AREA AND ENERGY EFFICIENT VLSI ARCHITECTURES FOR LOW-DENSITY PARITY-CHECK DECODERS USING AN ON-THE-FLY COMPUTATION

A Dissertation

by

KIRAN KUMAR GUNNAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2006

Major Subject: Computer Engineering

AREA AND ENERGY EFFICIENT VLSI ARCHITECTURES FOR LOW-DENSITY PARITY-CHECK DECODERS USING AN ON-THE-FLY COMPUTATION

A Dissertation

by

KIRAN KUMAR GUNNAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Co-Chairs of Committee,	Gwan Choi
	Scott Miller
Committee Members,	Jiang Hu
	Duncan Walker
Head of Department,	Costas Georghiades

December 2006

Major Subject: Computer Engineering

ABSTRACT

Area and Energy Efficient VLSI Architectures for Low -Density Parity-Check Decoders
Using an On-the-Fly Computation. (December 2006)

Kiran Kumar Gunnam, M.S., Texas A&M University

Co-Chairs of Advisory Committee: Dr. Gwan Choi
Dr. Scott Miller

The VLSI implementation complexity of a low density parity check (LDPC) decoder is largely influenced by the interconnect and the storage requirements. This dissertation presents the decoder architectures for regular and irregular LDPC codes that provide substantial gains over existing academic and commercial implementations. Several structured properties of LDPC codes and decoding algorithms are observed and are used to construct hardware implementation with reduced processing complexity. The proposed architectures utilize an on-the-fly computation paradigm which permits scheduling of the computations in a way that the memory requirements and re-computations are reduced. Using this paradigm, the run-time configurable and multi-rate VLSI architectures for the rate compatible array LDPC codes and irregular block LDPC codes are designed. Rate compatible array codes are considered for DSL applications. Irregular block LDPC codes are proposed for IEEE 802.16e, IEEE 802.11n, and IEEE 802.20. When compared with a recent implementation of an 802.11n LDPC decoder, the proposed decoder reduces the logic complexity by 6.45x and memory complexity by 2x for a given data throughput. When compared to the latest reported multi-rate decoders, this decoder design has an area efficiency of around 5.5x and energy efficiency of 2.6x for a given data throughput. The numbers are normalized for a 180nm CMOS process.

Properly designed array codes have low error floors and meet the requirements of magnetic channel and other applications which need several Gbps of data throughput. A high throughput and fixed code architecture for array LDPC codes has been designed. No modification to the code is performed as this can result in high error floors. This parallel decoder architecture has no routing congestion and is scalable for longer block lengths. When compared to the latest fixed code parallel decoders in the literature, this design has an area efficiency of around 36x and an energy efficiency of 3x for a given data throughput. Again, the numbers are normalized for a 180nm CMOS process. In summary, the design and analysis details of the proposed architectures are described in this dissertation. The results from the extensive simulation and VHDL verification on FPGA and ASIC design platforms are also presented.

To my family.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Gwan Choi, for his financial support and encouragement for my research. He supported me in all the difficult situations where I needed help. I would like to thank Dr. Scott Miller for his time in serving on my committee. His suggestions made me focus exclusively on LDPC decoder architectures though initially I set out to work on a conglomeration of different topics. Dr. Mark Yeary has been very helpful and he spent a lot of time improving my papers. I would also like to thank Dr. Duncan Walker who suggested that I look into scalabilty issues of the decoder architectures. I would like to thank Dr. Jiang Hu for his time and suggestions to improve the presentation aspects of my research.

I would like to take this opportunity to express my thanks to Intel, Schlumberger and Starvision Technologies for supporting my research. Dr. James Ochoa and Mr. Mike Jacox of Starvision Technologies in conjunction with Dr. Gwan Choi and Dr. John Junkins have supported my PhD program.

Several students and other people at Texas A&M helped me in my research work also. Thanks to Weihuang Wang, in particular, for working on the matlab simulation model for my architecture on the layered decoding for array codes and on the verification of some of the HDL modules. In addition, he spent several weeks with me working on writing the paper. Most of the figures presented in this dissertation were drawn by him. I appreciate the help of Mr. Abhiram Prabhakar and Mr. Euncheol Kim in providing the useful reviews for some of my work. Several members of the computer engineering group helped also. In

addition, Ms. Linda Crenwelge, associate editor of Choice magazine, provided me help with the editing of my papers. I am thankful for the additional staff at Texas A&M University for assisting in my degree program.

Several other researchers and professors outside Texas A&M University provided feedback on my work. Dr. Jinghu Chen of Qualcomm provided a review on one of my papers and supplied me with his software on density evolution. Dr. Zhongfeng Wang of Oregon State University provided several suggestions to improve the presentaion of the papers. In addition, I received several anonymous reviewers' comments as part of my paper submissions. Those suggestions are incorporated into the papers, as well as, into the dissertation.

Dr. Roger Robbins has been my career mentor for the last four years. His advice helped me see my career and life more clearly. Kanu Chadha gave his time to listen to me and to offer suggestions. My lovely wife, Anu, has supported me in many more ways than meet the eye. She did the difficult task of completing 36 credit hours in one year at Texas A&M for her masters degree course requirements while taking care of different things at home. I would like to thank my parents, and brother Ramakrishna, for their constant support and encouragement through every major decision in my life.

TABLE OF CONTENTS

Page

ABSTRACT iii

DEDICATION....... v

ACKNOWLEDGMENTS vi

TABLE OF CONTENTS viii

LIST OF FIGURES xi

LIST OF TABLES....... xiii

CHAPTER

I INTRODUCTION 1

1.1. Motivation 1
1.2 Problem Overview 5
1.3. Main Contributions 6

II QUASI-CYCLIC LOW-DENSITY PARITY-CHECK CODES AND DECODING....... 11

2.1.Introduction 11
2.2.Cyclotomic Cosets 12
2.3.Array LDPC Codes 13
2.4.Rate-compatible Array LDPC Codes 14
2.5.Irregular Quasi-Cyclic LDPC Codes (Block LDPC codes) 15
2.6.Irregular QC-LDPC Codes for Other Wireless Standards(802.11n and 802.20)....... 16
2.7 Two Phase Message Passing (TPMP) and Decoding of LDPC 16
2.8 Turbo Decoding Message Passing (TDMP) or Layered Decoding 18

III MULTI-RATE TPMP ARCHITECTURE FOR REGULAR QC-LDPC CODES.. 19

3.1 Introduction 19
3.2.Block Message Independence Property for Regular QC-LDPC Codes 20
3.3.Architecture 23

CHAPTER Page

3.4.Performance Comparison ... 29
3.5.FPGA Implementation Results ... 30
3.6.ASIC Implementation Results ... 31

IV VALUE-REUSE PROPERTIES OF OMS AND MICRO-ARCHITECTURES FOR CHECK NODE UNIT BASED ON OMS ... 35

4.1.Value-reuse Properties ... 35
4.2.Serial CNU for OMS ... 36
4.3.Parallel CNU ... 38

V FIXED CODE TPMP ARCHITECTURE FOR REGULAR QC-LDPC CODES ... 41

5.1.Introduction ... 41
5.2.Reduced Message Passing Memory and Router Simplification ... 42
5.3.Check Node Unit Micro-architecture ... 43
5.4.Architecture ... 45
5.5.Results and Performance Comparsion ... 48

VI MULTI-RATE TDMP ARCHITECTURE FOR RATE-COMPATIBLE ARRAY LDPC CODES ... 53

6.1.Introduction ... 53
6.2.Background ... 54
6.3.TDMP for Array LDPC ... 57
6.4.Value-reuse Properties of OMS ... 59
6.5.Multi-rate Architecture Using TDMP and OMS ... 61
6.6.Implementation Results and Discussion ... 68
6.7.Conclusion ... 75

VII MULTI-RATE TDMP ARCHITECTURE FOR IRREGULAR QC-LDPC CODES ... 81

7.1.Introduction ... 81
7.2.LDPC Codes and Decoding ... 82
7.3.Multi Rate Decoder Architecture Using TDMP and OMS ... 85
7.4.Discussion and Implementation Results ... 93
7.5.Conclusion ... 100

CHAPTER Page

VIII FIXED CODE TDMP ARCHITECTURE FOR REGULAR QC-LDPC CODES..........105

8.1.Introduction 105
8.2.Parallel Architecture Using TDMP and OMS 105
8.3.ASIC Implementation Results 108
8.4.Conclusion 111

IX SUMMARY 113

9.1 Key Contributions 113
9.2 Future Work 117
9.3 Conclusion 118

REFERENCES 120

VITA 126

LIST OF FIGURES

FIGURE Page

1.1 Block diagram of a digital communication system 1

3.1 Block diagram of the decoder architecture... 25

3.2 Pipeline of the decoder 26

3.3 Comparison of architecture for (3,k=6,…30) rate compatible array codes of up to length 1830……..………………………………………………….………...30

4.1 Serial CNU for OMS using value-reuse property 36

4.2 Finder for the two least minimum in CNU (a) binary tree to find the least minimum.……………………………………………………………………..39

4.3 Parallel CNU based on value-reuse property of OMS……...…………..…………40

5.1 Check node processing unit, Q: variable node message, R: check node message.……………………………………………………….............................. 44

5.2 Architecture ……………………………………………………………………..45

5.3 Pipeline ……………………………………………………………………..……47

5.4 Results comparison with M. Karkoot et al.,[37] and T. Brack, et al., [41] ….…….50

6.1 Serial CNU for OMS using value-reuse property.…………………………………60

6.2 LDPC Decoder using layered decoding and OMS …………………..…………62

6.3 Block serial processing and 3-stage pipelining for TDMP using OMS a) detailed diagram b) simple diagram 66

FIGURE Page

6.4 . (a) Bit error rate performance of the proposed TDMP decoder using OMS(j=3,k=6,p=347,q=0) Array LDPC code of length N=2082 and (j=5,k=25,p=61,q=0) array LDPC code of length N=1525.. ....…74

7.1 Operation of CNU (a) no time-division multiplexing (b) time-division multiplexing…………………………………………………………………….…87

7.2 Multi-rate LDPC decoder architecture for block LDPC codes…..……………..........88

7.3 Three-stage pipeline of the multi-rate decoder architecture ……………………….89

7.4 Out of order processing for R_{new} selection ……………………………….……….89

7.5 Proposed master-slave router to support different cyclic shifts that arise due to a wide range of expansion factors z(=24,28,..,96) and shift coefficients (0,1,..,z-1) ……………………………………………………………… 93

7.6 User data throughput of the proposed decoder vs. the expansion factor of the code, z, for different numbers of decoder parallelization, M ………….………..… 97

7.7 Frame-error rate results……….. ……………...………………………………….97

8.1 Parallel architecture for layered decoder…………………………….…………….106

8.2 (a) Illustration of connections between message processing units to achieve cyclic down shift of (n-1) on each block column n (b) Concentric layout to accommodate 347 message processing units 109

8.3 BER performance of the decoder for (3,6) array code of N=2082 ……………...111

LIST OF TABLES

TABLE Page

1.1 BER performance for different codes 3

1.2 Quick summary of the proposed multi-rate decoder architectures 8

1.3 Quick summary of the proposed fixed-code decoder architectures 9

3.1 Occupation of resources for a decoding iteration in terms of clock cycles 26

3.2 Snapshot of partial sum registers in p CNUs operating in parallel to compute p R messages 27

3.3 Snapshot of partial sum registers in p VNUs operating in parallel to compute p Q messages 28

3.4 Memory requirement comparison 30

3.5 FPGA results (Device: Xilinx 2v8000ff1152-5) for (3,30) code of length 1830 31

3.6 ASIC Implementation of the proposed TPMP multi-rate decoder architecture 33

3.7 Area distribution of the chip for (3, k) rate compatible array codes, 130nm CMOS 33

3.8 Power distribution of the chip for (3, k) rate compatible array codes, 130nm CMOS 34

4.1 Parallel CNU implementation 40

5.1 FPGA results (Device: Xilinx 2v8000ff1152-5) 49

5.2 Summary of the proposed fixed-code decoder architecture, Code 1 50

5.3 Summary of the proposed fixed-code decoder architecture, Code 2 51

5.4 Summary of the proposed fixed-code decoder architecture, Code 3 and Code 4 51

TABLE Page

5.5 Area distribution of the fixed code TPMP architectures for array codes, 130nm CMOS 52

5.6 Power distribution of the fixed-code TPMP architectures for array codes, 130nm CMOS 52

6.1 FPGA implementations and performance comparison 76

6.2 Memory implementation for optimally scaled architecture (*j=5,k=10,…, k_{max} (=61), p=61,M=p*) 77

6.3 Memory implementation for scalable architecture (*j=3,k=6,…,k_{max} (=32), p=347,M=61*) 78

6.4 ASIC Implementation of the proposed TDMP multi-rate decoder architecture 79

6.5 Area distribition of the chip for (5,k) rate compatible array codes, 130nm 80

6.6 Power distribution of the chip for (3,k) rate compatible array codes, 130nm 80

7.1 FPGA Implementation results of the multi-rate decoder (supports z=24, 48 and 96 and all the code rates) 95

7.2 FPGA Implementation results of the multi-rate decoder, fully compliant to WiMax (supports z=24,28,32,…,and 96 and all the code rates)………………… 96

7.3 Implementation comparison 96

7.4 ASIC Implementation of the proposed TDMP Multi-rate decoder architecture 101

7.5 Area distribution of the chip for WiMax LDPC codes 101

7.6 Power distribution of the chip for WiMax LDPC codes 102

7.7 ASIC Implementation of the proposed TDMP Multi-rate decoder architecture for 802.11n LDPC codes 102

7.8 Area distribution of the chip for IEEE 802.11n LDPC codes 103

7.9 Power distribution of the chip for IEEE 802.11n LDPC codes 103

TABLE Page

7.10 FPGA implementation results for the multi-rate decoder, fully compliant to IEEE 802.11n (Device, XILINX2V8000FF152-5, frequency =110MHz)............. 104

7.11 ASIC implementation results for the multi-rate decoder for M=81 (Frequency = 500MHz)... 104

8.1 Proposed decoder work as compared with other authors 112

CHAPTER I

INTRODUCTION

1.1. Motivation

The insatiable demand for data and connectivity at the user level, driven primarily by advances in integrated circuits, has dramatically impacted the evolution of the communications market. The period of the last 25 years witnessed the progress from 300 baud modems to multi-terabit fiber backbones, multi-gigabit wired communication links and multi-megabit wireless communication links.

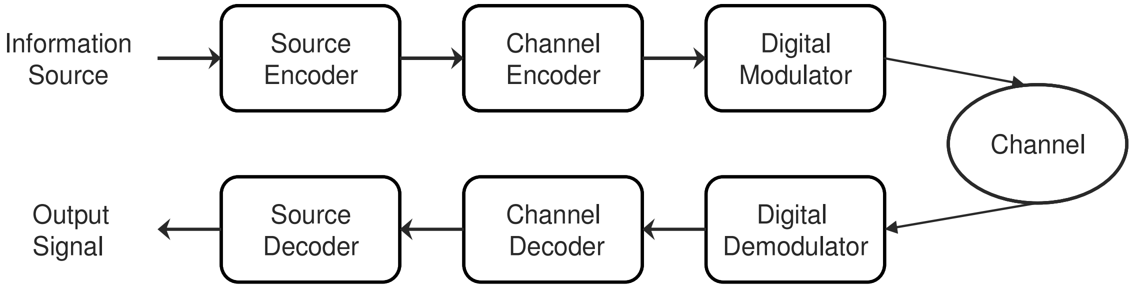


Fig 1.1. Block diagram of a digital communication system

Figure 1.1 shows a basic block diagram of a digital communication system [1]. First, an information signal, such as voice, video or data is sampled and quantized to form a digital sequence, then it passes through the source encoder or data compression to remove any unnecessary redundancy in the data.

This dissertation follows the style and format of *IEEE Transactions on Circuits and Systems*.

Then, the channel encoder codes the information sequence so that it can recover the correct information after passing through a channel. Error correcting codes such as convolutional [2], turbo [3] or LDPC codes [4] are used as channel encoders. The binary sequence then is passed to the digital modulator to map the information sequence into signal waveforms. The modulator acts as an interface between the digital signal and the channel. The communication channel is the physical medium that is used to send the signal from the transmitter to the receiver. At the receiving end of the digital communications system, the digital demodulator processes the channel-corrupted transmitted waveform and reduces the waveforms to a sequence of digital values that feeds into the channel decoder. The decoder reconstructs the original information by the knowledge of the code used by the channel encoder and the redundancy contained in the received data. Then, a source decoder decompresses the data and retrieves the original information. The probability of having an error in the output sequence is a function of the code characteristics, the type of modulation, and channel characteristics such as noise and interference level, etc [1].

Low-Density Parity Check (LDPC) codes and Turbo codes are among the best known near Shannon limit codes that can achieve good BER performance for low SNR applications [3]-[14] as shown in Table 1.1. When compared to the decoding algorithm of Turbo codes, LDPC decoding algorithm has more parallelization, low implementation complexity, low decoding latency, as well as no error-floors at high signal-to-noise ratios (SNRs). LDPC decoders require simpler computational processing. While initial LDPC decoder designs [15] suffered from complex interconnect issues, structured LDPC codes [10-11], [4], [16-25] simplify the interconnect complexity. Recently, Low-Density Parity-

Check (LDPC) codes have widely been considered as a promising error-correcting coding scheme for many real applications in telecommunications and magnetic storage, because of their superior performance and suitability for hardware implementation. LDPC codes are adopted/being adopted in the next generation digital video broadcasting (DVB-S2), MIMO-WLAN 802.11n, 802.12, 802.20, Gigabit Ethernet 802.3, magnetic channels (storage/recording systems), and long-haul optical communication systems .

Table 1.1

BER performance for different codes

Rate ½ Code	SNR required for BER <1e-5
Shannon, Random Code	0 dB
(255,123) BCH	5.4 dB
Convolutional Code	4.5 dB
Iterative Code Turbo	0.7 dB
Iterative Code LDPC	0.0045 dB

LDPC codes can be decoded by Gallager's iterative two-phase message passing algorithm (TPMP), which involves check-node update and variable-node update as a two phase schedule. Various algorithms are available for check-node updates and widely used algorithms are the sum of products (SP), min-sum (MS), and Jacobian-based BCJR (named after its discoverers Bahl, Cocke, Jelinik, and Raviv) [26-35]. The authors in [20] introduced the concept of turbo decoding message passing (TDMP, also called layered decoding) using BCJR for their architecture-aware LDPC (AA-LDPC) codes. TDMP offers 2x throughput and significant memory advantages when compared to TPMP. TDMP is later studied and applied for different LDPC codes using the sum of products algorithm and its variations in [38]-[39]. TDMP is able to reduce the number of iterations required by up to 50% without performance degradation when compared to the standard message passing algorithm. A quantitative performance comparison for different check updates was given by Chen and Fossorier et al. [32]. Their research showed that the offset min-sum (OMS) decoding algorithm with 5-bit quantization could achieve the same bit-error rate (BER) performance as that of floating point SP and BCJR with less than 0.1 dB penalty in SNR.

Most of the current LDPC decoder architecture research is focusing on increasing throughput or reducing implementation complexity, neglecting power analysis. In fact, power consumption presents a critical issue in computing, particularly in portable and mobile platforms, because of battery life and power dissipation. Designing a practical architecture must consider the trade-off among throughput, power consumption and hardware complexity. An LDPC decoder architecture can be implemented in parallel message passing and/or serial message passing. In the parallel decoder architecture [15], the nodes in the bipartite graph are directly mapped into message computation units and the edges of the graph are mapped into network of interconnects. The parallel architecture achieves high throughput at the cost of interconnect complexity. In the architecture [16], a fully pipelined implementation with two memory buffers per stage, alternating between read/write, was introduced. In [18], a joint code decoder design approach was adapted to construct a class of (3,k)-regular LDPC codes and a partly parallel decoder architecture was proposed to reduce the hardware complexity and achieve reasonable throughput.

1.2. Problem Overview

A parallel decoder implementation [15] exploiting the inherent parallelism of the algorithm is constrained by the complexity of the physical interconnect required to establish the graph connectivity of the code and, hence, does not scale well for moderate (2K) to large code lengths. Long on-chip interconnect wires present implementation challenges in terms of placement, routing, and buffer-insertion to achieve timing closure. For example, the average interconnect wire length of the rate-0.5, length 1020, 4-bit LDPC decoder of [15] is 3 mm using 160nm CMOS technology, and has a chip area of 52.5 mm^2 of which only 50% is utilized due to routing congestion. On the other hand, serial architectures [16] in which computations are distributed among a number of function units that communicate through memory instead of a complex interconnect, are slow and do not meet the practical data throughputs considered in the present standards.

The authors in [19] reported that 95% of power consumption of the decoder chip developed in [18] results from memory accesses. The implementation [20] reports that 50% of it power is due to memory accesses in message passing. There are several other architectures presented in [22]-[24], [37-38], [42], [45]. However, all of these implementations focused on improving the throughput while ignoring the power consumption issue due to message passing memory.

The check-to-bit message update equation is prone to quantization noise since it involves the nonlinear function and its inverse. The function has a wide dynamic range which requires the messages to be represented using a large number of bits to achieve a fine resolution, leading to an increase in memory size and interconnect complexity (e.g., for a regular (3, 6)-LDPC code of length 1020 with 4-bit messages, an increase of 1 bit increases the memory size and/or interconnect wires by 25%). The min-sum decoding algorithm [29], [32]-[33], [34] is an approximation for the Sum of Products algorithm to decode LDPC codes. The min-sum decoding algorithm does not have the complexity associated with non-linear functions used in the sum of products algorithm [26].

1.3. Main Contributions

The main contributions of this work are the following:

1 The On-the-fly computation paradigm by which the structured properties of LDPC codes are used to reduce computations, memory and interconnect.

2 New micro-architecture structures for switching network and check node processing.

3 Efficient decoder architectures and implementations for regular and irregular LDPC codes that offers substantial gains over the existing academic and commercial implementations Three unique run time configurable and multi-rate cores, each tailored in the design phase based on the class of code and the application, are designed. Two very high throughput and fixed code architectures are designed. Characteristics of these decoder ASIC implementations are briefly summarized in Table 1.2 and Table 1.3 along with the other recent state-of-the-art implementations. Details of each decoder implementation are given in the next chapters.

Rate compatible array codes are considered for DSL applications. Irregular block LDPC codes are proposed for IEEE 802.16e, IEEE 802.11n, IEEE 802.20 and being considered for other wireless standards. The total savings in memory translate to around 55% for the IEEE 802.11n LDPC decoder, when compared to a very recent state of the art decoder. In addition to the above savings, a master-slave router is proposed to accommodate 114 different parity check matrices in run time for IEEE 802.16e. This approach eliminates the control memory requirements by generating the control signals for the data router (slave) on-the-fly with the help of a self routing master network. If the memory approach is used for this as in the present state of the art, it would have resulted in a large chip area of around 140 mm^2 (in 180 nm technology) just for storing the control signals.

Properly designed regular array codes have low error floors and meet the requirements of magnetic recording channel and other applications which need several Gbps of data throughput. A high throughput and fixed code architecture for array LDPC codes has been designed. No modification to the code is done as this can result in early error floors. This parallel decoder architecture has no routing congestion and is scalable for longer block lengths. When compared to the latest state of the art decoders, this design has an area efficiency of around 10x for a given data throughput. In summary, all of these findings are explained in the text of this dissertation, with extensive theoretical simulations and VHDL verification on FPGA and ASIC design platforms.

Table 1.2

Quick summary of the proposed multi-rate decoder architectures

	Semi-Parallel multi-rate LDPC decoder [26]	Multi-rate TPMP Architecture regular QC-LDPC (Chapter III)	Multi-rate TDMP Architecture for regular QC-LDPC (Chapter VI)	Multi-rate TDMP Architecture for irregular QC-LDPC (Chapter VII)
LDPC Code	AA-LDPC, (3,6) code, rate 0.5, length 2048	(3,*k*) rate compatible array codes *p*=347. *k*=6,7,..12	(5,*k*) rate compatible array codes *p=61*. *k*=10,11,..61	Irregular codes up to length 2304 IEEE 802.16e WiMax LDPC codes
Decoded Throughput, t_d,	640 Mbps	2.37 Gbps	590 Mbps	1.37 Gbps
Area	14.3 mm^2	7.62 mm^2	1.6 mm^2	2.1 mm^2
Frequency	125 MHz	500 MHz	500 MHz	500 MHz
Nominal Power Dissipation	787 mW	821 mW	257 mW	282 mW
CMOS Technology	180 nm, 1.8V	130 nm, 1.2V	130 nm,.1.2V	130 nm, 1.2V
Decoding Schedule	TDMP, BCJR, it_{max}=10	TPMP, SP, it_{max}=20	TDMP, OMS, it_{max}=10	TDMP, OMS, it_{max}=10
Area Efficiency for t_d,	44.75 Mbps/mm^2	311 Mbps/ mm^2	369 Mbps/ mm^2	649.5 Mbps/ mm^2
Energy Efficiency for t_d,	123 pJ/Bit/Iteration	17 pJ/Bit/Iteration	44.2 pJ/Bit/Iteration	21 pJ/Bit/Iteration
Est. Area for 180 nm	14.3 mm^2	14.6 mm^2	3.06 mm^2	4.02 mm^2
Est. Frequency for 180 nm	125 MHz	360 MHz	360 MHz	360 MHz
Est. Decoded Throughput(t_d) ,180 nm	640 Mbps	1.71 Gbps	426 Mbps	989 Mbps
Est. Area Efficiency for t_d, 180 nm	44.75 Mbps/mm^2	117 Mbps/ mm^2	139.2 Mbps/mm^2	246 Mbps/mm^2
Est. Energy Efficiency for t_d, 180 nm	123 pJ/Bit/Iteration	38.3 pJ/Bit/Iteration	99.5 pJ/Bit/Iteration	47.3 pJ/Bit/Iteration
Application	Multi-rate application as well as fixed code application	DSL, Wireless	DSL, Wireless	Wireless, IEEE 802.11n, IEEE 802.16e, IEEE 802.22
Bit error rate Performance	Good	Good	Good	Very good and close to capacity
Scalability of Design for longer lengths	Yes	Yes	Yes	Yes

Table 1.3

Quick summary of the proposed fixed-code decoder architectures

	Fully Parallel LDPC decoder [15]	TPMP Architecture regular Array QC-LDPC (Chapter V)	TDMP Architecture for regular Array QC-LDPC (Chapter VIII)
Decoded Throughput, t_d,	1 Gbps	1.5 Gbps	6.94 Gbps
Area	52.5 mm^2	3.39 mm^2	5.39 mm^2
Frequency	64 MHz	500 MHz	100 MHz
Nominal Power Dissipation	690 mW	156.5 mW	75 mW
LDPC Code	Random LDPCr code, rate 0.5, length 1024	(4,30) array code of length 1830	(3,6) array code of length 2082
CMOS Technology	160 nm, 1.5V	130 nm, 1.2V	130 nm, 1.2V
Decoding Schedule	TPMP, SP, it_{max}=64	TPMP, SP, it_{max}=20	TDMP, OMS, it_{max}=10
Area Efficiency for t_d,	19 Mbps/mm^2	442.4 Mbps/mm2	1288 Mbps/mm^2
Energy Efficiency for t_d,	10.1 pJ/Bit/Iteration	5.6 pJ/Bit/Iteration	1.1 pJ/Bit/Iteration
Est Area for 180 nm	66.4 mm^2	6.5 mm^2	10.1 mm^2
Est Frequency for 180 nm	56.8 MHz	360 MHZ	72 MHz
Est Decoded Throughput $t_{d,}$ 180 nm	887.5 Mbps	1.08 Gbps	4.98 Gbps
Est Area efficiency for $t_{d\,,}$ 180 nm	13.36 Mbps/mm^2	166.1 Mbps/mm2	493 Mbps/mm^2
Est Energy efficiency for $t_{d\,,}$ 180 nm	14.5 pJ/Bit/Iteration	12.6 pJ/Bit/Iteration	4.8 pJ/Bit/Iteration
Scalability of Design for other code parameters and longer lengths	No	Yes	Yes
Application	Fixed code application	Very High throughput and low error-floor applications such as magnetic recording channels, Ethernet and optical links	Very High throughput and low error-floor applications such as magnetic recording channels, Ethernet and optical links.
Bit error rate Performance	Good	Good	Good

By examining the above implementation results for multi-rate architectures, we can conclude that irregular QC LDPC codes perform well and also their implementation complexity is less among the above 3 architectures. The implementation for irregular codes is more efficient as fewer number of non-zero blocks in the parity check matrix are needed to achieve excellent BER performance close to the capacity. Note that the underlying data flow graph for both regular QC-LDPC codes (Chapter VI) and irregular QC-LDPC codes (Chapter VII) is the same. This new data flow graph has several advantages which are discussed more fully in Chapters VI and VII. Scheduling of layered decoding, out-of-order processing, and bypassing techniques are employed to deal with irregularity. This is discussed fully in Chapter VII.

By examining the above implementation results, we can conclude that parallel TDMP architecture for array QC LDPC codes have the least complexity for very high throughput applications. A parallel layered architecture for irregular QC-LDPC codes can also be implemented based on this. However, the routing will be a problem and in addition irregular QC-LDPC will have a high error floor phenomenon. All of the above architectures are described in the following chapters.

In summary, the multi-rate and fixed code LDPC decoder architectures described in this dissertation achieve the best reported energy and area efficiencies while achieving the highest throughputs. The foundation of these architectures is based on minimizing the message passing and computation requirements by performing a thorough and systematic study.

CHAPTER II

QUASI-CYCLIC LOW-DENSITY PARITY-CHECK CODES AND DECODING

2.1. Introduction

LDPC codes are linear block codes described by an $m \times n$ sparse parity check matrix H. LDPC codes are well represented by bipartite graphs. One set of nodes, the variable or bit nodes correspond to elements of the code word and other set of nodes, viz. check nodes, correspond to the set of parity check constraints satisfied by the code words. Typically the edge connections are chosen at random. The error correction capability of the LDPC code is improved if cycles of short length are avoided in the graph. In a (r,c) regular code, each of the n bit nodes $(b_1, b_{2,} ..., b_n)$ has connections to r check nodes and each of the m check nodes $(c_1, c_{2,} ..., c_m)$ has connections to c bit nodes. In an irregular LDPC code, the check node degree is not uniform. Similarly the variable node degree is not uniform. We focus on the construction which structures the parity check matrix H into blocks of $p \times p$ matrices such that: 1. a bit in a block participates in only one check equation in the block and 2. each check equation in the block involves only one bit from the block. These LDPC codes are termed as Quasi cyclic LDPC codes: Cyclic shift of code word by *p* results in another code word. Here *p i*s the size of square matrix which is either a zero matrix or circulant matrix. This is a generalization of cyclic code in which cyclic shift of code word by 1 results in another code word.

2.2. Cyclotomic Cosets

One method to perform this construction is through cyclotomic cosets [49]. Another method is to achieve this property by employing random bit filling algorithm (for low rate codes such as rate ½ codes) and deterministic constructions (for high rate codes such as rate 8/9 codes) [9]-[11]. The work [49] reports no performance degradation for a (3, 5) - LDPC code of length 1055, rate 0.4; constructed from cyclotomic cossets. The H matrix can be constructed with filling the matrices obtained by permuting identity matrix by the appropriate shift coefficients [49]. Say $B_{j,k}\ \forall j = 1,2..r; k = 1,2,..c$ is a $p \times p$ matrix, located at the j^{th} block row and k^{th} block column of H matrix. The scalar value $s(j,k)$ denotes the shift applied to $I_{p\times p}$ identity matrix to obtain the $(j,k)^{th}$ block, $B_{j,k}$, and the rows in the $I_{p\times p}$ identity matrix are cyclically shifted to the right $s(j,k)$ positions for $s(j,k) \in \{0,1,2,..., p-1\}$. Let us define S as a $c \times r$ shift coefficient matrix in which

$$S_{j,k} = s(j,k)\ \forall j = 1,2..r; k = 1,2,..c\,. \tag{2.1}$$

The cyclotomic cosset containing the integer s is the set $\{s, sq, sq^2,..., sq^{m_s-1}\}$ where m_s is the smallest positive integer satisfying $sq^{m_s} \equiv s(\operatorname{mod} p)$ and q satisfies the relation $q^c = 1(\operatorname{mod} p)$. If $c = 5, r = 3$ and the desired length of code is in the vicinity of 1020. We find by trial and error that the values $p = 211$ and $q = 71$ result in cyclotomic cossets and the resulting code length n is $1055(= cp)$. One possible construction for S is

$$\begin{bmatrix} Cosset_1 \\ \\ Cosset_r \end{bmatrix} \text{.So } S_{3\times5} = \begin{bmatrix} 2 & 3 & 110 & 142 & 165 \\ 5 & 64 & 96 & 113 & 144 \\ 7 & 50 & 75 & 116 & 174 \end{bmatrix}$$

The H matrix can be now easily constructed with filling the matrices obtained by permuting $I_{211\times211}$ matrix by the above shift coefficients. So an H matrix, in this construction, can be completely characterized by these two simple matrices viz. $I_{p\times p}$ and

$S_{c\times r}$. To define H matrix, we start with fixing c, r and finding an appropriate p and shift coefficient matrix S such that the BER performance is maintained when compared to a random construction.

2.3. Array LDPC Codes

The reader is referred to [9]-[10], [36], [50-54] for a comprehensive treatment on array LDPC codes. The array LDPC parity-check matrix is specified by three parameters: a prime number p and two integers k, and j such that $j, k < p$.

It is given by,

$$H_A = \begin{bmatrix} I & I & I & \cdots & I \\ I & \alpha & \alpha^2 & \ldots & \alpha^{k-1} \\ I & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(k-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ I & \alpha^{j-1} & \alpha^{(j-1)2} & \cdots & \alpha^{(j-1)(k-1)} \end{bmatrix} \qquad (2.2)$$

where I is the $p \times p$ identity matrix, and α is a $p \times p$ permutation matrix representing a single left or right cyclic shift of I. Power of α in H denote multiple cyclic shifts, with the number of shifts given by the value of the exponent. In the following discussion, we use the α as a $p \times p$ permutation matrix representing a single left cyclic shift of I.

2.4. Rate-compatible Array LDPC Codes

Rate-compatible array LDPC codes are a modified version of the above for efficient encoding and multi-rate compatibility in [10] and their H matrix has the following structure

$$H = \begin{bmatrix} I & I & I & \cdots & I & I & \cdots & I \\ O & I & \alpha & \cdots & \alpha^{j-2} & \alpha^{j-1} & & \alpha^{k-2} \\ O & O & I & \cdots & \alpha^{2(j-3)} & \alpha^{2(j-2)} & & \alpha^{2(k-3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ O & O & \cdots & \cdots & I & \alpha^{(j-1)} & \cdots & \alpha^{(j-1)(k-j)} \end{bmatrix} \qquad (2.3)$$

where O is the $p \times p$ null matrix. The LDPC codes defined by H in (2.3) have a codeword length $M = jp$, number of parity-checks $M = kp$, and an information block length $K = (k-j)p$. The family of rate-compatible codes is obtained by successively puncturing the left most p columns, and the topmost p rows. According to this construction, a rate-compatible code within a family can be uniquely specified by a single parameter, say, q with $0 < q \leq j-2$. To have a wide range of rate-compatible codes, we can also fix j, p, and select different values for the parameter k. Since all the codes share the same base matrix size p; the same hardware implementation can be used. It is worth mentioning that this specific form is suitable for efficient linear-time LDPC encoding [10]. The systematic encoding procedure is carried out by associating the first $N-K$ columns of H with parity bits, and the remaining K columns with information bits.

2.5. Irregular Quasi-Cyclic LDPC Codes (Block LDPC Codes)

The block irregular LDPC codes have competitive performance and provide flexibility and low encoding/decoding complexity [12]-[13]. The entire *H* matrix is composed of the same style of blocks with different cyclic shifts, which allows structured decoding and reduces decoder implementation complexity. For the LDPC codes proposed for IEEE 802.16e, each base *H* matrix in block LDPC codes has 24 columns, simplifying the implementation. Having the same number of columns between code rates minimizes the number of different expansion factors that have to be supported. There are four rates supported: 1/2, 2/3, 3/4, and 5/6, and the base *H* matrix for these code rates are defined by systematic fundamental LDPC code of M_b-by-N_b where M_b is the number of rows in the base matrix and N_b is the number of columns in the base matrix. The following base matrices are specified: 12 x 24, 8 x 24, 6 x 24, and 4 x 24. The base model matrix is defined for the largest code length (*N* = 2304) of each code rate. The set of shifts in the base model matrix are used to determine the shift sizes for all other code lengths of the same code rate. Each base model matrix has 24 ($=N_b$) block columns and M_b block rows. The expansion factor *z* is equal to *N*/24 for code length *N*. The expansion factor varies from 24 to 96 in the increments of 4, yielding codes of different length. For instance, the code with length *N* = 2304 has the expansion factor *z*=96 [10]. Thus, each LDPC code in the set of WiMax LDPC codes is defined by a matrix *H* as :

$$H=\begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,N_b} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,N_b} \\ \cdots & \cdots & \cdots & \cdots \\ P_{M_b,1} & P_{M_b,2} & \cdots & P_{M_b,N_b} \end{bmatrix}=P^{H_b} \qquad (2.4)$$

where $P_{i,j}$ is one of a set of *z-by-z* cyclically right shifted identity matrices or a *z-by-z* zero matrix. Each 1 in the base matrix H_b is replaced by a permuted identity matrix while each 0 in H_b is replaced by a negative value to denote a *z-by-z* zero matrix.

2.6. Irregular QC LDPC Codes for Other Wireless Standards (802,11n and 802.20)

The LDPC codes proposed in other wireless standards area similar to the above structure. But the base matrices are different. So the same architectures can be re-used with minor changes.

2.7. Two Phase Message Passing (TPMP) and Decoding of LDPC

A quantitative performance comparison for different check updates [26]-[35] was given by Chen et al. [32]. Their research showed that the performance loss for OMS decoding with 5-bit quantization is less than 0.1dB in SNR compared with that of optimal floating point SP (Sum of Products) and BCJR. Assume binary phase shift keying (BPSK) modulation (a 1 is mapped to -1 and a 0 is mapped to 1) over an additive white Gaussian noise (AWGN) channel. The received values y_n are Gaussian with mean $x_n = \pm 1$ and variance σ^2. The reliability messages used in belief propagation (BP)-based offset min-sum algorithm can be computed in two phases: 1. check-node processing and 2. variable-node processing. The two operations are repeated iteratively until the decoding criterion is satisfied. This is also referred to as standard message passing or two-phase message passing (TPMP). For the i^{th} iteration, $Q_{nm}^{(i)}$ is the message from variable node n to check node m, $R_{mn}^{(i)}$ is the message from check node m to variable node n, $\mathrm{M}(n)$ is the set of the neighboring check nodes for variable node n, and $\mathrm{N}(m)$ is the set of the neighboring variable nodes for check node m.

The message passing for TPMP is described in the following three steps as given in [32] to facilitate the discussion on TDMP in the next section:

Step 1. *Check-node processing*: for each m and $n \in \mathrm{N}(m)$,

Sum of Products (SP) Check-node update

$$R_{mn}^{(i)} = \psi^{-1}\left[\left(\sum_{n' \in N(m)\backslash n} \psi\left(Q_{n'm}^{(i)}\right)\right)\right].\delta_{mn}^{(i)} \tag{2.5}$$

Here $\psi(x) = -\log(|\tanh(x/2)|$ is the Gallager's function which is invariant under its inverse.

Offset min-sum(OMS) Check-node update (approximation to (2.5))

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \max\left(\kappa_{mn}^{(i)} - \beta, 0\right), \tag{2.6}$$

$$\kappa_{mn}^{(i)} = \left|R_{mn}^{(i)}\right| = \min_{n' \in \mathrm{N}(m)\backslash n} \left|Q_{n'm}^{(i-1)}\right|. \tag{2.7}$$

where β is a positive constant and depends on the code parameters [32]. For (3, 6) rate 0.5 array LDPC code, β is computed as 0.15 using the density evolution technique presented in [12].

The sign of check-node message $R_{mn}^{(i)}$ is defined as

$$\delta_{mn}^{(i)} = \left(\prod_{n' \in \mathrm{N}(m)\backslash n} \mathrm{sgn}\left(Q_{n'm}^{(i-1)}\right)\right), \tag{2.8}$$

Step 2. *Variable-node processing*: for each n and $m \in \mathrm{N}(n)$,

$$Q_{nm}^{(i)} = L_n^{(0)} + \sum_{m' \in \mathrm{M}(m)\backslash m} R_{m'n}^{(i)}, \tag{2.9}$$

where the log-likelihood ratio of bit n is $L_n^{(0)} = y_n$.

Step 3. *Decision: for final decoding*

$$P_n = L_n^{(0)} + \sum_{m \in M(n)} R_{mn}^{(i)} . \tag{2.10}$$

A hard decision is taken by setting $\hat{x}_n = 0$ if $P_n(x_n) \geq 0$, and $\hat{x}_n = 1$ if $P_n(x_n) < 0$. If, $\hat{x}H^T = 0$, the decoding process is finished with $\hat{x}_n$ as the decoder output; otherwise, repeat steps (1-3). If the decoding process doesn’t end within predefined maximum number of iterations, $it_{\max}$, stop and output an error message flag and proceed to the decoding of the next data frame.

2.8.Turbo Decoding Message Passing (TDMP) or Layered Decoding

In TDMP, the LDPC code with j block rows can be viewed as concatenation of j layers or constituent sub-codes similar to observations made for AA-LDPC codes in [20]. After the check-node processing is finished for one block row, the messages are immediately used to update the variable nodes (in step 2, above), whose results are then provided for processing the next block row of check nodes (in step 1, above).

CHAPTER III

MULTI-RATE TPMP ARCHITECTURE FOR REGULAR QC-LDPC CODES

3.1. Introduction

This chapter provides efficient multi-rate TPMP architectures for regular QC-LDPC codes. This architecture is targeted for Cyclotomic coset based LDPC and array LDPC. This architecture works for rate compatible array LDPC codes with a minor change in implementation to accommodate the slight irregularity in the parity check matrix.

The QC-LDPC codes are discussed in Chapter II. For the continuity of presentation, some of the material discussed in Chapter II is briefly summarized in this section. The H matrix can be constructed with filling in with matrices obtained by permuting identity matrix by the appropriate shift coefficients [49]. Say $B_{j,k}\ \forall j=1,2..r;k=1,2,..c$ is a $p\times p$ matrix, located at the j^{th} block row and k^{th} block column of H matrix. The scalar value $s(j,k)$ denotes the shift applied to $I_{p\times p}$ identity matrix to obtain the $(j,k)^{th}$ block, $B_{j,k}$, and the rows in the $I_{p\times p}$ identity matrix are cyclically shifted to the right $s(j,k)$ positions for $s(j,k)\in\{0,1,2,...,p-1\}$. Let us define S as a $c\times r$ shift coefficient matrix in which

$$S_{j,k}=s(j,k)\ \forall j=1,2..r;k=1,2,..c\,. \tag{3.1}$$

So an H matrix, in this construction, can be completely characterized by these two simple matrices viz. $I_{p\times p}$ and $S_{c\times r}$.To define H matrix, we start with fixing c,r and finding an appropriate p and shift coefficient matrix S such that the BER performance is maintained when compared to a random construction.

For example if $c=5, r=3$ and $p=211$ the use of cyclotomic cosets [49] results in the following shift coefficient matrix for the code of length $1055(n=cp)$.

$$S_{3\times5}=\begin{bmatrix}2 & 3 & 110 & 142 & 165\\ 5 & 64 & 96 & 113 & 144\\ 7 & 50 & 75 & 116 & 174\end{bmatrix} \tag{3.2}$$

For regular array LDPC codes with similar parameters, this is given by

$$S_{3\times5}=\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 1 & 2 & 3 & 4\\ 0 & 2 & 4 & 6 & 8\end{bmatrix}$$

3.2. Block Message Independence Property for Regular QC-LDPC Codes

The reliability messages used in Gallager's Belief Propagation algorithm can be computed in two phases viz., check-node processing (3.3) and variable node processing (3.4) and this is repeated iteratively till the decoding criterion is satisfied (see Chapter II). The message passing equations are given by

$$R_{cj,bi}=\psi^{-1}\left[\left(\sum_{i^{'}=Row[cj][1]}^{Row[cj][c]}\psi\left(Q_{i^{'},cj}\right)\right)-\psi\left(Q_{bi,cj}\right)\right].\delta(cj,bi) \tag{3.3}$$

$$Q_{bi,cj}=\left(\sum_{j^{'}=Col[bi][1]}^{Col[bi][r]}R_{j^{'},bi}\right)-R_{cj,bi}+\wedge(bi) \tag{3.4}$$

where $R_{cj,bi}$ is the message from check c_j to bit b_i , $Q_{bi,cj}$ is the message from bit b_i to check c_j , $\psi(x)=-\log(|\tanh(x/2)|)$ is the Gallager's function which is invariant under its inverse, $\delta(cj,bi)$ is ±1 and is given by

$$\delta(cj,bi) = \left(\operatorname{sgn}\left(Q_{bi,cj}\right) . \prod_{i' \in Row[cj]} \operatorname{sgn}\left(Q_{i',cj}\right) \right) . (-1)^{|Row[cj]|} \tag{3.5}$$

$(-1)^{|Row[cj]|} = 1$ for codes constructed with even parity. $\wedge(bi)$ is the intrinsic reliability metric of bit i . $Row\lfloor c_j \rfloor[1...c]$ ($Col[b_i][1...r]$) gives the locations of bits (checks) connected to the check node c_j (bit node b_i).

We can represent R and Q messages by the following matrices for deriving the new data independence property. This arrangement is similar to physical message storage employed in [16] except that these matrices are not really stored in the proposed architecture.

$$Rm = \begin{bmatrix} R_{1,Row[1][1]} & R_{1,Row[1][2]} & \cdots & R_{1,Row[1][c]} \\ R_{2,Row[2][1]} & R_{2,Row[2][2]} & \cdots & R_{2,Row[2][c]} \\ \vdots & \vdots & \vdots & \vdots \\ R_{p \bullet r,Row[p \bullet r][1]} & R_{p \bullet r,Row[p \bullet r][1]} & \cdots & R_{p \bullet r,Row[p \bullet r][c]} \end{bmatrix}$$

$$Qm = \begin{bmatrix} Q_{1,Col[1][1]} & Q_{1,Col[1][2]} & \cdots & Q_{1,Col[1][r]} \\ Q_{2,Col[2][1]} & Q_{2,Col[2][2]} & \cdots & Q_{2,Col[2][r]} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{p \bullet c,Col[p \bullet c][1]} & Q_{p \bullet c,Col[p \bullet c][2]} & \cdots & Q_{p \bullet c,Col[p \bullet c][r]} \end{bmatrix} \tag{3.6}$$

If we employ the partitioning of H matrix into r rows and c columns of p x p matrices, the R and Q messages in a p x p block can be processed simultaneously. The recent architectures [17]-[18], [37], [49] exploit this property to store messages in the memory partitioned into p independent memory banks and employ p copies of message computation units.

We now represent the R and Q messages in a p x p block as p x 1 vectors

$$\vec{R}_{j,k} = \left[Rm_{1+(j-1)p,k}, \ldots, Rm_{l+(j-1)p,k}, \ldots, Rm_{p+(j-1)p,k} \right]^T$$
$$\vec{Q}_{k,j} = \left[Qm_{1+(k-1)p,j}, \ldots, Qm_{l+(k-1)p,j}, \ldots, Qm_{p+(k-1)p,j} \right]^T \tag{3.7}$$

$l = 1,2,\ldots, p \ \forall j = 1,2,\ldots, r, k = 1,2,\ldots, c$

Then R and Q messages in block matrix format are:

$$\vec{R} = \begin{bmatrix} \vec{R}_{1,1} & \vec{R}_{1,2} & \ldots & \vec{R}_{1,c} \\ \vec{R}_{2,1} & \vec{R}_{2,1} & \ldots & \vec{R}_{2,c} \\ \vdots & \vdots & \vdots & \vdots \\ \vec{R}_{r,1} & \vec{R}_{r,1} & \ldots & \vec{R}_{r,c} \end{bmatrix}$$

$$\vec{Q} = \begin{bmatrix} \vec{Q}_{1,1} & \vec{Q}_{1,2} & \ldots & \vec{Q}_{1,r} \\ \vec{Q}_{2,1} & \vec{Q}_{2,1} & \ldots & \vec{Q}_{2,r} \\ \vdots & \vdots & \vdots & \vdots \\ \vec{Q}_{c,1} & \vec{Q}_{c,1} & \ldots & \vec{Q}_{c,r} \end{bmatrix} \tag{3.8}$$

Now the Gallager's equations can be written as

$$\vec{R}_{j,k} = \psi\left[\left(\sum_{k=1}^{c} \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right) \right) - \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right) \right] . \vec{\delta}_{k,j} \tag{3.9}$$

$$\vec{Q}_{k,j} = \left(\sum_{j=1}^{r} \vec{R}_{j,k}^{p-s(j,k)} \right) - \vec{R}_{j,k}^{p-s(j,k)} + \vec{\wedge}_k \tag{3.10}$$

$$\vec{\delta}_{k,j} = \left(\operatorname{sgn}\left(\vec{Q}_{k,j}\right) . \prod_{k=1}^{r} \operatorname{sgn}\left(\vec{Q}^{s(k,j)}{}_{k,j}\right) \right) \tag{3.11}$$

$$\vec{\wedge}_k = \left[\wedge \left(1 + (k-1)p\right), \ldots, \wedge\left(p + (k-1)p\right) \right] \tag{3.12}$$

where $\vec{Q}_{k,j}^{s(j,k)}$ ($\vec{R}_{j,k}^{p-s(j,k)}$) is the modified $p \times 1$ vector $\vec{Q}_{k,j}$ ($\vec{R}_{j,k}$), whose elements are circularly shifted in location by the amount $s(j,k)$ ($p - s(j,k)$).

Say $\vec{A}_j = \sum_{k=1}^{c} \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right)$, $\vec{B}_{k,j} = \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right)$ (3.13)

$$\vec{C}_k = \sum_{j=1}^{r} \vec{R}_{j,k}^{p-s(j,k)}, \vec{D}_{j,k} = \vec{R}_{j,k}^{p-s(j,k)} \tag{3.14}$$

Now

$$\vec{R}_{j,k} = \psi\left[\vec{A}_j - \vec{B}_{k,j}\right].\vec{\delta}_{k,j} \tag{3.15}$$

$$\vec{Q}_{k,j} = \vec{C}_k - \vec{D}_{j,k} + \vec{\wedge}_k \tag{3.16}$$

We can observe that the j^{th} block row of R messages is only dependent on the j^{th} block column of Q messages and similarly the k^{th} block row of Q messages is only dependent on the k^{th} block column of R messages. Only one class of messages has to be stored if we schedule the pipeline of the R and Q message computation unit such that either one of R and Q message units output the block row at once and multiplexing the other units schedule such that it is able to produce the output in block column fashion.

If p Check to Bit serial message computation units, which have internal FIFOs of size $(c \times (r-1)+1) \approx c.r$ are employed, this is approximately equivalent to storage requirement of one class of messages $(p.c.r)$. We do not need any additional memory for storing R and Q messages. By scheduling we can efficiently use the internal memory of the computational units.

3.3. Architecture

For the example (3, 5) - LDPC code of length 1055 described in Section 3.2, $r = 3, c = 5$ and $p = 211$. We can generalize the following discussion to any LDPC code with similar structure. A multi-rate architecture is obtained by designing the architecture such that it can support the maximum values of r and c.

According to the observation made in Section 3.2, the pipeline is designed such that Q messages are produced block row wise and R messages are produced in block column fashion (Fig. 3.1). Initially the Q messages are available in row wise as they are set to soft log likelihood information of the bits coming in chunks of p (10). The Q Initializer (Q Init) is an SRAM of size $n+p$ and holds the $\wedge$ values of two different frames. It can supply p intrinsic values to the BCUs each clock cycle and also can simultaneously read p intrinsic values from the channel at the start of iterations of the next frame. The data path of the design is set to 5 bits. ψ and ψ^{-1} are implemented based on uniform quantization and according to the scheme of [12]. The maximum number of iterations is set to 20 and the iterations will stop when the decoded vector d (using Majority function of Bit to check messages)satisfies the relation $dH^T = 0$.

The p by p cyclic shifter is constructed with two input - two output switches and $\log 2(p)$ stages of $p/2$ switches are used. The Switching Sequence (SS) unit supplies the binary sequences to toggle switches in order to produce the shifts in the matrix $S_{3\times 5}$ (2). The cyclic shifters of R and Q messages will receive sequences column wise corresponding to the shifts (2, 5, 7, 3… 174) for cyclic shift up and down respectively (refer to (3.9) and (3.10)). The check node processing array is composed of p serial Check Node Units (CNU) which computes the partial sum for each block row in a multiplexed fashion to produce the R messages in block column fashion. The registers ps1, ps2 and ps3 correspond to the partial sum for block row 1, 2 and 3 respectively.

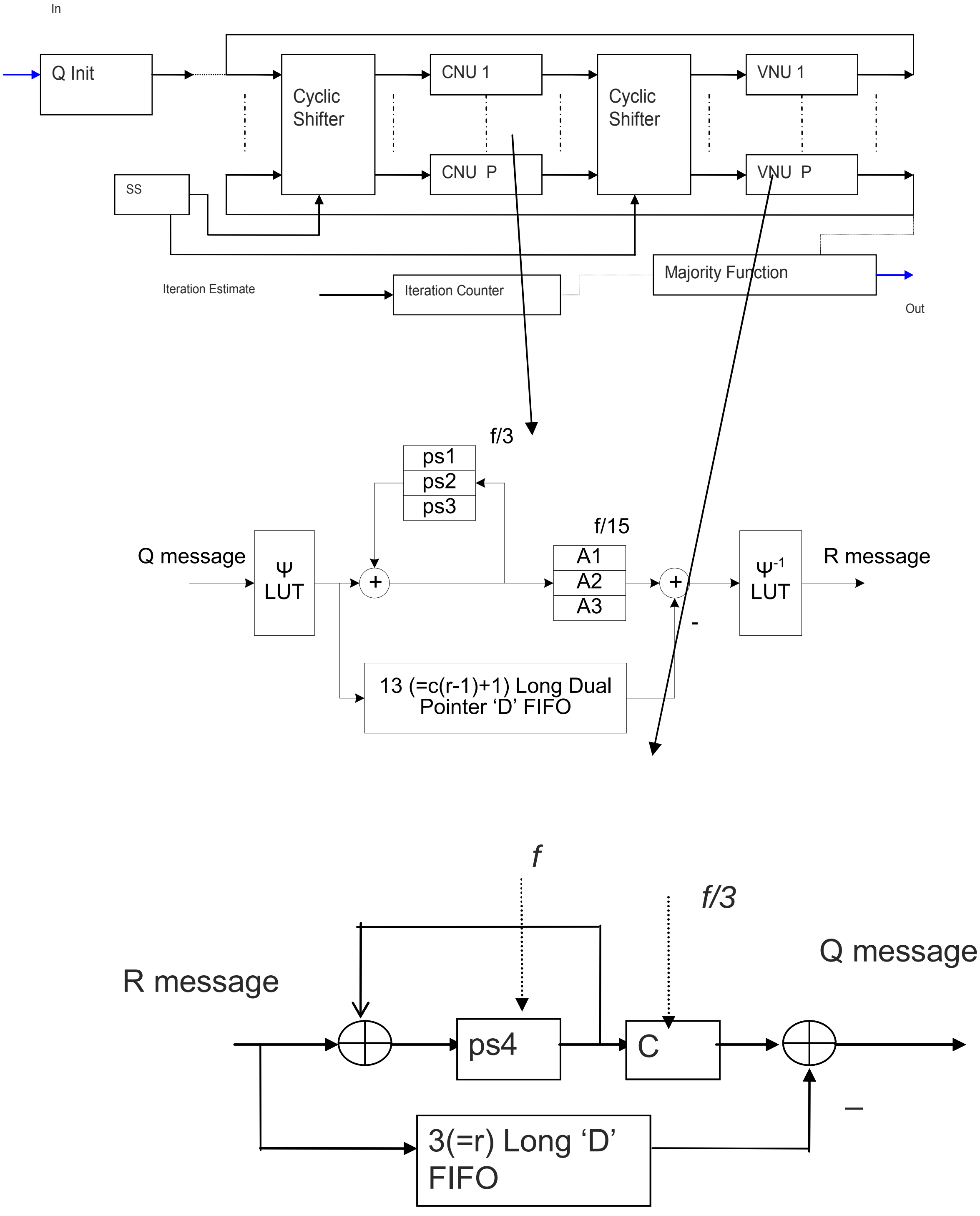


Fig. 3.1. Block diagram of the decoder architecture

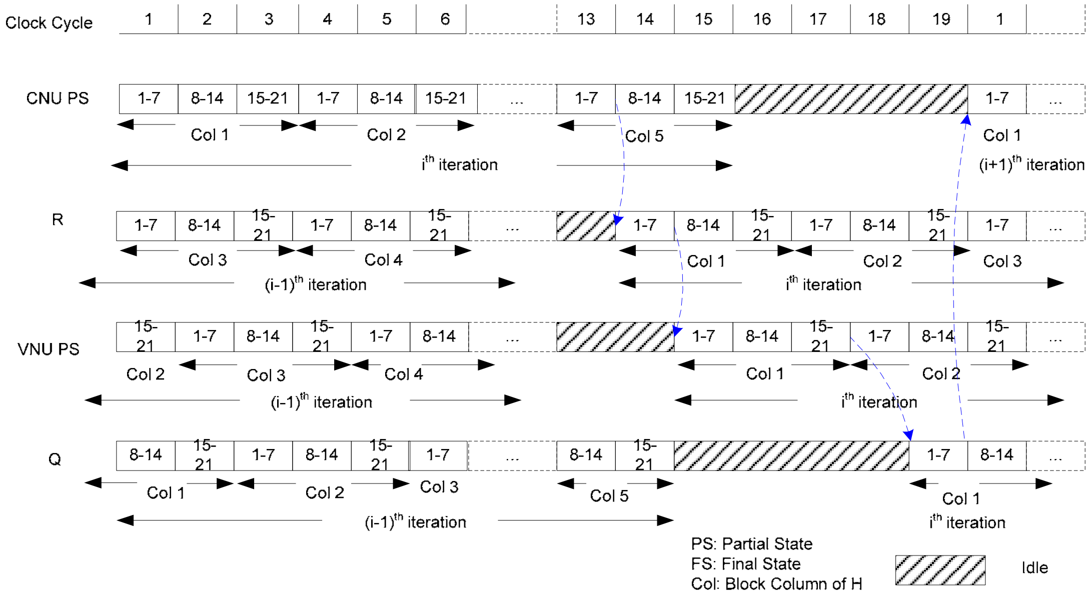


Fig. 3.2. Pipeline of the decoder

Table 3.1.

Occupation of resources for a decoding iteration in terms of clock cycles. (Shown for two iterations.)

I	CBU Adders	CBU Subtractors	BCU Adders	BCU Subtractors
1	1-15	14-28	15-29	19-33
2	20-34	35-49	34-48	38-52

I=Iteration Number.

Table 3.2.

Snapshot of partial sum registers in p CNU s operating in parallel to compute p R messages

Clock, I	13,1	15,1	22,1
$ps\vec{1}$	$\sum_{k=1}^{5}\psi\left(\vec{Q}_{k,1}^{s(1,k)}\right)$	$\sum_{k=1}^{5}\psi\left(\vec{Q}_{k,1}^{s(1,k)}\right)$	$\sum_{k=1}^{1}\psi\left(\vec{Q}_{k,1}^{s(1,k)}\right)$
$ps\vec{2}$	$\sum_{k=1}^{4}\psi\left(\vec{Q}_{k,2}^{s(2,k)}\right)$	$\sum_{k=1}^{5}\psi\left(\vec{Q}_{k,2}^{s(2,k)}\right)$	0
$ps\vec{3}$	$\sum_{k=1}^{3}\psi\left(\vec{Q}_{k,3}^{s(3,k)}\right)$	$\sum_{k=1}^{5}\psi\left(\vec{Q}_{k,3}^{s(3,k)}\right)$	0

The CNU B FIFO corresponds to (3.13) stores the intermediate computations. Its snapshot at 15th clock cycle is $\left[\vec{B}_{5,3,},\vec{B}_{5,2,}\vec{B}_{5,1,}...,\vec{B}_{1,1,}\right]$. The registers A1, A2 and A3 (which correspond to (3.13)) latch the ps1, ps2 and ps3 (Table 3.3) in 14,15 and 16 clock cycles respectively and one of these values (from 14- 28th clock cycle for 1st iteration) will be selected sequentially as one of the inputs to the subtractor and each subtraction operation during this period produces R messages in block column fashion. The variable node processing array is composed of p serial Variable Node Units (VNU) which compute the partial sum ps4 for each block row in a sequential fashion to produce the Q messages in block row fashion. The pipeline is shown in Fig. 3.2.

Table 3.3.

Snapshot of partial sum registers in p VNUs operating in parallel to compute p Q messages

Clock,I	15,1	17,1	29,1
$ps\vec{4}$	$\sum_{j=1}^{1} \vec{R}_{j,1}^{p-s(j,1)}$	$\sum_{j=1}^{3} \vec{R}_{j,1}^{p-s(j,1)}$	$\sum_{j=1}^{3} \vec{R}_{j,5}^{p-s(j,5)}$

The VNU D FIFO corresponds to (3.14). Its snapshot at 17th clock cycle is $\left[\vec{R}_{1,1}, \vec{R}_{2,1}, \vec{R}_{3,1}\right]$ and at 29th clock cycle is $\left[\vec{R}_{1,5}, \vec{R}_{2,5}, \vec{R}_{3,5}\right]$. The register C (which corresponds to (314)) latch the ps4 (Table 3.4), every three clock cycles and is one of the inputs to the subtractor and each subtraction operation during this period produces Q messages in block row fashion.

While this architecture is targeted for regular array LDPC codes and cyclotomic coset based regular QC-LDPC codes, this architecture works for rate compatible array LDPC codes with a minor change in implementation to accommodate the slight irregularity in the parity check matrix. Note that due to the slight irregularity in rate-compatible array LDPC matrix, each block row l has a node degree $j-l+1$. The variable-nodes in each block column n has a node degree equal to $\min(n, j)$. We have to devise a simple control mechanism to address this irregularity. One simpler way to facilitate implementation is to assume that all the block rows have equal check-node degrees and set the check-node messages corresponding to null blocks in H matrix to zero in order not to affect the variable-node processing. $\vec{R}_{l,n}^{(i)} = 0$ if $n < l$ in each iteration *i*. Similarly the variable-node messages belonging to the null blocks are always set to positive infinity in order not to affect the check-node processing. $\bar{Q}_{l,n}^{(i)} = \infty \text{ if } n < l$. For check-node update, the message with maximum reliability won't affect the CNU output.

3.4. Performance Comparison

3.4.1. Memory Advantage

Table 3.4 shows the comparison with the related work. The memory savings are 75% and savings in memory accesses are 66% when compared to [16]. When compared to [17], [20] the memory accesses are 50% less while the memory requirement is almost the same and this results in better low power characteristic for the proposed architecture. For example [20] reported that the NA-Mm accounts for 50% of their decoder power.

3.4.2. Throughput Advantage

The architecture presented here does the overlapping of the CNU processing and VNU processing. So this architecture has around 2x throughput advantage similar to overlapped TPMP architectures [17] when compared to the other TPMP architectures [18]. In addition, this architecture is efficiently pipelined as compared to other architectures. This will lead to frequency advantage as well as the reduction of glitches.

Table 3.4.

Memory requirement comparison

	[3]	[4]	[5]	[8]	Proposed
Mm	$4p.c.r$	$2p.c.r$	$p.c.r$	$p.c.r$	0
Mc	$p.c$	$p.c$	$p.r$	$p.c$	$p.c.r$
NA_Mm	$4p.c.r$	$4p.c.r$	$2p.c.r$	$2p.c.r$	0
NA_Mc	$2p.c.r$	$2p.c.r$	$2p.c.r$	$2p.c.r$	$2p.c.r$

Mm: Memory for message storage Mc: Internal Memory in Check to Bit Serial Computational Units NA_Mm: No. of R/W accesses from Mm for a decoding iteration NA_Mc: No. of R/W accesses from Mc for a decoding iteration

3.5 FPGA Implementation Results

Fig.3.3 gives the comparison with the implementation of [42]. Table 3.5 gives the FPGA implementation details for the proposed architecture.

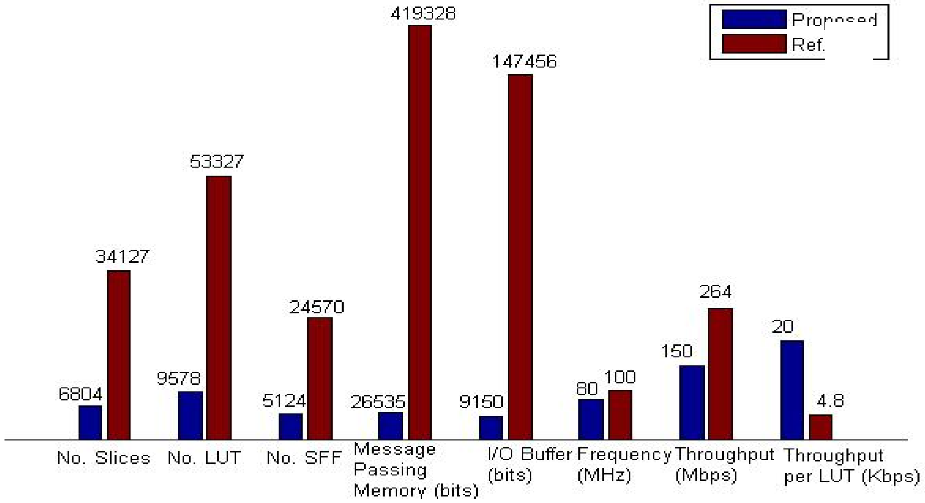


Fig. 3.3. Comparison of architecture for (3,k=6,…30) rate compatible array codes of up to length 1830. Reference is [42]

Table 3.5.

FPGA results (Device: Xilinx 2v8000ff1152-5) for (3,30) code of length 1830

	No. Slices	No. 4-input LUT	No. Slice Flip-flops	BRAM
CNU simple	39	66	35	
CNU-TDM3	51	56	59	
Array of 61 CNU-TDM3	3111	3416	3599	
VNU	18	25	26	
VNU array	1098	1525	1586	
Routers	2070	3600	0	
Message Computation FIFOs 61 FIFOs of depth 87 and word length 5				26535 bits 5 BRAMS configured as depth 87 and word length 61
Input Buffer 1830x5 bits				9150 bits 5 BRAMS configured as depth 30 and word length 61
Total number available	46592	93184	93184	168
Top-TDM3	6279	8541	5185	Frequency 80 MHz
Throughput				150 Mbps

3.6. ASIC Implementation Results

The proposed decoder architecture is implemented using the open source standard cells v*sclib013* [62] in 130nm technology. The synthesis is done by using the synopsys design analyzer tool, while layout is done using the cadence's Silicon Ensemble tool. Tables 3.6, 3.7 and 3.8 give the performance comparison as well as the decoder chip characteristics. Even though TDMP is a better alternative for implementation (Chapters VI, VII and VIII) as shown in later chapters, the original TDMP decoder [12] is based on more complicated BCJR algorithm. The CNU for BCJR takes more area due to the need of several internal FIFOs. In addition, the Omega network is used in [20] instead of logarithmic shifter. The use of logarithmic shifter saves area to store the control signals as well as the the absence of control wires make the logarithmic shifter’s layout much more compact. Note that the memory requirements for both the decoder is similar. However the number of memory accesses in [20] is much higher leading to low energy efficiency. The proposed decoder has a frequency advantage also, as the CNU stage has 2 pipeline stages, the VNU stage has 2 pipeline stages. The decoder in [20] has fewer pipeline stages due to the nature of feedback loops in the CNU processing.

Table 3.6.

ASIC Implementation of the proposed TPMP multi-rate decoder architecture

	Semi-Parallel multi-rate LDPC decoder [20]	Multi-rate TPMP Architecture regular QC-LDPC
LDPC Code	AA-LDPC, (3,6) code, rate 0.5, length 2048	(3,*k*) rate compatible array codes *p*=347. *k*=6,7,..12. length *=pk(2082,..,4164)*
Decoded Throughput, t_d,	640 Mbps	2.37 Gbps
Area	14.3 mm^2	7.62 mm^2
Frequency	125 MHz	500 MHz
Nominal Power Dissipation	787 mW	821 mW
Memory	51,680 bits	62,465 bits (including the channel LLR memory)
CMOS Technology	180 nm 1.8V	130 nm, 1.2V
Decoding Schedule	TDMP, BCJR, it_{max}=10	TPMP, SP, it_{max}=20
Area Efficiency for t_d,	44.75 Mbps/mm^2	311 Mbps/ mm^2
Energy Efficiency for t_d,	123 pJ/Bit/Iteration	17 pJ/Bit/Iteration
Est. Area for 180 nm	14.3 mm^2	14.6 mm^2
Est. Frequency for 180 nm	125 MHz	360 MHz
Decoded Throughput(t_d) ,180 nm	640 Mbps	1.71 Gbps
Area Efficiency for t_d, 180 nm	44.75 Mbps/mm^2	117 Mbps/ mm^2
Energy Efficiency for t_d, 180 nm	123 pJ/Bit/Iteration	38.25 pJ/Bit/Iteration
Application	Multi-rate application as well as fixed code application	Multi-rate application as well as fixed code application Rate-compatible array codes are considered for DSL applications.
Bit error rate Performance	Good	Good and similar to AA-LDPC

it_{max}= Maximum number of iterations.

Table 3.7.

Area distribution of the chip for (3, *k*) rate compatible array codes, *130 nm CMOS*

	Area (mm^2)
CNU Array (FIFO is not included)	2.19
VNU Array	1.43
Message Passing Memory+ Channel LLR memory	2.28
2 Cyclic shifters	1.73
Total chip area	7.63

Table 3.8.

Power distribution of the chip for (3,*k*) rate compatible array codes, *130 nm CMOS*

	Power (mW)
Logic(CNU, VNU and shifters)	482.5
Memory	199.7
Leakage	0.3
Clock	96.5
Wiring	48.2
Total	827.2

CHAPTER IV

VALUE-REUSE PROPERTIES OF OMS AND MICRO-ARCHITECTURES FOR CHECK NODE UNIT BASED ON OMS

4.1. Value-reuse Properties

This chapter provides the novel micro-architecture structures for check node message processing unit (CNU) for the min-sum decoding of Low-Density Parity-Check codes (LDPC). The construction of these CNU structures is based on a less known property of the min-sum processing step that it produces only two different output magnitude values (or three signed 2's complement values) irrespective of the number of incoming bit-to check messages [26]. These new micro-architecture structures would employ the minimum number of comparators by exploiting the concept of survivors in the search. These would result in the reduced number of comparisons and consequently reduced energy use.

For each check node m , $\left|R_{mn}^{(i)}\right| \ \forall n \in \mathrm{N}(m)$ takes only 2 values. The least minimum and the second least minimum of the entire set of the messages can be defined from various variable-nodes to the check-node m as,

$$M1_m^{(i)} = \min_{n' \in \mathrm{N}(m)} \left|Q_{mn'}^{(i-1)}\right| .,\tag{4.1}$$

$$M2_m^{(i)} = 2nd \min_{n' \in \mathrm{N}(m)} \left|Q_{mn'}^{(i-1)}\right| .\tag{4.2}$$

Now (2.7) in Chapter II becomes

$$\left|R_{mn}^{(i)}\right| = M1_m^{(i)} ,\ \forall n \in \mathrm{N}(m) \backslash M1\_index\tag{4.3}$$

$= M2_m^{(i)}, n = M1\_index$.

Since $\forall n \in \mathrm{N}(m)$, $\delta_{mn}^{(i)}$ takes a value of either $+1$ or -1 and $\left|R_{mn}^{(i)}\right|$ takes only two values. So (2.6) in Chapter II, gives rise to only three possible values for the whole set $R_{mn}^{(i)}$ $\forall n \in \mathrm{N}(m)$. In a VLSI implementation, this property significantly simplifies the logic and reduces the memory.

4.2. Serial CNU for OMS

This section presents the micro-architecture of serial CNU for OMS, which is used in TPMP architecture (Chapter V) and in TDMP architectures(Chapter VI and VII).

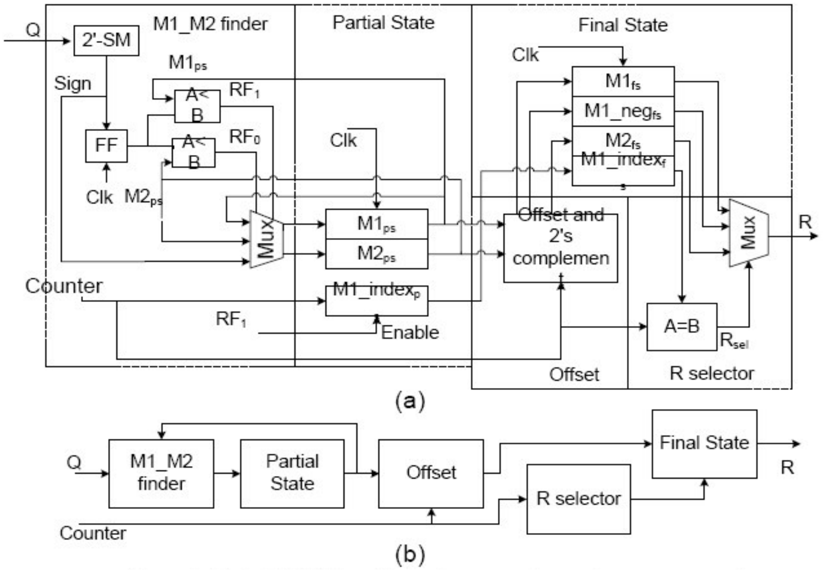


Fig 4.1. Serial CNU for OMS using value-reuse property.

Fig. 4.1(a) shows the CNU micro-architecture for (5, 25) code while Fig. 4.1(b) shows the block diagram of the same. In the first 25 clock cycles of the check-node processing, incoming variable messages are compared with the two up-to-date least minimum numbers (partial state, PS) to generate the new partial state, M1 which is the first minimum value, M2 which is the second minimum value and index of M1. The final state (FS) is then computed by offsetting the partial state. It should be noted that the final state includes only *M1, -M1, +/-M2* with offset correction. Fig. 4.1(b) is the block diagram of the same architecture. M1_M2 finder computes the two least numbers, according to the incoming data and the current minimum numbers stored in partial state. The offset module applies the offset correction, and stores the results in the Final State module. R selector then assigns one out of these 3 values, based on the index of M1 and the sign of R message generated by sign XOR logic (4), to the output R messages. While the final state has dependency on offset correction, the offset is dependent on the completion of the partial state. In operation, the final state and partial state will operate on different check-nodes. The serial CNU finds the least two minimum numbers with 2 comparators in a serial fashion and reduces the number of offset-correction computation from k to 2. Normally, CNU (check-node unit) processing is done using the signed magnitude arithmetic for (2.7) and VNU (variable-node unit processing) (2.9) is done in 2's complement arithmetic. This requires 2's complement to the signed conversion at the inputs of CNU and signed to the 2's complement at the output of CNU. In the proposed scheme, 2's complement is applied to only 2 values instead of k values at the output of CNU. The value re-use property also reduces the memory requirement significantly. Conventionally, the number of messages each CNU stores is equal to the number of edges it has, that is k. Now only four units of information are needed: the three values that $R_{mn}^{(i)}$ may take and the location of $M1_m^{(i)}$, then check-node message to the VNU is readily chosen by multiplexing.

4.3. Parallel CNU

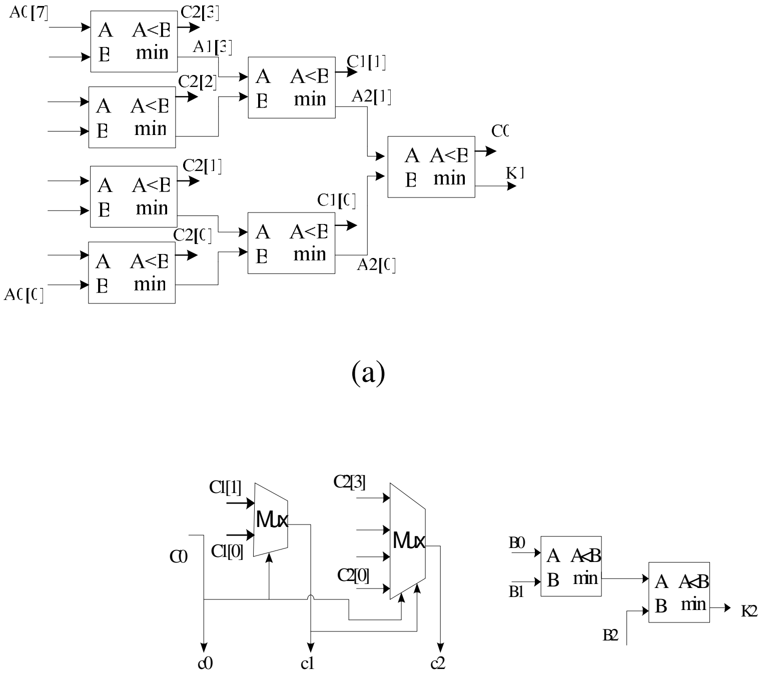
In procedures 1 and 2 below, consider the case of rate 0.5 (4, 8) code so that $k = 8$ and assume the word length of signed magnitude variable node messages is 5 so that there are 4 bits allocated for magnitude.

Procedure 1: Locate the two minimum magnitude values of the input vector. *Procedure 1.1*: Find the first minimum in the input vector of length 8 using the binary tree of comparators, see Fig. 4.2.a. *Procedure 1.2*: Select the survivors by using the comparator output flags as the control inputs to multiplexes. For example in the last stage of the comparator tree the value other than the least minimum is the survivor. No further comparisons are necessary along the tree path to the survivor. We trace back the survivors using the comparator outputs. At any stage of the binary tree we have only one survivor. So there would be $\log_2 k$ survivors. *Procedure 1.3*: Perform $\log_2 k - 1$ comparisons among survivors to find the least minimum of survivors (i.e., the second minimum of input vector) using another smaller binary tree (Fig 4.2.b).

In Fig. 4.2, C0, C1, and C2 are 1-bit outputs of comparators. The comparator's output is 1 if A<B and is 0 otherwise. '0' in C0 notation is used to denote the first level of outputs from the right and so on. C2[0] denotes the output of the first comparator (from bottom) at third level outputs from the right. A2, A1, and A0 are 4-bit magnitudes of variable node messages Q. '0' in A0 notation is used to denote the first level of inputs. A0[0] denotes the 4-bit input word at the first input of the first level of inputs. A similar naming convention is used for other symbols. K1 = A0 [C0 C1[C0] C2[C0C1[C0]]] is the least minimum. The 3 bit trace back C0 C1[C0] C2[C0C1[C0]] gives the index of K1 in the input vector A0. Next, the survivors are obtained from the intermediate nodes of the

search tree. We use 2-in-1, 4-in-1 and 8-in-1 multiplexers respectively to obtain the following survivors: B2= A2[c0], B1= A1[!c1 c0]] and B0= A0[!c2 c1 c0]. Here, the notation !x denotes logical inversion of the bit x. The second minimum is obtained from these survivors (Fig. 4.2.b).

Procedure 2: This procedure produces the R outputs according to (4.3). Apply the offset to K1 and K2 to produce M1 and M2. Next compute –M1 and/or –M2. Then, based on the computed sign information by XOR logic (2.8) and index of K1, R is set to one of the 3 possible values M1, -M1, +/- M2 (see Fig. 4.3).



(a)

(b)

Fig. 4.2. Finder for the two least minimum in CNU (a) Binary tree to find the least minimum. (b) Trace-back multiplexers and comparators on survivors to find the second minimum. Multiplexers for selecting survivors are not shown.

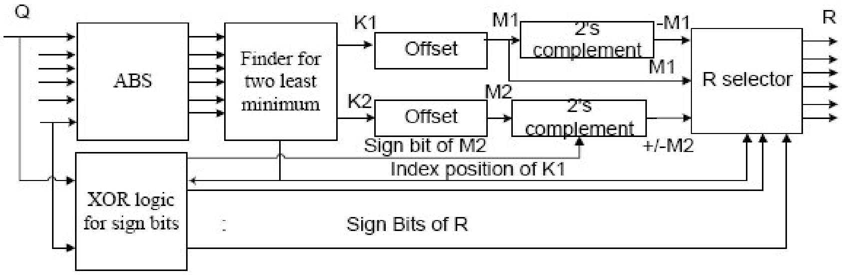


Fig. 4.3. Parallel CNU based on value-reuse property of OMS.

Table 4.1 presents the complexity comparison of parallel CNU for min-sum variants. The Parallel CNU in Fig. 4.3 can work as a regular min-sum (MS) CNU if the offset modules are removed. Note that the recently published CNU work on regular MS in [44], used a pseudo-rank order filter to find M1 and M2, which is more complex than our proposed method based on survivors [55]. In addition, the value-reuse property is not exploited completely as *k* instances of 2's complement adder are used and the BER performance degradation is 0.5 dB when compared to floating point SP. Also, note that the overall decoder architecture in both [20] (which is based on normalized MS, NMS) and [44] are based on TPMP, while the work presented here uses TDMP

Table 4.1

Parallel CNU implementation

CNU	Complexity in terms of equivalent adders	MS Variant (loss in dB against floating point SP)
[20]	$4k + 0.5(k(\lceil \log\ 2(k) \rceil - 1))$	NMS (~0.1)
[44]	$5k/2 + 3(\lceil \log 2(k) \rceil - 1)$	MS (~0.5)
Proposed	$2k + \lceil \log 2(k) \rceil + 1$	MS (~0.5)
Proposed	$2k + \lceil \log 2(k) \rceil + 5$	OMS (~0.1 dB)

CHAPTER V

FIXED CODE TPMP ARCHITECTURE FOR REGULAR QC-LDPC CODES

5.1. Introduction

Message passing memory takes around 30% of chip area and consumes from 50%-90% power of the typical semi-parallel decoders for the Low Density Parity Check Codes (LDPC) [18], [20]. We propose a new LDPC Decoder architecture based on the Min Sum algorithm that reduces the need of message passing memory by 80% and the routing requirements by more than 50%. This novel architecture is based on the scheduling of computation that results in "on the fly computation" of variable node and check node reliability messages. The results are memory-efficient and router-less implementations of (3,30) code of length 1830 and (3,6) code of length 1226; each on a Xilinx Virtex 2V8000 FPGA device achieved 1.27 Gbps and 585 Mbps respectively

We present a new architecture that exploits the various properties of structured Array LDPC codes [9] and the value–reuse properties offset min-sum algorithm to reduce the memory, routing and computational requirements. The key features of this architecture are:

1. 80% savings in message passing memory requirements when compared to other semi-parallel architectures based on MS and its variants [37], [41]
2. Scalable for any code length due to the concentric and regular layout unlike the fully parallel architecture [15]
3. Reduction of router multiplexers from 50% and beyond based on dynamic state concept.

5.2. Reduced Message Passing Memory and Router Simplification

Array codes are defined in [9] and have three parameters (d_v, d_c) and length N. Here d_v is the variable node degree and d_c is the check node degree. The size of the circulant matrix block in array code is a prime number and is given by $p = N / d_c$. Refer Chapter II for details on array codes. Most of the previous work is in the area of semi-parallel implementation of structured LDPC codes, however most of them are based on SP, for instance [17], [18], [51], [37], [41] proposed architectures based on MS and its variants. In the architecture of [51], (Chapter III) the check node messages in the H matrix are produced block column wise so that all the variable messages in each block column can be produced on the fly. Again these variable-node messages can be immediately consumed by the partial state computation sub-units in Check Node Units. This scheduling results in savings in message passing memory that is needed to store intermediate messages. This work extends above concepts used for SP to the offset MS.

Cyclic shifters take around 10%-20% of chip area based on the decoder's parallelization and constitute the critical path of the decoder. We make an observation that if all the block rows are assigned to different computational unit arrays of Check Node Unit(CNU) and serial CNU processing across block row is employed, then we need to have a constant wiring to achieve any cyclic shift as each subsequent shift can be realized using the feedback of previous shifted value. This leads to the elimination of forward router between CNU and Variable Node unit (VNU) as well as the reverse router between VNU and CNU. This is possible due to the fact that block-serial processing is employed and Array codes have a constant incremental shift in each block row. For the first block row, the shift and incremental shift is 0. For the second block row, the shifts are [$0,1,2,\ldots,d_c-1$] and the incremental shift is 1. For the third block row, the shifts are [0, 2 ,$\ldots, 2X(d_c-1)$] and the incremental shift is 2.

5.3. Check Node Unit Micro-architecture

The proposed serial check node unit design, discussed fully in Chapter IV and in [55] utilizes a less known property of the min-sum algorithm that the check node processing produces only two different output magnitude values irrespective of the number of incoming variable-node messages. [26]. The work in [37] resorts to the use of $2\,d_c$ comparators and additional processing such as offset correction and 2's complement for all d_c messages and does not utilize this property. This property would greatly simplify the number of comparisons required as well as the memory needed to store CNU outputs. Fig. 5.1. shows the serial CNU architecture for (3, 30) code. In the first 30 clock cycles of the check node processing, incoming variable messages are compared with the two up-to-date least minimum numbers (partial state, PS) to generate the new partial state, which include the least minimum value, M1, the second minimum value M2 and index of M1. Final state (FS) is then computed by offsetting the partial state. It should be noted that the final state includes only three signed numbers, i.e. M1, -M1, +/-M2 with offset correction, and index of M1. VNU micro-architecture is implemented as a parallel unit as the number of inputs is small. It takes 3 check node messages and one channel value. It is a binary tree adder followed by subtractors and 2's complement to signed magnitude conversion to generate variable node messages [16].

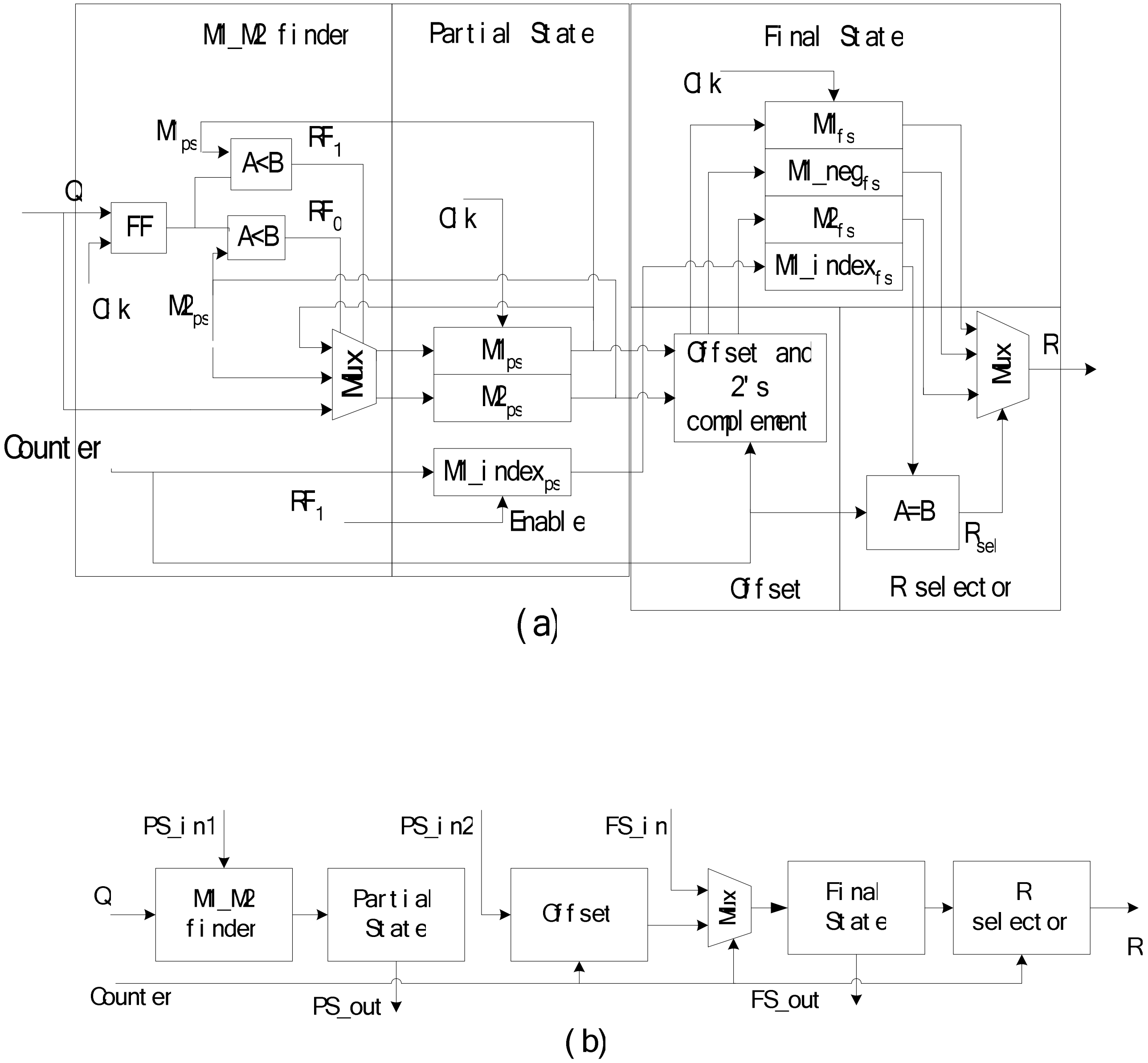


Fig 5.1. Check node processing unit, Q: Variable node message, R: Check node message. (a) simple scheme; (b) dynamic scheme.

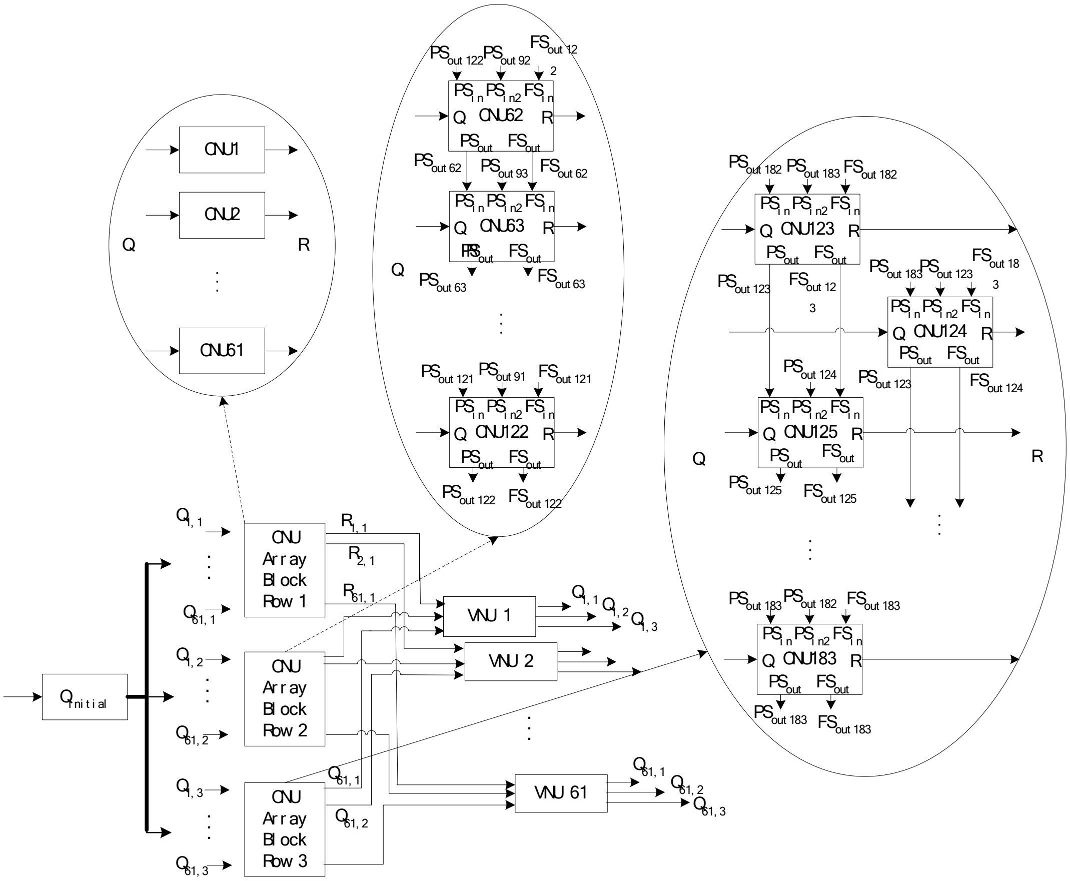


Fig 5.2. Architecture

5.4. Architecture

Figures 5.2 and 5.3 present the proposed architecture and pipeline scheduling for the implementation of (3, 30) – array LDPC code of length 1830 with the circulant matrix size of 61. The check node processing unit array is composed of 3 sub-arrays. Each sub-array contains 61 serial CNUs which compute the partial state for each block row to produce the check-node messages for each block column of H. Block row 1 is array of 61 simple CNUs. CNU array block row 2 and 3 are composed of dynamic CNUs (Fig 2b). The variable node processing array is composed of 61 parallel VNU units which can process 3 x 61 messages at each clock cycle. The sign bits will be stored in a FIFO (implemented as RAM), however, there is no need to subject these values to shifts as these values are not modified in check node processing partial state processing.

In the array of simple serial CNU that is designed to do check node processing for first block row in H matrix, the check node processing for each row in H matrix is done such that all the comparisons are performed locally with in one CNU to update the partial state each clock cycle and transfer the partial state to final state d_c once every cycle. In the array of dynamic CNU designed for second block row in H matrix, CNU 122 gets its partial state from CNU 121, CNU 121 gets its partial state from CNU 120 and so on. Array of dynamic CNU designed for the third block row in H matrix such that the connection between partial state registers among various units achieve cyclic shifts of [0,2,..,58]. Similar principle is used when making connections for the final state in the CNU array to achieve reverse routing.

As shown in Figure 5.3, initially the variable messages are available in row wise as they are set to soft log likelihood information (LLR) of the bits coming from the channel. Q Init is an SRAM of size $2N$ and holds the channel LLR values of two different frames. It can supply p intrinsic values to the VNUs each clock cycle. The data path of the design is set to 5 bits to provide the same BER performance as that of the floating point sum of products algorithm with 0.1-0.2 dB SNR loss [32]. Each iteration takes $d_c + 3$ clock cycles. For (3, 30) code this results in 6 x 33 clock cycles to process each frame when a maximum number of iterations set to 6. For (3,6) code this results in 20 x 9 clock cycles to process each frame when the number of iterations is set to 20.

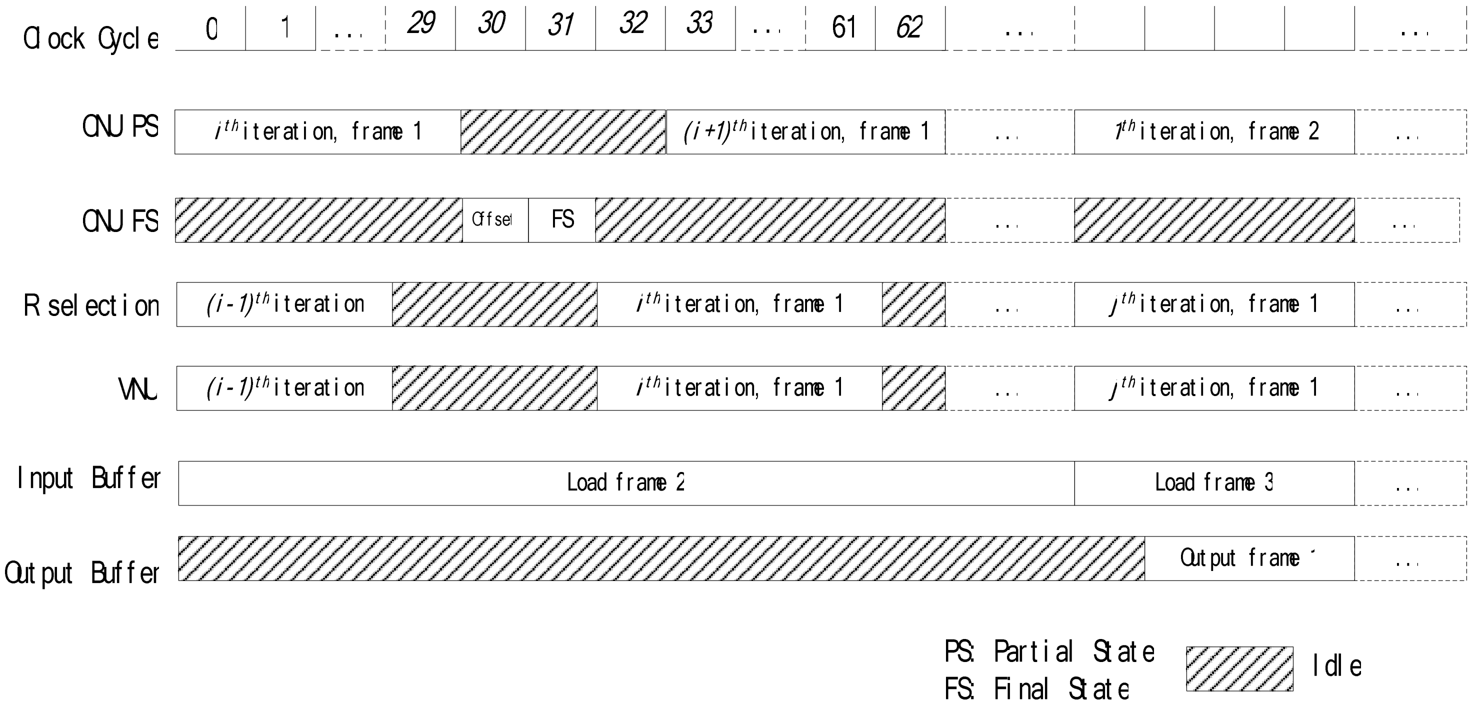


Fig 5.3. Pipeline

5.5. Results and Performance Comparison

The savings in message passing memory due to scheduling are 80% as we need to store only the sign bits of variable node messages. Forward router and reverse routers are eliminated. This results in the reduction of the number of multiplexers from $2\times(d_c-1)\times\log 2(p)\times p\times wl$ (as routers are eliminated) to $(d_c-1)\times p\times(3\times wl+\lceil\log 2(d_c)\rceil+1)$ (to support transfer of partial state to final state in the array of dynamic CNU). Here $wl=5$ and is the word length of the data path.

Table 5.1 shows resource consumption of different components used in the design for (3, 30) code of length 1830. Implementations for (3, 30) codes of lengths 1830 and (3,6) code of length 1226 on a Xilinx Virtex 2V3000 device achieved 1.2 Gbps (system frequency 153 MHz) and 340 Mbps (system frequency 140 MHz) respectively. Up to our best knowledge our LDPC implementations achieves the highest throughput per given FPGA resources. Figure 5.4 gives comparison of design metrics for our designs with the other designs [37], [41] based on similar code parameters and min sum implementation.

Tables 5.2-5.4 gives the ASIC implementation results and comparisons with the state-of-the-art fixed code decoder architectures. The design in [21] based on 1-bit data path. Though the routing congestion is decreased based on broadcasting and hard decision, it is still an issue with soft decoders. In addition, the hard decision decoder has very poor BER performance. The fully parallel decoder architecture in [15] is based on modified array codes to reduce the routing congestion and these exhibit early error floors. Thus this decoder may not be suitable for low error floor applications which require BERs of less than 1e-12. However, the advanced circuit techniques described in [], may be applicable to any LDPC decoder. The PLAs are used to implement non-linear functions needed for SP decoding algorithm occupied most of the area. All the other logic is implemented as standard static and dynamic CMOS logic. In the proposed architecture, the non-linear function is not needed in the offset min-sum. So there is no logic that can readily benefit from using the PLA based logic design. However, recent research indicates that it is possible to do a mix of PLA based logic and standard cell design to improve the frequency.

Table 5.1.

FPGA results (Device: Xilinx 2v8000ff1152-5)

	No. Slices	No. 4-input LUT	No. Slice Flip-flops	Operating frequency(MHz)
CNU simple	45	70	53	236
CNU dynamic	55	102	55	193
CNU array block row 1	2599	4285	2980	196
CNU array block row 2,3	3464	6438	2967	
CNU array	9230	17143	8875	
VNU	27	42	25	169
VNU array	1623	2562	1525	169
Top	11695	19732	10733	153
Total number available	46592	93184	93184	

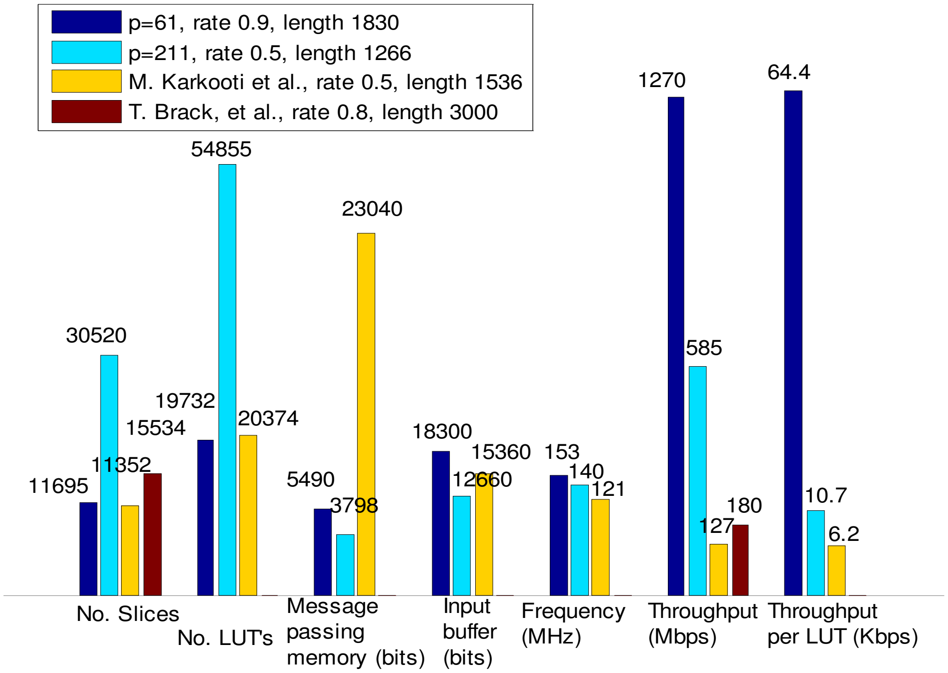


Fig 5.4. Results comparison with M. Karkoot et al.,[37] and T. Brack, et al., [41]

Table 5.2

Summary of the proposed fixed-code decoder architecture, code 1

	Fully Parallel LDPC decoder [15]	TPMP Architecture regular Array QC-LDPC
Decoded Throughput, t_d,	1 Gbps	5.78 Gbps
Area	52.5 mm^2	9.9 mm^2
Frequency	64 MHz	500 MHz
Power Dissipation	690 mW	695 mW
Memory	34816 bits (scattered flip-flops)	20820 bits for 2 input buffers 6246 bits for sign memory
LDPC Code	Random LDPC code, rate 0.5, length 1024	(3,6) array code, rate 0.5, length 2082
CMOS Technology	160 nm, 1.5V	130 nm, 1.2V
Decoding Schedule	TPMP, SP, it_{max}=64	TPMP, SP, it_{max}=20
Area Efficiency for t_d,	19 Mbps/mm^2	585.2 Mbps/mm^2
Energy Efficiency for t_d,	10.1 pJ/Bit/Iteration	6.01 pJ/Bit/Iteration
Scalability of Design for other code parameters and longer lengths	No	Yes

Table 5.3

Summary of the proposed fixed-code decoder architecture, code 2

	Fully Parallel LDPC decoder [21]	TPMP Architecture regular Array QC-LDPC
Decoded Throughput, t_d,	3.2 Gbps	3.0176 Gbps
Area	17.64 mm^2	8.04 mm^2
Frequency	100 MHz	500 MHz
Power Dissipation	NA	324.4 mW
Memory		46350 bits for 2 input buffers 23175 bits for sign memory
LDPC Code	RS-LDPC, (6,32) code, rate 0.8413, length 2048	(5,45) array code, rate 0.8889, length 4635
CMOS Technology	180 nm,. 1.8V	130 nm, μ , 1.2V
Decoding Schedule	TPMP, Hard decision SP, it_{max}=32	TPMP, 5-bit soft decoding, offset Min-sum, it_{max}=16
Area Efficiency for t_d,	181.4 Mbps/mm^2	375.6 Mbps/mm^2
Energy Efficiency for t_d,	NA	6.72 pJ/Bit/Iteration
Scalability of Design	No	Yes

Table 5.4

Summary of the proposed fixed-code decoder architecture, code 3 and code 4

	TPMP Architecture regular Array QC-LDPC	TPMP Architecture regular Array QC-LDPC
Decoded Throughput, t_d,	1.5 Gbps	9.85 Gbps
Area	3.39 mm^2	19.26 mm^2
Frequency	500 MHz	500 MHz
Power Dissipation	156.5 mW	890 mW
Memory	18300 bits for 2 input buffers 7320 bits for sign memory	104100 bits for 2 input buffers 41640 bits for sign memory
LDPC Code	(4,30) array code of length 1830	(4,30) array code of length 10410
CMOS Technology	130 nm, 1.2V	130 nm, 1.2V
Decoding Schedule	TPMP, SP, it_{max}=16	TPMP, SP, it_{max}=16
Area Efficiency for t_d,	442.4 Mbps/mm^2	512 Mbps/mm^2
Energy Efficiency for t_d,	5.64 pJ/Bit/Iteration	5.64 pJ/Bit/Iteration
Scalability of Design	Yes	Yes

Table 5.5

Area distribution of the fixed code TPMP architectures for Array codes, 130 nm CMOS[62]

	Code1, (3,6) array code, rate 0.5, length 2082 Area (mm^2)	Code 2, (5,45) array code, rate 0.8889, length 4635 Area (mm^2)	Code 3, (4,30) array code of length 1830 Area (mm^2)	Code 4, (4,30) array code of length 10410 Area (mm^2)
CNU Array (sign FIFO is not included)	5.7	2.8	1.3	7.6
VNU Array	1.6	0.8	0.4	2.1
Message Passing Memory+ Channel LLR memory	1.3	3.4	1.2	7.1
Wiring	1.3	1.0	0.5	2.5
Total chip area	9.9	8.0	3.4	19.3

Table 5.6

Power distribution of the fixed code TPMP architectures for array codes, 130 nm CMOS [62]

	Code1, (3,6) array code, rate 0.5, length 2082 Power (mW)	Code 2, (5,45) array code, rate 0.8889, length 4635 Power (mW)	Code 3, (4,30) array code of length 1830 Power (mW)	Code 4, (4,30) array code of length 10410 Power (mW)
Logic(CNU,VNU)	442.3	219.3	103.1	586.9
Memory	137.0	50.8	27.2	154.1
Leakage	0.4	0.3	0.1	0.8
Clock	75.3	35.1	17.0	96.3
Wiring	40.6	18.9	9.1	51.9
Total	695.6	324.4	156.5	890.0

CHAPTER VI

MULTI-RATE TDMP ARCHITECTURE FOR RATE-COMPATIBLE ARRAY LDPC CODES

6.1. Introduction

The main contribution of this chapter is an efficient turbo decoding message passing (TDMP) architecture which utilizes the value–reuse property of OMS, cyclic shift property of structured array LDPC codes, and the extension of block serial scheduling [5]. The resulting decoder architecture has the following key advantages: 1) removal of memory needed to store the sum of the variable node messages and the channel log-likelihood ratios (LLR) when compared to other semi-parallel architectures [20], [38], [51], [37], [41]. 2) 40%-72% savings in storage of extrinsic messages depending on the rate of the codes when compared to other semi-parallel architectures [20], [38], [51], [37], [41], 3) need of only one cyclic shifter instead of two cyclic shifters when compared to the work in [20], [38], [51], [37], [41]. 4) removal of memory needed to store variable node messages when compared to [37]-[41] and finally, 5) increase of throughput by 2x as number of required iterations decrease by 50% when compared to [37], [41], [51]. The last two advantages are also shared by other TDMP architectures [20], [38].

The rest of the chapter is organized as follows. Section 6.2 introduces the background of array LDPC codes, and OMS, the decoding algorithm. Section 6.3 presents the equations which facilitate the decoding process. Section 6.4 presents the value-reuse property and micro-architecture structure for CNU. The new data flow graph and architecture for TDMP using OMS is included in section 6.5. Section 6.6 shows the

FPGA implementation results, and performance comparison with related work. Section 6.7 concludes the chapter.

6.2. Background

6.2.1. Array LDPC Codes

The array LDPC parity-check matrix is specified by three parameters: a prime number p and two integers k (check-node degree) and j (variable-node degree) such that $j, k \leq p$ [9]. This is given by

$$H = \begin{bmatrix} I & I & I & \cdots & I \\ I & \alpha & \alpha^{2} & \cdots & \alpha^{k-1} \\ I & \alpha^{2} & \alpha^{4} & \cdots & \alpha^{2(k-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ I & \alpha^{j-1} & \alpha^{(j-1)2} & \cdots & \alpha^{(j-1)(k-1)} \end{bmatrix} \qquad (6.1.a)$$

where I is a $p \times p$ identity matrix, and α is a $p \times p$ permutation matrix representing a single left cyclic shift (or equivalently down cyclic shift) of I. The exponent of α in H is called the shift coefficient and denotes multiple cyclic shifts, with the number of shifts given by the value of the exponent. Rate-compatible array LDPC codes are modified versions of the above for efficient encoding and multi-rate compatibility in [10] and their H matrix has the following structure

$$H = \begin{bmatrix} I & I & I & \cdots & I & I & \cdots & I \\ O & I & \alpha & \cdots & \alpha^{j-2} & \alpha^{j-1} & & \alpha^{k-2} \\ O & O & I & \cdots & \alpha^{2(j-3)} & \alpha^{2(j-2)} & & \alpha^{2(k-3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ O & O & \cdots & \cdots & I & \alpha^{(j-1)} & \cdots & \alpha^{(j-1)(k-j)} \end{bmatrix}, \qquad (6.1.b)$$

where O is the $p \times p$ null matrix. The LDPC codes defined by H in (6.1b) have a codeword length $N = kp$, number of parity-checks $M = jp$, and an information block length $K = (k - j)p$. The family of rate-compatible codes is obtained by successively puncturing the left most p columns, and the topmost p rows. According to this construction, a rate-compatible code within a family can be uniquely specified by a single parameter, say, q with $0 < q \leq j - 2$. To have a wide range of rate-compatible codes, we can also fix j, p, and select different values for the parameter k. Since all the codes share the same base matrix size p; the same hardware implementation can be used. It is worth mentioning that this specific form is suitable for efficient linear-time LDPC encoding [10]. The systematic encoding procedure is carried out by associating the first $N - K$ columns of H with parity bits, and the remaining K columns with information bits.

6.2.2. Offset Min-sum Decoding of LDPC

Assume binary phase shift keying (BPSK) modulation (a 1 is mapped to -1 and a 0 is mapped to 1) over an additive white Gaussian noise (AWGN) channel. The received values y_n are Gaussian with mean $x_n = \pm 1$ and variance σ^2. The reliability messages used in belief propagation (BP)-based offset min-sum algorithm can be computed in two phases: 1.) check-node processing and 2.) variable-node processing. The two operations are repeated iteratively until the decoding criterion is satisfied. This is also referred to as standard message passing or two-phase message passing (TPMP). For the i^{th} iteration, $Q_{nm}^{(i)}$ is the message from variable node n to check node m, $R_{mn}^{(i)}$ is the message from check node m to variable node n, $\mathrm{M}(n)$ is the set of the neighboring check nodes for variable node n, and $\mathrm{N}(m)$ is the set of the neighboring variable nodes for check node m. The message passing for TPMP based on OMS is described in the following three steps as given in [132] to facilitate the discussion on TDMP in the next section:

Step 1. *Check-node processing*: for each m and $n \in \mathrm{N}(m)$,

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \max\left(\kappa_{mn}^{(i)} - \beta, 0\right), \tag{6.2}$$

$$\kappa_{mn}^{(i)} = \left|R_{mn}^{(i)}\right| = \min_{n' \in \mathrm{N}(m) \setminus n} \left|Q_{n'm}^{(i-1)}\right|, \tag{6.3}$$

where β is a positive constant and depends on the code parameters [32]. For (3, 6) rate 0.5 array LDPC code, β is computed as 0.15 using the density evolution technique presented in [11]. The sign of check-node message $R_{mn}^{(i)}$ is defined as

$$\delta_{mn}^{(i)} = \left(\prod_{n' \in \mathrm{N}(m) \setminus n} \operatorname{sgn}\left(Q_{n'm}^{(i-1)}\right)\right), \tag{6.4}$$

Step 2. *Variable-node processing*: for each n and $m \in \mathrm{N}(n)$,

$$Q_{nm}^{(i)} = L_n^{(0)} + \sum_{m' \in \mathrm{M}(m) \setminus m} R_{m'n}^{(i)}, \tag{6.5}$$

where the log-likelihood ratio of bit n is $L_n^{(0)} = y_n$.

Step 3. *Decision: for final decoding*

$$P_n = L_n^{(0)} + \sum_{m \in M(n)} R_{mn}^{(i)}. \tag{6.6}$$

A hard decision is taken by setting $\hat{x}_n = 0$ if $P_n(x_n) \geq 0$, and $\hat{x}_n = 1$ if $P_n(x_n) < 0$. If $\hat{x}H^T = 0$, the decoding process is finished with $\hat{x}_n$ as the decoder output; otherwise, repeat steps (1-3). If the decoding process doesn't end within predefined maximum number of iterations, $it_{\max}$, stop and output an error message flag and proceed to the decoding of the next data frame.

6.3. TDMP for Array LDPC

In TDMP, the array LDPC with j block rows can be viewed as concatenation of j layers or constituent sub-codes similar to observations made for AA-LDPC codes in [20]. After the check-node processing is finished for one block row, the messages are immediately used to update the variable nodes (in step 2, above), whose results are then provided for processing the next block row of check nodes (in step 1, above). We first illustrate the vector equations for TDMP for array LDPC codes assuming that the H matrix has the structure in (6.1.a).

[Initialization for each new received data frame]

$$\vec{R}_{l,n}^{(0)} = 0, \vec{P}_n = \vec{L}_n^{(0)}, \tag{6.7}$$

$\forall i = 1,2,\cdots,it_{\max}$, [Iteration loop]

$\forall l = 1,2,\cdots,j$, [Sub-iteration loop]

$\forall n = 1,2,\cdots,k$, [Block column loop]

$$\left[\vec{Q}_{l,n}^{(i)}\right]^{S(l,n)} = \left[\vec{P}_n\right]^{S(l,n)} - \vec{R}_{l,n}^{(i-1)}, \tag{6.8}$$

$$\vec{R}_{l,n}^{(i)} = f\left(\left[\vec{Q}_{l,n'}^{(i)}\right]^{S(l,n')}, \forall n' = 1,2,\cdots,k\right), \tag{6.9}$$

$$\left[\vec{P}_n\right]^{S(l,n)} = \left[\vec{Q}_{l,n}^{(i)}\right]^{S(l,n)} + \vec{R}_{l,n}^{(i)}, \tag{6.10}$$

where the vectors $\vec{R}_{l,n}^{(i)}$ and $\vec{Q}_{l,n}^{(i)}$ represent all the R and Q messages in each $p \times p$ block of the H matrix, $s(l,n)$ denotes the shift coefficient for the block in l^{th} block row and n^{th}

block column of the H matrix. $\left[\vec{Q}_{l,n}^{(i)}\right]^{S(l,n)}$ denotes that the vector $\vec{Q}_{l,n}^{(i)}$ is cyclically shifted down by the amount $s(l,n)$ and k is the check-node degree of the block row. A negative sign on $s(l,n)$ indicates that it is cyclic up shift (equivalent cyclic right shift). $f(\cdot)$ denotes the check-node processing, which can be done using BCJR or SP or OMS. For this work, we use OMS as defined in (2.6). If we are processing a block row in serial fashion using p check-node units (6.9), then the output of the CNU will also be in serial form. As soon as the output vector $\vec{R}_{l,n}^{(i)}$ corresponding to each block column n in H matrix for a block row l is available, this could be used to produce the updated sum $\left[\vec{P}_n\right]^{S(l,n)}$ (6.10). This could be immediately used in (6.8) to process block row $l+1$ except that the shift $s(l,n)$ imposed on $\vec{P}_n$ has to be undone and a new shift $s(l+1,n)$ has to be imposed. This could be simply done by imposing a shift corresponding to the difference of $s(l+1,n)$ and $s(l,n)$.

Note that due to the slight irregularity in array LDPC matrix defined in (6.1.b), each block row l has a node degree $j-l+1$. The variable-nodes in each block column n has a node degree equal to $\min(n,j)$. We have to devise a simple control mechanism to address this irregularity. One possible way to deal with this check-node irregularity is setting the check-node degrees in the CNU processor unit based on the block row that is being processed. Another simpler way to facilitate implementation is to assume that all the block rows have equal check-node degree and set the check-node messages corresponding to null blocks in H matrix to zero in order not to affect the variable-node processing. $\vec{R}_{l,n}^{(i)} = 0$ if $n < l$ in each iteration *i*. Similarly the variable-node messages belonging to the null blocks are always set to positive infinity in order not to affect the check-node processing. $\vec{Q}_{l,n}^{(i)} = \infty$ if $n < l$. For check-node update based on SP or OMS, the message with maximum reliability won't affect the CNU output. In the specific case of OMS, it is easy to see this as the CNU magnitude is dependent on the two least minimum.

6.4. Value-reuse Properties of OMS

This section presents the micro-architecture of serial CNU for OMS, which was used in our recent work on TPMP architecture [55]-[52],(see Chapters IV and V). The same CNU can be used in TDMP architecture presented in the next section. For the sake of continuity, the CNU is explained here again. For each check node m ,$\left|R_{mn}^{(i)}\right|$ $\forall n \in \mathrm{N}(m)$takes only two values, which are the two least minimum of input magnitude values. Since $\forall n \in \mathrm{N}(m)$, $\delta_{mn}^{(i)}$ takes a value of either $+1$or -1 and $\left|R_{mn}^{(i)}\right|$takes only 2 values, (6.3) gives rise to only three possible values for the whole set, $R_{mn}^{(i)}$ $\forall n \in \mathrm{N}(m)$ (chapter IV). In a VLSI implementation, this property significantly simplifies the logic and reduces the memory.

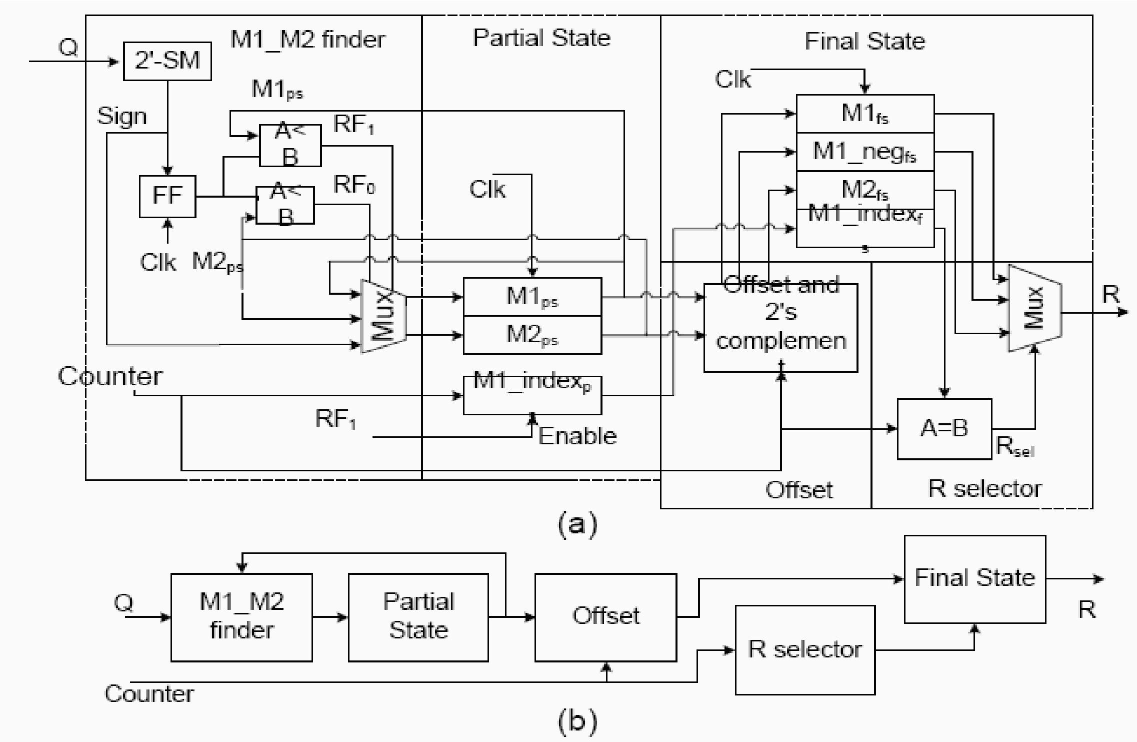


Fig 6.1. Serial CNU for OMS using value-reuse property.

Fig. 6.1(a) shows the CNU micro-architecture for (5, 25) code while Fig. 6.1(b) shows the block diagram of the same. In the first 25 clock cycles of the check-node processing, incoming variable messages are compared with the two up-to-date least minimum numbers (partial state, PS) to generate the new partial state, M1 which is the first minimum value, M2 which is the second minimum value and index of M1. The final state (FS) is then computed by offsetting the partial state. It should be noted that the final state include only *M1,-M1, +/-M2* with offset correction. Fig. 6.1(b) is the block diagram of the same architecture. M1_M2 finder computes the two least numbers, according to the incoming data and the current minimum numbers stored in partial state. The offset module applies the offset correction, and stores the results in the final state module. R selector then assigns one out of these 3 values, based on the index of M1 and the sign of

R message generated by sign XOR logic (6.4), to the output R messages. While the final state has dependency on offset correction, the offset is dependent on the completion of partial state. In operation, the final state and partial state will operate on different check-nodes. The serial CNU finds the least two minimum numbers with 2 comparators in a serial fashion and reduces the number of offset-correction computation from k to 2. Normally, CNU (check-node unit) processing is done using the signed magnitude arithmetic for (2.7) and VNU (variable-node unit processing) (2.9) is done in 2's complement arithmetic. This requires 2's complement to signed conversion at the inputs of CNU and signed to 2's complement at the output of CNU. In the proposed scheme, 2's complement is applied to only 2 values instead of k values at the output of CNU. The value re-use property also reduces the memory requirement significantly. Conventionally, the number of messages each CNU stores is equal to the number of edges it has, that is k. Now only four units of information are needed: the three values that $R_{mn}^{(i)}$ may take and the location of $M1_m^{(i)}$, then check-node message to the VNU is readily chosen by multiplexing.

6.5. Multi-rate Architecture Using TDMP and OMS

6.5.1. Block Serial Architecture

A new data flow graph is designed based on the TDMP, and on the value reuse property of min-sum algorithm described above (see Fig. 6.2.). For ease of discussion, we will illustrate the architecture for a specific structured code: array code of length 1525 described in section II, $j = 5$, $k = 25$ and $p = 61$, the discussion can be easily generalized to any other structured codes. First, functionality of each block in the architecture is explained. A check-node process unit (CNU) is the serial CNU based on OMS described in the previous section. The CNU array is composed of *p* computation units that compute the partial state for each block row to produce the R messages in block serial fashion. Since final state of previous block rows, in which the compact information for CNU messages is stored is needed for TDMP, it is stored in register banks.

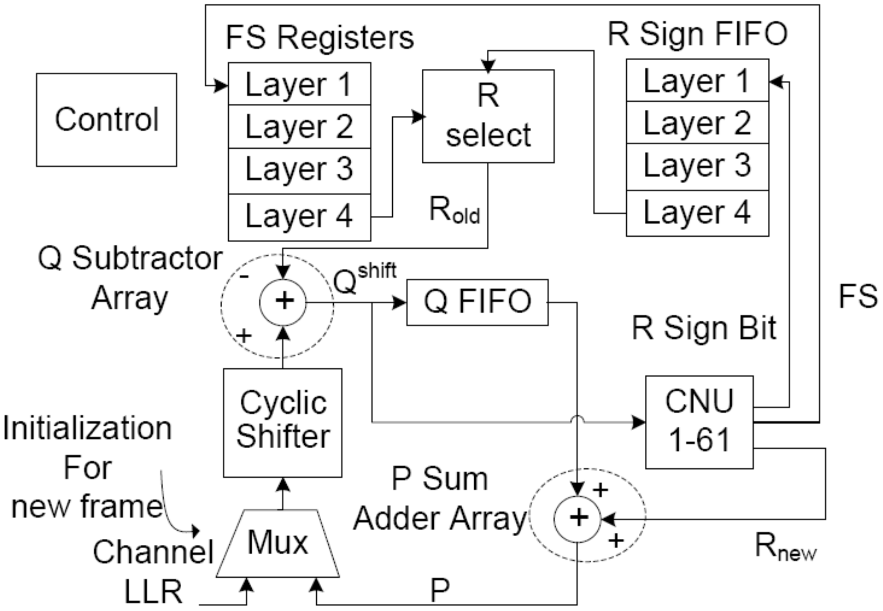


Fig. 6.2. LDPC decoder using layered decoding and OMS

There is one register bank of depth $j-1$, which is 4 in this case, connected with each CNU. Each final state is the same as the final state register bank in the CNU. Besides the shifted Q messages, the CNU array also take input of the sign information for previous computed R messages in order to perform R selection operation. The sign bits are stored in sign FIFO. The total length of sign FIFO is k and each block row has p one bit sign FIFOs. We need $j-1$ of such FIFO banks in total. p number of R select units is used for

R_{old} . An R select unit generates the R messages for $25(=k)$ edges of a check-node from three possible values stored in final state register associated with that particular check-node in a serial fashion. Its functionality and structure is the same as the block denoted as R select in CNU. This unit can be treated as de-compressor of the check node edge information which is stored in compact form in FS registers. The generation of R messages for all the layers in this way amounts to significant memory savings, which would be quantified in a later section. The shifter is constructed as cyclic down logarithmic shifter to achieve the cyclic shifts specified by the binary encoded value of the shift. The logarithmic shifter is composed of $\log 2(p)$ stages of p switches. Since cyclic up shift is also needed in the operation of the decoder, cyclic up shift by u can be simply achieved by doing cyclic down shift with $p-u$ on the vector of size p . The decoding operation proceeds as per the vector equations described in section III. In the beginning of the decoding process, P vector is set to receive channel values in the first k clock cycles (i.e. the first sub-iteration) as the channel values arrive in chunks of p , while the output vector of R select unit is set to zero vector. The multiplexer array at the input of cyclic shifter is used for this initialization. The CNU array takes the output of the cyclic shifter serially, and the partial state stage will be operating on these values. After k clock cycles, partial state processing will be complete and the final state stage in CNU array will produce the final state for each check-node in 2 clock cycles. Then R select unit within the each CNU unit starts generating k values of check-node messages in serial fashions. The CNU array thus produces the check-node messages in a block serial fashion as there are p CNUs are operating in parallel. The P vector is computed by adding the delayed version of the Q vector (which is stored into a FIFO SRAM to till the serial CNU produces the output) to the output vector R of the CNU. Note that the P vector that is generated can be used immediately to generate the Q vector as the input to the CNU array as CNU array is ready to process the next block row. This is possible because CNU processing is split into three stages as shown in the pipeline diagram and partial state stage and final state stage can operate simultaneously on two different block rows. Now, the P message vector will undergo a cyclic shift by the amount of difference of the shifts of the block row that is processed, and the previous block row that was just processed. This shift value can be either positive or negative indicating that a down shift or up shift need to be performed by the cyclic shifter. The shifted P sum messages are subtracted by R message to get the shifted version of Q messages.

The snapshot of the pipeline of the decoder is shown in Fig. 6.3.a. and 6.3.b Here, the partial state stage in CNU (CNU PS) is operating on the 2nd block row from clock cycles labeled as 0 to 24 (note that these numbers will not denote the actual clock numbers as the snapshot is shown in the middle of the processing). Final state stage in CNU (CNU FS) can not start until the end of PS processing, that is clock cycle 25. As soon as the FS is done in clock cycle 26, R select is able to select the output R messages, and P and Q messages processing starts. With the first block of Q message ready, PS for the next block row can be started immediately. Note that all the logic blocks (other than the storage elements) are active over 90% of the time. The only exception is the offset module, which is composed of two 5-bit adders, in each CNU. The overall proportion of all the CNU FS logic in the overall decoder is less than 4%.The control unit also contains the information of array code parameters such as j,k,q– these could be changed to support multi-rate decoding. The family of rate-compatible codes is obtained by successively puncturing the left most p columns and the topmost p rows in the H matrix defined in (1b) q times. Changing q from 0 to 3(=j -2) gives the code rates of 0.8 to 0.9. Changing k values from 15 to 61 while fixing $j=5$ results in code rates from 0.666 to 0.91. The Q FIFO needs to be of maximum depth p as the k can take a maximum value equal to p.

6.5.2. Scalable Architecture

Note that the throughput of the architecture is increased by increasing p of the code, and scaling the hardware accordingly. While the complexity of computational units scale linearly with p, the complexity of cyclic shifter increases with the factor $(p/2)\log_2 p$. So, it is necessary to change the architecture for large values of p. Alternatively it may be needed in low throughput applications to have low parallelization. To suit this requirement, minor changes in the proposed architecture are necessary. Let us assume the desired parallelization is $M < p$. For the ease of implementation, choose M close to the powers of 2. The cyclic shifter needed is $M \times M$. Since it is needed to achieve $p \times p$ cyclic shift with consecutive shifts of $M \times M$, it is necessary that the complete vector of size p is available in M banks with the depth $s = (ceil(p/M))$ and shifting is achieved in part by the cyclic shifter, and in part by the address generation. Now, all the CNU and variable node processing is done in a time division multiplexed fashion for each sub-vector of length M, so as to process the vector of size p to mimic the pipeline in Fig. 6.3. Now, instead of taking one clock cycle to process a block column in a block row

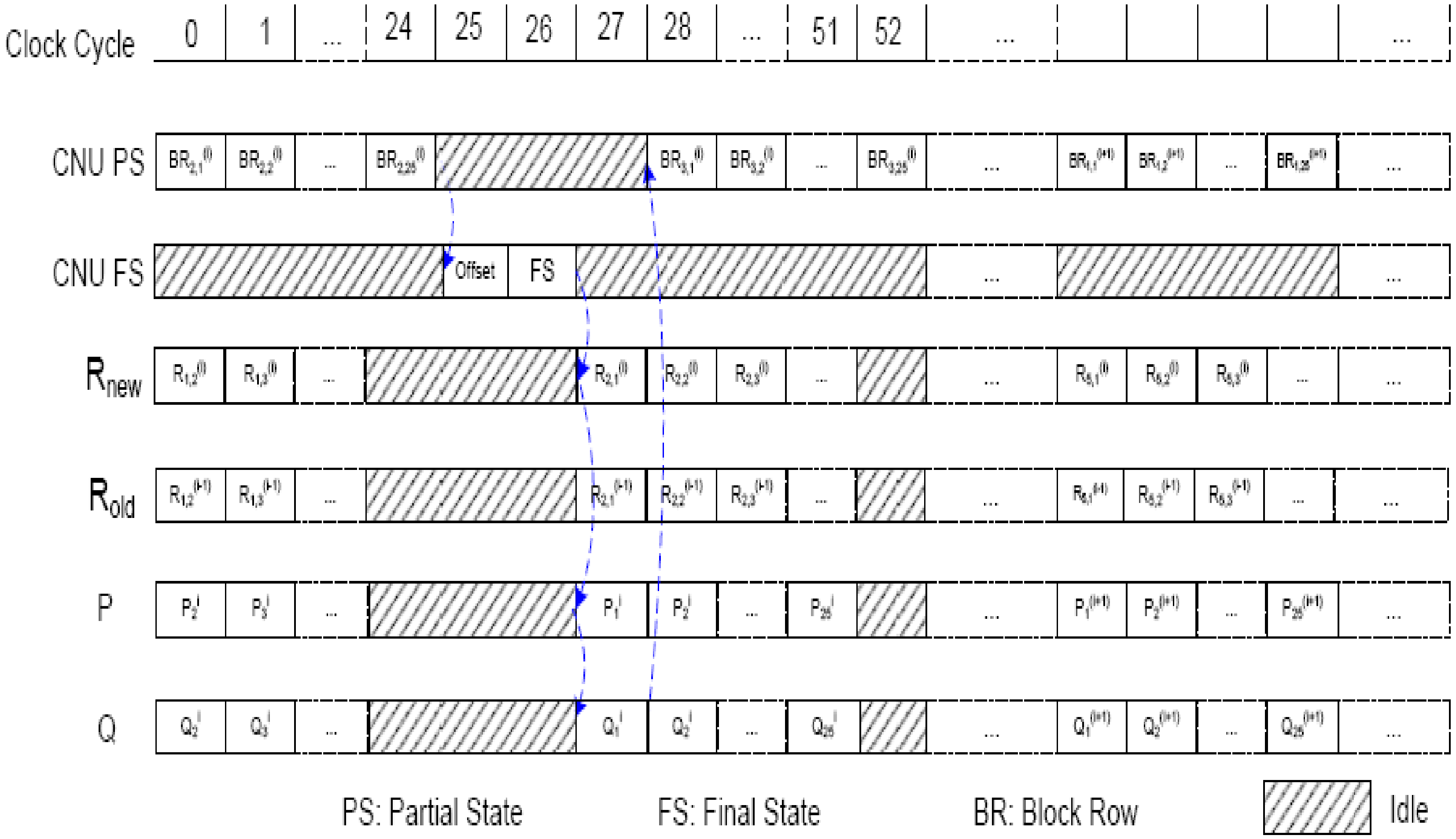


Fig. 6.3. Block serial processing and 3-stage pipelining for TDMP using OMS a) Detailed Diagram

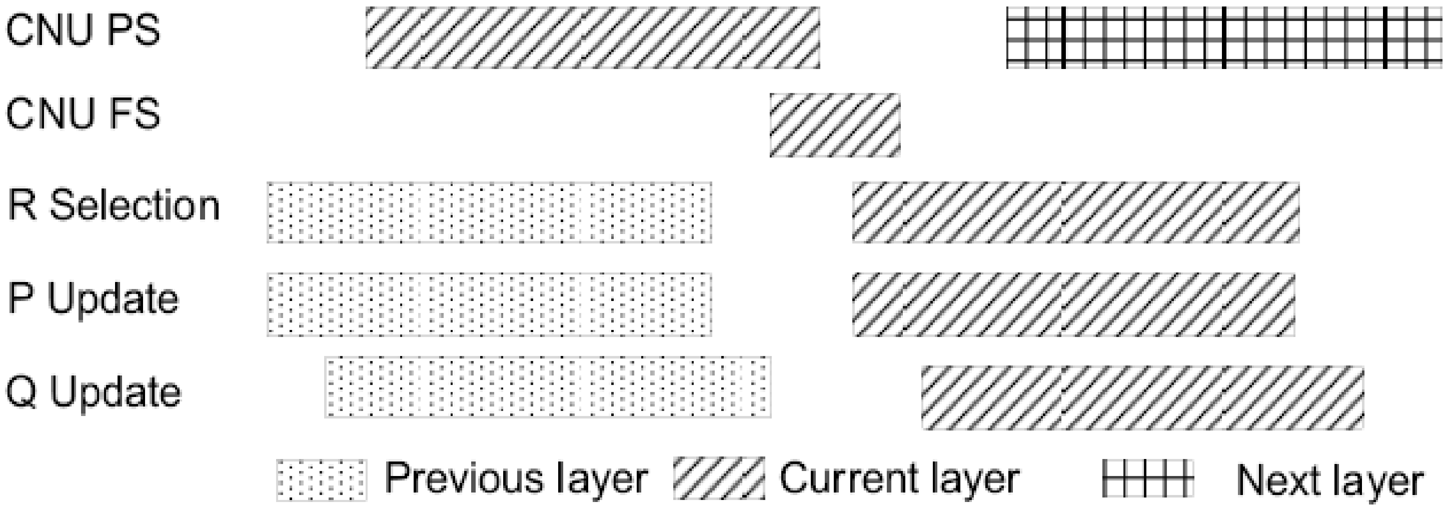


Fig. 6.3 continued b) Simple diagram

(layer), it takes s clock cycles. The FS register bank external to the CNU and FS registers in CNU are now implemented as M banks of memory with depths equal to js and word length is equal to the total number of bits to represent the four information entities of final state of each check node (FS) viz, M1, -M1, M2 and M1 Index. In addition, we need another memory with M banks with depth equal to s to store the partial state (M1, M2, M1 Index and cumulative sign). Channel values need to be stored in a buffer of size p as the decoder needs any M values out of this buffer at each clock cycle. Note that this architecture is consuming less numbers of logic when compared to fully scaled architecture, its memory requirements increased slightly. One way to look at the memory requirements of the scalable architecture ($M < p$) is: all the logic resources, FS registers, and PS registers are scaled down from p to M ,while having an external SRAM of size equal to the number of FS, and PS registers used in the case of fully scaled architecture (i.e. $M = p$). Note that the exact memory bank organization can be changed by grouping different messages together, so less numbers of memory banks are possible.

6.6. Implementation Results and Discussion

We prototyped the proposed multi-rate decoder architectures on Xilinx Virtex 2V8000-5 device. The synthesis results and performance comparison with other recent state of the art implementations are given in Table 6.1. More details on FPGA implementation for required memory is given in Tables 6.2 and 6.3. The work in [42] can be directly compared with the present work as both target different coding rates and support code lengths of up to 10,000 and the BER performance of the regular codes considered in both the architectures is similar. The work in [38] and [22] only supports one fixed code length. Similarly the work in [41] supports one code rate only while supporting different lengths from 1000-3000. The amount of memory that needs to be used in multi-rate architectures is dependent on the maximum values of code parameters that need to be supported. Also note that almost all the recent semi-parallel architectures [3]-[8] are based on LDPC codes constructed from cyclically shifted identity matrices for the ease of implementation. So, for a fair and uniform comparison, memory savings in the next paragraphs are calculated on assuming the same kind of code parameters (size of identity matrix used, *p*, check node degree, *k*, variable node degree, *j* and code length, N).

We also implemented the proposed decoder architecture using the open source standard cells v*sclib013* [62] in 130 nm technology. The synthesis is done using synopsys design analyzer tool, while layout is done using cadence's silicon ensemble tool. Tables 6.4, 6.5 and 6.6 give the performance comparison as well as the decoder chip characteristics. The original TDMP decoder [20] is based on more complicated BCJR algorithm. The CNU for BCJR takes more area due to the need of several internal FIFOs. In addition, Omega network is used in [20] instead of logarithmic shifter. The use of logarithmic shifter saves area to store the control signals as well as the the absence of control wires make the logarithmic shifter's layout much more compact. The proposed decoder has a frequency advantage also, as the CNU stage has 3 pipeline stages. The decoder in [20] has fewer pipeline stages.

6.6.1. Memory Savings

1) Block Serial Architecture

Consider the proposed implementation for (5,*k*) array LDPC codes. The parameters for this family of codes are $j = 5$, $k = 10,11,\cdots k_{\max}(= 60)$, $p = 61, q = 0,1,2,3.$ where $k_{\max}$ is the maximum check-node degree of all the codes that need to be supported. The proposed TDMP architecture features large memory savings, up to 2x throughput advantage, as well as 50% less interconnection complexity. The TDMP permits us to use a running sum, which is initialized to channel log-likelihood ratio (LLR) values in the first iteration. So, there is no memory needed to store the channel LLR values as these values are implicitly stored in the *Q* messages. Since the maximum number of *Q* messages that need to be stored are equal to $k_{\max} \times p \times 5$, as opposed to storing $5 j k_{\max} p$ messages in TPMP architectures. So, the total savings in Q memory are 80% as a direct result of employing TDMP proposed in [3].

The proposed architecture offers further advantages. Instead of storing all the R messages, the compressed information, *M1*, -*M1*, +/-*M2*, and index of *M1* is stored (FS memory). R select unit can generate the R message by the use of an index comparator and the sign bit of the R message which comes from the sign FIFO. This results in a reduction of around 50%-90% of R memory based on the rate of the code when compared to BCJR algorithm or Sum of Products algorithm. The total savings in R memory is

$$\frac{5 j k_{\max} p - \left[k_{\max} + 5 \times 3 + \lceil \log 2(k_{\max}) \rceil\right] jp}{5 j k_{\max} p} \times 100\% .$$

The factor 5 comes due to the use of 5-bit quantization for R messages. So the savings of R memory for (5,k) codes with $k_{\max} = 61$is 73%

Also note that, due to the nature of block serial scheduling and the principle of on-the-fly computation in the architecture, there is no need to store the P messages. The total savings of memory bits is $\left(k_{\max} \times p \times 6\right)$. Note that the factor 6 comes due to the number of bits used to represent the P message.

The total savings in memory for the proposed block serial architecture, accounting for R memory, Q memory, and P memory, when compared to TPMP architectures based on SP and min-sum [18], [37], [41] is **82%**. When compared to TDMP-BCJR architecture [20] and TDMP-SP architecture [48], the total memory savings due to our TDMP-OMS architecture is **72%** since all TDMP architectures have the same savings in Q memory. The total memory needed in our prototyped block serial architecture is 37210 bits.

2) Scalable Architecture

In the case of scalable architecture, the above savings will apply for R memory, Q memory. The savings for P memory will change slightly. Due to the nature of block serial scheduling in the architecture, there is only the need to store the P messages for only two blocks. The total savings of memory bits is $\frac{\left(6k_{\max} p - 2 \times 6 \times p\right)}{6k_{\max} p} \times 100\%$. Note that the factor 6 comes due to the number of bits used to represent the P message.

Note that the scalable architecture features a partial state memory when compared to the block serial architecture. However, this is small as it must contain the partial state for only one block row at any time, is equal to $\left[k_{\max} + 5 \times 2 + \left\lceil \log 2\left(k_{\max}\right)\right\rceil\right] p$ bits.

Consider the proposed implementation for (3,*k*) array LDPC codes. The parameters for this family of codes are $j = 3$, $k = 6,7,\cdots k_{\max}\left(= 32\right)$, $p = 347, q = 0,1.$ The total savings in memory for the proposed scalable architecture, accounting for R memory, Q memory, and P memory, when compared to TPMP architectures based on SP and min-sum [6]- [8], is **67%**. When compared to TDMP-BCJR architecture [3] and TDMP-SP [4], the total memory savings due to our TDMP-OMS scalable architecture is **54%** since all TDMP architectures have the same savings in Q memory. The total memory needed in our prototyped scalable architecture is 131860 bits.

6.6.2.. Savings in Logic

We designed a low complexity serial CNU based on OMS using the value reuse properties. This has significant logic savings when compared to other implementations of SP, BCJR. There is no need of costly look up tables as in the case of SP. There is no need of internal message FIFOs as in the case of BCJR. A parallel CNU can also be designed based on these value re-use properties [55], (Chapter IV) which out performs the parallel CNU presented in [20]. In addition, we used the properties of layered decoding and array codes to reduce the complexity from two cyclic shifters [20], [38], [51], [37], [41] to one cyclic shifter. The removal of an *M* x *M* cyclic shifter for 6 bit messages results in a savings of $6M \log 2(M)$ multiplexers and associated wiring congestion.

6.6.3.. Throughput

Each TDMP iteration consists of *j* sub-iterations and each sub-iteration takes *k(=check node degree)* clock cycles as defined in sections 6.2 and 6.3. Note that *k* can take any value that is less than $k_{\max}$ supported by the decoder implementation. This feature along with the ability to control the number of layers by puncturing with the parameter *q*, makes the decoder to decode a wide range of different rate compatible array LDPC codes.

User data throughput t_u is given by $t_u = rate \times t_d = ((k - j + q)/k)t_d$, (6.11)

where t_d is decoded throughput and is given by

$$t_d = Nf/(it_{\max} CCI) = pkf/(it_{\max} \times CCI), \tag{6.12}$$

where *f* is the decoder chip frequency and *CCI* stands for number of clock cycles required to complete one iteration.

$$CCI = (j - q)CCL, \tag{6.13}$$

where CCL stands for the number of clock cycles to process one layer and is given by

$$\mathrm{CCL} = \mathrm{k}\lceil \mathrm{p/M} \rceil + 2. \tag{6.14}$$

To achieve the same BER as that of the TPMP schedule on SP (or equivalent TPMP schedule on BCJR), the TDMP schedule on OMS needs half the number of iterations (Fig. 6.4) having similar convergence gains reported for TDMP-BCJR [20] and TDMP-SP [22]. However, the choice of finite precision OMS results in a performance degradation of less than 0.2 dB. 5-bit uniform quantization for R and Q messages and 6-bit uniform quantization for P messages is used. The step size for quantization, Δ, and offset parameter, β are set based on the code parameters [32].

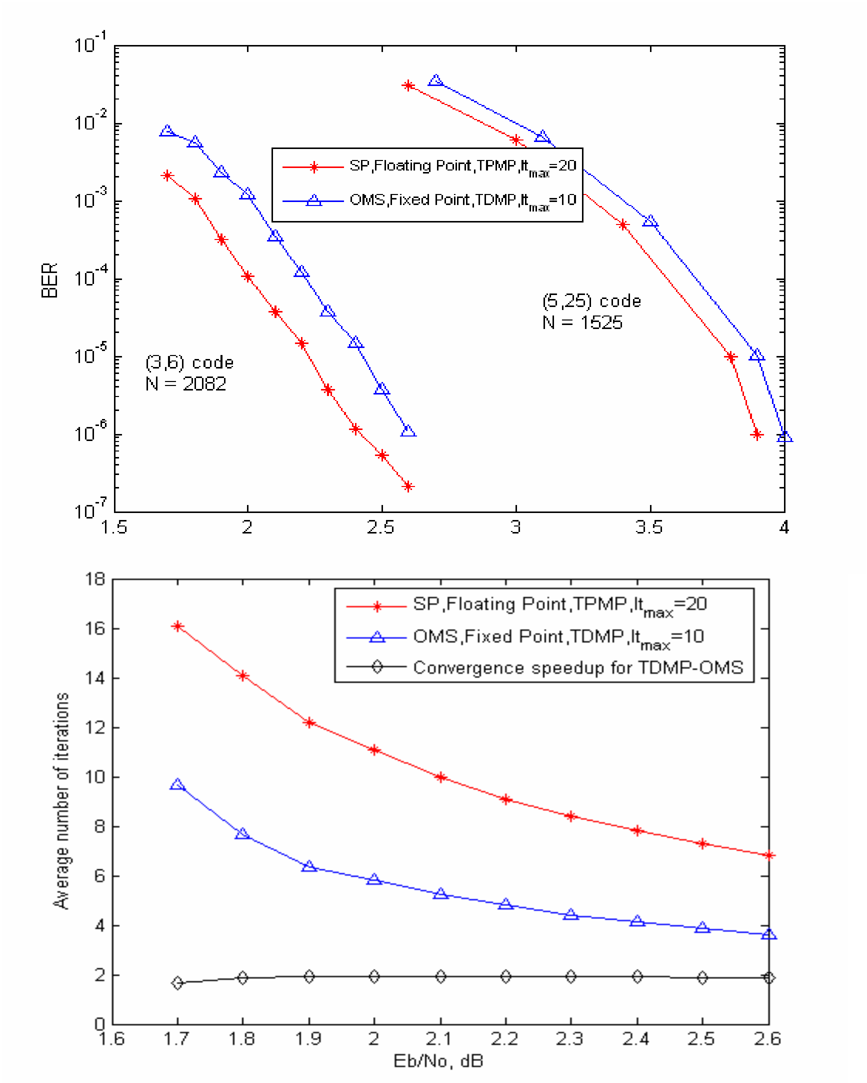


Fig. 6.4. (a) Bit error rate performance of the proposed TDMP decoder using OMS(j=3,k=6,p=347,q=0) array LDPC code of length N=2082 and (j=5,k=25,p=61,q=0) array LDPC code of length N=1525. (b) Convergence speed up of TDMP-OMS over TPMP-SP. Results shown for (j=3,k=6,p=347,q=0) array LDPC code of length N=2082. Here It$_{max}$= maximum number of iterations.

6.7. Conclusion

We present memory efficient multi-rate decoder architecture for turbo decoding message passing of structured LDPC codes, using the min-sum algorithm for check-node update. Our work offers several advantages, when compared to the other state of the art LDPC decoders, in terms of significant reduction in logic, memory, and interconnect. This work retains the key advantages offered by the original TDMP work – however our contribution is in using the value-reuse properties of OMS algorithm and devising a new TDMP decoder architecture to offer significant additional benefits. The contribution of this work is in using the value-reuse properties of OMS algorithm to reduce the memory storage requirement up to 70% and devising a new TDMP decoder architecture to reduce the router requirements by 50% and reduce the memory requirements for the storage of sum of channel values and variable node reliability messages. We also presented the variation of the architecture, scalable architecture that offers a throughput-hardware resource trade-off.

Table 6.1

FPGA implementations and performance comparison

	T. Brack Et al.[41]	D. Hocevar et al. [38],[22]	L. Yang et al. [42].	Optimally scaled, parallelization (*M*)=*p=61*	Scalable *(M<p* *M=61* *p=347)*
Slices	11,729	NA[1],	34,127	6,002	6,182
LUTs	NA	72,621	24,570	7,713	8,022
Slice flip-flops	NA	6,779	53,327	9,981	10,330
BRAM	76	32	102	12	129
Actual Memory used (in bits)	NA	173,892 (86 x 337 x 6)	566,784	37,210	131,860
Xilinx FPGA Device used[2]	Virtex4-LX100	Virtex2-V8000	Virtex2-V8000	Virtex2-V8000	Virtex2-V8000
Frequency	100	44	100	112	112
Code rate	0.8	0.5	5/8,7/8,1/2	0.5-0.9167	0.5-0.9063
Code parameters Supported,	Code length N= 1000-3000	Fixed N=8888, *p=337,* *Number of non-zero blocks in H,* N_{nz}*=86*	Different Coding Rates N=10,000	*p* = 61, *j* = 5, *k* = 10,..61, *q* = 0,…,3 N=*pk*= 610-3,721 *Max.* N_{nz}=295	*p* = 347, *j* = 3, *k* = 6,..32, *q* = 0,1 N=*pk*= 2082-11,104 *Max.* N_{nz}=93
Code Construction	Structured	Structured	Structured	array LDPC	array LDPC
Check Node Update	Normalized MS	SP	SP	OMS	OMS
Decoding Schedule	TPMP	TPMP or TDMP	TPMP	TDMP	TDMP
Maximum Iterations	10 TPMP	25 TPMP [22] 12 TDMP [38]]	24 TPMP	10 TDMP	10 TDMP
User Data Throughput, t_u	180 Mbps	40 Mbps [22] 80 Mbps [38]	66 Mbps	68-329 Mbps	113-319 Mbps

[1] One Virtex-2 slice has 2 LUTs, 2 slice flip-flops and other logic. [2] Virtex 4 family has higher capabilities than Virtex 2 family.

Table 6.2

Memory implementation for block serial architecture ($j=5,k=10,\ldots,k_{max}$ (=61), $p=61,M=p$)

	Implementation	Required Memory in Bits	Number of BRAMs	Data access(bits), per clock cycles, Read,r,/Write,w.
R memory(External FS registers, *j-1* layers)	Register bank FIFO of depth *j-1*	$\left[5\times 3+\lceil \log 2(k_{\max})\rceil\right](j-1)p$	LUTs are configured as shift registers (320 LUTs are used to store 5124 bits)	1281, *k, w* 305 , 1, *r*
R memory(Internal FS and PS registers,1 layer)	Registers as part of CNU	FS: $\left[5\times 3+\lceil \log 2(k_{\max})\rceil\right]p$ PS: $\left[5\times 2+\lceil \log 2(k_{\max})+1\rceil\right]p$	Slices flipflops are used (2318 slice flip flops to store 2318 bits)	1281, *k, w* 305 , 1, *r* 1037(max),1,*w* 1037,1,r
R memory, Sign FIFO(*j-1 layers*)	2 Dual Port BRAM	$k_{\max}(j-1)p$	1, Width =31, Depth = 244 1, Width = 30, Depth = 244	61 ,1,*w* 61 ,1,*r*
Q FIFO	10 Dual Port BRAM	$5k_{\max}p$	9, Width =30, Depth = 61 1, Width = 35, Depth = 61	305 ,1,*w* 305 ,1,*r*
P Memory	NA	0	Not needed	0

Table 6.3

Memory implementation for scalable architecture (*j=3,k=6,…,k$_{max}$ (=32),p=347,M=61*)

	Implementation	Required Memory in Bits	Number of BRAMs
R memory(External FS registers, *j-1* layers)	34 Dual Port BRAM	Same formula as in Table 6.2	34, Width =36, Depth = 12
R memory(Internal FS and PS registers,1 layer)	62 Dual Port BRAM	Same formula as in Table 6.2	34, Width =36, Depth = 6 28, Width =36, Depth = 6
R memory, Sign FIFO(*j-1 layers*)	2 Dual Port BRAM	Same formula as in Table 6.2	1, Width =31, Depth = 384 1, Width = 30, Depth = 384
Q FIFO	10 Dual Port BRAM	Same formula as in Table 6.2	9, Width =30, Depth = 192 1, Width = 35, Depth = 192
P Memory	21 Dual Port BRAM	$2\times 6\times p$	21, Width= 36, Depth =6

Table 6.4

ASIC Implementation of the proposed TDMP multi-rate decoder architecture

	Semi-Parallel multi-rate LDPC decoder [20]	Multi-rate TDMP Architecture regular QC-LDPC
LDPC Code	AA-LDPC, (3,6) code, rate 0.5, length 2048	((5,*k*) rate compatible array codes *p=61*. *k*=10,11,..61length *=pk(610,.,3721)*
Decoded Throughput, t_d,	640 Mbps	590 Mbps
Area	14.3 mm^2	1.6 mm^2
Frequency	125 MHz	500 MHz
Nominal Power Dissipation	787 mW	257 mW
Memory	51,680 bits	37,210 bits
CMOS Technology	180 nm, 1.8V	130 nm, 1.2V
Decoding Schedule	TDMP, BCJR, it_{max}=10	TDMP, OMS, it_{max}=10
Area Efficiency for t_d,	44.75 Mbps/mm^2	369 Mbps/ mm^2
Energy Efficiency for t_d,	123 pJ/Bit/Iteration	44.2 pJ/Bit/Iteration
Est. Area for 180 nm	14.3 mm^2	~3.06 mm^2
Est. Frequency for 180 nm	125 MHz	~360 MHz
Est. Decoded Throughput(t_d) , 180 nm	640 Mbps	426 Mbps
Est. Area Efficiency for t_d, 180 nm	44.75 Mbps/mm^2	139.2 Mbps/mm^2
Energy Efficiency for t_d, 180 nm	123 pJ/Bit/Iteration	99.45 pJ/Bit/Iteration
Application	Multi-rate application as well as fixed code application	Multi-rate application as well as fixed code application Rate-compatible array codes are considered for DSL applications.
Bit error rate Performance	Good	Good and similar to AA-LDPC

it_{max}= Maximum number of iterations.

Table 6.5

Area distribution of the chip for (5, *k*) rate compatible array codes, 130 nm (note that the CNU array includes CNUs as well as FS registers)

	Area (mm^2)
CNU Array	0.67
VNU Array	0.05
Memory	0.71
Pipeline flip-flops	0.02
Cyclic shifter	0.08
Wiring	0.07
Total chip area	1.6

Table 6.6

Power distribution of the chip for (3,*k*) rate compatible array codes, 130 nm

	Power (mW)
Logic(CNU,VNU and shifters)	162.7
Memory	45.4
Leakage	~0.1
Clock	32.5
Wiring	16.2
Total	257.0

CHAPTER VII

MULTI-RATE TDMP ARCHITECTURE FOR IRREGULAR QC-LDPC CODES

7.1. Introduction

In this work, we propose to apply TDMP for the offset MS for block LDPC codes used in IEEE 802.16e (Mobile WiMax) and IEEE 802.11n (High speed wireless local area network). WiMax technology involves microwaves for the transfer of data wirelessly. It can be used for high-speed, mobile wireless networking at distances up to a few miles. The main contribution of this work is an efficient architecture that utilizes the value–reuse property of OMS, cyclic shift property of structured LDPC codes and enhancement of our previous work of block serial scheduling [51], (Chapter III). The proposed architecture utilizes the value–reuse property of offset min-sum, block-serial scheduling of computations and turbo decoding message passing algorithm. The decoder has the following advantages: 55% savings in memory, reduction of routers by 50%, and increase of throughput by 2x when compared to the recent state-of-the-art decoder architectures.

The rest of the chapter is organized as follows. Section 7.2 gives the background about structured block LDPC codes, and TDMP. The data flow graph and architecture for TDMP using offset MS is shown in Section 7.3. Section 7.4 presents the FPGA and ASIC implementation results and discussion. Section 7.5 concludes the chapter.

7.2. LDPC Codes and Decoding

7.2.1. Block LDPC Codes of WiMax

The block irregular LDPC codes have competitive performance and provide flexibility and low encoding/decoding complexity [12]. The entire *H* matrix is composed of the same style of blocks with different cyclic shifts, which allows structured decoding and reduces decoder implementation complexity. Each base *H* matrix in block LDPC codes has 24 columns, simplifying the implementation. Having the same number of columns between code rates minimizes the number of different expansion factors that have to be supported. There are four rates supported: 1/2, 2/3, 3/4, and 5/6, and the base *H* matrix for these code rates are defined by systematic fundamental LDPC code of M_b-by-N_b where M_b is the number of rows in the base matrix and N_b is the number of columns in the base matrix. The following base matrices are specified: 12 x 24, 8 x 24, 6 x 24, and 4 x 24. The base model matrix is defined for the largest code length (*N* = 2304) of each code rate. The set of shifts in the base model matrix are used to determine the shift sizes for all other code lengths of the same code rate. Each base model matrix has 24 $(= N_b)$ block columns and M_b block rows. The expansion factor *z* is equal to *N*/24 for code length *N*. The expansion factor varies from 24 to 96 in the increments of 4, yielding codes of different length. For instance, the code with length *N* = 2304 has the expansion factor *z*=96 [12]. Thus, each LDPC code in the set of WiMax LDPC codes is defined by a matrix *H* as

$$H = \begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,N_b} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,N_b} \\ \cdots & \cdots & \cdots & \cdots \\ P_{M_b,1} & P_{M_b,2} & \cdots & P_{M_b,N_b} \end{bmatrix} = P^{H_b} \tag{7.1}$$

where $P_{i,j}$ is one of a set of *z-by-z* cyclically right shifted identity matrices or a *z-by-z* zero matrix. Each 1 in the base matrix H_b is replaced by a permuted identity matrix while each 0 in H_b is replaced by a negative value to denote a *z-by-z* zero matrix.

7.2.2. Block LDPC Codes of IEEE 802.11n

These codes have the same structure as the Block LDPC codes of WiMax. The expansion factor, defined as the size of the identity matrix *z* can be 27, 54 or 81 [13]. All the base matrices have the same number of block columns $N_b = 24$, and the code length N is $N_b \times z$ .The code lengths (648; 1296 and 1944) and all the code rates (1/2; 2/3; 3/4 and 5/6) are specified in IEEE 802.11n standard draft [13].

In TDMP, the block LDPC with j block rows can be viewed as concatenation of j layers or constituent sub-codes similar to observations made for AA-LDPC codes in [20]. In TDMP, after the check-node processing is finished for one block row, the messages are immediately used to update the variable nodes, whose results are then provided for processing the next block row of check nodes. This differs from TPMP, where all check nodes are processed first and then the variable-node messages will be computed. Each decoding iteration in the TDMP is composed of j number of sub-iterations. In the beginning of the decoding process, variable messages are initialized as channel values and are used to process the check nodes of the first block row. After completion of that block row, variable messages are updated with the new check- node messages. This concludes the first sub-iteration. In similar fashion, the result of check-node processing of the second block row is immediately used in the same iteration to update the variable-node messages for third block row. The completion of check-node processing and associated variable-node processing of all block rows constitutes one iteration.

The TDMP can be described with (7.2-7.5):

[Initialization for each new received data frame],

$$\vec{R}^{(0)}_{l,m} = 0, \vec{P}_m = \vec{L}^{(0)}_m \ , \ \forall l = 1, 2, \cdots, j \, , \forall m = 1,2,\cdots, N_b \tag{7.2}$$

$\forall i = 1,2,\cdots, it_{\max}$, [Iteration loop]

$\forall l = 1, 2, \cdots, j$, [Sub-iteration loop]

$\forall n = 1,2,\cdots, k$, [Block column loop]

$$\left[\vec{Q}^{(i)}_{l,n}\right]^{S(l,n)} = \left[\vec{P}_n\right]^{S(l,n)} - \vec{R}^{(i-1)}_{l,n}, \tag{7.3}$$

$$\vec{R}^{(i)}_{l,n} = f\left(\left[\vec{Q}^{(i)}_{l,n'}\right]^{S(l,n')}\right), \forall n' = 1,2,\cdots, k \, , \tag{7.4}$$

$$\left[\vec{P}_n\right]^{S(l,n)} = \left[\vec{Q}^{(i)}_{l,n}\right]^{S(l,n)} + \vec{R}^{(i)}_{l,n} \ , \tag{7.5}$$

where the vectors $\vec{R}^{(i)}_{l,n}$ and $\vec{Q}^{(i)}_{l,n}$ represent all the R and Q messages in each non-zero block of H matrix, $s(l,n)$ denotes the shift coefficient for the l^{th} block row and n^{th} non-zero block of the H matrix (note that null blocks in the H matrix need not be processed); $\left[\vec{R}^{i-1}_{l,n}\right]^{S(l,n)}$ denotes that the vector $\vec{R}^{i-1}_{l,n}$ is cyclically shifted up by the amount $s(l,n)$, k is the check-node degree of the block row. A negative sign on $s(l,n)$ indicates that it is cyclic down shift (equivalent cyclic left shift). $f(\cdot)$ denotes the check-node processing, which can be done using BCJR, SP or MS. For the proposed work we use OMS as defined in

Chapter II and IV. In addition, we apply the connection for a message also to have two dimensional correction for min-sum decoding. [33].

7.3. Multi rate Decoder Architecture Using TDMP and OMS

7.3.1. Architecture Description

A new data flow graph is designed based on the TDMP and on the value-reuse property of the OMS algorithm described in Chapter IV. For ease of discussion, we will illustrate the architecture for the specific structured code denoted as rate ¾ code A. Note that all the codes have the same number of block columns. By changing the parameter *k* supplied to the CNU and by varying the parameter *j*, the number of block rows to be processed, this architecture supports all the codes in the 802.16e standard. For rate 3/4 code A of length 1152 has *j* =6 block rows and check-node degree of the 6 block rows is given by *k_v*= [13 12 12 12 12 13] and the block size is $z = 48$. Assume that the desired parallelization is *M*=24. The cyclic shifter needed is $M \times M$. The $z \times z$ cyclic shift is achieved with cyclic shifts of $M \times M$ in combination with the appropriate address generation and this works for only z=24,48 and 96. The complete P vector of size z is available in M memory banks of depth $s = ceil(z / M) = 2$. The shifter is constructed as a cyclic down logarithmic shifter to achieve the cyclic shifts specified by the binary encoded value of the shift. The logarithmic shifter is composed of $\log 2(M)$ stages of M 2-in-1 multiplexers. Cyclic up shift by u can be simply achieved by doing cyclic down shift with $z - u$ on the vector of size. Say now if we want to change the parallelization *M* to 48. If we construct a single 48 x 48 cyclic shifter, it can only handle z=48. So, we use two 24 x 24 cyclic logarithmic shifters to construct the 48 x 48 shifter while being able to work as two independent 24x24 shifters to support the expansion factor z=24. We need to introduce some additional multiplexers to achieve this. This way the decoder can support the expansion factors of 24, 48 and 96. Similarly, the cyclic shifter implementation for M=96, is constructed out of 4 24 x 24 cyclic logarithmic shifters. One should note that it is not possible to achieve cyclic shifts specified by $s(l,n)$,(=0,1,..z-1) on a vector of length z with a cyclic shifter of size $M \times M$ if M is not a integer multiple of z, z = M. This issue will be dealt with the use of a master-slave Benes network as explained later.

7.3.2. Decoder Operation

Details on CNU processing are given in Fig 7.1. The decoder architecture is presented in Fig. 7.2. Notice the similarity with the data flow graph for regular QC-LDPC codes. (Fig.6.2.) All the check-node processing and variable-node processing is done in a time division multiplexed fashion for each sub-vector of length as shown in Fig. 7.3. To process a block in a block row (layer), it takes s clock cycles. A check-node process unit (CNU) is the serial CNU based on OMS described in the previous section. The CNU array is composed of M serial CNUs described in Chapter IV. As shown in the pipeline (Fig. 7.3), the CNU array operates on the R messages and partial states of two adjacent block rows. While the final state has dependency on partial states, P and Q messages are dependent on the final states. Since the final state of the previous block rows, in which the compact information for CNU messages is stored, is needed for TDMP. It is stored in the FS memory. There is one memory bank of depth j, which is 12 in this case, connected with each CNU. The FS memory for the entire CNU array is implemented as M banks of memory with depth js and word length 20 bits, constituted of {M1, -M1, +/-M2} with offset correction, and M1 index. In addition, we need another memory with

M banks with depths equal to s to store the partial state, with the word length 16 bits as we need to store and retrieve (M1, M2, M1 index and cumulative sign). Note that we need to store partial state for only one block row at any time.

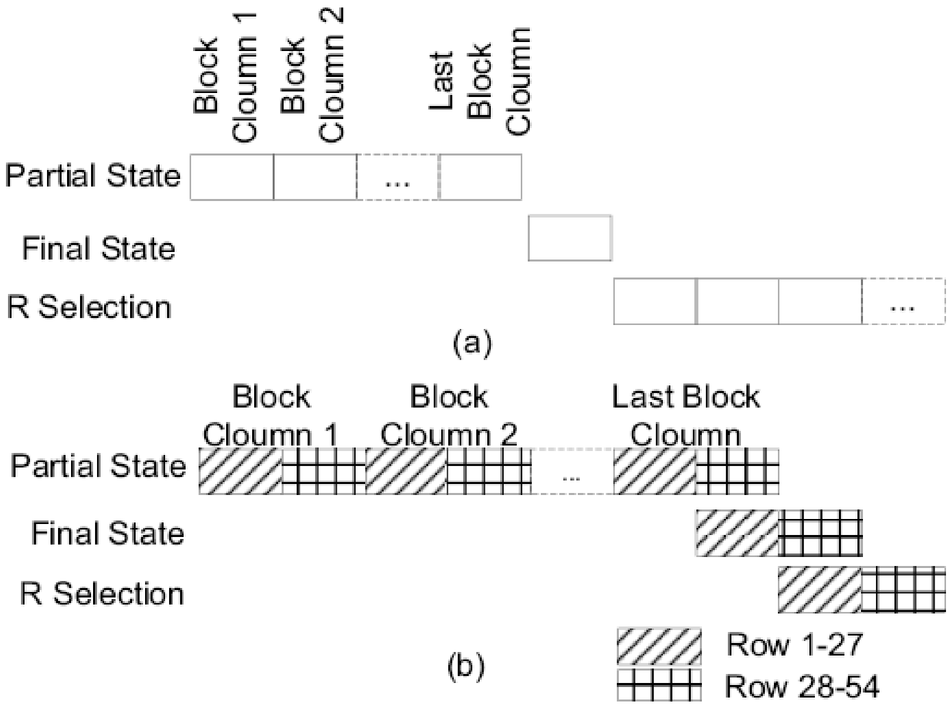


Fig.7.1. Operation of CNU (a) no time-division multiplexing (b) time-division multiplexing

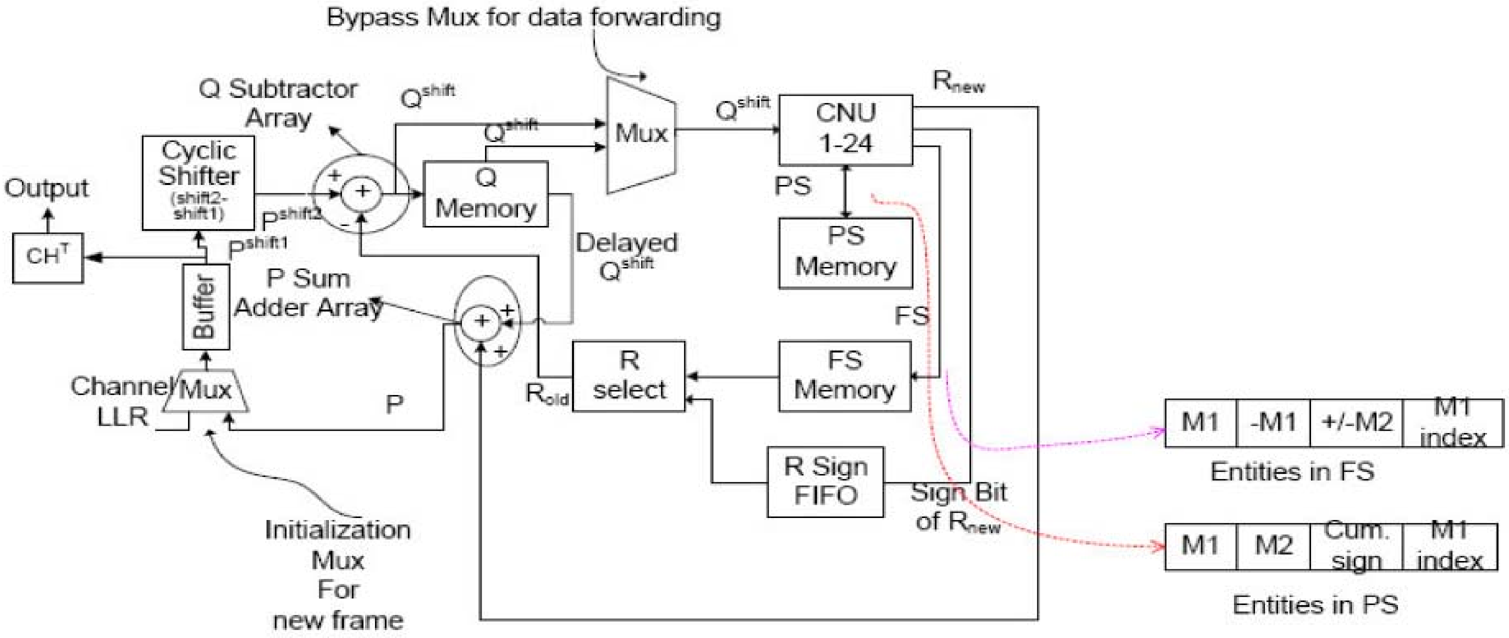
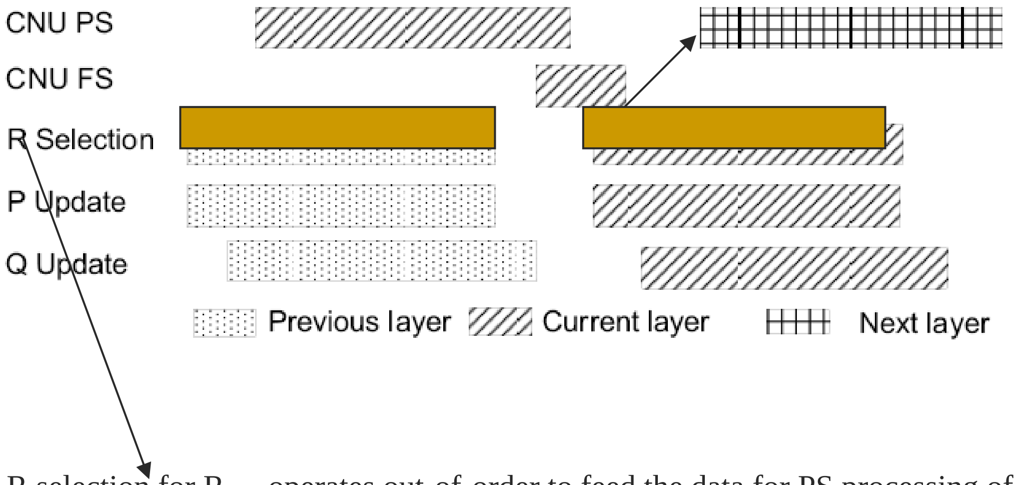
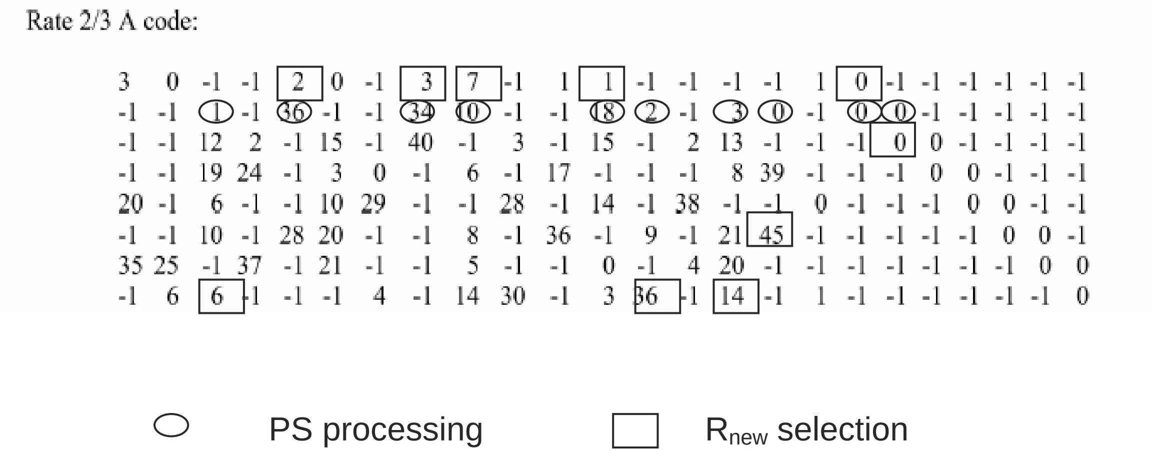


Fig.7.2. Multi-rate LDPC decoder architecture for Block LDPC codes



R selection for R_{new} operates out-of-order to feed the data for PS processing of next layer

Fig.7.3. Three-stage pipeline of the multi-rate decoder architecture.



Rate 2/3 A code:

$$\begin{array}{cccccccccccccccccccccccc}
3 & 0 & -1 & -1 & \boxed{2} & 0 & -1 & \boxed{3} & \boxed{7} & -1 & 1 & \boxed{1} & -1 & -1 & -1 & -1 & 1 & \boxed{0} & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & \enclose{circle}{1} & -1 & \enclose{circle}{36} & -1 & -1 & \enclose{circle}{34} & \enclose{circle}{10} & -1 & -1 & \enclose{circle}{18} & \enclose{circle}{2} & -1 & \enclose{circle}{3} & \enclose{circle}{0} & -1 & \enclose{circle}{0} & \enclose{circle}{0} & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 12 & 2 & -1 & 15 & -1 & 40 & -1 & 3 & -1 & 15 & -1 & 2 & 13 & -1 & -1 & -1 & \boxed{0} & 0 & -1 & -1 & -1 & -1 \\
-1 & -1 & 19 & 24 & -1 & 3 & 0 & -1 & 6 & -1 & 17 & -1 & -1 & -1 & 8 & 39 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\
20 & -1 & 6 & -1 & -1 & 10 & 29 & -1 & -1 & 28 & -1 & 14 & -1 & 38 & -1 & -1 & 0 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
-1 & -1 & 10 & -1 & 28 & 20 & -1 & -1 & 8 & -1 & 36 & -1 & 9 & -1 & 21 & \boxed{45} & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\
35 & 25 & -1 & 37 & -1 & 21 & -1 & -1 & 5 & -1 & -1 & 0 & -1 & 4 & 20 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
-1 & 6 & \boxed{6} & -1 & -1 & -1 & 4 & -1 & 14 & 30 & -1 & 3 & \boxed{36} & -1 & \boxed{14} & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{array}$$

Fig.7.4. Out of Order Processing for R_{new} selection

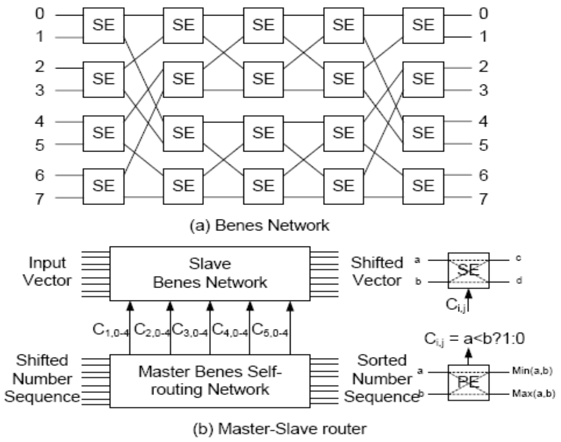
For the decoding of one layer, (7.3-7.5) are performed in sequence and these steps are repeated for all the layers. As shown in Fig. 7.3, the CNU array operates on the R messages and partial states of two adjacent block rows (layers). While the final state has dependency on partial states, P and Q messages are dependent on the final states. In the decoding process, a block row of check nodes are processed in serial fashion, the output of the CNU is also in serial form. As soon as the output vector $\vec{R}_{l,n}^{(i1)}$ corresponding to each block column n, in H matrix, for a block row l is available. This could be used to produce the updated sum $\left[\vec{P}_n\right]^{S(l,n)}$ in (7.5), which is then immediately used in (7.3) to process the shifted vector for block row $l+1$. The shift $S(l,n)$ imposed on $\vec{P}_n$ has to be undone and a new shift $S(l+1,n)$ imposed. This could be achieved by simply imposing a shift corresponding to the difference of $S(l+1,n)$ and $S(l,n)$.

To accommodate the irregularity in block LDPC codes, the R selection unit for R_{old}($\vec{R}_{l,n}^{(i-1)}$ in (7.3)) and PS processing are executed in linear order for the current layer (i.e. first non-zero block, second non-zero block in a layer), while order of R generation for $\mathrm{R}_{\mathrm{old}}$ processing is determined by the non-zero blocks of the next layer that has to be processed because $\vec{Q}_{l,n}^{(i)}$ in (7.3) has dependency on $\vec{R}_{l,n}^{(i1)}$ in (7.4) of the previous layer. Furthermore, since check node degree of each layer in Irregular Block codes may vary widely, it is not efficient that each layer executes for the number of clock cycles equal to the maximum check-node degree. In addition, due to data dependencies the processing of the next layer may have to be stalled. To address these inefficiencies, we propose out-of-order processing on $\mathrm{R}_{\mathrm{new}}$ generation. The R select unit for $\mathrm{R}_{\mathrm{new}}$ may be operating on any of the previous layers (see Fig. 7.4). It should be pointed out that R generation is independent of PS or FS processing, so it's out-of-order processing will not impose any additional restriction on the architecture. Even though it does require careful scheduling and there will be some additional logic to account for selecting based on the M1 index.

P values for a block are stored in a buffer each of size z as the decoder needs M values out of this buffer at each clock cycle. We need to store only two blocks to permit pipelined operation (i.e. one buffer is filled while the other buffer is used to produce Q messages) and these blocks are accessed in ping-pong fashion for processing of each new layer. Besides the shifted Q messages, the CNU array also takes input of the sign information for previous computed R messages in order to perform the R selection. An R select unit generates the R messages for k edges of a check node from three possible values stored in final state memory word associated with that particular check node in a serial fashion. Its functionality and structure is the same as the block denoted as R select in CNU. This unit can be treated as a de-compressor of the check-node edge information, which is stored in compact form in FS memory. It is possible to do the decoding using a different sequence of layers instead of processing the layers from 1 to j which is typically used to increase the parallelism such that it is possible to process two block rows simultaneously [38]. In this work, we use the same concept of re-ordering, but *also* for low complexity memory implementation. We need to schedule the order of layer processing and partition the Q memory to limit the number of read/write accesses to two for a memory bank. Q memory is partitioned into three dual port memory banks, with each supporting two read/write accesses. Note that Q memory has to be further partitioned to support vector processing. Also note that the decoder stopping criterion can be done for each layer using cH^T block similar to the process in [38]. However, the proposed decoder needs 1 bit wide m x m cyclic shifter on hard decision values of P in this process.

Master-slave Benes network

To be able to accommodate different shifts needed for the WiMax LDPC codes, we can use a Benes network as in [60], which is of complexity $2\log 2(M)-1$stages of M 2-in-1 multiplexers. A memory can be used to store control inputs needed for different shifts in case of supporting one expansion factor [20], [60]. [20] uses Omega network, which is less complex than Benes network [60]. However both [20] and [60] will support only base H matrix. Note that this memory for providing control signals to this network is equal to $\frac{M}{2}(2\log 2(M)-1)$ bits for every shift value that needs to be supported. This will be a very huge requirement for supporting all the WiMax codes. Note that the memory needed for storing control signals for the Omega network is around 1.22 mm^2 in, out of the decoder chip area of 14.1 mm$^{2.}$[20]. This is equivalent to storing the control signals for one expansion factor and one base H matrix. So, if the same kind of scheme is used to support 19 different expansion factors and 6 types of base H matrices *in run time*, the control signal memory needs approximately 139 mm$^{2.}$ So, this approach clearly will not work. We propose a simpler approach to generate the control signals using a Master-Slave Benes router (Fig. 7.5). Assume that we need to perform a cyclic shift of 2 on a message vector of length 4 using a 8x8 Slave Benes network. Supply the integers (2, 3, 0, 1, 4, 5, 6, 7) to the Master Benes network which is always configured to sort the inputs and output (0,1,2,…7). During the sorting process, the Master Benes network can generate the control signals on by virtue of comparators [40]. These signals can be used in the Master network to accomplish sorting. Also, these signals can be used in the Slave network to achieve the desired shift of 2. Note that the complexity of this approach adds, almost doubles the overall logic requirements of the router.



(a) Benes Network

(b) Master-Slave router

Fig.7.5. Proposed Master-slave router to support different cyclic shifts that arise due to a wide range of expansion factors z(=24,28,..,96) and shift coefficients(0,1,..,z-1).

7.4. Discussion and Implementation Results

Table 7.2-7.4 gives the FPGA implementation and comparison results. The proposed TDMP architecture features large memory savings up to 2x throughput advantage, as well as 50% less interconnection complexity. The TDMP permits us to use a running sum, which is initialized to channel log-likelihood ratio (LLR) values in the first iteration. So, there is no memory needed to store the channel LLR values as these values are implicitly stored in the *Q* messages. Since the maximum number of *Q* messages that need to be stored are equal to $N_b \times z_o \times 5$, as opposed to storing $N_{nz} \times z_o \times 5$ messages in TPMP architectures where N_b is the maximum number of block columns of all the codes that need to be supported, z_o is the maximum expansion factor of the base matrix, N_{nz} is the number of non-zero blocks by considering the base H matrix, which has the maximum number of non-zero blocks among all the base H matrices that need to be supported. For WiMax LDPC codes, these parameters are $N_b = 24$, $z_o = 96$, and $N_{nz} = 76$. So, the total savings in Q memory are 68% as a direct result of employing TDMP proposed in [2].

The proposed architecture offers further advantages. Instead of storing all the R messages, the compressed information cumulative sign, *M1, -M1*, *+/-M2,* and index of *M1* is stored. R select unit can generate the R message by the use of an index comparator and the XOR of the cumulative sign and the sign bit of the corresponding Q message which comes from the sign FIFO. The total savings in R memory is $\frac{5kN_l z_o - [k + 5\times 3 + ceil(\log_2(k)) + 1]N_l z_o}{5kN_l} \times 100\%$, where N_l is the number of layers or block rows by considering the base H matrix, which has the maximum number of non-zero blocks among all the base H matrices that need to be supported. The factor 5 comes due to the use of 5-bit quantization for R messages. Among the different base LDPC codes in WiMax, rate 5/6 code has the maximum check-node degree, $k = 19$ and the maximum number of block rows in the *H* matrix is 12. So the savings of R memory is 57%.

Also note that, due to the nature of block serial scheduling and the scheduling of layered processing in the architecture, there is only the need to store the P messages for only two blocks. The total savings of memory bits is $\left(N_b \times z_o \times 6 - 2\times z_o \times 6\right)/\left(N_b \times z_o \times 6\right)\times 100\%$. Note that the factor 6 comes due to the number of bits used to represent the P message. So the savings are around 91% as $k_{\max} = 19$ and, z_0=*96* for block LDPC codes in 802.16e.

The total savings in memory accounting for R memory, Q memory, and P memory, when compared to TPMP architectures based on SP [13] and min-sum [7], [8], [14] is

63%.When compared to TDMP architecture based on BCJR [2], the total memory savings is 55% since both architectures have the same savings in Q memory.

In terms of throughput and interconnect advantage, to achieve the same BER as that of TPMP schedule on OMS, TDMP schedule on OMS needs half the number of iterations. This essentially doubles the throughput. However, the choice of finite precision OMS results in a performance degradation of less than 0.1 dB. 5-bit uniform quantization for R and Q messages and 6-bit uniform quantization for P messages is used. The step size for quantization, Δ, and offset parameter, β are set based on the code parameters [11].

Table 7.1.

FPGA Implementation results of the multi-rate decoder (supports z=24, 48 and 96 and all the code rates)

(Device, *Xilinx 2V8000ff152-5*, frequency *110MHz)*

	Used			Available
	M=24	M=48	M=96	
Slices	1640	3239	6568	46592
LUT	2982	5664	11028	93184
SFF	1582	3165	6330	93184
BRAM	38	73	100	168
Memory (bits)	65760	65760	60288	
Through-put (Mbps)	41~70	57~139	61~278	

Table 7.2.

FPGA Implementation results, the multi-rate decoder, Fully Compliant to WiMax (supports z=24,28,32,…,and 96 and all the code rates)

(Device, *Xilinx 2V8000ff152-5*, frequency *110MHz)*

	Used			Available
	M=24	M=48	M=96	
Slices	3746	8369	18664	46592
LUT	7939	15579	30858	93184
SFF	1582	3165	6330	93184
BRAM	38	73	100	168
Memory (bits)	65760	65760	60288	
Through-put (Mbps)	41~70	57~139	61~278	

Table7.3

Implementation comparison

	M. Karkooti *et al* [37]	T. Brack *et al[41]*.
Slices	11352	14475
LUT	20374	N/A
SFF	N/A	N/A
BRAM	66	165
Throughput(Mbps)	127	180
Codes supported	1 code of length 1536 and rate 0.5	3 codes of length 1000,2000, 3000 and rate 0.8
Decoding	TPMP-MS	TPMP-MS
WiMax Code support	No	No

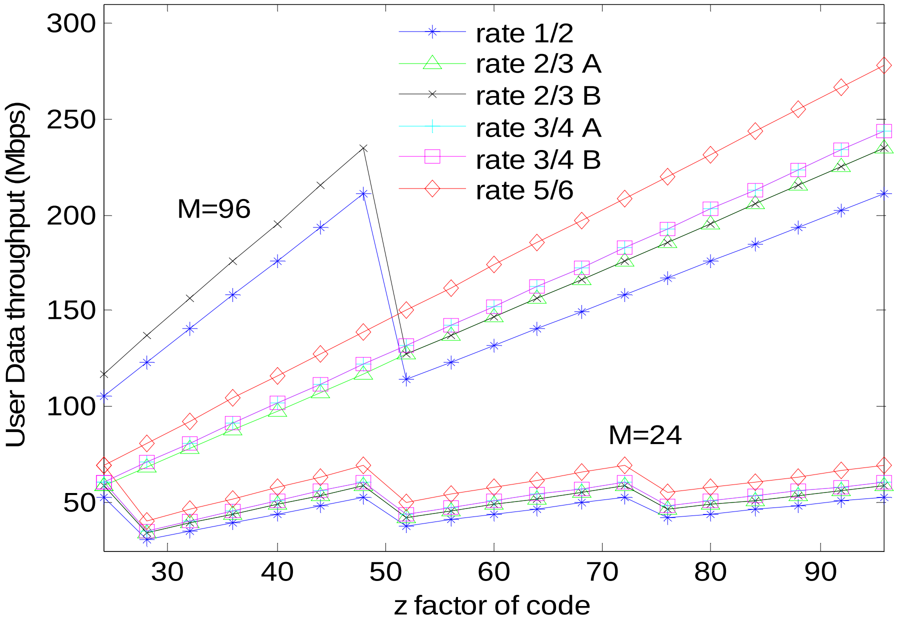


Fig. 7.6 User data throughput of the proposed decoder vs. the expansion factor of the code,z, for different numbers of decoder parallelization,M

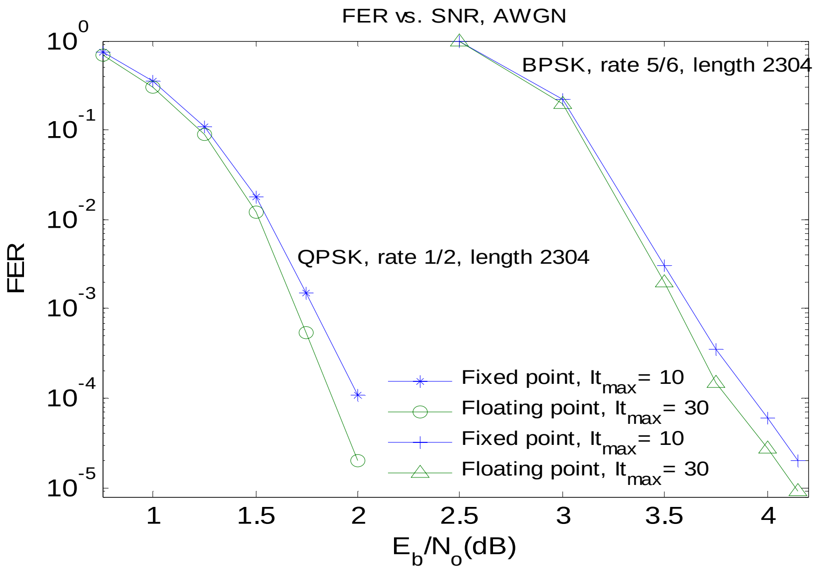


Fig.7.7. Frame-error rate results.

Moreover, this architecture requires only one cyclic shifter instead of two cyclic shifters [2], [4]. Note that the architecture features a partial state memory when compared to other architectures. However, this is small as it must contain the partial state for only one block row at any time, is equal to 1536 bits. In the case of parallelization equal to $M= z_0$, then there is no need for P buffer. Also, PS memory bank and FS memory bank need to store only R messages belonging to 11 layers. The P buffer is not needed as the shifter employed is $z_o \times z_o$ and it can perform the shift without the need of a buffer since the input messages are available in the chunks of z_0. There is no need for a PS memory bank, since there are z_0 CNU to handle the maximum number of rows in a block row (z_0), and consequently there is no time folding. The data throughput results are presented in Fig. 7.6. Note that the throughput is dependent on the z factor of the code as this determines the percentage usage of the available parallelization M. The implementation has a performance penalty of less than 0.15 dB in SNR when compared to floating point TDMP decoding (see Fig. 7.7.).User data throughput t_u is given by $t_u = rate \times t_d$, where t_d is decoded throughput and is given by $t_d = Nf / \left(it_{\max} CCI\right)$, where f is the decoder chip frequency and CCI stands for number of clock cycles required to complete one iteration. CCI is given by $CCI = \left(N_b\right)CCL$. N_b is the number of block rows (or layers) of the code and CCL stands for number of clock cycles to process one layer and is given as, $\mathrm{CCL} = \mathrm{k}_{\text{max-code}} \lceil \mathrm{z/M} \rceil + 2$. where $\mathrm{k}_{\text{max-code}}$ is the maximum check node degree of the chosen base H matrix. For instance, rate 3/4 code A, [12], explained in previous sections, the check node degrees of 6 block rows is specified as [13 12 12 12 12 13] . So here $\mathrm{k}_{\text{max-code}}$ =13. Here z is the expansion factor of the code used and M is the decoder parallelization as explained before. Note that some codes support processing of two layers in parallel and the decoder can accommodate this if sufficient parallelism is available as can be seen from Fig. 7.6.

7.4.1. ASIC Implementation Results for WiMax LDPC Codes

We have implemented the proposed decoder architecture using the open source standard cells v*sclib013* [14] in 0.13 micron technology. The synthesis is done using a synopsys design analyzer tool, while layout is done using a cadence's Silicon Ensemble tool. Tables 7.4-7.9 give the performance comparison as well as the decoder chip characteristics. The original TDMP decoder [20] is based on more complicated BCJR algorithm. The CNU for BCJR takes more area due to the need of several internal FIFOs. In addition, Omega network is used in [20] instead of logarithmic shifter. The use of logarithmic shifter saves area to store the control signals as well as the the absence of control wires make the logarithmic shifter's layout much more compact. The proposed decoder has a frequency advantage also, as the CNU stage has 3 pipeline stages. The decoder in [20] has fewer pipeline stages.

7.4.2. ASIC Implementation Results for 802.11n LDPC Codes

All the code lengths (648, 1296 and 1944, according to different expansion factors z = 27, 54, and 81 respectively) and code rates (1/2, 2/3, 3/4 and 5/6) as specified in the IEEE 802.11n standard [13] are supported in this architecture. Table 7.10 gives the FPGA implementation and ASIC results for M = 81 are shown in Table 7.11, in which VNU constitutes the P adder array and Q subtractor array. Note that the relatively large memory area in ASIC implementation is due to the 133 shallow memory banks required for the total number of 55344 bits. Similar memory area overheads are reported in [3]. Here, all calculations for the decoded throughput are based on an average of 5 decoding iteration to achieve a frame error rate of 1e-6, while it_{max} is set to 15. The total power dissipation is estimated to be 238.4mW by the Synopsys design analyzer. Recent work on IEEE 802.11n LDPC decoder [15] consumes 375:14K logic gates and 88452 bits of memory for 940 Mbps throughput. So the proposed decoder, when compared to this work reduces the logic gate complexity by 6.45x and memory complexity by 2x for a given data throughput (based on the results in Table 7.11).

7.5. Conclusion

We present a memory efficient multi-rate decoder architecture for turbo decoding message passing of block LDPC codes of IEEE 802.16e and IEEE 802.11n using the OMS algorithm for check-node update. Our work offers several advantages when compared to the other-state-of -the-art LDPC decoders in terms of significant reduction in logic, memory, and interconnect. This work retains the key advantages offered by the original TDMP work. However, our contribution is in using the value-reuse properties of offset MS algorithm and devising a new TDMP decoder architecture to offer significant additional benefits.

Table 7.4.

ASIC Implementation of the proposed TDMP multi-rate decoder architecture

	Semi-Parallel multi-rate LDPC decoder [12]	Multi-rate TDMP Architecture regular QC-LDPC
LDPC Code	AA-LDPC, (3,6) code, rate 0.5, length 2048	Irregular codes up to length 2304 IEEE 802.16e WiMax LDPC codes
Decoded Throughput, t_d,	640 Mbps	1.37 Gbps
Area	14.3 mm^2	2.1 mm^2
Frequency	125 MHz	500 MHz
Nominal Power Dissipation	787 mW	282 mW
Memory	51,680 bits	60,288 bits
CMOS Technology	180,nm 1.8V	130 nm.2V
Decoding Schedule	TDMP, BCJR, it_{max}=10	TDMP, OMS, it_{max}=10
Area Efficiency for t_d, 180 nm	44.75 Mbps/mm^2	649.5 Mbps/mm^2
Energy Efficiency for t_d, 180 nm	123 pJ/Bit/Iteration	21 pJ/Bit/Iteration
Est. Area for 180 nm	14.3 mm^2	~4.02 mm^2
Est. Frequency for 180 nm	125 MHz	~360 MHz
Est Decoded Throughput(t_d),180nm	640 Mbps	989 Mbps
Est Area Efficiency for t_d, 180 nm	44.75 Mbps/mm^2	246 Mbps/mm^2
Est Energy Efficiency for t_d, 180 nm	123 pJ/Bit/Iteration	47.25 pJ/Bit/Iteration
Application	Multi-rate application as well as fixed code application	IEEE 802.16e Multi-rate application .
Bit error rate Performance	Good	Very good and close to capacity

Table 7.5.

Area distribution of the chip for WiMax LDPC codes

Architecture 1 : supports z=24,48 and 96 and all the code rates)

Architecture 2: Fully Compliant to WiMax supports z=24,28,32,…,and 96 and all the code rates. The only difference is the replacement of logarithmic cyclic shifter with Master-slave Benes router. This will increase the complexity of router by almost 5x and increase the power dissipation of the decoder by 70% when compared to the Architecture 1.

	Architecture 1,Area (mm^2)	Architecture 2,Area (mm^2)
CNU Array	0.53	0.53
VNU Array	0.08	0.08
Memory	1.23	1.23
Pipeline flip-flops	0.03	0.03
Cyclic shifter	0.15	0.74
Wiring	0.09	0.12
Total chip area	2.11	2.73

Table 7.6.

Power distribution of the chip for WiMax LDPC codes

(supports z=24,48 and 96 and all the code rates)

	Architecture 1,Power (mW)	Architecture 2, Power (mW)
Logic(CNU,VNU and shifters)	160.41	273.31
Memory	73.88	73.88
Leakage	0.09	0.11
Clock	32.08	54.66
Wiring	16.04	27.34
Total	282.5	429.3

Table 7.7.

ASIC Implementation of the proposed TDMP multi-rate decoder architecture for 802.11n LDPC codes

	Semi-Parallel multi-rate LDPC decoder [12]	Multi-rate TDMP Architecture regular QC-LDPC
LDPC Code	AA-LDPC, (3,6) code, rate 0.5, length 2048	Irregular codes up to length 1944 IEEE 802.11n LDPC codes
Decoded Throughput, t_d,	640 Mbps	1.1571 Gbps
Area	14.3 mm^2	1.782 mm^2
Frequency	125 MHz	500 MHz
Nominal Power Dissipation	787 mW	238 mW
Memory	51,680 bits	52,488 bits
CMOS Technology	180 nm, 1.8V	130 nm, 1.2V
Decoding Schedule	TDMP, BCJR, it_{max}=10	TDMP, OMS, it_{max}=10
Area Efficiency for t_d, 180 nm	44.75 Mbps/mm^2	649.5 Mbps/mm^2
Energy Efficiency for t_d, 180 nm	123 pJ/Bit/Iteration	21 pJ/Bit/Iteration
Est. Area for 180 nm	14.3 mm^2	~3.41 mm^2
Est. Frequency for 180 nm	125 MHz	~360 MHz
Decoded Throughput(t_d) ,180nm	640 Mbps	833 Mbps
Area Efficiency for t_d, 180nm	44.75 Mbps/mm^2	244 Mbps/mm^2
Energy Efficiency for t_d, 180 nm	123 pJ/Bit/Iteration	47.25 pJ/Bit/Iteration
Application	Multi-rate application as well as fixed code application	IEEE 802.11n Multi-rate application
Bit error rate Performance	Good	Very good and close to capacity

Table 7.8.

Area distribution of the chip for IEEE 802.11n LDPC codes

Supports z=27,54 and 81 and all the code rates. Fully compliant to IEEE 802.11n

	Area (mm^2)
CNU Array	0.44
VNU Array	0.07
Memory	1.04
Pipeline flip-flops	0.02
Cyclic shifter	0.12
Wiring	0.08
Total chip area	1.78

Table 7.9.

Power distribution of the chip for IEEE 802.11n LDPC codes

Supports z=27,54 and 81 and all the code rates. Fully compliant to IEEE 802.11n

	Power (mW)
Logic(CNU,VNU and shifters)	135.35
Memory	62.34
Leakage	0.07
Clock	27.07
Wiring	13.53
Total	238.40

Table 7.10

FPGA Implementation results for the multi-rate decoder . Fully compliant to IEEE 802.11n (Device, XILINX2V8000FF152-5, frequency = 110MHZ)

	M=27	M=54	M=81	Available
Slices	1836	3647	5514	46592
LUT	3317	6335	9352	93184
SFF	1780	3560	5341	93184
BRAM	46	89	133	168
Memory(bits)	56640	56640	55344	
Throughput(Mbps)				
z = 81	119	238	356	
z = 54	119	238	178	
z =27	119	119	119	

Table 7.11

ASIC Implementation results for the multi-rate decoder for M= 81 (Frequency = 500MHZ)

Resource	Area(mm^2)	Equivalent NAND-2 gates
CNUs	0.45	67500
VNUs	0.07	10125
Storage	1.04	N/A
Flip-flops	0.03	3375
Shifter and wiring	0.22	18900
Total	1.85	99900
Throughput(Mbps)	541, 1082 and 1618 z = 27, 54 and 81	

CHAPTER VIII

A PARALLEL VLSI ARCHITECTURE FOR LAYERED DECODING FOR ARRAY LDPC CODES

8.1. Introduction

In this chapter, we use the novel parallel micro-architecture structure (Chapter IV) for the check-node message processing unit (CNU) for the offset min-sum (OMS) decoding of LDPC codes based on value-reuse and survivor concepts. In addition, a novel physical-layout-driven architecture for TDMP, using the OMS for array LDPC codes, is proposed. The resulting decoder architecture has significantly lower requirements of logic and interconnects when compared to the published decoder implementations.. Section 2.3 introduced the background of array LDPC codes and OMS, the decoding algorithm. Section 6.3 presented the TDMP and its properties for array LDPC codes. Section 4.1 presented the value-reuse property and proposed micro-architecture structure of CNU. The data flow graph and parallel architecture for TDMP using OMS is included in section 8.2. Section 8.3 shows the ASIC implementation results and performance comparison with related work and section 8.4 concludes the chapter.

8.2. Parallel Architecture Using TDMP and OMS

A new data flow graph architecture (see Fig. 8.1) is designed based on the properties of TDMP and on the value reuse property of OMS. For ease of discussion and also for the sake of relevant comparisons with the state of the art work, we will illustrate the architecture for a specific structured code: array LDPC code of length N=2082 and

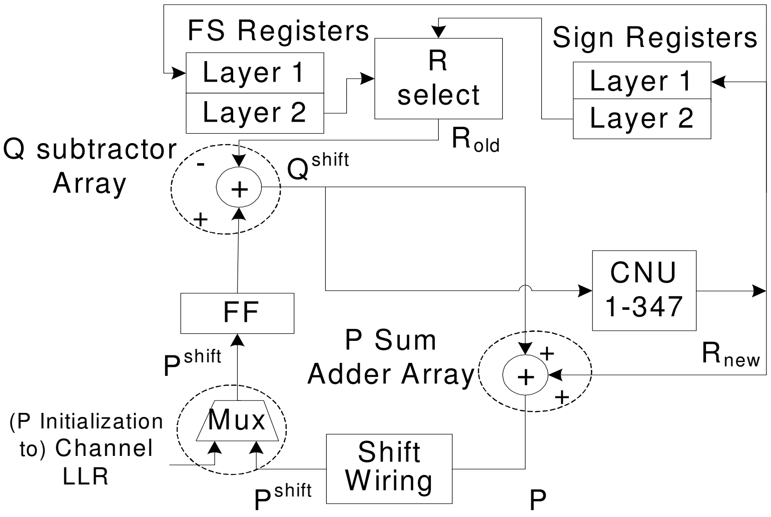


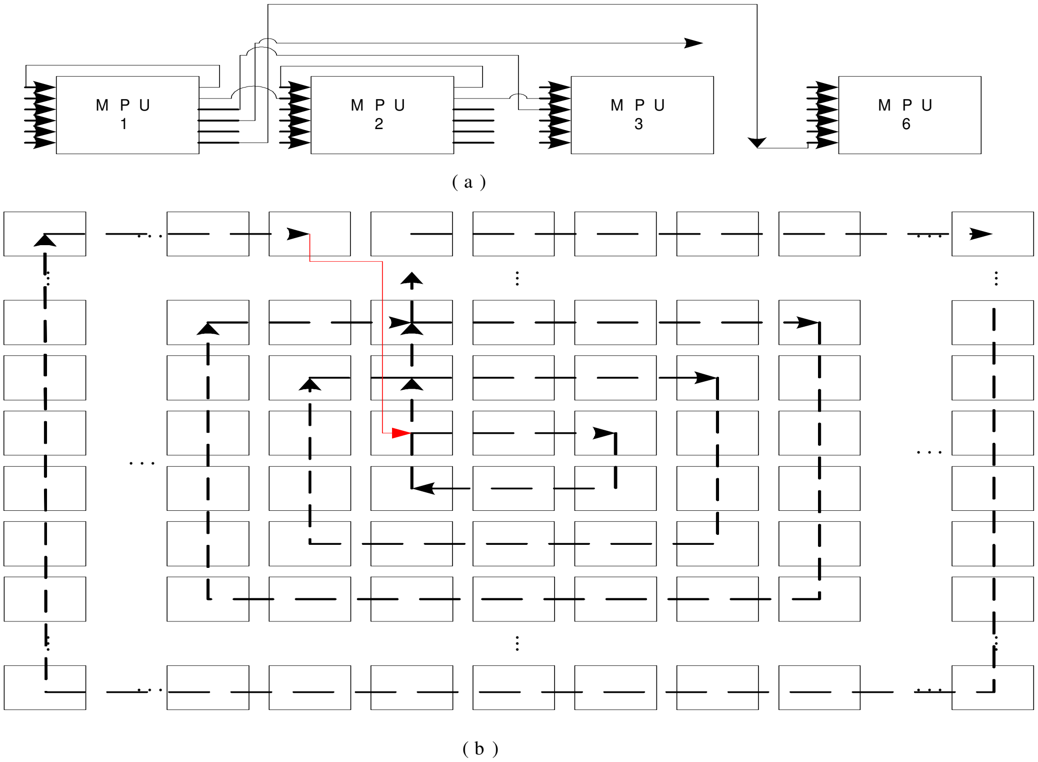
Fig.8.1. Parallel architecture for layered decoder.

K=1041 described in section 2, $j=3$, $k=6$ and $p=347$. A parallel CNU with input vector of length 6 is based on the design described in Chapter IV. The CNU array is composed of p CNU computation units that compute the R messages for each block row in fully parallel fashion. Since R messages of previous $j-1$ block rows are needed for TDMP, the *compressed* information of each row is stored in final state (FS) register banks. Each final state register in a FS register bank contains M1, -M1, +/-M2 and index for M1. The depth of FS register bank is $j-1$, which is 2 in this case. There are a total of p such register banks, each one associated with one CNU. The sign bits of R messages are stored in sign flip-flops. The total number of sign flip-flops for each row of R messages is k and each block row has pk sign flip-flops. We need $j-1$ of such sign flip-flop banks in total. A total of p R select units is used for R_{old} . An R select unit, whose functionality and structure is the same as the block denoted as R selector in CNU (Fig.4.3), generates the R messages for $6(=k)$ edges of a check node from 3 values stored in final state register in parallel fashion. In the beginning of the decoding process, i.e., the first sub-iteration of the first iteration for each new received data block, *P* matrix (of dimensions *p* x *k*) is set to received channel values in the first clock cycle (i.e. the first sub-iteration), while the output matrix of R select unit is set to zero matrix (6.7). The multiplexer array at the input of P buffer is used for this initialization. Note that due to parallel processing, each sub-iteration (6.8)-(6.10) takes one clock cycle. So, except for the first sub-iteration of the first iteration, i.e., from the 2nd clock cycle, the *P* matrix is computed by adding the shifted *Q* matrix (labeled as Q^{shift} in Fig. 3) to the output matrix *R* (labeled as R_{new}) of the CNU array (6.10). The compressed information of *R* matrix stored in the register banks FS is used to generate R_{old} for the l^{th} sub-iteration in the next iteration (6.8). This results in a reduction of R memory that is around 20%-72% for 5-bit quantized messages based on the check-node degree k of the code. The proposed decoder supports a fixed value of *k,* which is determined in the design time based on the error correction performance required by application.

Note that the *P* matrix that is generated can be used immediately to generate the *Q* matrix as the input to the CNU array as the CNU array is ready to process the next block row (6.8). Now each block column in the *P* message matrix will undergo a cyclic shift. This shift is given by the amount of difference of the shifts of the block row that is processed and the previous block row that was just processed in the previous sub-iteration. A concentric layout is designed to accommodate routing and 347(=*p*) message processing units (MPU) as shown in Fig. 8.2. An MPU consists of a parallel CNU, a parallel VNU, and associated registers belonging to each row in the *H* matrix. The 2*k* adder units, 1 R select unit associated with each parallel CNU is termed as the parallel variable-node unit (VNU). MPU $i(0,1,2,\ldots,346(=p\text{-}1))$ communicates with its $5(=k\text{-}1)$ adjacent neighbor MPUs (whose numbers are mod$(i+1,p)$...mod$(i+5,p)$) to achieve cyclic down shifts of 1,2,...,5 ($=n$-1) respectively for block columns 2, 3, …,6 ($=n$)in the H matrix (1). Similarly MPU i communicates with its 5($=k$-1) adjacent neighbor MPUs (whose numbers are mod$(p\text{-}i\text{-}2,p)$...mod$(p\text{-}i\text{-}10,p)$) to achieve cyclic up shifts of 2,4,..10($=(j\text{-}1)(n\text{-}1)$), respectively for block columns 2, 3, …,6($=n$) as noted in section 8.3. so the upshift needed on each block column n is 2n as j=3.

8.3. ASIC Implementation Results

We have implemented the proposed parallel layered decoder architecture for (3,6) code of length 2082 using the open source standard cells *vsclib013* [62] in 0.13 micron technology. The synthesis is done using Synopsys design analyzer tool, while layout is done using Cadence’s Silicon Ensemble tool. The chip area is 2.3 mm x 2.3 mm and the post routing frequency is 100 MHz. However, the additional IO circuitry (the serial-to-parallel and parallel-to-serial conversion circuitry around the chip), which is application dependent, is not accounted for in the chip area and is estimated not to exceed 15% of chip area. Note that the only memory needed is to store compressed R messages and this is implemented as scattered flip fops associated with each CNU. The ASIC implementation of the proposed parallel architecture achieves a decoded throughput of 6.9 Gbps for 10 TDMP iterations and user data throughput of 3.45 Gbps. Each TDMP iteration consists of $j(=3)$ sub-iterations and each sub-iteration takes one clock cycle.



(a)

(b)

Fig.8.2. a) Illustration of connections between message processing units to achieve cyclic down shift of (n-1) on each block column n; b) Concentric layout to accommodate 347 message processing units. Rectangles indicate MPUs while the arrowed lines represent connections between adjacent MPUs. Connections for cyclic up shift of 2n are not shown

User data throughput t_u is calculated by the following formulae: $t_u = rate * t_d = (K/N) * t_d$,where t_d is decoded throughput and is given by $t_d = N * f / (it_{\max} * CCI)$,where f is the decoder chip frequency and *CCI* stands for the number of clock cycles required to complete one iteration. The symbols $it_{\max}$, *K,* and *N* are defined in section 8.2. The design metric *CCI* is equal to the number of layers in array code i.e., *j(=3).* To achieve the same BER as that of the TPMP schedule on SP (or equivalent TPMP schedule on BCJR), the TDMP schedule on OMS needs half the number of iterations (Fig. 8.3) having similar convergence gains reported for TDMP-BCJR [20]. However, the choice of finite precision OMS results in a performance degradation of 0.2 dB. 5-bit uniform quantization for R and Q messages and 6-bit uniform quantization for P messages is used. The step size for quantization, Δ, and offset parameter, β are set to 0.15[32]. Table 8.1 gives the performance comparison with the recent state-of-the-art work. The design data of 180 nm process for [15] and the present work is extrapolated based on linear scaling in frequency and quadratic scaling in the area of 180 nm CMOS process. The design in [21] is based on 1-bit data path. Though the routing congestion is decreased based on broadcasting and hard decision, it is still an issue with soft decoders. In addition, the hard decision decoder has very poor BER performance. The fully parallel decoder architecture in [25] is based on modified array codes to reduce the routing congestion and these exhibit early error floors. Thus this decoder may not be suitable for low error floor applications which require BERs of less than 1e-12. However, the advanced circuit techniques described in [25] may be applicable to any LDPC decoder. The PLAs are used to implement non-linear functions needed for SP decoding algorithm and occupied most of the area. All the other logic is implemented as standard static and dynamic CMOS logic. In the proposed architecture, the non-linear function is not needed in the offset min-sum. So there is no logic that can readily benefit from using the PLA based logic design. However, recent research indicates that it is possible to do a mix of PLA based logic and standard cell design to improve the frequency.

When compared to the works in [15], [20], [21] and other published work, the work presented here shows significant gains in area efficiency for decoded thrashput (t_d)user data throughput ($t_{u,}$) while having similar and good BER performance as that of [20].

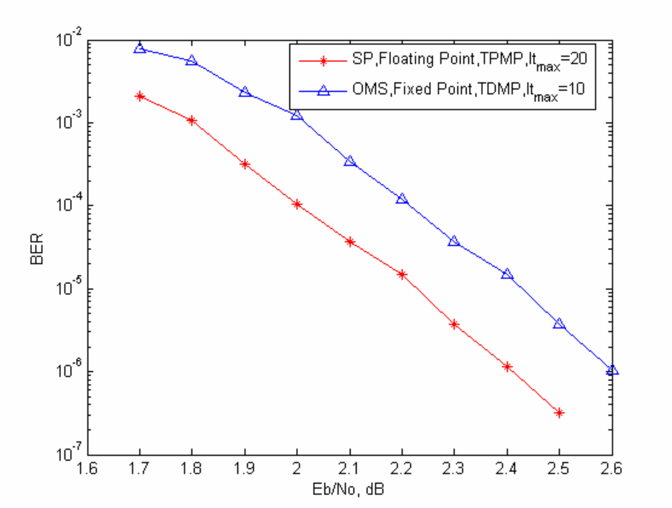


Fig.8.3. BER performance of the decoder for (3,6) array code of N=2082.

8.4. Conclusion

This chapter presented physical-layout-driven parallel decoder architecture for TDMP of array LDPC codes. We showed the key properties of OMS such as value-reuse and survivor, and designed a low complexity CNU with memory savings of around 20%-72%. In addition, the properties of TDMP for array LDPC codes are used to remove the interconnect complexity associated with parallel decoders. Our work offers several advantages when compared to the other state-of-the-art LDPC decoders in terms of significant reduction in logic, memory and interconnects.

Table 8.1

Proposed decoder work as compared with other authors.

	[20]	[15]	[21]	This work
Decoded Throughput, t_d	640 Mbps	1.0 Gbps	3.2 Gbps	6.9 Gbps
Area	14.3 mm^2	52.5 mm^2	17.64 mm^2	5.29 mm^2
Decoder's Internal memory	51680 bits (SRAM) 9216 bits (flip-flops)	34816 bits (scattered flip-flops)	98944 bits (scattered flip-flops)	27066 bits (scattered flip-flops)
Router/Wiring	3.28 mm^2-Network	26.25 mm^2-Wiring	Details unknown	0.89 mm^2-Wiring
Frequency, f	125 MHz	64 MHz	100 MHz	100 MHz
Nominal Power Dissipation	787 mW	690 mW	NA	75 mW
Area Efficiency for t_d,	44.7 Mbps/mm^2	19.04 Mbps/mm^2	181.4 Mbps/mm^2	493.0 Mbps/mm^2
Energy Efficiency for t_d,	123 pJ/Bit/Iteration	10.1 pJ/Bit/Iteration	NA	1.1 pJ/Bit/Iteration
LDPC Code	AA-LDPC, (3,6) code, rate 0.5	Random and Irregular code, rate 0.5	RS-LDPC, (6,32) code, rate 0.8413	Array code, (3,6) code, rate 0.5
Check Node Update	BCJR	SP	SP	Offset Min-Sum,OMS
Decoding Schedule	TDMP, it_{max}=10	TPMP, it_{max}=64	TPMP, it_{max}=32	TDMP, it_{max}=10
Block Length, N	2048	1024	2048	2082
SNR(E_b /N_o) for BER of 1e-6	2.4 dB	2.8 dB	6.4 dB	2.6 dB
Average CCI due to pipelining	40	1	1	3
CMOS Technology	180 nm, 1.8V	160 nm, 1.5V	180 nm, 1.8V	130 nm, 1.2V
Est. Area for 180 nm	14.3 mm^2	~66.4 mm^2	17.64 mm^2	~10.1 mm^2
Est. Frequency for 180nm	125 MHz	~56.8 MHz	100 MHz	~72 MHz
Est. Decoded Throughput(t_d) 180 nm,	640 Mbps	887.5 Mbps	3.2 Gbps	4.98 Gbps
Est. Area Efficiency for t_d, 180 nm	44.7 Mbps/mm^2	13.36 Mbps/mm^2	181.4 Mbps/mm^2	493.0 Mbps/mm^2
Est. Energy Efficiency for $t_{d,,}$ 180 nm	123 pJ/Bit/Iteration	14.5 pJ/Bit/Iteration	NA	4.8 pJ/Bit/Iteration
Scalability of Design	Yes	No	No	Yes

CHAPTER IX

SUMMARY

9.1. Key Contributions

This dissertation presented a systematic design of decoder architectures for QC-LDPC codes using an on-the-fly computation paradigm. The ingredients of this paradigm are reflected in the data flow graph design to minimize the logic, message passing memory and router requirements

9.1.1. Key Contributions for Multi-rate Architectures for QC-LDPC Codes

Chapter III presented the multi-rate decoder architecture using the classic two phase message passing schedule and describes the ways to achieve highly efficient pipelined structures to minimize the message passing requirements and improve the throughput. The required message passing memory is realized as the SRAM-FIFO which also served as the internal FIFO for computations. The number of memory accesses is 50% less than the state-of-the-art decoders and this lead to improved energy efficiency. In addition, due to the efficient pipelining techniques employed, the decoder has a throughput advantage. The decoder in [12] has fewer pipeline stages due to the nature of feedback loops in the CNU processing. In [5], the partial sums will go through a router and reverse router and the final sum will have to go through another reverse router. This would affect the timing as well as increase the complexity of the design. In addition, with the code construction used in [5], the shifts needed are not necessarily cyclic. This would result in costly implementation of a switching network instead of simple multi-stage cyclic shifters.

Chapter IV presented the value-reuse properties of offset min-sum decoding algorithm and the micro-architecture structures for the serial and parallel check-node units. These implementations are better when compared to other implementation of min-sum algorithm and its variants (offset min-sum and normalized min-sum).

Chapter VI presented the multi-rate architecture for rate compatible array LDPC codes. This utilized the value-reuse properties of offset min-sum presented in Chapter III and block-serial scheduling of computations presented in Chapter V, along with layered decoding or TDMP proposed in [20]. This novel architecture has the following advantages: removal of memory needed to store the sum of the variable node-messages and the channel values, removal of memory needed to store the variable node-messages, 40%-72% savings in storage of extrinsic messages depending on the rate of the codes, reduction of routers by 50%, an increase of throughput up to 2x. This architecture works for any other regular QC-LDPC codes without any need of modifications in the hardware.

Chapter VII presented the ways to adapt the layered architecture (presented in Chapter V) to the irregular QC- LDPC codes, Block LDPC codes. Block LDPC codes are considered for various wireless standards such as IEEE 802.11n, IEEE 802.16e and IEEE 802.22. The techniques of data-forwarding and out-of-order processing are used to deal with the irregularity of the codes. Another key advantage of layered decoder architectures presented in Chapters VI and VII are the reduction of routers from 2 to 1. The architectures benefit from the properties of layered scheduling which are not used in the existing layered decoder architectures. Due to the fact that fewer number of non-zero blocks have to be processed in the Block LDPC codes, the presented architecture for Block LDPC codes achieves the best energy efficiency and area efficiencies when compared to the decoder architectures presented for regular QC-LDPC codes. When compared with recent implementation of an 802.11n LDPC decoder [15], the proposed decoder targeted for IEEE 802.11n, reduced the logic gate complexity by 6.45x and memory complexity by 2x for a given data throughput. When compared to the latest state of the art multi-rate decoders [20], this decoder design has an area efficiency of around 5.5x and energy efficiency of 2.6x for a given data throughput. The numbers are normalized for a 180nm CMOS process.

In addition to the above key savings that apply for all the Block-LDPC codes, Chapter VII described simpler ways to accommodate the parity check matrices with different expansion factors. For the case of limited expansion factors, a base cyclic shifter is used to achieve shifts for the vector lengths that are multiples of the input vector size of the base cyclic shifter. For the case of wide range of expansion factors: A master-slave router is proposed to accommodate different permutations that arise due to the need to support 114 different parity check matrices in run time for IEEE 802.16e. This approach eliminates the control memory requirements by generating the control signals for slave data router with the help of a self routing master network.

9.1.2. Key Contributions for Fixed Code Architectures for Regular QC-LDPC Codes

Some applications such as magnetic recording channel, high speed ether net and optical links require very high throughputs. Here the channel is known beforehand, so a fixed code can be designed based on the requirements of the application. This is in contrast to the flexibility needed to accommodate different codes for varying channel conditions in wireless applications.

Chapter V presented a new fixed code architecture for regular array codes based on the offset min-sum algorithm that reduces the need of message passing memory by 80% and the routing requirements by more than 50% when compared to other state-of-the-art decoder architectures. This architecture is based on the scheduling of computation that results in “on the fly computation” of variable node and check-node reliability messages. This schedule is the variation of the scheduling scheme presented in Chapter III. This architecture is scalable for any code length due to the concentric and regular layout unlike the fully parallel architecture [3].

Chapter VIII used the novel parallel micro-architecture structure (Chapter IV) for the check-node message processing unit (CNU) for the offset min-sum (OMS) decoding of LDPC codes based on value-reuse and survivor concepts. In addition, a novel physical-layout-driven architecture for TDMP, using the OMS for array LDPC codes, is presented. The resulting decoder architecture has significantly lower requirements of logic and interconnects when compared to the published decoder implementations. When compared to the latest state-of-the-art multi-rate decoders, this design has an area efficiency of around 10x and an energy efficiency of 25x for a given data throughput. When compared to the proposed multi-rate decoders for Block LDPC codes, this layered parallel decoder design has an area efficiency of around 2x and an energy efficiency of 10x for a given data throughput. When compared to the latest state-of-the-art fixed code parallel decoders, this design has an area efficiency of around 36x and the energy efficiency of 3x for a given data throughput. The numbers are normalized for a 180nm CMOS process. Note that the layered parallel decoder architecture has the best energy efficiency among all the presented architectures here. This advantage comes from the fact that the parallel architectures have lower switching activities when compared to the multi-rate architectures.

9.1.3. Comparison with Turbo Decoders

It is interesting to note that the Semi Parallel Turbo Codec of 3GPP- HSDA [63] has an energy efficiency of 10 nJ/ Bit/Iteration and an area efficiency of 1.65 Mbps/mm^2. This codec is not multi-rate and not code programmable. The work reported here for a multi-rate and code programmable TDMP decoder for Block LDPC achieves an energy efficiency of 47.3 pJ/Bit/Iteration and an area efficiency of 246 Mbps/mm^2. Note that the comparison is done for 180 nm CMOS technology.

Note that the Parallel Turbo Codec of 3GPP- HSDA [64] has an energy efficiency of 2.72 nJ/ Bit/Iteration and an area efficiency of 5.14 Mbps/mm^2. This codec does not have the features of multi-rate, code programmability and scalability for higher code lengths. The work reported here for a fixed code parallel TDMP decoder for regular array LDPC codes achieves an energy efficiency of 4.8 pJ/Bit/Iteration and an area efficiency of 493 Mbps/mm^2. This decoder is also not code programmable but is scalable for any code length and parameters in the design time. The comparison is done for 180 nm CMOS technology.

9.2. Future Work

We showed that, if the memory approach is used for storing the control signals for supporting the base parity check matrices as in the present state of the art [20], (for IEEE 802.16e fully compliant LDPC decoder in Chapter VII), it would have resulted in a large chip area of around 140 mm^2 (in 180 nm technology; 73 mm^2 in 130 nm technology) just for storing the control signals. The proposed approach of Master-Slave router removes this control memory overhead. However, this approach increased the area of the router from 0.15 mm^2 to 0.74 mm^2 in 130 nm technology. This area overhead is much smaller than the control memory overhead. However, by utilizing the fact that the limited set of permutations (only cyclic shifts on different vector lengths) are needed, further simplification of the master Benes router (that generates the control signals by sorting the integer sequence) is possible. For this a different sorting algorithm needs to be developed. In addition, the precision of the comparators at different stages can be adapted based on the maximum number of bits that can differ at the inputs of the comparator.

The parallel layered architecture proposed for regular array LDPC codes can be easily adapted for other regular QC-LDPC codes. However, in this case, the routing requirements will increase. Several research efforts are underway to design regular QC-LDPC codes which have better error performance than the existing regular QC-LDPC codes such as array LDPC codes. One criterion that can be considered in this design is: having a limited set of differences of shifts among the block column of the regular QC-LDPC. This would permit the decoder architecture to support a limited number of shifts for each block column.

As of now, the semiconductor industry is seeing the integrated circuits at 65 nm technology node. The migration of the designs presented here to the latest process will give significant gains to the numbers reported here.

9.3. Conclusion

The multi-rate and fixed code LDPC decoder architectures described in this dissertation achieve the best reported energy and area efficiencies while achieving the highest throughputs. These architectures are based on minimizing the message passing and computation requirements.

REFERENCES

[1] J. G. Proakis, *Digital Communications,* 4th ed. New York: McGraw-Hill: 2000.

[2] S. Lin and D. J. Costello, Jr., *Error Control Coding*, 2nd ed., Englewood Cliffs: Prentice Hall, 2004.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-Codes" in *Proc. IEEE Int. Conf. on Communications (ICC'93)*, Geneva, Switzerland, May 1993, pp.1064-1070.

[4] R. G. Gallager, *Low-Density Parity-Check Codes*, M.I.T Press, 1963. Available: http://justice.mit.edu/people/gallager.html

[5] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[6] D. MacKay and R. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, pp. 1645-1646, Aug. 1996.

[7] S. Chung, Jr., G. D. Forney, T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the shannon limit," *IEEE Communications Letters*, vol. 5, issue 2, pp. 58-60, Feb. 2001.

[8] A. Shokrollahi T.J. Richardson and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47 issue 2, pp. 619-637, Feb. 2001.

[9] J. L. Fan, "Array codes as low density parity check codes," in *Proc. 2nd International Symposium on Turbo Codes and Related Topics*, Brest, France, Sept. 2000, pp. 543–546.

[10] A. Dholakia and S. Olcer, "Rate-compatible low-density parity-check codes for digital subscriber lines," in *Proc. IEEE International Conference on Communications*, Jun. 2004, pp. 415–419.

[11] M.P.C. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. on Information Theory*, vol. 50, no. 8, pp. 1788- 1793, August 2004.

[12] "*Part 16: air interface for fixed and mobile broadband wireless access systems amendment for physical and medium access control layers for combined fixed and mobile operation in licensed bands*", IEEE P802.16e-2005, October 2005.

[13] *IEEE 802.11 Wireless LANsWWiSE Proposal: High Throughput extension to the 802.11 Standard.* IEEE 11-04-0886-00-000n.

[14] J. Castura, E. Boutillon and F.R. Kschischang, "Decoder first code design," in *Proc. 2nd International Symposium on Turbo codes and Related Topics*, Brest, France, September 2000, pp. 459-462.

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

[16] E. Yeo, P. Pakzad, B. Nikolic and V. Anantharaman, "High throughput low-density parity-check decoder architectures," in *IEEE Global Telecommunication Conference*, 2001 (GLOBECOM'01), vol. 5, pp. 3019-3024.

[17] A. Selvarathinam, G. Choi, K. Narayanan, A. Prabhakar, and E. Kim, "A massively scaleable decoder architecture for low-density parity-check codes," in*Proc. IEEE International Symposium on Circuits and Systems 2003 (ISCAS'03),* Bangkok, Thailand, vol. 2, May 2003, pp. 61-64.

[18] T. Zhang and K. Parhi, "A 56Mb/s (3, 6)-Regular FPGA LDPC decoder," in *Proc. IEEE SIPS 2002* San Diego, CA, Oct. 16–18, 2002, pp. 127-132.

[19] Y. Li; M. Elassal,; M. Bayoumi, "Power efficient architecture for (3, 6)-regular low-density parity-check code decoder," *in Proc. IEEE International Symposium on Circuits and Systems* 2004 (ISCAS '04), vol.4, May 2004, pp 23-26.

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

[21] A. Darabiha, A. C. Carusone and F. R. Kschischang, "Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity," in *Proc. IEEE International Symposium on Circuits and Systems 2005 (ISCAS'05), Kobe, Japan, May 2005.*

[22] Y. Chen and D. Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder," in *Proc. IEEE GLOBECOM,* San Francisco, CA, Dec. 2003, pp. 113–117.

[23] Flarion Technology, Vector-LDPC Coding Solution Data Sheet. Available: http://www.flarion.com/products/vector.asp.

[24] R. Singhal, G.S. Choi, and R. N. Mahapatra, "Programmable LDPC decoder based on the bubble-sort algorithm,", in *Proc. IEEE VLSI Design 2006*, Jan 2006, pp. 203-208.

[25] V. Nagarajan, N. Jayakumar, S. Khatri, and G. Milenkovic, "High throughput VLSI implementations of iterative decoders and related code construction problems", in *Proc. Global Telecommunications Conference,* 2004 (GLOBECOMM ‘04), vol. 1, 29 Nov.-3 Dec. 2004, pp. 361-365.

[26] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, “Reduced complexity iterative decoding of low density parity check codes based on belief propagation,” *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.

[27] A. Prabhakar and K. Narayanan, “A memory efficient serial LDPC decoder architecture,” in *IEEE International conference on Acoustics, Speech and Signal Processing,* 2005 *(*ICASSP 2005), Philadelphia, PA, vol. 5, 18-23 March 2005, pp. v/41: v/44.

[28] C. Jones, E. Valles, M. Smith and J. Villasenor, “Approximate-min constraint node updating for LDPC code design,” *in IEEE Conference on Military Communications, 2003*(MILCOM 2003), 13-16 Oct 2003, pp. 57-162.

[29] J. Chen and M. Fossorier, “Near optimum universal belief propagation based decoding of low-density parity check codes,” in *IEEE Transactions on Communications*, vol. COM-50, pp. 406-414, March 2002.

[30] J. Chen and M. Fossorier, “Density evolution for two improved BP-based decoding algorithms of LDPC codes,” *IEEE Communication Letters*, vol. 6, pp. 208–210, May 2002.

[31] F. Guilloud, E. Boutillon and J.L. Danger “λ-Min decoding algorithm of regular and irregular codes,” in *Proc. 3rd International Symposium on Turbo Codes & Related Topics*, Brest, France, Sept 2003, pp. 451-454.

[32] J. Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier and X.Y. Hu, “Reduced-complexity decoding of LDPC codes,” *IEEE Trans. on Communications*, vol. 53, no. 8, pp. 1288-1299, August 2005.

[33] J. Zhang, M. Fossorier, D. Gu and J. Zhang, “Two-dimensional correction for min-sum decoding of irregular codes,” *IEEE Communication letters*, vol. 10, issue 3, pp. 180-182, March 2006.

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

[35] S.Y. Chung, T.J. Richardson and R.L. Urbanke, “Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation,” *IEEE Trans. Inform. Theory*, vol. 47, no. 2, Feb 2001.

[36] E. Eleftheriou and S. Olcer, “Low density parity-check codes for digital subscriber lines,” in *Proc. Intl. Conf. on Communication 2002*, New York, pp.1752-1757.

[37] M. Karkooti, and J.R. Cavallaro, “Semi-parallel reconfigurable architectures for real-time LDPC decoding,” in *Proc. International Conference on Information Technology: Coding and Computing,* 2004 (ITCC 2004), vol. 1, pp. 579 – 585.

[38] D. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes”, in *IEEE Workshop on Signal Processing Systems (IEEE SIPS)*, October 2004, pp. 107-112.

[39] H. Sankar and K. R. Narayanan, “Memory-efficient sum-product decoding of LDPC codes,” *IEEE Trans. Comm.*, vol. 52, no. 8, pp. 1225- 1230, August 2004.

[40] B. Gocal, “Bitonic sorting on Bene networks”, in *Proceedings of the 10th International Parallel Processing Symposium* (April 15 - 19, 1996). IPPS. IEEE Computer Society, Washington, DC, 749-753.

[41] T. Brack, F. Kienle, and N. Wehn, “Disclosing the LDPC code decoder design space,” in *Proceedings of Design Automation and Test in Europe (DATE) Conference*, March 2006, pp. 200-205.

[42] L. Yang, M. Shen, H. Liu, and C. Shi, "An FPGA implementation of low-density parity-check code decoder with multi-rate capability," in *Proceedings of the Asia and South Pacific Design Automation Conference*, 18-21 Jan. 2005, vol. 2, pp. 760- 763.

[43] E. Kim and G. Choi, "Diagonal low-density parity-check code for simplified routing in decoder," in *IEEE Workshop on Signal Processing Systems (IEEE SIPS)*, Nov. 2005, pp. 756-761.

[44] Z. Wang and Z. Cui, "A Memory Efficient Partially Parallel Decoder Architecture for QC-LDPC Codes," *Conference Record of the Thirty-Ninth Asilomar Conf. on Signals, Systems and Computers*, 28 October-1 November 2005, pp. 729- 733

[45] H. Zhong and T. Zhang, “Block-LDPC: A practical LDPC coding system design approach,” *IEEE Trans. on Circuits and Systems-I*, vol. 52, no. 4, pp. 766-775, April 2005.

[46] T. Zhang and K.K. Parhi, “A 54 MBPS (3, 6)-regular FPGA LDPC decoder,” in *Proc. IEEE SIPS*, pp.127-132, 2002.

[47] S. Olcer, "Decoder architecture for array-code-based LDPC codes," in *Global Telecommunication Conference, 2003 (GLOBECOM'03)*, vol. 4, Dec 2003, pp. 2046-2050.

[48] E. Liao, E. Yeo and B. Nikolic, "Low-density parity-check code constructions for hardware implementations," in *IEEE Intl. Conf. on Communications, (ICC 2004)*, vol. 5, 20-24 June 2004, pp. 2573-2577.

[49] M. M. Mansour and N. R. Shanbhag, "Low power VLSI decoder architectures for LDPC codes," in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, Monterey, CA, August 2002, pp. 284-289.

[50] P. Bhagawat, M. Uppal and G. Choi, "FPGA based implementation of decoder for array low-density parity-check codes," in *IEEE International Conference on Acoustics, Speech and Signal processing, 2005* (ICASSP 2005), vol. 5, 18-23 Mar 2005, pp. 29-32.

[51] K. Gunnam, G. Choi and M. B. Yeary, "An LDPC decoding schedule for memory access reduction", in *IEEE International Conference on Acoustics, Speech and Signal processing, 2004 (*ICASSP 2004), vol. 5, 17-21 May 2004, pp. V- 173-6.

[52] K. Gunnam, W. Wang, E. Kim, G. Choi and M.B. Yeary, "Decoding of quasi-cyclic LDPC codes using an on-the-fly computation," Accepted for *40th Asilomar Conf. on Signals, Systems and Computers*, October 2006.

[53] K. Gunnam, G. Choi and M. B. Yeary, "A parallel layered decoder architecture for array LDPC codes," Accepted for IEEE VLSI Design Conference, January 2007

[54] K. Gunnam, G. Choi, W. Wang and M.B. Yeary, "VLSI architectures for turbo decoding message passing using min-sum for rate-compatible array LDPC codes," Accepted for *International Symposium on Wireless Pervasive Computing* February 2007.

[55] K. Gunnam and G. Choi, "A low power architecture for min-sum decoding of LDPC codes," TAMU, ECE Technical Report, May 2006, TAMU-ECE-2006-02. Available: http://dropzone.tamu.edu/techpubs

[56] K. Gunnam and G. Choi, "Architectures for decoding of structured LDPC codes using the on-the-fly computation paradigm", TAMU, ECE Technical Report, May 2006, TAMU-ECE-2006-04. Available: http://dropzone.tamu.edu/techpubs

[57] K. Gunnam, G. Choi, M. B. Yeary and M.Atiquzzaman, "VLSI architectures for layered decoding for irregular LDPC codes of WiMax", TAMU, ECE Technical Report, July 2006, TAMU-ECE-2006-08. Available: http://dropzone.tamu.edu/techpubs

[58] K. Gunnam, G. Choi, W. Wang and M. B. Yeary, “VLSI architectures for layered decoding for irregular LDPC codes of IEEE 802.11n,” TAMU, ECE Technical Report, July 2006, TAMU-ECE-2006-11. Available: http://dropzone.tamu.edu/techpubs.

[59] H. Zhong and T. Zhang, "Block-LDPC: A practical LDPC coding system design approach", *IEEE Trans. on Circuits and Systems I*, vol. 52, no. 4, pp. 766-775, April 2005.

[60] G. Malema and M. Liebelt, "Interconnection network for structured low-density parity-check decoders," *Asia-Pacific Conference on Communications*, 03-05 Oct. 2005, pp. 537- 540.

[61] M. Rovini, N. E. Insalata, F. Rossi, L. Fanucci, “VLSI design of a high throughput multi-rate decoder for structured LDPC codes,” in *Proc. 8th Euromicro Conference on Digital System Design*, Sept. 2005, pp. 202-209.

[62] Open source standard cell library. Available online: http://www.vlsitechnology.org Accessed January, 2006.

[63] M. Bikerstaff L. Davis, C. Thomas, D. Garrett, and C. Nicol, “A 24 Mb/s radix-4 LogMAP turbo decoder for 3GPP- HSDPA mobile wireless,” in *IEEE Int. Solid-State Circuits Conf.(ISSCC)* Dig. Tech. Papers, 2003, pp. 150–151.

[64] M. Bougard B. Bougard, A. Giulietti, V. Derudder, J. Weijers, S. Dupont, L. Hollevoet, F. Catthoor, L. Van der Perre, H. De Man, R. Lauwereins “A scalable 8.7 nJ/bit 75.6 Mb/s parallel concatenated convolutional (turbo-) codec,” in *IEEE Int. Solid-State Circuits Conf.(ISSCC)* Dig. Tech. Papers, 2003, pp. 152–153.

VITA

Kiran Kumar Gunnam is a Ph.D. candidate in the department of electrical and computer engineering at Texas A&M University. He obtained his MS degree in electrical engineering from the same department in May 2003. He has 6+ years of research and development work experience in real time implementation of communication and signal processing systems on VLSI and programmable platforms. He has 3+ years of industry research work experience at Intel, Schlumberger and Starvision Technologies. He has 3+ years of academic research work experience in Texas Engineering Experiment Station at Texas A&M University. His academic research contributed in a novel navigation sensor signal processing design (Visnav) which is now a commercial product and is considered for unmanned aerial refueling and space docking applications.

He is a recipient of the TAMU Ph.D. scholarship of $10,000 for VLSI architectures for communication systems and TAMU-Starvision Ph.D. scholarship of $11,000. In addition, he received TAMU tuition waivers and tuition scholarships of around $39,000 covering his graduate education for 2000-02 and 2005-06.

His contact e-mail address is kiran-k-gunnam@ieee.org. He can also be contacted through,

Dr. Gwan Choi
Associate Professor
Dept of Electrical & Computer Engineering
320D WERC MS-3259
Texas A&M University
College Station, TX 77843
gchoi@ece.tamu.edu   979-845-7486