified min-sum without correction and with conditional correction step with different $d=dstep$ and $y=ystep$. The sum-product curve is also given for comparison.

Figure 81. BER curves for (8000,4000) code, decoded by 6-bit modified min-sum without correction and with conditional correction step with different $d=d_{\text{step}}$ and $y=y_{\text{step}}$. The sum-product curve is also given for comparison.

Figure 82. BER curves for (273,191) code, decoded by unclipped min-sum, modified unclipped min-sum, 5-bit min-sum and 5-bit min-sum with optimum conditional correction step $d_{\text{step}} = 3, 3, 4, 3, 2, 3$ and $y_{\text{step}} = 1, 1, 1, 2, 2, 1, 2$, for $E_b/N_0 = 2, 2.5, 3, 3.5, 4, 4.5$, respectively. The sum-product result is also given for comparison.

Figure 83. BER curves for (273,191) code, decoded by unclipped min-sum, modified unclipped min-sum, 6-bit min-sum and 6-bit min-sum with optimum conditional correction step $d_{\text{step}} = 6, 5, 8, 4, 4, 4$ and $y_{\text{step}} = 1, 1, 1, 2, 3, 4, 2$, for $E_b/N_0 = 2, 2.5, 3, 3.5, 4, 4.5$, respectively. The sum-product result is also given for comparison.

Figure 84. BER curves for (1268,456) code, decoded by unclipped min-sum, modified unclipped min-sum, 5-bit min-sum and 5-bit min-sum with optimum conditional correction step for each E_b/N_0 ($d_{\text{step}} = 3, 3, 4, 4$ and $y_{\text{step}} = 3, 2, 2, 2$ for $E_b/N_0 = 1, 1.5, 2, 2.5$, respectively). The sum-product is also given for comparison.

Figure 85. BER curves for (1268,456) code, decoded by unclipped min-sum, modified unclipped min-sum, 6-bit min-sum and 6-bit min-sum with optimum conditional correction step for each E_b/N_0 ($d_{\text{step}} = 8, 8, 8, 8$ and $y_{\text{step}} = 3, 5, 3, 2$ for $E_b/N_0 = 1, 1.5, 2, 2.5$, respectively). The sum-product is also given for comparison.

Appendix 6. Unconditional correction step results for 5 and 6 quantization bits

Figure 86. BER curves for (273,191) code, decoded by 5-bit min-sum and 5-bit min-sum with different unconditional correction step. The sum-product curve is also given for comparison.

Figure 87. BER curves for (273,191) code, decoded by 6-bit min-sum and 6-bit min-sum with different unconditional correction step. The sum-product curve is also given for comparison.

Figure 88. BER curves for (8000,4000) code, decoded by 5-bit min-sum and 5-bit min-sum with different sizes of unconditional correction step . The sum-product curve is also given for comparison.

Figure 89. BER curves for (8000,4000) code, decoded by 6-bit min-sum and 6-bit min-sum with different sizes of unconditional correction step. The sum-product curve is also given for comparison.

Figure 90. BER curves for (1268,456) code, decoded by 5-bit min-sum with different sizes of unconditional correction step. The sum-product curve is also given for comparison.

Figure 91. BER curves for (1268,456) code, decoded by 6-bit min-sum with different sizes of unconditional correction step. The sum-product curve is also given for comparison.

Appendix 7. Proof of associativity for applying correction factor

Consider m is the absolute value of the smallest message (min-sum output) and m_1 and m_2 are absolute values of two extrinsic messages with differences d_1, d_2 with respect to m , respectively. For the case that simplified correction factor is first applied to m_1 , the final message m_f is:

$$m_f = \{m + f(d_1)\} - f(d_2 + \log(1 + e^{-d_1}))$$

Where $d_2 + \log(1 + e^{-d_1})$ is the difference of m_2 and the message achieved after applying simplified correction factor to m and m_1 .

$$m_f = \{m - \log(1 + e^{-d_1})\} - \log(1 + \exp[-(d_2 + \log(1 + e^{-d_1}))])$$

$$m_f = \{m - \log(1 + e^{-d_1})\} - \log(1 + e^{-d_2} \exp[-\log(1 + e^{-d_1})])$$

$$m_f = \{m - \log(1 + e^{-d_1})\} - \log(1 + e^{-d_2} \exp[\log(\frac{1}{1 + e^{-d_1}})])$$

$$m_f = m - \log(1 + e^{-d_1}) - \log(1 + \frac{e^{-d_2}}{(1 + e^{-d_1})})$$

$$m_f = m - \log\left\{(1 + e^{-d_1})\left(1 + \frac{e^{-d_2}}{(1 + e^{-d_1})}\right)\right\}$$

$$m_f = m - \log(1 + e^{-d_1} + e^{-d_2})$$

In the case that correction first is applied to m and m_2 and then to m_1 , m_f would be the same. Using the same discussion for three messages we can write:

$$m_f = \{m + f(d_1) - f(d_2 + \log(1 + e^{-d_1}))\} - f(d_3 + -\log(1 + e^{-d_1} + e^{-d_2}))$$

$$m_f = \{m - \log(1 + e^{-d_1} + e^{-d_2})\} - \log(1 + \exp[-(d_3 + \log(-\log(1 + e^{-d_1} + e^{-d_2}))])$$

$$m_f = \{m - \log(1 + e^{-d_1} + e^{-d_2})\} - \log(1 + e^{-d_3} \exp[-\log(1 + e^{-d_1} + e^{-d_2})])$$

$$m_f = \{m - \log(1 + e^{-d_1} + e^{-d_2})\} - \log(1 + e^{-d_3} \exp[\log(\frac{1}{1 + e^{-d_1} + e^{-d_2}})])$$

$$m_f = m - \log(1 + e^{-d_1} + e^{-d_2}) - \log(1 + \frac{e^{-d_3}}{1 + e^{-d_1} + e^{-d_2}})$$

$$m_f = m - \log\{(1 + e^{-d_1} + e^{-d_2})(1 + \frac{e^{-d_3}}{1 + e^{-d_1} + e^{-d_2}})\}$$

$$m_f = m - \log(1 + e^{-d_1} + e^{-d_2} + e^{-d_3})$$

This is obvious that the final result would be the same if the order of applying the correction is changed.

Appendix 8. Average number of iterations for other codes

The average number of iterations for quantized versions of min-sum algorithm for (273,191) and (8000,4000) codes, are as follows:

E_b/N_0 (dB)	6-bit MS	5-bit MS	4-bit MS	3-bit MS	UN- MS	UN- LRSP
2	14.7	16.1	18.6	32	14.5	14
2.5	12.1	13.2	15.8	28	12	11.2
3	8.9	9.8	12.2	19	9	8.7
3.5	5.2	5.9	7.1	14	5.1	4.5
4	4.7	5.3	6.6	12	4.5	4.1
4.5	4	4.7	5.3	9	4	3.9

Table 8-1. Average number of iterations for (273,191) code decoded by min-sum (MS), unclipped min-sum (UN-MS) and unclipped log-ratio SP (UN-LRSP), without quantization and with different number of quantization bits.

E_b/N_0 (dB)	6-bit MS	5-bit MS	4-bit MS	3-bit MS	UN- MS	UN- LRSP
1.7	32	40.1	44.2	N/A	46	22.6
1.8	28.3	28.9	29.2	N/A	31.8	17.1
1.9	20.8	21.2	23.8	N/A	27.6	12.4
2	16.4	16.7	17.9	33	22.3	11.3
2.1	13.9	14.7	15.3	28	17.7	9.2

Table 8-2. Average number of iterations for (8000,4000) code decoded by min-sum (MS), unclipped min-sum (UN-MS) and unclipped log-ratio SP (UN-LRSP), without quantization and with different number of quantization bits. The average number of iterations for 3-bit min-sum at low SNR values is not given due to the fact that none of the codewords converges at this range of SNR.