DESIGN AND IMPLEMENTATION FOR A MULTI-STANDARD TURBO DECODER USING A RECONFIGURABLE ASIP

By

Eid Mohamed Abdel-Hamid Abdel-Azim

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND ELECTRICAL COMMUNICATIONS
ENGINEERING

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2013

DESIGN AND IMPLEMENTATION FOR A MULTI-STANDARD TURBO DECODER USING A RECONFIGURABLE ASIP

By
Eid Mohamed Abdel-Hamid Abdel-Azim

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND ELECTRICAL COMMUNICATIONS
ENGINEERING

Under the Supervision of

Dr. Ahmed F. Shalash	Dr. Hossam A. H. Fahmy
………………………………..	………………………………..
Associate Professor, ELECTRONICS AND ELECTRICAL COMMUNICATIONS Department Faculty of Engineering, Cairo University	Associate Professor, ELECTRONICS AND ELECTRICAL COMMUNICATIONS Department Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2013

DESIGN AND IMPLEMENTATION FOR A MULTI-STANDARD TURBO DECODER USING A RECONFIGURABLE ASIP

By
Eid Mohamed Abdel-Hamid Abdel-Azim

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND ELECTRICAL COMMUNICATIONS
ENGINEERING

Approved by the
Examining Committee

____
Dr. Emad Eldin Mahmoud Hegazi, External Examiner

____
Prof. Dr. Mohamed M. Khairy, Internal Examiner

____
Dr. Ahmed F. Shalash, Thesis Main Advisor

____
Dr. Hossam A. H. Fahmy, Member

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
May – 2013

Engineer's Name: Eid Mohamed Abdel-Hamid Abdel-Azim

Date of Birth: 26/8 /1986
Nationality: Egyptian
E-mail: Eid_mohamed300@yahoo.com
Phone: 01280826660
Address: 139 Tahrir street, Dokki, Giza
Registration Date: …./…./……..
Awarding Date: …./…./……..
Degree: Master of Science
Department: Electronics And Electrical Communications Engineering

Supervisors:
Dr. Ahmed F. Shalash
Dr. Hossam A. H. Fahmy

Examiners:

Dr. Emad Eldin Mahmoud Hegazi	(External examiner)
Prof. Dr. Mohamed M. Khairy	(Internal examiner)
Dr. Ahmed F. Shalash	(Thesis main advisor)
Dr. Hossam A. H. Fahmy	(Member)

Title of Thesis:

Design And Implementation For A Multi-Standard Turbo Decoder Using A Reconfigurable ASIP

Key Words:
Turbo Decoder; ASIP; High-throughput; Parallel Architecture; Memory Conflict

Summary:

This thesis presents an efficient architecture to implement a turbo decoder using a scalable low energy application specific instruction-set processor (ASIP). The parallelism on the ASIP architecture is proposed to achieve the high-throughput demand for turbo decoder which is one of the most important requirements of the Fourth Generation (4G) wireless communication systems. We show the effects on the throughput, the area, and the hardware utilizations of the different turbo decoder schemes.

Acknowledgments

I would like to thank my supervisors, Dr. Ahmed F. Shalash and Dr. Hossam A. H. Fahmy, for their continuous support, advice, and guidance throughout my work.

Special thanks to my dear mother, my dear father for their continuous support and encouragement during all working days.

I also wish to express my appreciation to my colleagues at the Center of Wireless Studies (CWS).

Dedication

To my mother, my father, my brothers and my sister.

Contents

Acknowledgments **i**

Dedication **ii**

Table of Contents **iii**

List of Tables **vi**

List of Figures **vii**

List of Symbols **x**

List of Abbreviations **xi**

Abstract **xii**

1 Introduction **1**
- 1.1 Introduction 1
- 1.2 Prior Work 2
  - 1.2.1 Parallel Memory Access 3
  - 1.2.2 Unified and ASIP Turbo Decoder Works 3
- 1.3 Design Flow 4

2 Turbo Codes **6**
- 2.1 Introduction 6
- 2.2 WiMAX convolution Turbo code 6
  - 2.2.1 Duo-binary Turbo Encoding 6
  - 2.2.2 WiMAX encoder 6
  - 2.2.3 Internal interleaver 7
  - 2.2.4 Circular state encoding 8
  - 2.2.5 Rates and puncturing block 9
- 2.3 3GPP-LTE convolution Turbo code 9
  - 2.3.1 Single binary Turbo Encoding 9
  - 2.3.2 LTE encoder 10
  - 2.3.3 Trellis termination 10
  - 2.3.4 Internal interleaver 11

3 Turbo Decoder Algorithms 12
3.1 SISO Decoding . . . 12
3.2 Log-MAP . . . 13
3.3 Max-log-MAP . . . 14
3.3.1 Branch State Metric . . . 14
3.3.2 Forward State Metric . . . 14
3.3.3 Backward State Metric . . . 15
3.3.4 Extrinsic LLR . . . 16
3.4 Unified Radix-4 decoding algorithm . . . 17
3.5 Enhancement Max Log MAP . . . 18
3.6 Decoder Design Strategies . . . 18
3.7 Sliding Window Max-Log-MAP . . . 19
3.8 Parallel Sliding Window First Scheme . . . 21
3.9 Parallel Sliding Window Second Scheme . . . 23
3.10 Trellis Termination . . . 23

4 Simulations of WiMAX and 3GPP-LTE Turbo Codes 24
4.1 Enhancement MAX Log MAP . . . 24
4.2 Effect of number of iterations . . . 24
4.3 Effect of Turbo interleaver block sizes . . . 24
4.4 Effect of Turbo Modulation Schemes . . . 25
4.5 Effect of Symbol selection (Puncturing) . . . 25
4.6 Effect of the Rayleigh selective fading channel on LTE . . . 25
4.7 Sliding Window MAX Log Map approximations . . . 26
4.7.1 Parallel Sliding Window Effects Using Guard Window and border states techniques . . . 26
4.8 Fixed point analysis . . . 27

5 Memory Conflict Analysis 37
5.1 Maximum Contention Free Interleavers . . . 37
5.2 Effect of Window Size on Memory Contentions . . . 38
5.3 The Second Scheme of Parallel Decoding Analysis . . . 38
5.3.1 Decreasing the Number of Conflicts . . . 38
5.3.2 Handling of the Conflicts . . . 41
5.3.3 Simulations Results for Memory Conflict . . . 42
5.4 Memory Conflict Handling of The First Scheme . . . 43

6 ASIP Architecture 46
6.1 ASIP Architecture . . . 46
6.2 State Metric Unit . . . 46
6.2.1 First Configuration . . . 48
6.2.2 Second Configuration . . . 49
6.2.3 Third Configuration . . . 49
6.3 Memory Access . . . 53
6.4 Branch Metric Unit . . . 54
6.5 LLR Calculation Unit . . . 56
6.6 Instructions . . . 57
6.7 Interfacing Between SISOs . . . 58

7 Inside The CPU **61**
7.1 Micro-instructions 63
7.1.1 NOP Instruction 63
7.1.2 Call Instruction 64
7.1.3 Ret Instruction 65
7.1.4 Goto Instruction 65
7.1.5 ZOL Instruction 66
7.1.6 LOOPNE Instruction 67
7.1.7 ParSISO Instruction 69
7.1.8 Initialize Instruction 71
7.1.9 Mov Instruction 72
7.1.10 StrData Instruction 73
7.1.11 Decode Instruction 73
7.1.12 Config Instruction 74

8 Results and Future Works **75**
8.1 Varying of the Window Sizes & the Parallel SISOs 75
8.2 Comparisons 75
8.3 Future Works 78
8.4 Conclusion 80

Bibliography **81**

List of Tables

2.1 Wimax Turbo code permutation parameters 8
2.2 Circulation state lookup table (S_C) 8
2.3 LTE Turbo code internal interleaver parameters 11

5.1 main parameters in different standards 43
5.2 Implementation parameters 43
5.3 Comparison between two designs for data alignment block 44
5.4 Throughput comparison between two designs for LTE standard 44
5.5 Memory analysis for radix-2 implementations (all block sizes for each standard) 44
5.6 Memory analysis for radix-4 implementations (all block sizes for each standard) 44
5.7 Comparison of memory conflict for HSPA+ (Radix-4 scheme) with 2 parallel SISOs 44

6.1 Comparison of Three Configuration for one SISO 51

7.1 The op-code of the Instructions 64
7.2 The NOP Instruction Description 64
7.3 The Call Instruction description 64
7.4 The Return Instruction description 65
7.5 The Goto Instruction description 66
7.6 The ZOL Instruction description 68
7.7 The LOOPNE Instruction description 68
7.8 The ParSISO Instruction description 69
7.9 The Initialize Instruction description 72
7.10 The Mov Instruction description 73
7.11 The StrData Instruction description 73
7.12 The Decode Instruction description 74
7.13 The Config Instruction description 74

8.1 Effect of Window Size on Power, area and throughput for p=16 & p=1 SISO 75
8.2 Turbo Decoder Parameters 78
8.3 Memory Sizes 78
8.4 Comparison with existing Turbo decoder architectures 79

List of Figures

1.1 Contention on memory bank 2, three simultaneous writing attempts from processing elements (PE) 1, 3 and 4 to memory bank 2. 2
1.2 Design flow approach from the top level desgin to ASIC/FPGA design . 5

2.1 Block diagram of Duo-binary CTC encoder 7
2.2 WiMAX CTC encoder 7
2.3 Sub-block interleaving and grouping 9
2.4 Block diagram of single binary CTC encoder 10
2.5 3GPP-LTE CTC encoder 11

3.1 General architecture for Turbo decoding: the interleaved/deinterleaved extrinsic information are passed between the two decoders 12
3.2 Calculation of one forward state metric, alpha, (a) for Duo-binary codes and (b) for single binary codes 15
3.3 Calculation of one backward state metric, beta, (a) for Duo-binary codes and (b) for single binary codes 16
3.4 Calculation of LLR 17
3.5 Conversion from radix-2 to radix-4 single binary turbo codes of LTE (HSPA+) trellis 18
3.6 Extrinsic Log Likelihood Ratio (LLR) Scaling 19
3.7 Two schemes for the decoding process: (a)sequential scheme, (b) butterfly scheme 20
3.8 Timing sequences for Sliding Window Max-Log-MAP showing the operation of how states are computed for different sub-blocks with time . . . 20
3.9 Sliding Window operation using a guard window technique 21
3.10 Timing sequences for Parallel SISOs 22
3.11 Timing sequences for Parallel SISOs Butterfly Scheme 23

4.1 Enhancement MAX Log MAP on WiMAX with N=240 couples, 4 iter and R=1/3 25
4.2 Enhancement MAX Log MAP on LTE with k=120 bits, 4 iter and R=1/3 . 26
4.3 EEffect of iteration numbers on WiMAX with N=240 couples, 4 iter and R=1/3 27
4.4 EEffect of iteration numbers on LTE with k=400 bits, 4 iter and R=1/3 . . 28
4.5 Interleaver block size effect on WiMAX with 4 iter and R=1/3 28
4.6 Interleaver block size effect on LTE with 4 iter and R=1/3 29
4.7 Effect of Turbo Modulation Schemes on WiMAX with N=240 couples, 4 iter and R=1/3 29
4.8 Effect of Turbo Modulation Schemes on LTE with K=400 bits, 4 iter and R=1/3 30

4.9 Symbol selection for WiMAX on AWGN with N=240 couples and 4 iter 30
4.10 Symbol selection for WiMAX on WiMAX channel model with N=240 couples and 4 iter 31
4.11 Effect of Symbol selection for LTE on AWGN channel with K=400 and 4 iter 31
4.12 Effect of number of iterations for LTE standard in rayleigh selective fading channel with 4 iter 32
4.13 Sliding Window MAX Log Map on AWGN N=240 couples 32
4.14 Sliding Window MAX Log Map on AWGN N=960 couples 33
4.15 Sliding Window MAX Log Map on WiMAX channel model 33
4.16 Performance between border memory and guard window technique for N=240 34
4.17 Performance between border memory and guard window technique for N=2400 34
4.18 Input quintizations (a)integer part only (b)fraction part only (c)both integer and fraction parts 35
4.19 Extrinsic LLRs quintizations (a)integer part only (b)fraction part only (c)both integer and fraction parts 36

5.1 An example for no collisions for WiMAX interleaver when M=1 37
5.2 An example for collisions happen for WiMAX interleaver when M=0.96 . 38
5.3 Maximum buffer sizes for 3GPP2 CDMA2K interleaver for different window sizes with 8 parallel windows 39
5.4 Maximum buffer sizes for LTE interleaver for different window sizes with 8 parallel windows 39
5.5 Maximum latency for 3GPP2 CDMA2K interleaver for different window sizes with 8 parallel windows 40
5.6 Parallel architecture with adding delayed buffers to reduce the number of concurrent values need to write by half at every clock 40
5.7 The data alignment block receives LLR values and stores them until processed by the targeted memory bank 41
5.8 The controller of the data alignment block with divided into p small controllers 42
5.9 The controller of the data alignment block with divided into p small controllers 45

6.1 Block diagram for pipelined ASIP architecture showing the different stages 47
6.2 State Metric Unit 49
6.3 parallel access with interleaed addresses with K=40, p=4, w=10 (a) for LTE, no conflicts happen, (b) for HSPA, conflicts happen on bank 4 . . . 50
6.4 Effect of three configurations on latency between the 2 decoders due to memory conflicts with W=64, P=16, for some block sizes of HSPA+ standard 52
6.5 Effect of three configurations on memory buffers in the data alignment block with W=64, P=16, for all block sizes of HSPA+ standard 52
6.6 Effect of three configurations on probability of simultenous accesses on memory banks with W=64, P=16, for all block sizes of HSPA+ standard . 53
6.7 Block diagram for channel data memory 55

6.8 Branch Metric Unit 56
6.9 Block diagram of the LLR calculation unit 57
6.10 Pseudo code description of Turbo decoding algorithm for single binary scheme 59
6.11 Interfacing between SISOs 60

7.1 General block diagram architecture for the processor 61
7.2 State diagram of non-pipelined processor 62
7.3 Block Diagram for generating the T_n and q_n signals 62
7.4 The process of the execution of the pipelined instructions 63
7.5 The formatting of the Return instruction 63
7.6 The formatting of the call instruction 64
7.7 The formatting of the Return instruction 65
7.8 The formatting of the Goto instruction 65
7.9 The description of how the ZOL instruction works 66
7.10 The formatting of the ZOL instruction 66
7.11 The block diagram of the ZOL instruction control parts including dedicated registers in details 67
7.12 A sample code using ZOL instruction is given in (a), the timing sequence of the executions of that sample is given in (b) 69
7.13 The formatting of the LOOPNE instruction 69
7.14 The description of how the LOOPNE instruction works 70
7.15 The formatting of the ParSISO instruction 70
7.16 The block diagram of SimDecoder unit 71
7.17 An example showing the two different modes for SimDecoder unit 71
7.18 The formatting of the Initialize instruction 72
7.19 The formatting of the Move instruction 72
7.20 The formatting of the StrData instruction 73
7.21 The formatting of the Decode instruction 74
7.22 The formatting of the Config instruction 74

8.1 Effect of varying Window size for single binary codes with P=16, K=6144, on Area, Power and Throughput 76
8.2 Effect of varying window sizes on buffer sizes with W=64, P=16, for all block sizes of LTE standard 76
8.3 Effect of varying number of parallel SISOs on the latency between the two decoders due to memory conflicts with W=64 and for all block sizes of 3GPP2 CDMA2000 standard 77

List of Symbols

N : The code block size in pair of bits
K : The code block size in bits
A : First systematic output sub-block of the CTC interleaver
B : Second systematic output sub-block of the CTC interleaver
Y : First Parity output sub-block of the CTC interleaver
W : Second Parity output sub-block of the CTC interleaver
$Y2$: Third Parity output sub-block of the CTC interleaver
$W2$: Fourth Parity output sub-block of the CTC interleaver
S_c : Circular state
$NSCH$: The number of available sub channels
$NCPC$: The modulation order
R : The coding rate
c :The input to the Turbo code internal interleaver
c' : The output from the Turbo code internal interleaver
z : First Parity output of the LTE interleaver
z' : Second Parity output of the LTE interleaver
$lnP(u_k|\bar{y})$: a-posteriori-probability
$P_{ext}(u_k|\bar{y})$:The extrinsic LLR in log domain
$\bar{y} = \{A, B, Y, W\}$:The noisy soft input values
W : Window size
WG : Guard window size
k : The time instant
s_k : The current state at time instant k
$\alpha_k(s)$: The forward state metric at time instant k
$\beta_k(s)$: The backward state metric at time instant k
γ_k : the branch state metric
T_k : The branch LLRs at time instant k
Λ_{ext} : The extrinsic LLR of received data
P : Number of the parallel SISOs
Rb : the decoding throughput
i : Iterations number
f_{clk} : the operating frequency
q : The bit-width of the fractional part
int : The bit-width of integer part

List of Abbreviations

RSC : Recursive Systematic Convolutional
CRSC : Circular Recursive Systematic Convolutional
CTC : Convolutional Turbo Codes
SISO : Soft In Soft Out
MAP : Maximum Aposterior Probability
LLR : Log Likely-hood Ratio
BER : Bit error rate
FER : Frame error rate
NII : The next iteration initialization
RPDP : Relative power delay product
ZOL : Zero overhead loop
FPGA : Field Programmable Gate Array
LDPC : Low Density Parity check
QAM : Quadrature Amplitude Modulation
QPSK : Quadrature Phase shift keying
WiMAX : Worldwide Interoperability for Microwave access
LTE : Long Term Evolution
HSPA : High Speed Packet Access
3*GPP* : 3rd Generation Partnership Projects
DVB − SH : Digital Video Broadcasting - Satellite services to Handhelds
DVB − RCS : Digital Video Broadcasting with Return Channel over Satellite
DVB − RCT : Digital Video Broadcasting Return Channel Terrestrial
CDMA : Code Division Multiple Access

Abstract

This thesis presents an efficient architecture to implement a turbo decoder using a scalable low energy application specific instruction-set processor (ASIP). The parallelism on the ASIP architecture is proposed to achieve the high-throughput demand for turbo decoder which is one of the most important requirements of the Fourth Generation (4G) wireless communication systems. The parallel architecture is achieved by using multiple soft-in/soft-out (SISO) decoders. A scalable Interfacing between the parallel SISOs are also proposed. Three implementations of the turbo decoder have been proposed. We show the effects on the throughput, the area, and the hardware utilizations of the different turbo decoder schemes. The parallel architecture leads to conflicts during memories accesses. A complete memory conflict analysis for different interleaver patterns has been performed and shows the effect of using different decoding configurations on the memory conflicts for different standards. Such a conflict adds latency and reduces the throughput significantly. A simple controller is designed to manage the conflicts on the fly. The proposed design is synthesized in 180 nm technology and achieves throughput of 171 Mbps, power of 236.9 mW using 16 parallel SISOs running at 100 MHz.

Chapter 1

Introduction

1.1 Introduction

In wireless communication systems, the channel coding block is an important tool for improving communications reliability. The discovery of turbo codes [1] was probably the most significant breakthrough in the field of channel coding since the introduction of trellis codes. Over the past few years, many communication standards such as Digital Video Broadcast - Return Channel Satellite (DVB-RCS) [2], HSPA+ [3], 3GPP-LTE [4], WiMAX [5] and 3GPP2-CDMA2000 [6] have adopted Turbo codes due to their near Shannon-capacity performance.

Lately, a need for one configurable engine to be used with these different standards emerged. This engine should include one reconfigurable Turbo decoder for the different Turbo codes used in these standards. The Turbo decoder is one of the most difficult blocks in any communication chain which requires high throughput, adequate area, and low power. The efficient implementation of Turbo decoders is essential to improve the overall system performance.

The future wireless devices which will be made from additional and high specifications than the current standards involved, need to be compatible with the same platforms which exist in the markets. There is no direct answer to the question of which is the most efficient platform. Many platforms are proposed and the cost of the design to meet the required performance varies. The general purpose processor (GPP) fulfills the complete flexibility at the expense of the power and the throughput requirements. The throughput of GPP is very low because the instructions and the architectures are not designed for wireless system domains. In addition, low power consumption is an important goal that should be achieved in wireless devices. The digital signal processor (DSP), which is the current heart of software defined radio (SDR), also is convenient for transferring from one standard to another. However, the power consumption is the restriction for such platforms as well.

On the other hand, the application specific instruction-set processors (ASIP) emerged as one of the most important platforms. The ASIP strikes a balance between general purpose platforms and dedicated platforms by targeting specific applications. The purpose of developing ASIP architecture is reducing the time between changing from one application to another with adequate resources. Time to market is a key motive behind using ASIP architectures. The ASIP architecture collects some flexibility and dedicated blocks to achieve the high demand requirements for the current and next generation applications. applications. The instruction of the ASIP is fully optimized for the target applications.

The authors in [7] show a comparison between four different platforms. These platforms can be categorized as a DSP, SDR processor, application-specific processor (ASP) and ASIP. The K-best LSD algorithm is implemented on the four programmable platforms taking into account the current trend in the wireless communication research. The ASIP implementation achieves the best results amongst these implementations.

Turbo decoding is based on an iterative algorithm. A sequential implementation will not be suitable to achieve a high throughput. There are two methods to speed up the decoding process:

- Using a parallel decoder architecture by dividing the whole data block length into a number of windows to allow parallel access. This method works for all types of Turbo codes such as: binary and double-binary Turbo codes [8] [9].
- Applying one-level look-ahead recursion [10] [11] which leads to doubling the throughput and may lead to a unified decoding architecture that is suitable to support multiple standards. Applying this method can reduce the trellis cycles by 50% for single binary Turbo codes.

Both approaches lead to contention on memory due to parallel access, resulting in memory conflicts. An example of these conflicts is shown in Figure 1.1 where four processing elements (PE) are writing simultaneously with interleaved addresses which lead to contentions on memory bank 2.

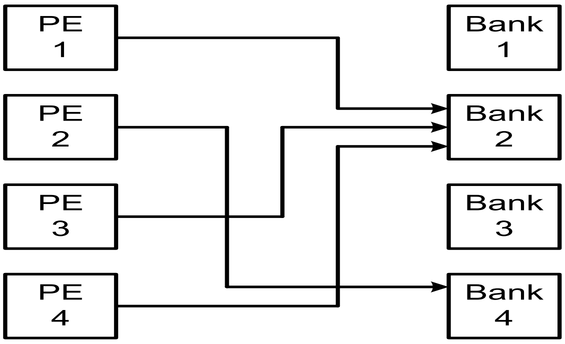


Figure 1.1 Contention on memory bank 2, three simultaneous writing attempts from processing elements (PE) 1, 3 and 4 to memory bank 2.

Another source of memory conflicts is the interleavers contained in Turbo codes. The interleaver patterns affect the memory conflicts. There are two types of interleavers, unconstrained interleavers and constrained interleavers. The constrained interleavers are called maximum contention free (MCF) [12] which mean no conflicts occur due to parallel accesses. There are few standards that include contention-free interleavers such as WiMAX and 3GPP-LTE.

1.2 Prior Work

Over the past few years, several research groups tried to propose ASIP architectures. Specifically, the parallel Turbo decoder designs and its effects on the memory conflicts have received much attention.

1.2.1 Parallel Memory Access

In [13] and [14], the authors propose a unified parallel radix-4 Turbo decoder architecture to support WiMAX and 3GPP-LTE standards in order to benefit from the maximum contention free properties for these standards. However, these works are limited by interleavers properties and can not support the other remaining standards.

In [15], the authors propose a memory mapping algorithm avoiding any contention by cascading three different layers of permutations. The proposed method does not optimize neither the cost of the architecture nor the throughput due to the latency resulting from executing the algorithm. In [16] the avoidance of conflicting access is done through building a network of General-Interleaved-Bottleneck-Breaker nodes (GIBB) based on a random graph generation algorithm. These nodes are not efficient to be a backbone of the network because they require complex routing and many resources.

In [17] buffering of the conflicting data until processed by the targeted memory was proposed to avoid memory contentions. However, the way to determine the size of the buffers was not efficient. The size was obtained at design time by profiling the RTL-model. In addition, the scheme supported only one standard, which is not suitable for multi-standard configurations.

In [18] the authors propose buffer architecture based on analysis of memory conflicts. This work reduces the area and offers higher operating frequencies by selecting the size of the buffers based on a standard case instead of the worst case. The stalling mechanism has been produced to treat the worst-case situations adding variable delay dependent on the block length. These features make the scheme not suitable for constrained real time systems.

A good analysis of memory conflicts for multi-standards Turbo decoders is provided in [19]. However, the memory conflicts for the unified parallel radix-4 Turbo decoder and some standards such 3GPP2-CDMA2000 have not been analyzed. The analysis of memory conflicts for a unified parallel radix-4 Turbo decoder is provided in [10]. A higher number of extra cycles have been added to avoid the contentions on memories per half-iteration for HSPA+ standard. Furthermore bigger FIFO sizes have been used.

1.2.2 Unified and ASIP Turbo Decoder Works

The authors in [20] proposed a unified architecture for supporting the Turbo codes of WiMAX and UMTS. The parallel decoding is used for Turbo codes of WiMAX which requires higher throughput than Turbo codes of UMTS. In this design, the Turbo decoder of UMTS is designed without parallel architecture due to a lower throughput requirement. This leads to a low efficiency in the usage of the hardware resources. For UMTS case, the design uses full hardware resources by reducing the clock frequency.In addition, the design can handle the collisions which happen due to the parallel access for UMTS case and without complicated mechanisms.

The authors in [21] show a pipelined ASIP processor which supports two types of Turbo codes. The throughput of the design is 34Mbps and 17Mbps at five iterations for duo-binary and binary Turbo codes respectively. Such design is not suitable for 4G wireless communication standards as it can not satisfy the throughput requirement.

A multi-processor platform based on a reconfigurable ASIP processor for the application domain of channel decoding is presented in [22]. The flexibility of that design is very poor as the duo-binary Turbo codes are not implemented. The duo-binary Turbo codes emerged in many standards such as WiMAX, DVB-RCS and DVB-RCT.

The proposed ASIP-based multiprocessor architecture in [23] is based on the shuffled decoding technique. The shuffled decoding technique allows two MAP decoders concurrently working and passing the extrinsic LLRs. The presented ASIP can process the duo-binary schemes twice faster than the single binary schemes. This leads to significant degradation in the hardware utilization in case of the single binary schemes.

A Turbo Decoder implementation on a multi-processor platform based on sub-block parallelization is proposed in [24]. The authors proposed two configurations to handle the topology to connect the multi-processors. Both configurations take large number of cycles for producing the LLRs which lead to large degradation on the throughput. The proposed work achieves a throughput of 22.64 Mb/s by using 16 parallel processors with 5 iterations. The proposed processor is not designed using ASIP technique. Such configurations are not suitable for high throughput requirements.

The authors in [25] presented a programmable SIMD-based (Single Instruction Multiple Data) DSP architecture for SDR that includes a set of architectural features to accelerate Turbo decoder computations. The proposed SIMD processor supports the parallel computations. However the proposed design is more flexible as it targets software designs at the expense of the dedicated blocks. This SW approach is not sufficient to achieve the throughput and the power requirements for wireless devices.

In this thesis, ASIP architecture for a scalable and reconfigurable multi-standard Turbo decoder is proposed. In our architecture, we avoid using complex techniques to handle the high-throughput demand and propose different implementations to achieve good hardware resources by building a unified architecture which is supporting different classes of Turbo codes. Our design offers a good compromise between the flexibility of the design which is required for transferring from one scheme to another and the demands of wireless communication devices such as power consumption and throughput. In addition, our work presents an analysis of the memory conflicts for a unified multi-standard Turbo decoder and provides efficient techniques to satisfy their requirements.

1.3 Design Flow

The typical RTL design approach, as shown in Figure 1.2, consists of three main steps:

- System simulation to ensure correct functionality, which is done using high level languages like MATLAB, C++ Ě
- Hardware design to implement the system towards the ASIC, which is done using Hardware description languages like VHDL, Verilog Ě
- Physical design to convert the obtained HDL into real chip.

The result comparisons are performed after each step to ensure the correctness of the design.

In the next chapters, the system simulations had been performed for the double-binary convolutional turbo Coding used in WiMAX IEEE 802.16e standard and single binary convolutional turbo Coding used in 3GPP-LTE. The influence of the turbo interleaver block sizes, number of iterations, code rates, sliding window MAX Log MAP, and quantization of the internal signals have all been studied. An enhancement, by applying the

Max log MAP to the decoder, to reduce the performance gap against log MAP decoder, has also been studied in detail.

In addition, the hardware design had been proposed in details. The reconfigurable turbo decoder processor architecture and different implementation schemes had been considered.

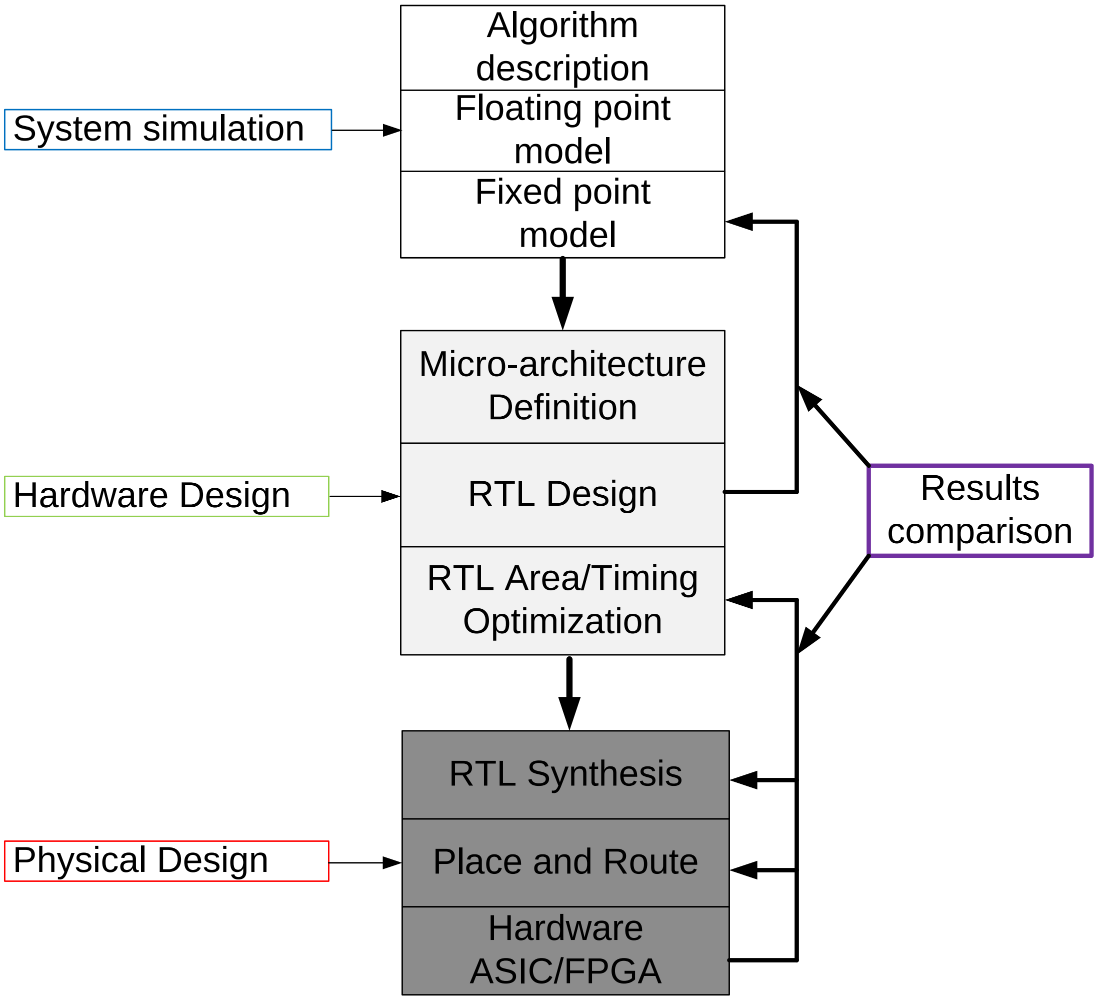


Figure 1.2 Design flow approach from the top level desgin to ASIC/FPGA design

Chapter 2

Turbo Codes

2.1 Introduction

There are two types of Turbo codes which are used in wireless communication systems: single binary and duo-binary Turbo codes. The single binary Turbo codes encode one bit at a time which is called radix-2 and is used in many standards such as 3GPP-LTE, HSPA+, and 3GPP2-CDMA2000. On the other side, the duo-binary Turbo codes encode two bits at a time which is called radix-4 and is used in WiMAX and DVB-RCS standards.

The details of the double-binary convolutional Turbo coding used in WiMAX IEEE 802.16e standard and single binary convolutional Turbo Coding used in 3GPP-LTE are presented. The decoding algorithms also are proposed for both types single and duo-binary Turbo codes.

2.2 WiMAX convolution Turbo code

2.2.1 Duo-binary Turbo Encoding

A typical duo-binary convolutional Turbo encoder consists of two identical Recursive Systematic Convolutional (RSC) encoders with parallel concatenation separated by an interleaver. A RSC encoder has typically R= 2/4 coding rate. Parallel concatenation means two RSC encoders encoding at the same time. Figure 2.1 shows the block diagram of the Turbo encoder that is used by WiMAX with rate R= 2/6. The inputs, A and B, are first coded in their natural order in encoder ENC1, retrieving parity bits Y1 and W1. Then the input is interleaved and encoded again in the equivalent encoder ENC2, retrieving parity bits Y2 and W2. The outputs from the two encoders are almost uncorrelated due to the interleaver.

2.2.2 WiMAX encoder

CTC encoder, including its constituent encoder, is depicted in Figure 2.2. It uses a double binary CRSC (Circular Recursive Systematic Convolutional) code. The bits of the data to be encoded are alternatively fed to A and B, starting with the MSB of the first byte being fed to A, followed by the next bit being fed to B. The encoder is fed blocks of k bits or N couples (k=2N bits), where k is a multiple of 8 and N is a multiple of 4.

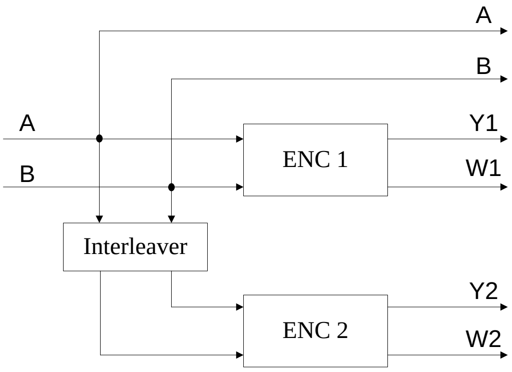


Figure 2.1 Block diagram of Duo-binary CTC encoder

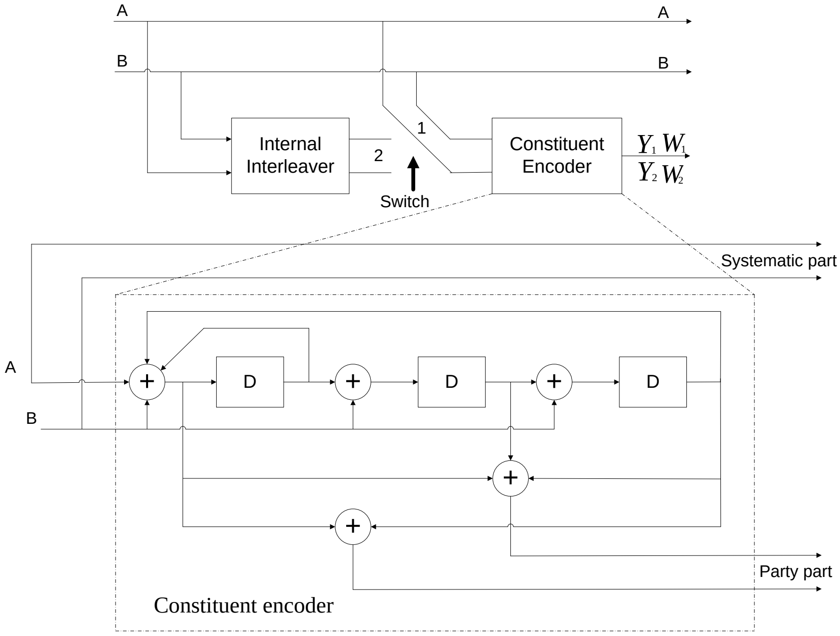


Figure 2.2 WiMAX CTC encoder

2.2.3 Internal interleaver

The CTC interleaver specified in IEEE802.16e consists of two permutation steps, one is a permutation on the level of each symbol individually, and the second is on the level of the sequence of all symbols. The two-step interleaver shall be performed by:

Step 1. Switch alternate couples

$For\ j = 0\ to\ N-1$

$If(j_{mod2} == 0)\ let(B,A) = (A,B)$

Step 2. $P_i(j)$

The function $P_i(j)$ provides the interleaved address i of the considered couple j.

$For\ j = 0\ to\ N-1$

Switch j_{mod4}

Case 0: $i = (P_0.j+1)_{mod\ N}$

Case 1: $i = (P_0.j+1+N/2+P_1)_{mod\ N}$

Case 2: $i = (P_0.j+1+P_2)_{mod\ N}$

Case 3: $i = (P_0.j + 1 + N/2 + P_3)_{mod\,N}$

Where P_0, P_1, P_2 and P_3 are coding parameters that are specific for each block size, N, and are provided in the standards. The address j is interleaved to address i. Table 2.3 provides the combinations of the default parameters to be used

N	P_0	P_1	P_2	P_3
24	5	0	0	0
48	13	24	0	24
96	7	48	24	72
120	13	60	0	60
192	11	96	48	144
216	13	108	0	108
240	13	120	60	180
480	13	240	120	360
960	17	1200	600	1800

Table 2.1 Wimax Turbo code permutation parameters

2.2.4 Circular state encoding

The tail biting scheme used in IEEE802.16e Turbo encoder is circular coding, this scheme guarantees that the initial state is the same as final state. The sequence of determination of circulation states S_{c1}, S_{c2} is:

- Initializing the encoder with state 0. Encode the sequence in the natural order for the determination of S_{c1} and in the interleaved order for determination of S_{c2}. In both cases the final state of the encoder is $S0_{N-1}$
- According to the length N of the sequence, determining S_{c1} or S_{c2} as given in Table 2.2

	$S0_{N-1}$							
N_{mod7}	0	1	2	3	4	5	6	7
1	0	6	4	2	7	1	3	5
2	0	3	7	4	5	6	2	1
3	0	5	3	6	2	7	1	4
4	0	4	1	5	6	2	7	3
5	0	2	5	7	1	3	4	6
6	0	7	6	1	3	4	5	2

Table 2.2 Circulation state lookup table (S_C)

2.2.5 Rates and puncturing block

The next step after encoding is to generate subpackets with various coding rates depending on channel conditions, the rate 1/3 CTC encoded codeword goes through interleaving block then puncturing is performed to generate subpackets. Sub-block interleaving is performed to get a robustness against burst errors and to rearrange the data so that puncturing of the data can be performed in a simple way. The output from the encoder is arranged in sub-blocks (A, B, Y1, Y2, W1 and W2) and each block is interleaved with a special sub-block interleaver as shown in Figure 2.3. An equation 2.1 states the interleaver function used for the sub-block interleaving. T_k represents the output addresses; m and J are interleaver parameters that are provided in a look up table and depend on the block size, N. If the result from the function is larger than the block size, the output address is disregarded and the variable k is increased and a new address is calculated. BRO is the m-bit bit reverse order.

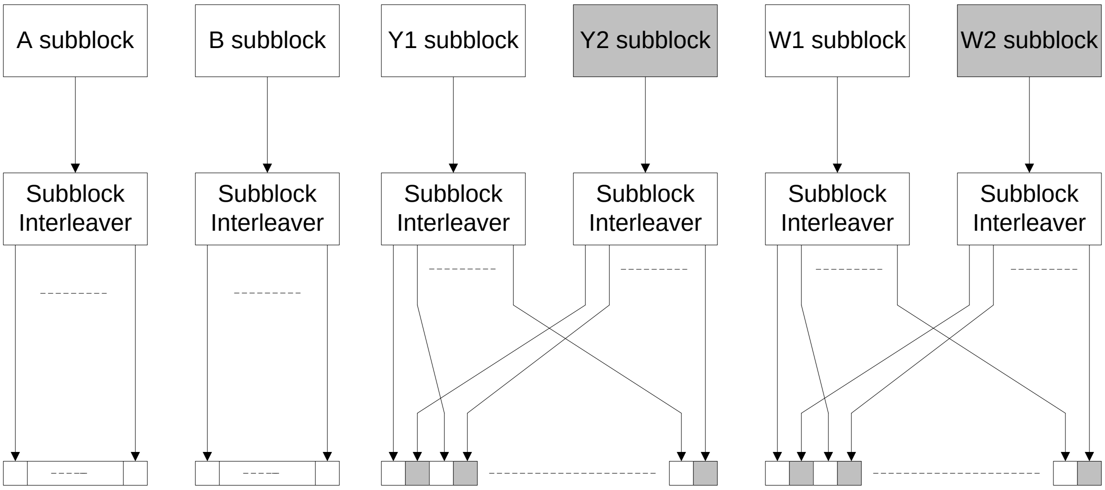


Figure 2.3 Sub-block interleaving and grouping

$$T_k = 2^m (K \, mod \, J) + BRO_m(\lfloor K/J \rfloor) \tag{2.1}$$

The output from the interleavers is combined serially after the sub-block interleaving. The systematic bits are grouped consecutively, and then the parity bits are grouped alternating one bit from Y1 and then one bit from Y2 et cetera. The puncturing is performed by selecting a number of consecutive bits according to Equation 2.2. The puncturing function depends on the block size, the number of available sub channels, NSCH, and the modulation order NCPC. Send Bits i=0,. . . ,L

$$L = 48 * N_{SCH} * N_{CPC} \tag{2.2}$$

2.3 3GPP-LTE convolution Turbo code

2.3.1 Single binary Turbo Encoding

A typical single binary Turbo encoder consists of two identical Recursive Systematic Convolutional (RSC) encoders with parallel concatenation separated by a random interleaver. An RSC encoder has typically 1/2 coding rate. Figure 2.4 shows a Turbo encoder with 1/3 coding rate. The input bits block C is first encoded by encoder 1. Since the encoder is systematic, the first output c is equal to the input bit c. The second output is the first parity bit z encoded by encoder 1. Encoder 2 receives interleaved input bit and outputs the second parity bit z'. The main purpose of the interleaver before encoder 2 is to avoid burst error and increase the minimum distance of Turbo codes.

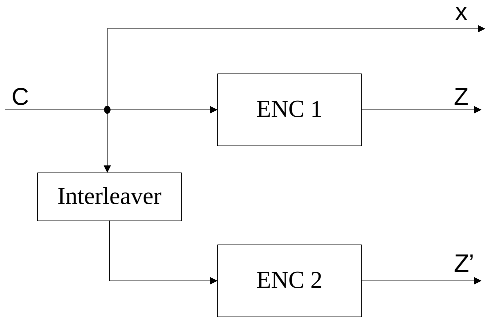


Figure 2.4 Block diagram of single binary CTC encoder

2.3.2 LTE encoder

LTE Turbo encoder employs a Parallel Concatenated Convolutional Code with two constituent encoders and one internal interleaver. The coding rate is 1/3. Figure 2.5 shows the structure of the Turbo encoder. The transfer function of the 8-state constituent code for the encoder is

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right], \tag{2.3}$$

where

$$g_1(D) = 1 + D^2 + D^3, \; and \; g_0(D) = 1 + D + D^3. \tag{2.4}$$

The initial value for the shift registers of the 8-state constituent encoders shall be all zeros when starting to encode the input bits. The output from the Turbo encoder is $d_k^{(0)} = x_k$, $d_k^{(1)} = z_k$, $d_k^{(2)} = z_k^{'}$ for $k = 0, 1, 2, \ldots, k-1$. K is the code block size from 40 to 6144 bits.

2.3.3 Trellis termination

Trellis termination is performed by taking the tail bits from the shift register feedback after all information bits are encoded. Tail bits are padded after the encoding of information bits. The first three tail bits shall be used to terminate the first constituent encoder (upper switch of Figure 2.5 in lower position) while the second constituent encoder is disabled. The last three tail bits shall be used to terminate the second constituent encoder (lower switch of Figure 2.5 in lower position) while the first constituent encoder is disabled. The output bits after trellis termination should be $x_k, z_k, x_{k+1}, z_{k+1}, x_{k+2}, z_{k+2}, x_k^{'}, z_k^{'}, x_{k+1}^{'}, z_{k+1}^{'}, x_{k+2}^{'}, z_{k+2}^{'}$

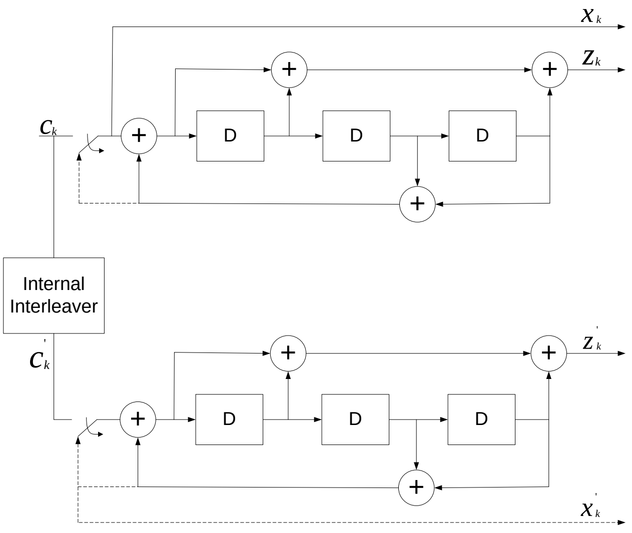


Figure 2.5 3GPP-LTE CTC encoder

2.3.4 Internal interleaver

The bits input to the Turbo code internal interleaver are denoted by $c_0, c_1, \ldots, c_{k-1}$, where K is the number of input bits. The bits output from the Turbo code internal interleaver are denoted by $c'_0, c'_1, \ldots, c'_{k-1}$. The relationship between the input and output bits is as follows:

$$c'_i = c_{\Pi(i)},\ i = 0, 1, \ldots, K-1 \tag{2.5}$$

where the relationship between the output index i and the input index $\Pi(i)$ satisfies the following quadratic form:

$$\Pi(i) = (f_1.i + f_2.i^2) mod\ K \tag{2.6}$$

The parameters f_1 and f_2 depend on the block size K and are summarized in Table 2.3.

K	f_1	f_2
40	3	10
120	103	90
200	13	50
400	151	40
640	39	80
1024	31	64

Table 2.3 LTE Turbo code internal interleaver parameters

Chapter 3

Turbo Decoder Algorithms

The Turbo decoding process is an iterative process consisting of two Soft In Soft Out (SISO) Maximum Aposterior Probability (MAP) decoders, as shown in Figure 3.1. Each MAP decoder receives the soft values from the transmitter through the communication channel. The input to the decoder is kept constant, the decoding is performed several times, and only extrinsic information (sub-results) is passed between the iterations. Each decoder calculates the Log Likely-hood Ratios (LLRs) which are the sum of two components: the intrinsic LLRs coming from the communication channel and the extrinsic LLRs added by the decoder itself. After each half-iteration, the decoders exchange their extrinsic information. The decoding algorithm requires several iterations to estimate the transmitted data. After the predetermined number of iterations, typically 4-8 depending on demands for BER (bit error rate) and FER (frame error rate), a final decision is made by using the extrinsic information from the two SISO decoders and the systematic soft bits from the demodulator.

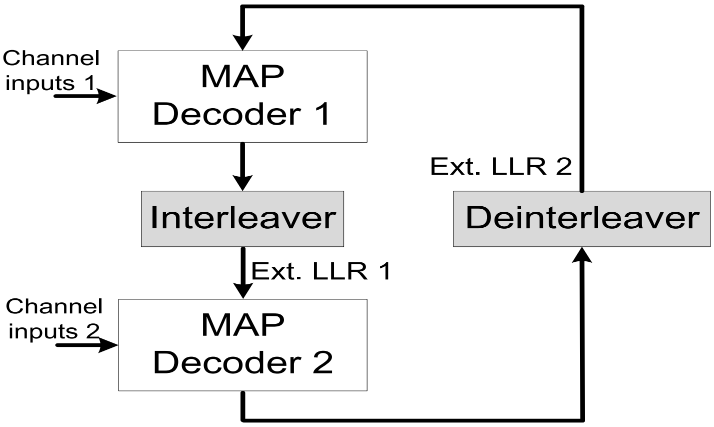


Figure 3.1 General architecture for Turbo decoding: the interleaved/deinterleaved extrinsic information are passed between the two decoders

3.1 SISO Decoding

BCJR is an optimal algorithm for estimating a-posteriori-probabilities of states and state transitions. Berrou [1] modified the algorithm to estimate the probability of each information bit (for binary codes, pair of information bits for duo binary codes). The modification of the algorithm is often referred to as the MAP (maximum a posteriori) algorithm. It is suitable for iterative turbo decoding because it is a SISO. An important property is that the soft output data can be used in the next iteration to calculate more accurate values. All calculations in the MAP algorithm are performed in the log domain to avoid numerical problems and unnecessary multiplications; the algorithm is then called the log-MAP algorithm. The MAP algorithm requires many multiplications and divisions. All calculations in the MAP algorithm are performed in the log domain to avoid numerical problems and unnecessary multiplications; the algorithm is then called the log-MAP algorithm. Nevertheless the log-MAP algorithm is complex and requires extensive hardware resources. Many simplified versions of MAP algorithm were proposed to be suitable for hardware implementation such as linear-log-MAP, constant-log-MAP and Max-log-MAP [26]. The performance degradation due to simplification in MAP algorithm can be compensated by using an enhancement Max-Log-MAP where a scaling factor scales the extrinsic LLRs.

3.2 Log-MAP

In the decoding process, the goal is to calculate an accurate a-posteriori-probability for the received block that can be used to make a hard decision by guessing on the largest APP for each information bit (for binary codes, pair of information bits for duo binary codes) when all iterations are complete. Equation 3.1 represents the APP that can be calculated iteratively by calculating the metrics in Equations 3.2, 3.3 and 3.4.

$$lnP(u_k|\bar{y}) = ln(\sum exp(\alpha_{k-1}(s) + \gamma_k(s,s') + \beta_k(s'))) \tag{3.1}$$

$$\alpha_k(s) = ln(\sum_{all s'}(\alpha_{k-1}(s') + \gamma_k(s,s'))) \tag{3.2}$$

$$\beta_{k-1}(s) = ln(\sum_{all s}(\beta_k(s) + \gamma_k(s,s'))) \tag{3.3}$$

$$\gamma_k = \sum(-1)^{b_0}.A + (-1)^{b_1}.B + (-1)^{b_2} + (-1)^{b_3}.W + lnP(u_k) \tag{3.4}$$

The values of $\bar{b} \in \{0,1\}$ depends on the encoding polynomial and can be pre-calculated for all state transitions, respectively. The extrinsic information from the last stage is denoted $lnP(u_k)$ and $\bar{y} = \{A,B,Y,W\}$ represents the noisy soft input values, where s is the current state and s' is the the transition state. The parameters α, β and γ are explained in detail later.

Extrinsic information for the next stage is calculated according to Equation 3.5.

$$P_{ext}(u_k|\bar{y}) = lnP(u_k|\bar{y}) - (-1)^{b_0}.A - (-1)^{b_1}.B - lnP(u_k) \tag{3.5}$$

The high complexity equations above can be implemented by using a rearranged version of the function,Equation 3.6, and using look up tables for the correction term. A better implemented correction term gives better error correction capability. Constant-log-MAP and linear-log-MAP are two implementation of the SISO algorithm with different correction terms.

$$ln(e^{x_1} + \cdots + e^{x_n}) = max(x_1, \ldots, x_n) + f(x_1, \ldots, x_n) \tag{3.6}$$

3.3 Max-log-MAP

Max-log-MAP is a simplification of the log-MAP algorithm that makes the estimation stated in Equation 3.7; i.e. disregard the correction term.

$$ln(e^{x_1} + \cdots + e^{x_n}) \approx max(x_1, \ldots, x_n) \tag{3.7}$$

Lower complexity usually brings some disadvantages and that is the case here: the error correcting capability is degraded for this algorithm. This improvement in robustness is explained by Berrou et al in [27].

In this thesis, the max-log-MAP decoding algorithm [28] will be used. As it has low computational complexity, it is widely used for SISO decoding in the turbo decoder. The max-log-MAP algorithm includes one forward and one backward recursion through the received soft input data and a number of different metrics are calculated, these will be explained in detail.

As mentioned earlier, there are two types for Turbo codes: single binary and duo-binary Turbo codes. The single binary Turbo codes decode one bit at time which called radix-2. On the other side, the duo-binary Turbo codes decode two bits at time which called radix-4. The extrinsic LLR is calculated from three metrics: forward states, backward states and branch metrics.

3.3.1 Branch State Metric

The Turbo codes can be defined using a trellis where at every time k there exist the same number of possible states. Define $\gamma_k^{(j)}(s_{k-1}, s_k)$ as the branch metrics between the state s_k at time k and state s_{k-1} at time $k-1$ and j represents the different combinations of the systematic bits, where $j \in [0,1]$ for radix-2 scheme and $j \in [00, 10, 01, 11]$ for radix-4 scheme. The metric $\gamma_k^{(j)}(s_{k-1}, s_k)$ represents the probability that a transition between these two states occurred. The branch metric $\gamma_k^{(j)}(s_{k-1}, s_k)$ can be calculated from

$$\gamma_k^{(j)}(s_{k-1}, s_k) = \sum_{j=1}^{m} x_{k,j} y_{k,j} + \Lambda_{apr,k}^{(j)} \tag{3.8}$$

where the branch metric depends on the received soft inputs $y_{k,j}$, the extrinsic information received from previous decoding iteration and re-named apriori information $\Lambda_{apr,k}^{(j)}$ and the possible transmitted bits $x_{k,j} \in \{0,1\}$.

3.3.2 Forward State Metric

The forward state metric, $\alpha_k(s)$ (alpha), represents the possibilities for the encoder to be in a specific state at the time instance k when considering all data received up to the moment k . This metric is calculated in a forward recursion through the trellis as shown in Figure 3.2. Equation 3.9 is the function used for the calculation of the forward state metric.

$$\alpha_k(s) = \max_{(s',j)}(\alpha_{k-1}(s') + \gamma_{k,j}(y_k, s, s')) \tag{3.9}$$

where the state considered is denoted s and the possible states that can result in a transition to this state s are denoted $s^{'}$. The state metric calculation in radix-2 is based

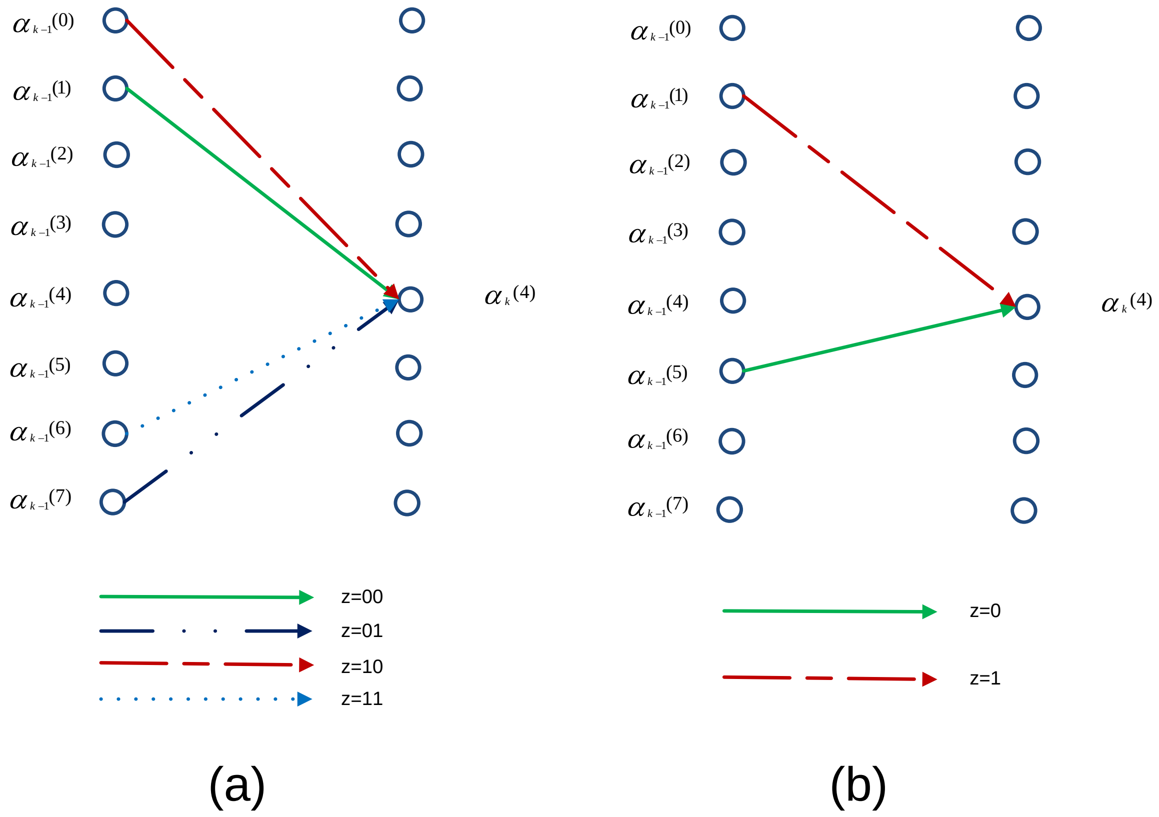


Figure 3.2 Calculation of one forward state metric, alpha, (a) for Duo-binary codes and (b) for single binary codes

on the maximum between two branches while it is the maximum between four branches in case of radix-4.

Figure 3.2 is an example of how the alpha calculation is performed. The value at time instance k for state 4 is calculated by taking the largest value from the alpha values in four states 0, 1, 6 and 7 at k-1 for duo-binary case while in the single binary case by taking the largest value from the alpha values in two states 1and 5. Each of the old alpha values are added with the corresponding gamma value from s to $s^{'}$. In duo-binary case, the upper arrow represents a transition caused by input $j = 10_2$ (the red line) so the gamma value includes APP information retrieved from the last iteration for a 10_2 -transition. The gamma value will also include a comparison of the soft bits with the output that the encoder would generate in a transition from state 0 with input $j = 10_2$.

3.3.3 Backward State Metric

The backward state vector , $\beta_k(s)$ (beta), represents the probability for the different states when considering all the data after the time instance k . The calculation of $\beta_k(s)$ is done in a similar manner as $\alpha_k(s)$ except that $\beta_k(s)$ is calculated backwards through the received soft input data as shown in Figure 3.3. The equation 3.10 states how the beta values are calculated.

$$\beta_k(s) = \max_{(s',j)}(\beta_{k-1}(s') + \gamma_{k,j}(y_k, s, s')) \tag{3.10}$$

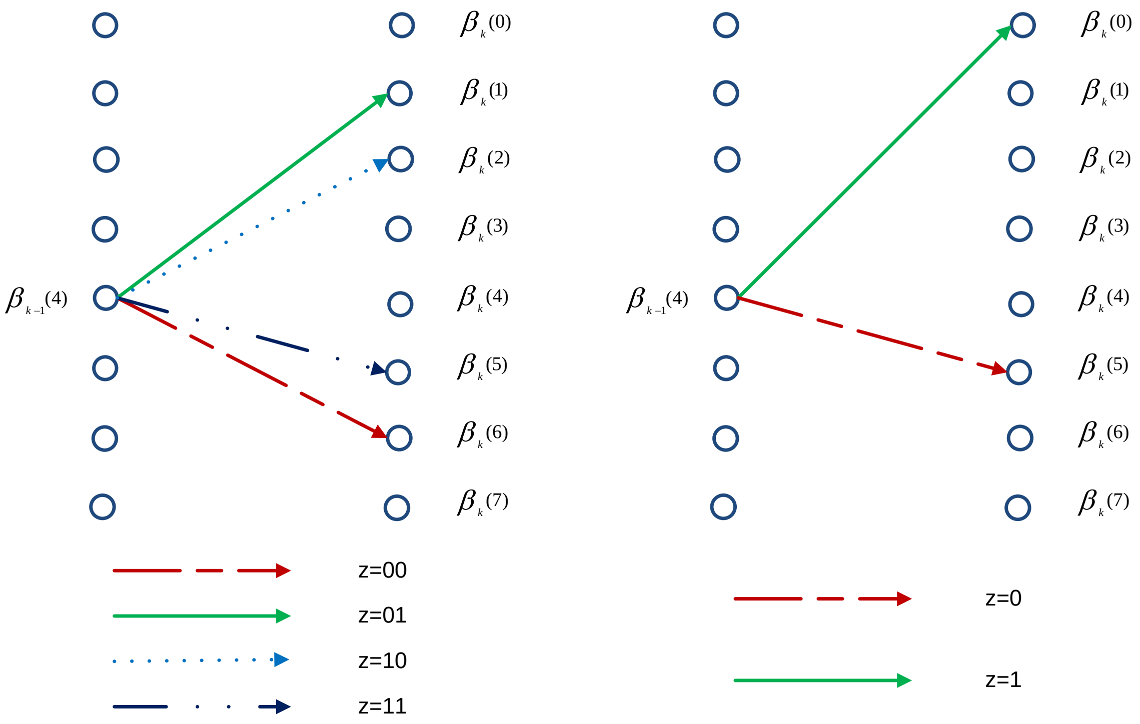


Figure 3.3 Calculation of one backward state metric, beta, (a) for Duo-binary codes and (b) for single binary codes

3.3.4 Extrinsic LLR

Based on the branch metrics, forward and backward metrics, the decoder calculates the branch LLRs as

$$T_k(j) = \max_{s \to s':(j)} (\alpha_k(s) + \beta_{k+1}(s') + \gamma_{k-1}(y,s,s')) \tag{3.11}$$

Where $T_k(j)$ represents Likelihood of the branch that corresponds to transition from state s to state s' for original input sequence (z). Figure 3.4 above represents all $j = 00_2$ transitions and calculation of the LLR for duo-binary scheme, the other combination of j are done in a similar manner. LLR is the maximum sum of an alpha value, a beta value and the transitions of the parity bits.

The MAP decoder estimates the probability of each information bit for binary codes. The extrinsic LLR of received bit is $\Lambda_{ext}^{(1)}$ and $\Lambda_{ext}^{(0)}$ for one and zero respectively. Also the probability of each information symbol, pair of bits, for duo binary codes. The probability of received symbol are $\Lambda_{ext}^{(00)}, \Lambda_{ext}^{(01)}, \Lambda_{ext}^{(10)}$ and $\Lambda_{ext}^{(11)}$ for 00, 01, 10 and 11 respectively.

A big advantage of max-log-MAP compared to log-MAP is that only the relative values of the metrics are interesting, not their actual values. The reduction of the number of exchanged estimated informations by each decoder is done by reference to $T_k(0)$ for single binary and $T_k(00)$ for duo-binary. So the number of exchanged informations is one for single binary and three for duo-binary as

$$L_k(j) = T_k(z) - T_k(0) \tag{3.12}$$

and we get that $L_k(0)$ always equals to zero. The extrinsic LLR $\Lambda_{ext,k}^{(j)}$ is calculated using the following equation:

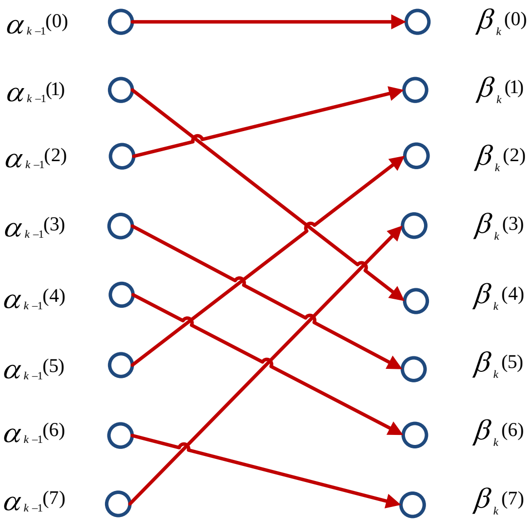


Figure 3.4 Calculation of LLR

$$\Lambda_{ext,k}^{(j)} = L_k(j) - \Lambda_{apr,k}^{(j)} - y_{k,j} \tag{3.13}$$

After calculation of branch LLRs, three extrinsic LLRs $\Lambda_{ext,k}^{(11)}$, $\Lambda_{ext,k}^{(10)}$, $\Lambda_{ext,k}^{(01)}$ should be bypassed to the other component decoder.

The final decision of decoded bits for duo-binary Turbo codes scheme is performed according to the sign of the output LLRs obtained from Equation 3.14.

$$\begin{aligned} L_k(A) &= \max(T_k(10), T_k(11)) - \max(T_k(01), T_k(00)) \\ L_k(B) &= \max(T_k(01), T_k(11)) - \max(T_k(10), T_k(00)) \end{aligned} \tag{3.14}$$

After Calculation of both $L_k(A)$, $L_k(B)$, we are able to estimate both original information bits $\hat{A}$, $\hat{B}$. This should be done at the last decoding iteration.

The final decision of decoded bits for single binary Turbo codes scheme is performed according to the sign of the output LLRs, $L_k(1)$, obtained from Equation 3.12.

3.4 Unified Radix-4 decoding algorithm

For single binary turbo codes, the trellis cycles can be reduced 50% by applying the one-level look-ahead recursion [29] [30] as illustrated in Fig. 3.5. Radix-4 α recursion is then given by:

$$\begin{aligned} \alpha_k(s_k) &= \max_{s_{k-1}}\{\max_{s_{k-2}}\{\alpha_{k-2}(s_{k-2}) + \gamma_{k-1}(s_{k-2}, s_{k-1})\} + \gamma_k(s_{k-1}, s_k)\} \\ &= \max_{s_{k-2}, s_{k-1}}\{\alpha_{k-2}(s_{k-2}) + \gamma_k(s_{k-2}, s_k)\} \end{aligned} \tag{3.15}$$

where $\gamma_k(s_{k-2}, s_k)$ is the new branch meteric for the two-bit symbol$\{A_{k-1}, A_k\}$ connecting state s_{k-2} and s_k: Similarly, Radix-4 β recursion is computed as:

$$\beta_k(s_k) = \max_{s_{k+2}, s_{k+1}}\{\beta_{k+2}(s_{k+2}) + \gamma_k(s_k, s_{k+2})\} \tag{3.16}$$

Since Radix-4 algorithm is based on the symbol level, and the calculation of the LLRs are done in a similar manner as in section 3.3.4 for duo-binary scheme.

Although the duo-binary Turbo codes is more complex than single binary, many designs are implemented based on the conversion from radix-2 to radix-4 to increase the hardware utilization and to form an unified architecture to support both classes of Turbo codes. Such conversion is suitable for low-throughput non-parallel architectures. In addition, such conversion has many drawbacks when targeting the parallel architecture as we will discuss this point in the next chapters.

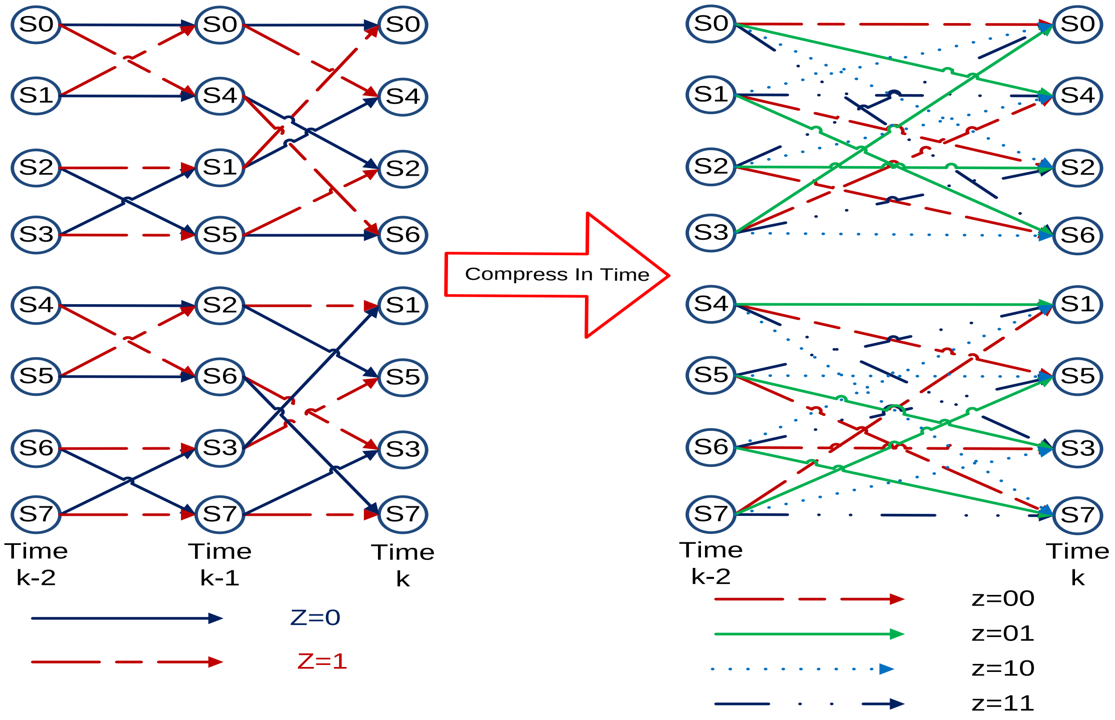


Figure 3.5 Conversion from radix-2 to radix-4 single binary turbo codes of LTE (HSPA+) trellis

3.5 Enhancement Max Log MAP

We can improve the decoder performance by multiplying the Extrinsic Log Likelihood Ratio (LLRs) by a scale factor [31] as shown in Figure 3.6.This method leads to reduce the performance gap against log MAP decoder.

3.6 Decoder Design Strategies

There are two used schemes for the decoding process design. The first scheme, *sequential scheme*, takes $2K$ cycles for the information block length K. The MAP decoder calculates the forward states and stores them to be used in the second K cycles. In the second K cycles, the MAP decoder calculates the backward states and generates one extrinsic LLR per clock cycle as shown in Figure 3.7. The second scheme, *butterfly scheme*, takes K cycles. In the first $K/2$ cycles, the MAP decoder calculates the forward states from the

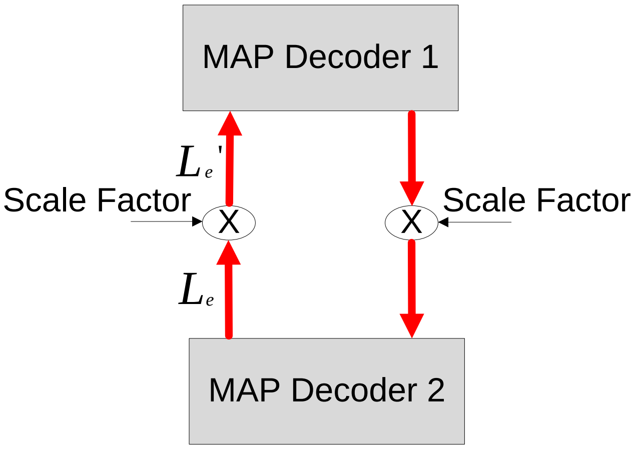


Figure 3.6 Extrinsic Log Likelihood Ratio (LLR) Scaling

first state up to the $K/2$ state and the backward states from the last state K back to the state $K/2$ simultaneously and stores them. In the second $K/2$ cycles, each clock cycle the MAP decoder calculates one forward state and one backward state and generates two extrinsic LLRs per clock cycle. So K LLRs are written to the memory within $K/2$ clock cycles.

The *sequential scheme* requires one state metric and one LLR calculation unit. It saves much area as it uses one state metric unit and reduces the number of LLRs which are written simultaneously to the memory. The conflict cycles in case of using parallel processing is reduced. Beginning the calculation with either the backward or with the forward state metrics is allowed. While the *butterfly scheme* uses two state metric and two LLR calculation units. The *butterfly scheme* generates two LLRs which increase the memory conflict and increase latency specially in case of parallel decoding [32].

3.7 Sliding Window Max-Log-MAP

The MAP decoder needs to wait for all the receiving block before starting the decoding process and requires a memory to save the states values. As the block size increases, the latency of the decoding process increases and the storage requirement is larger.

To avoid large storage requirement and achieve the latency constraints, the Sliding Window (SW) Max-Log-MAP approximation was proposed [33]. In SW Max-Log-MAP, the information block length K is divided into a number of windows, each window has the size W. This division reduces the storage to a constant value W where only one working window state metrics are stored instead of storing all state metrics. After the completion of reception of the first window, the forward states are calculated and are stored into the state memory to be used in the next calculations. Then, it is ready to calculate the backward states and extrinsic LLRs of symbols of the first window. The forward states of second window are calculated simultaneously as shown in Figure 3.8. The choice of the window size plays a role for decoding performance. Targeting large windows leads to performance close to the ideal case, Max-Log-MAP performance, at the expense of increasing in the storage requirements for the state memory and vice versa. Reasonable value for the window size should be adopted.

At the end of each sub-block, backward states are being calculated. A problem raises that no pre-estimation of values of state probabilities at the end of the window to initialize

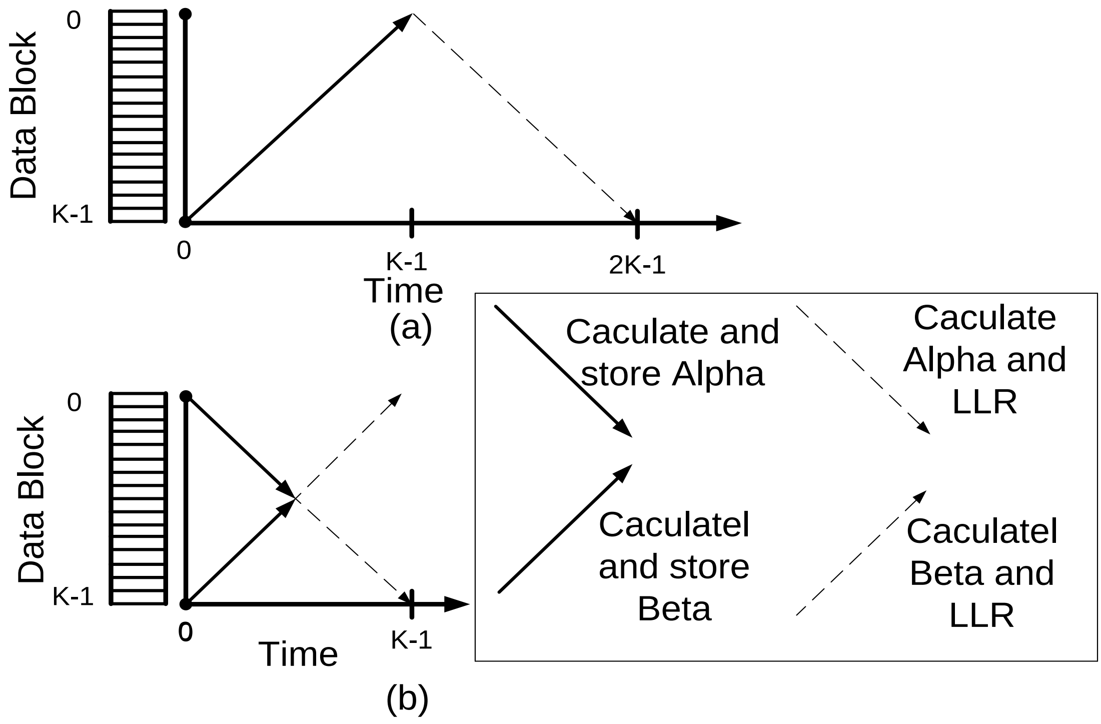


Figure 3.7 Two schemes for the decoding process: (a)sequential scheme, (b) butterfly scheme

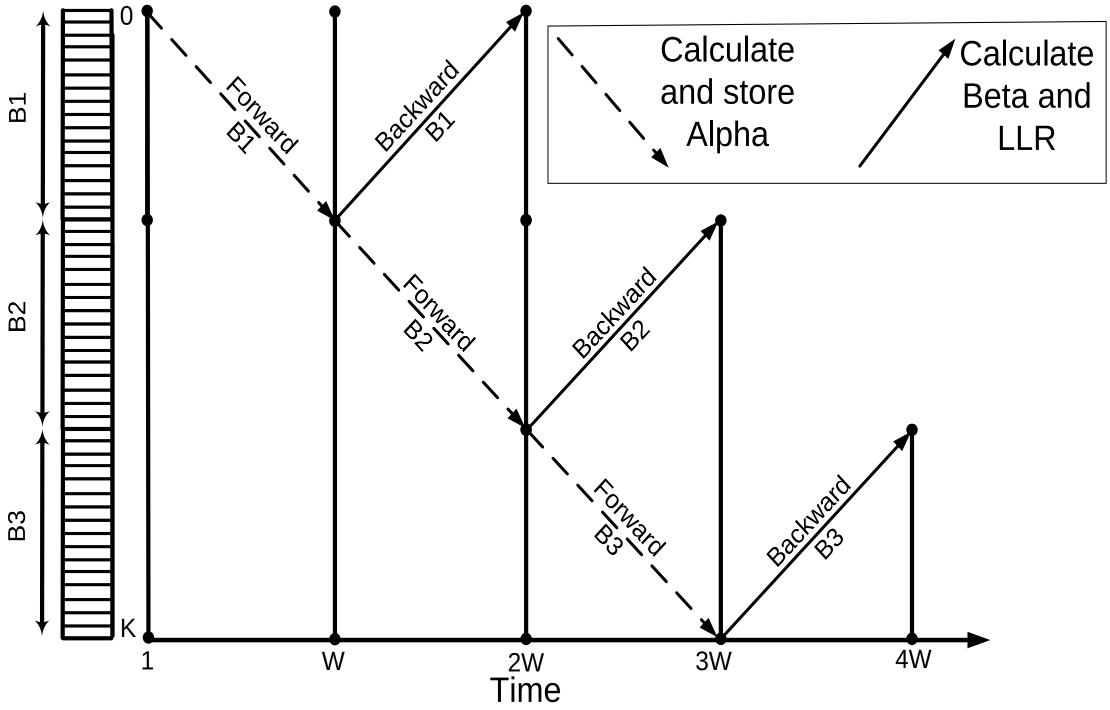


Figure 3.8 Timing sequences for Sliding Window Max-Log-MAP showing the operation of how states are computed for different sub-blocks with time

backward states. A possible solution is to assume equiprobable states at this time slot. This has its impact on degrading the system performance.

In order to overcome the effect of performance degradation, some proposed techniques use a guard window to have a rough estimation of initial value of backward state metrics. The guard window begins tracing back not from the end of the current window, but from a further time slot in the next window, this depends on the guard window size. As window size and guard window size increases, we have a better performance. The process of SW MAX Log MAP using a guard window is shown in Figure 3.9

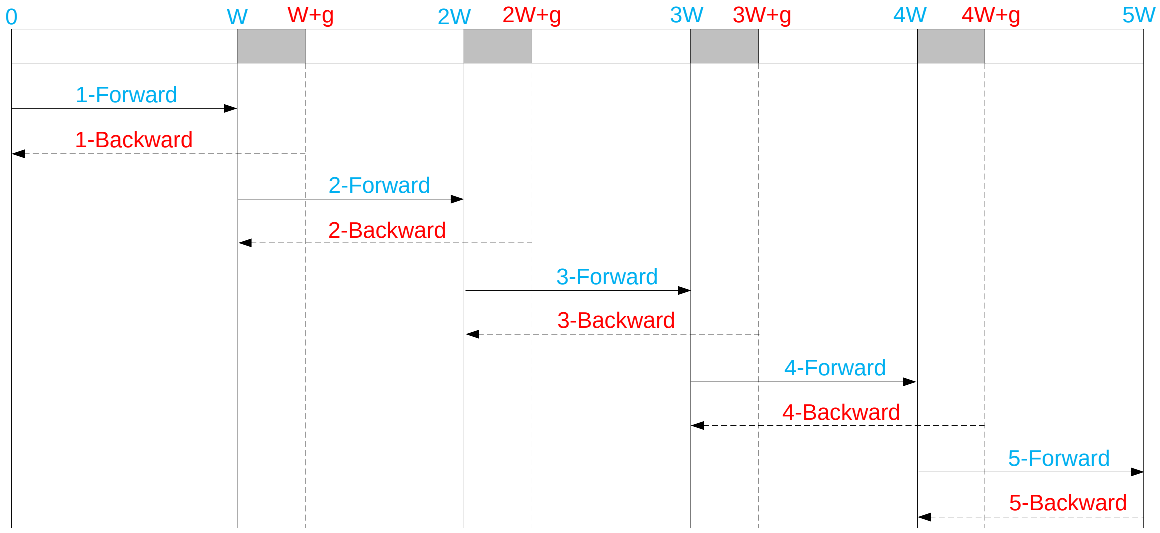


Figure 3.9 Sliding Window operation using a guard window technique

3.8 Parallel Sliding Window First Scheme

As mentioned earlier, parallel decoding is used to increase the throughput. The current wireless communication systems require high-throughput. The parallel decoding [8] [9] is used to increase the throughput. Parallel decoding can be employed by dividing the whole information block into p sub-blocks, each is processed independently by a dedicated SISO decoder [34]. Each sub-block is again divided into several windows of length W. Each window operation takes $2W$ cycles. In the first W cycles, each SISO decoder calculates backward states and stores them to be used in the second W cycles. In the second W cycles, the SISO decoders simultaneously calculate forward states and generate extrinsic LLRs. At the same time in which the forward calculations are executed, the backward calculations of the next window are processed. At the last W cycles, the forward states and extrinsic LLRs of the remaining windows are calculated. However there is an overlap between forward and backward states generations, only W LLRs are written to the memory within W clock cycles for each SISO. A timing sequence description of the Parallel SW Max-Log-MAP algorithm is provided in Figure 3.10.

Another problem is raised which is similar to SW Max-Log-MAP, due to handling the received block as independent windows, we need to have a pre-estimation for state probabilities at the end of the window to initialize the backward states and at the beginning of the window to initialize the forward states. The pre-estimation is required for both the forward and backward metrics which leads to large degradation in the decoding performance, counter to the SW Max-Log-MAP which requires a pre-estimation in one direction. The assumption of equal probabilities for all the states at these time slots leads to large degradation in the system performance.

Two different techniques were proposed in the literature instead of using the equal probability pre-estimation. In the first technique, the windows are overlapping by an interval called the guard window. The forward states are not calculated from the beginning of the window, they are calculated from an earlier time slot in the previous window located at a distance equivalent to the guard window. The same concept is used to calculate backward states, they are calculated from a further time slot in the next window. The guard window and window sizes play an important role to determine the system performance.

The second technique is called the next iteration initialization (NII). The NII does not need to perform dummy calculations for initializing the state metrics. In the NII, the values of the states border of the windows are stored into the memory to be used as estimation in the next iteration. However there is no information at the beginning of the decoding process, so the borders of the states of the windows are assumed equiprobable states at the first iteration for the two MAP decoders. The advantage of NII technique is that it does not waste clock cycles in the estimation of the states border, and also saves power dissipation during such dummy calculations. The NII has almost same performance as the previous technique. Two border memories, one for each MAP decoder, are required to store the states border. Small window sizes lead to an increase in the size of the border state memories and vice versa.

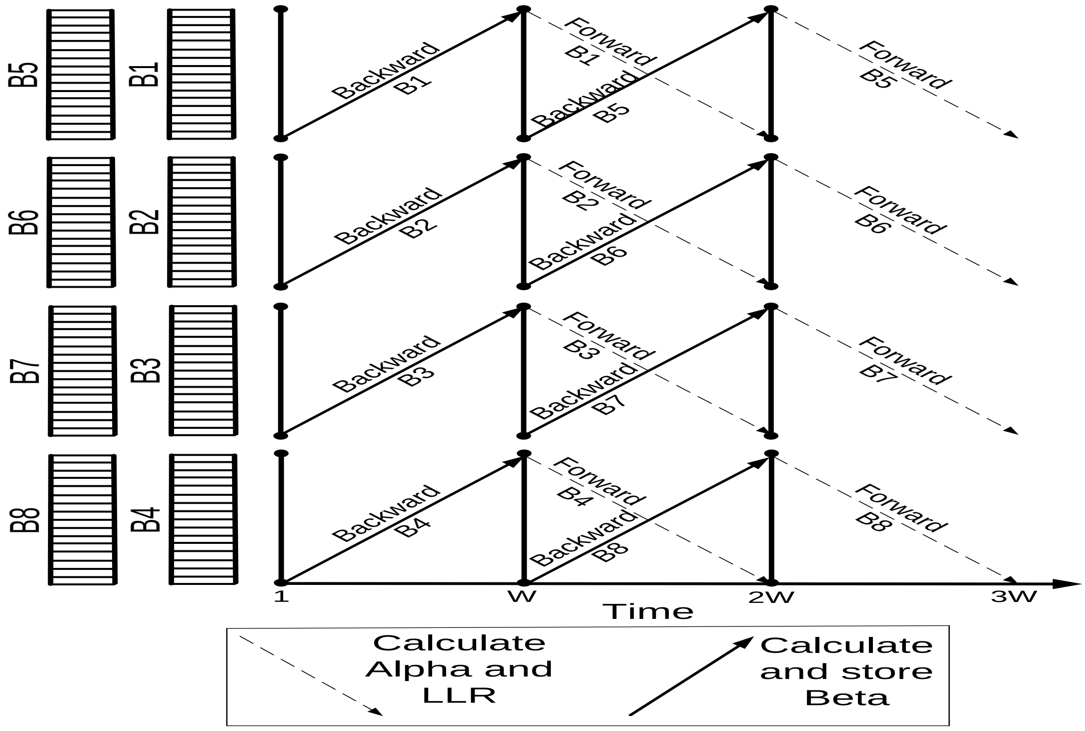


Figure 3.10 Timing sequences for Parallel SISOs

3.9 Parallel Sliding Window Second Scheme

The parallel decoding in second scheme can also be employed by dividing the whole information block into p sub-blocks similarly to the previous case. In this scheme, each window operation takes W cycles. In the first $W/2$ cycles, each SISO decoder calculates forward and backward states and stores them to be used in the second $W/2$ cycles. In the second $W/2$ cycles, each SISO decoder generates two extrinsic LLRs per clock cycle, which means that W LLRs are written to the memory within $W/2$ clock cycles as shown in Fig. 3.11.

This scheme takes less time than the previous parallel decoding. However large latency is added to the decoding time due to the increase of the number of LLR data which will be written to the memory simultaneously. Such large latency will degrade on the throughput significantly. To overcome such degradation, many methods are proposed to handle and to improve throughput than the original case. These methods will be discussed in the next chapters.

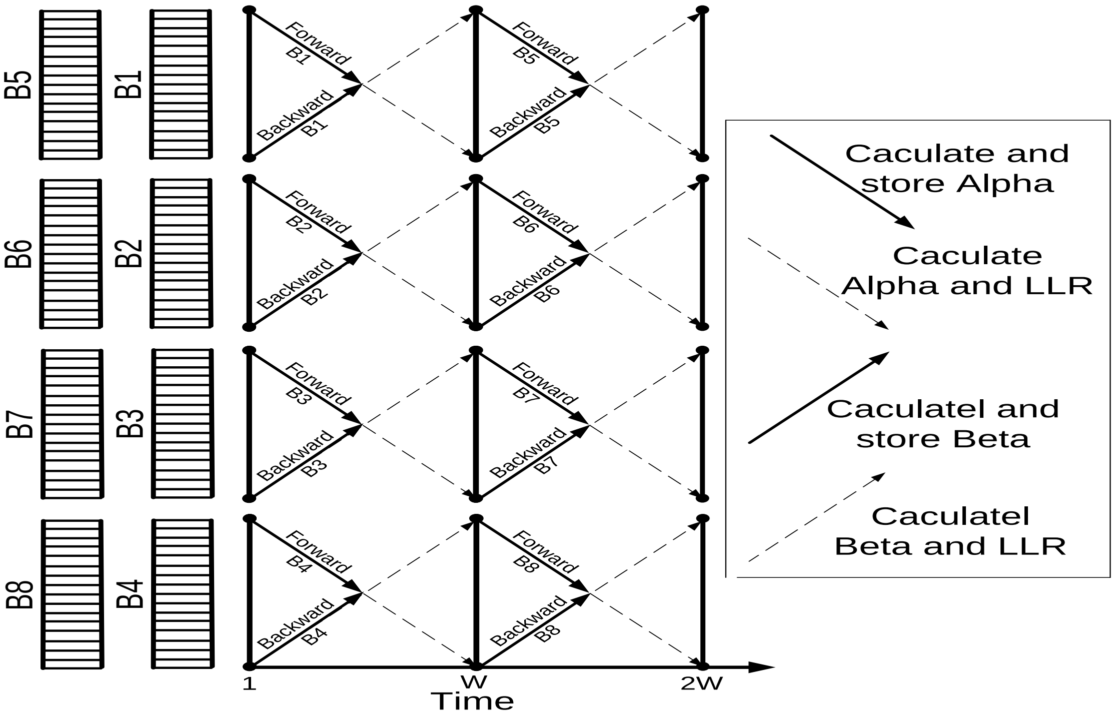


Figure 3.11 Timing sequences for Parallel SISOs Butterfly Scheme

3.10 Trellis Termination

Starting and ending from known states at the encoder results in better performance at the decoders. There are two trellis termination mechanisms used in the current standards. In the first mechanism, the encoder starts from the zero state and tail bitting is used to ensure that we end at the zero state. In the second mechanism, the encoder makes sure it starts and ends in the same state. This does not need tail bitting and does not affect the throughput of small block sizes.

Chapter 4

Simulations of WiMAX and 3GPP-LTE Turbo Codes

The simulations of double-binary convolutional Turbo coding which is used in WiMAX IEEE 802.16e standard and single binary convolutional Turbo Coding which is used in 3GPP-LTE [35] are presented, showing that influence of the Turbo interleaver block sizes, number of iterations, code rates, sliding window MAX Log MAP, quantization of the internal signals. In addition, applying the enhancement Max log MAP on the decoder to reduce the performance gap against log MAP decoder.

The simulations are done in AWGN and the fading channel model that proposed for IEEE802.16m standard for urban macrocell [36].

4.1 Enhancement MAX Log MAP

Figures 4.1 and 4.2 show that a 0.2 dB improvement is possible for a scaling factor of 0.75 compared to a scaling factor of 1 for both LTE and WiMAX with 4 iterations, OPSK scheme, rate 1/3, and in AWGN.

4.2 Effect of number of iterations

As illustrated earlier, Turbo decoding algorithms are based on iterative decoding. In this case, increasing the number of iterations provides an improvement in the original data estimation. Figures 4.3 and 4.4 illustrate the performance analysis of MAX Log MAP algorithm for LTE in AWGN and WiMAX in AWGN and the WiMAX channel model.

4.3 Effect of Turbo interleaver block sizes

Simulation results indicate that Turbo codes performance varies according to the interleaver block size. It is shown that the increase of CTC interleaver size enhances the BER performance for the same SNR. Figures 4.5 and 4.6 illustrate the performance of MAX Log MAP algorithm for WiMAX with interleaver block sizes of 24, 96, 192, 240, 480 and 960 couples respectively, and for LTE with interleaver block sizes of 40, 120, 200, 400, 480,640 and 1024 bits respectively. Simulation is performed for 4 turbo decoder iterations and coding rate of 1/3 in AWGN.

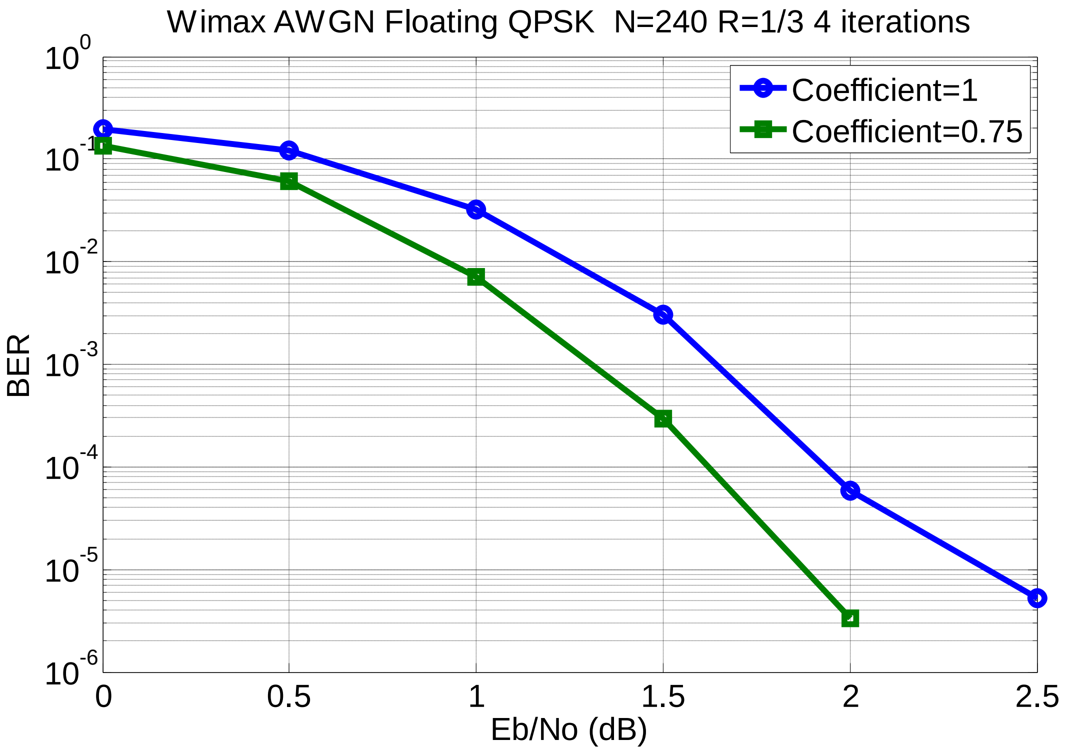


Figure 4.1 Enhancement MAX Log MAP on WiMAX with N=240 couples, 4 iter and R=1/3

4.4 Effect of Turbo Modulation Schemes

Figures 4.7, 4.8 show the performance of MAX Log MAP algorithm for WiMAX with QPSK and 16-QAM in AWGN and WiMAX channel model,and for LTE with QPSK, 16-QAM and 64-QAM in AWGN. Simulation is performed for 4 turbo decoder iterations and coding rate 1/3.

4.5 Effect of Symbol selection (Puncturing)

Symbol selection is performed to reduce number of coded bits per information symbol. Simulation results indicate that puncturing affects the BER performance of Turbo codes. In 802.16 CTC encoder, variable code rates of 1/2, 2/3, 3/4, and 5/6 are defined. It is shown that the increase in the code rate results in a degradation of Turbo codes performance. The process of puncturing should be adaptive according to the channel conditions. Figures 4.9, 4.10 show comparison between various Coding rates in AWGN and WiMAX channel model.Simulation is performed for 4 turbo decoder iterations and N=240 couples. The effect of puncturing in the single binary codes is shown in Figure 4.11.

4.6 Effect of the Rayleigh selective fading channel on LTE

The effect of the Rayleigh selective fading channel on LTE turbo codes is shown in Figure 4.12. The performance of the decoding iterations is improved when the channel is

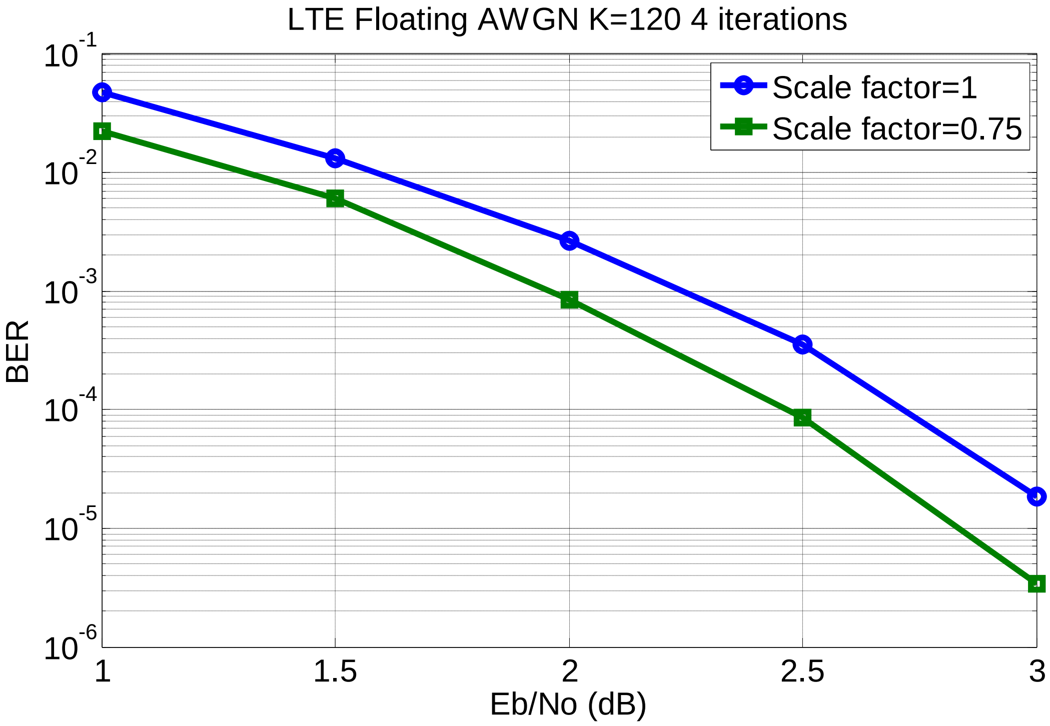


Figure 4.2 Enhancement MAX Log MAP on LTE with k=120 bits, 4 iter and R=1/3

selective and not correlated as the exchanged information from one MAP decoder to the other is informative.

If there is a way to measure the correlation of the channel, it may be used to change the number of iterations in order to reduce the power consumption. As the correlation of the channel is increased, the number of iteration is reduced which helps to reduce the power consumption which will not degrade the performance.

4.7 Sliding Window MAX Log Map approximations

In this section, the effect of Sliding window MAX Log MAP approximation is illustrated. The BER performance is tested for different window sizes (Ws) and guard window sizes (Wg). The simulation results are shown in Figures 4.13, 4.14. It is obvious that the system performance is exposed to some degradation with the change of the guard window size (Wg). The simulation results indicate that for the same window sizes (Ws) and guard window sizes (Wg) and increasing the block sizes lead to degradation in performance as shown in Figure 4.15.

4.7.1 Parallel Sliding Window Effects Using Guard Window and border states techniques

The guard window technique used to initialize the borders of the windows has degradation effects on the BER performance for parallel SW Max log MAP than the sequential SW Max log MAP. The parallel access of SW Max log MAP initializes the borders of

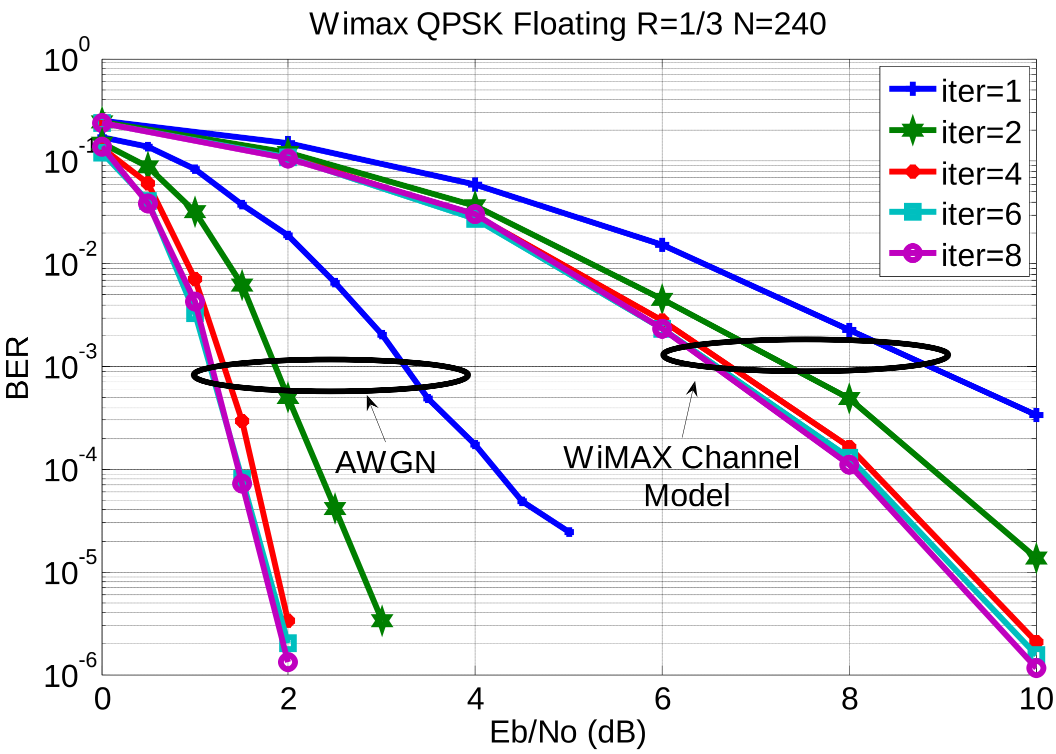


Figure 4.3 EEffect of iteration numbers on WiMAX with N=240 couples, 4 iter and R=1/3

both forward and backward states while the sequential SW Max log MAP initializes the backward states only.

For parallel access, the border memory technique has better performance than the guard window technique as shown in Figures 4.16 and 4.17

4.8 Fixed point analysis

In this section, fixed point simulation results are presented showing the optimal number of quantization bits for both input signals and internal signals. The notation $< int, q >$ is used to describe the fixed-point representation, where *int* represents the bit-width of integer part and *q* represents the bit-width of the fractional part.

In Figures 4.18(a), 4.18(b), and 4.18(c) quantizations of input signals are indicated, it is shown that 2 bits for integer part and 2 bits for fraction part have a good performance, it approaches the performance of the floating point model.

In Figures 4.19(a), 4.19(b), and 4.19(c) quantizations of Extrinsic LLRs signals are indicated, it is shown that 5 bits for integer part and 1 bit for fraction part have a good performance, it approaches the performance of the floating point model.
Simulation parameters are done with rate 1/3, AWGN channel, Block size N=240 couples, Window size (Ws)=32, guard window (Wg)=4 and with 4 iterations.

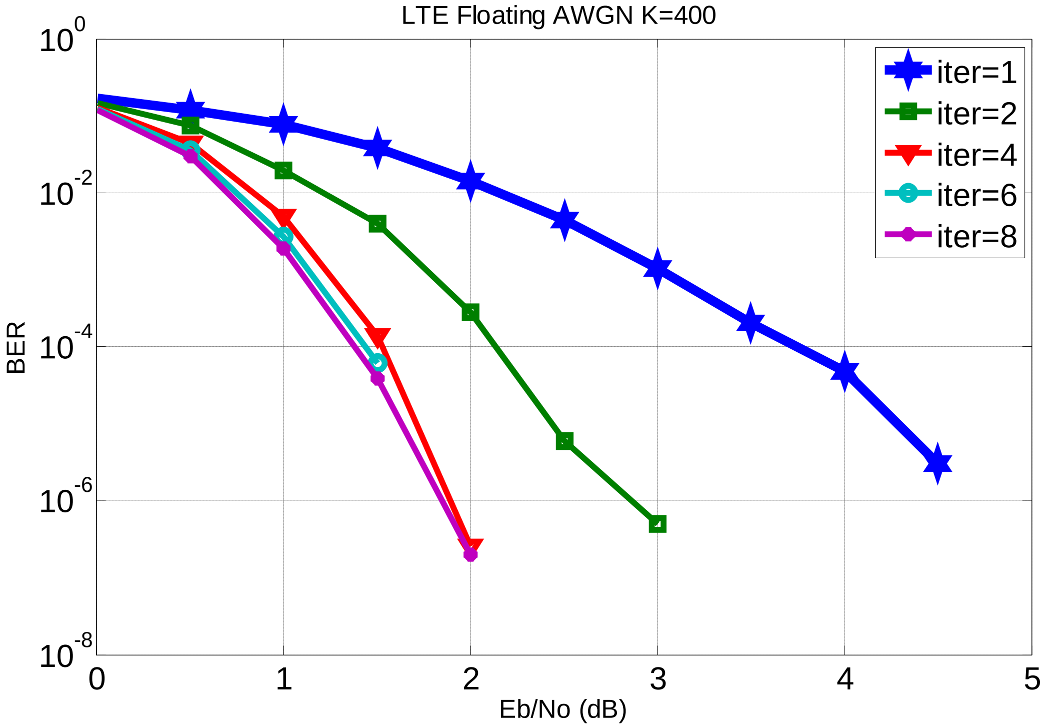


Figure 4.4 EEffect of iteration numbers on LTE with k=400 bits, 4 iter and R=1/3

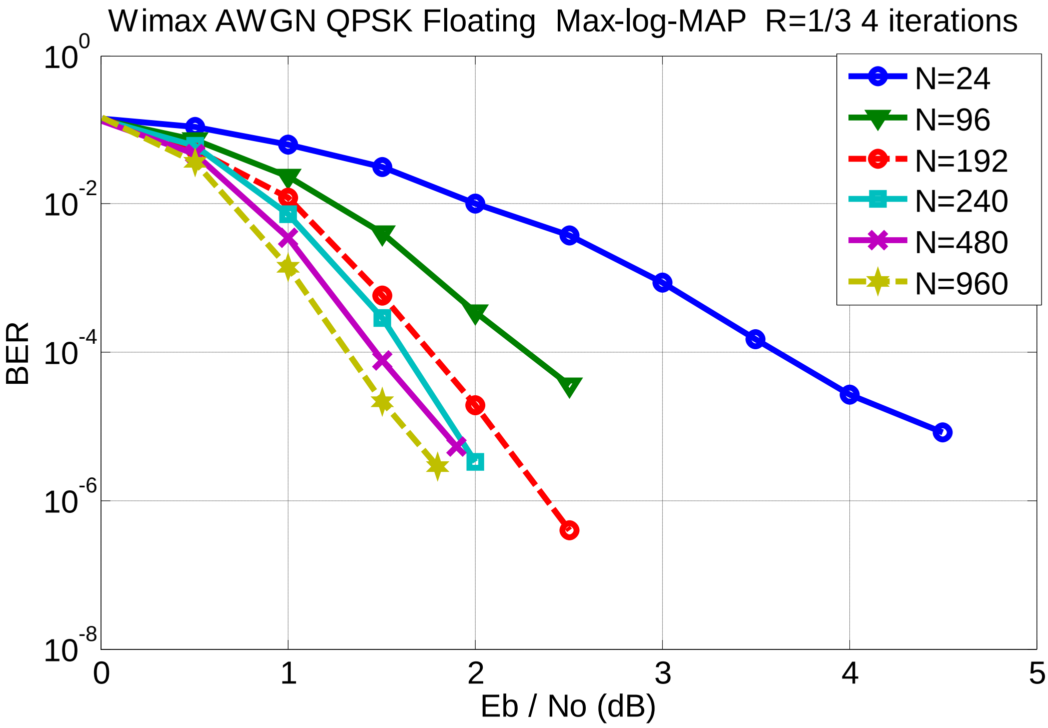


Figure 4.5 Interleaver block size effect on WiMAX with 4 iter and R=1/3

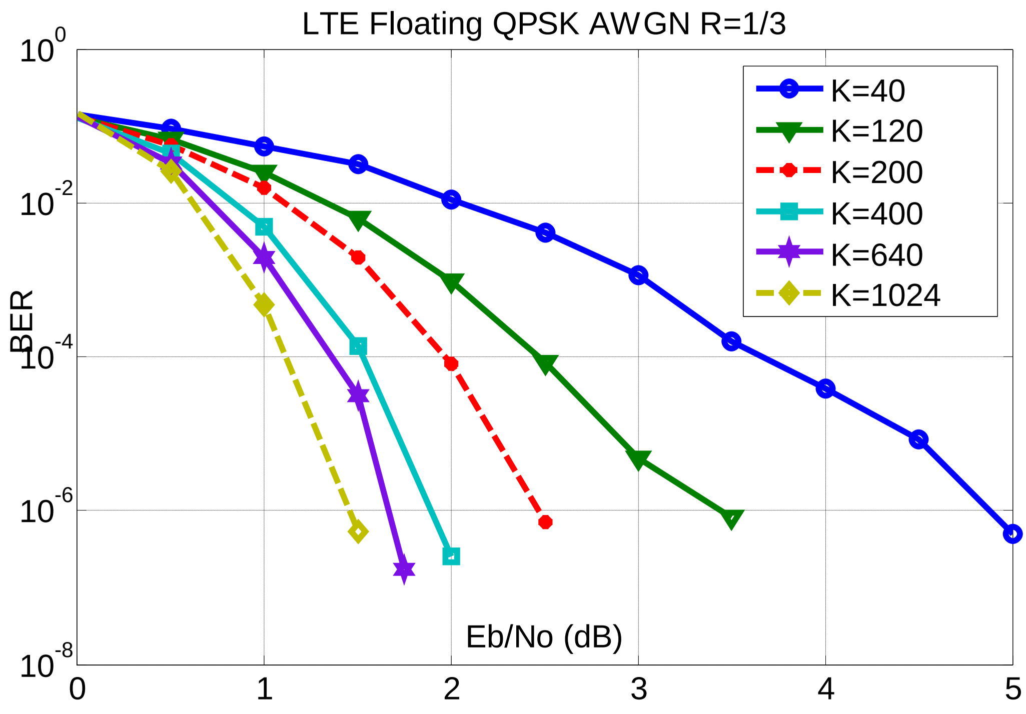


Figure 4.6 Interleaver block size effect on LTE with 4 iter and R=1/3

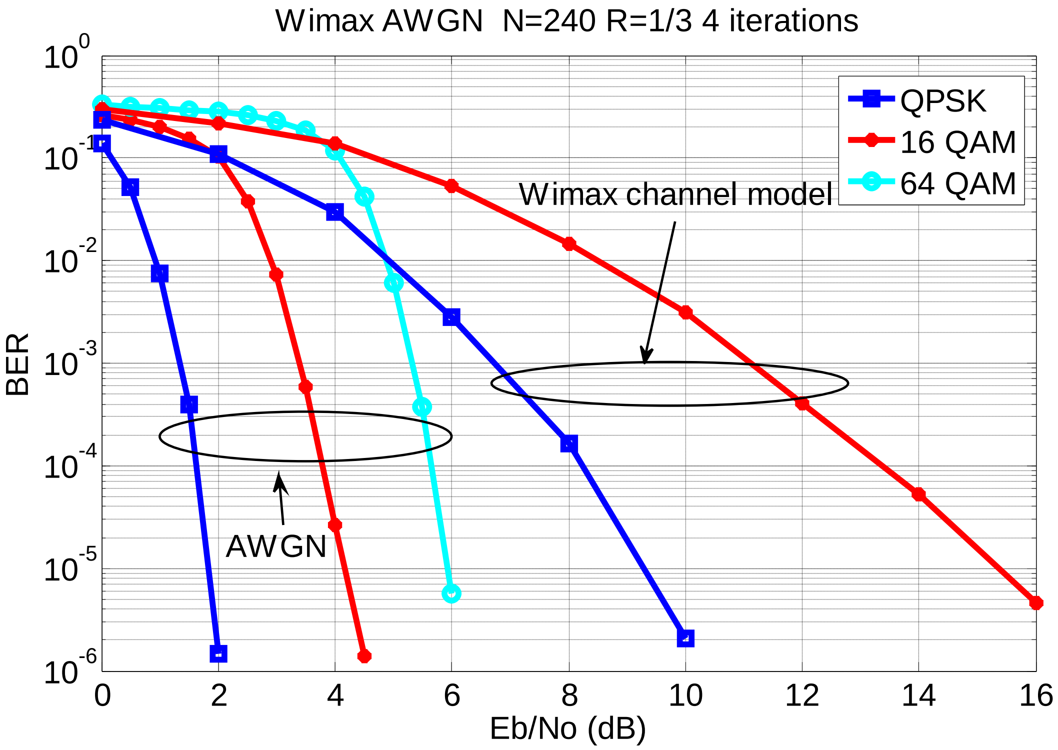


Figure 4.7 Effect of Turbo Modulation Schemes on WiMAX with N=240 couples, 4 iter and R=1/3

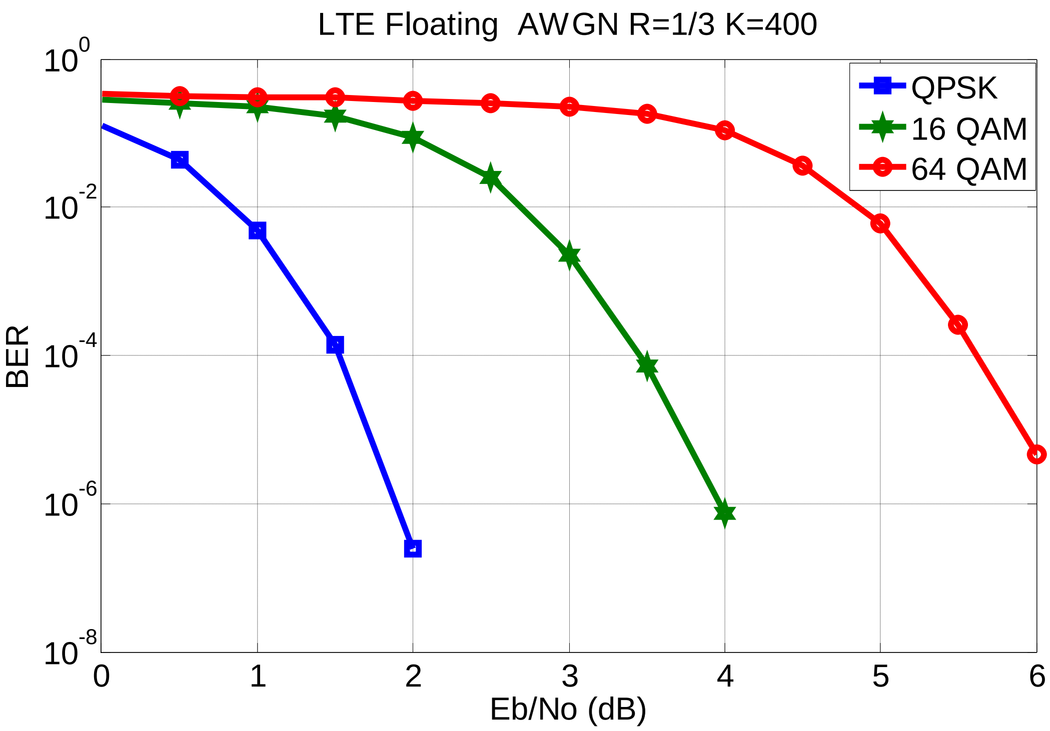


Figure 4.8 Effect of Turbo Modulation Schemes on LTE with K=400 bits, 4 iter and R=1/3

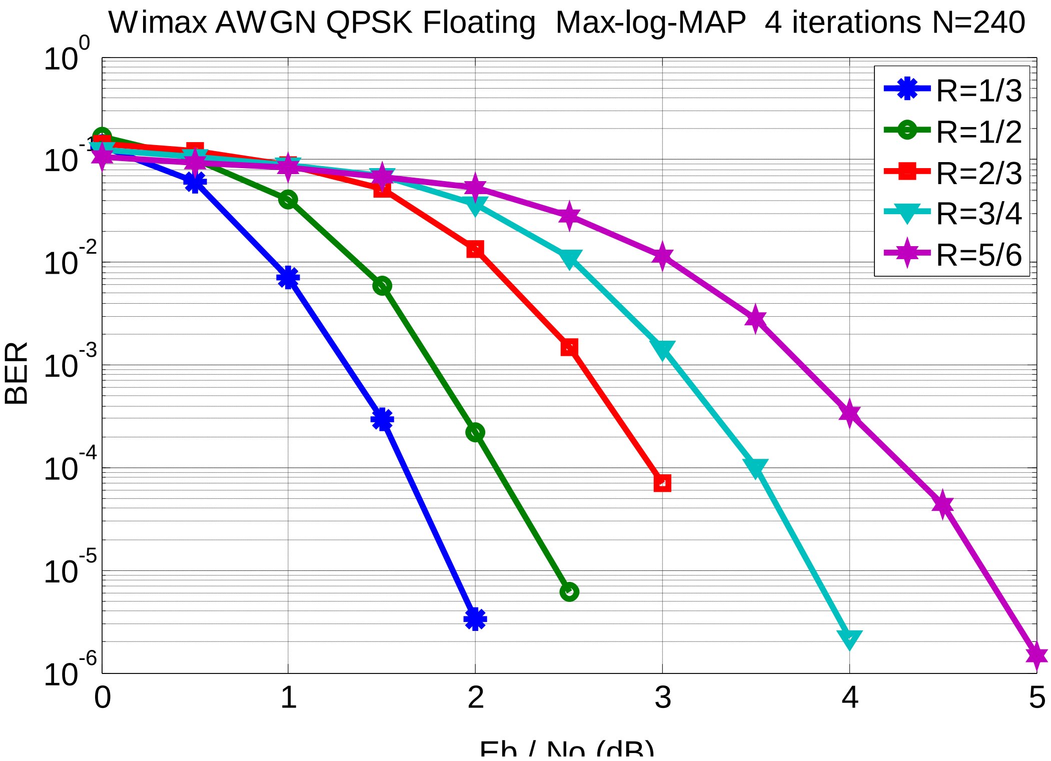


Figure 4.9 Symbol selection for WiMAX on AWGN with N=240 couples and 4 iter

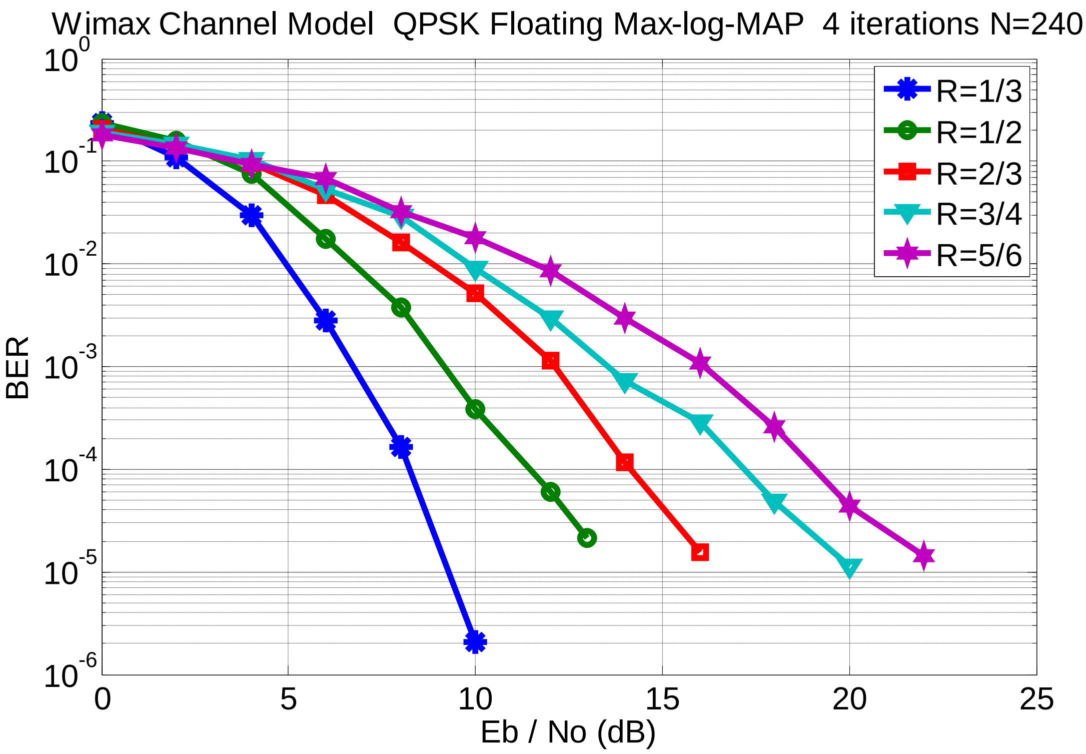


Figure 4.10 Symbol selection for WiMAX on WiMAX channel model with N=240 couples and 4 iter

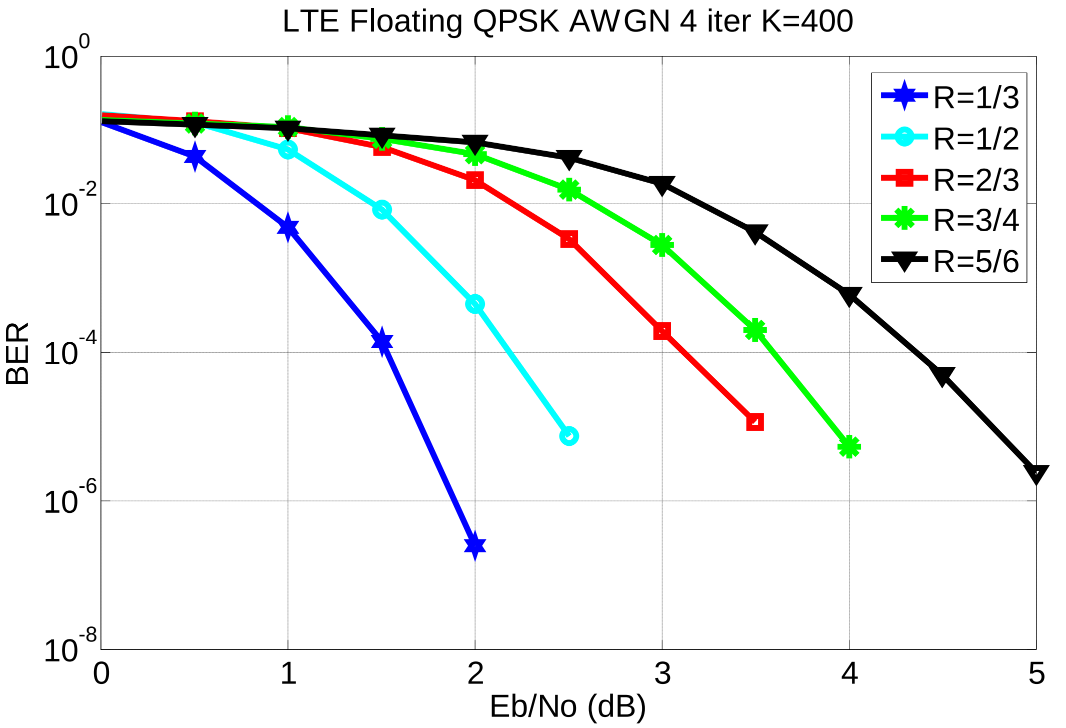


Figure 4.11 Effect of Symbol selection for LTE on AWGN channel with K=400 and 4 iter

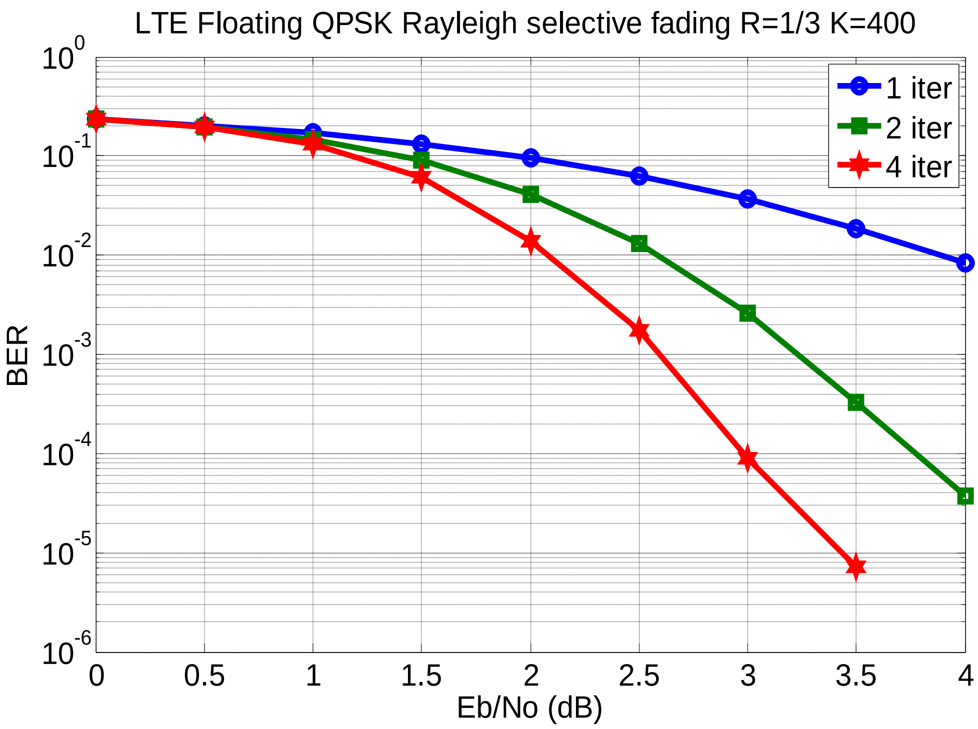


Figure 4.12 Effect of number of iterations for LTE standard in rayleigh selective fading channel with 4 iter

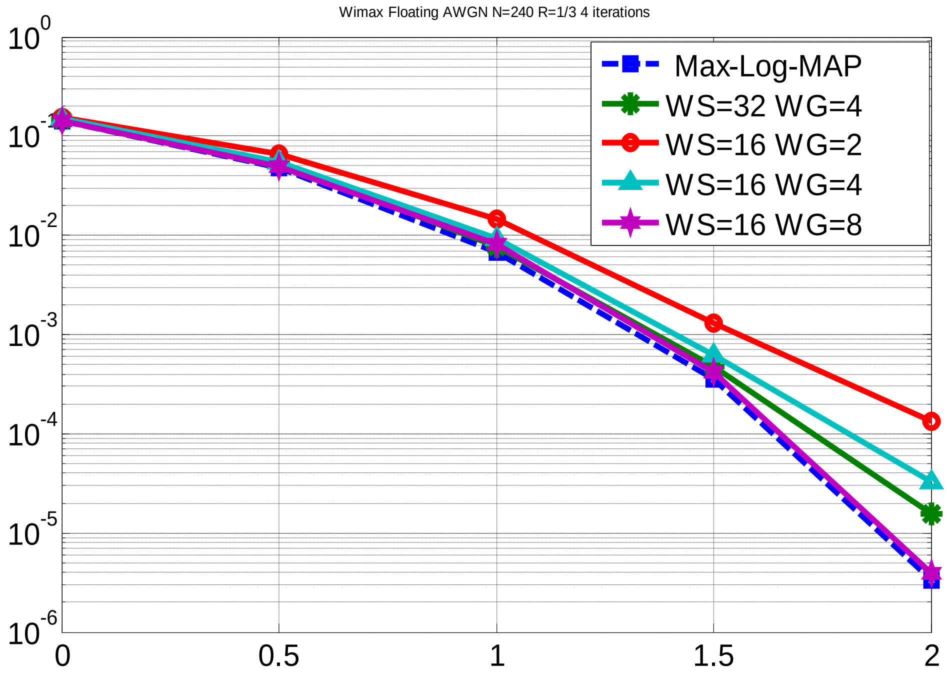


Figure 4.13 Sliding Window MAX Log Map on AWGN N=240 couples

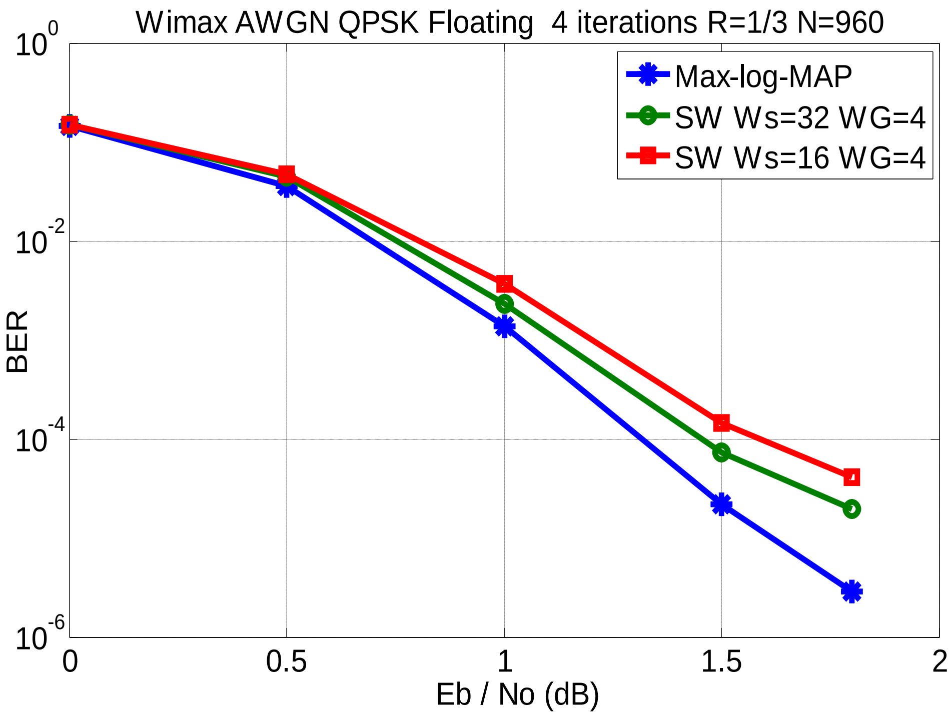


Figure 4.14 Sliding Window MAX Log Map on AWGN N=960 couples

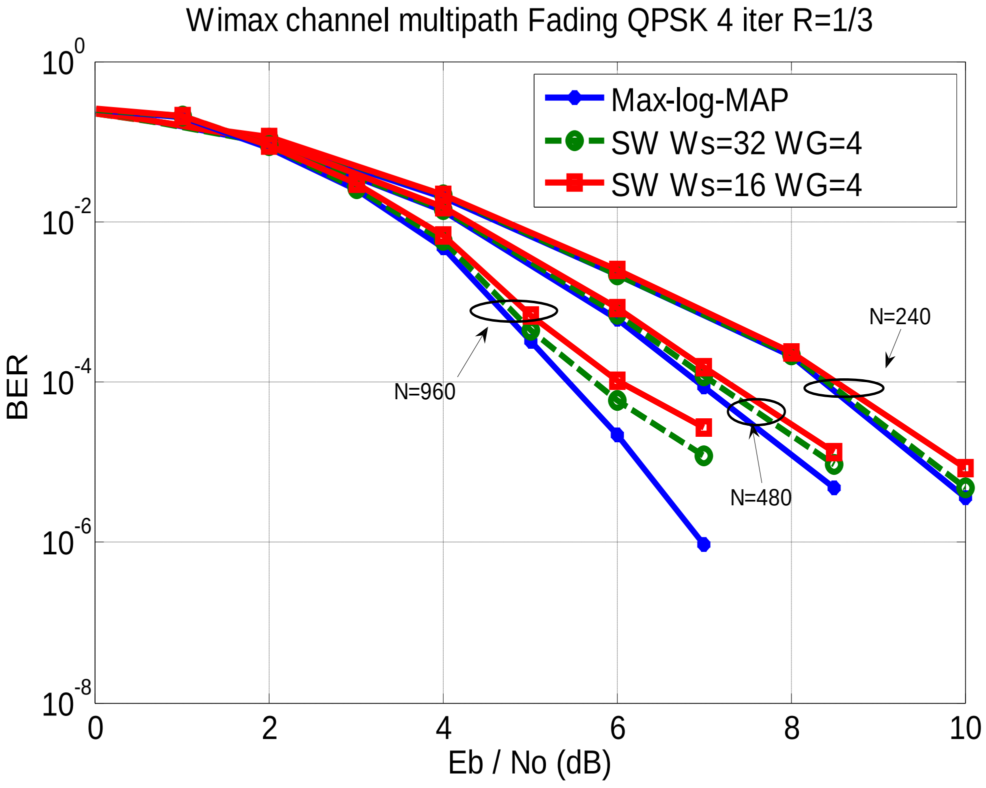


Figure 4.15 Sliding Window MAX Log Map on WiMAX channel model

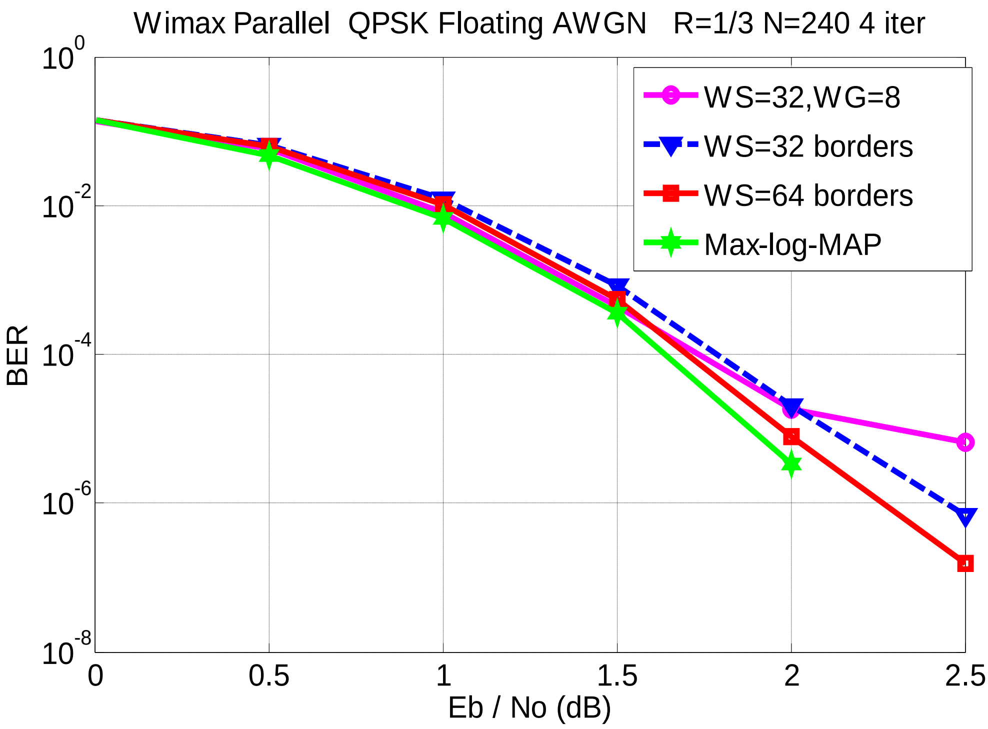


Figure 4.16 Performance between border memory and guard window technique for N=240

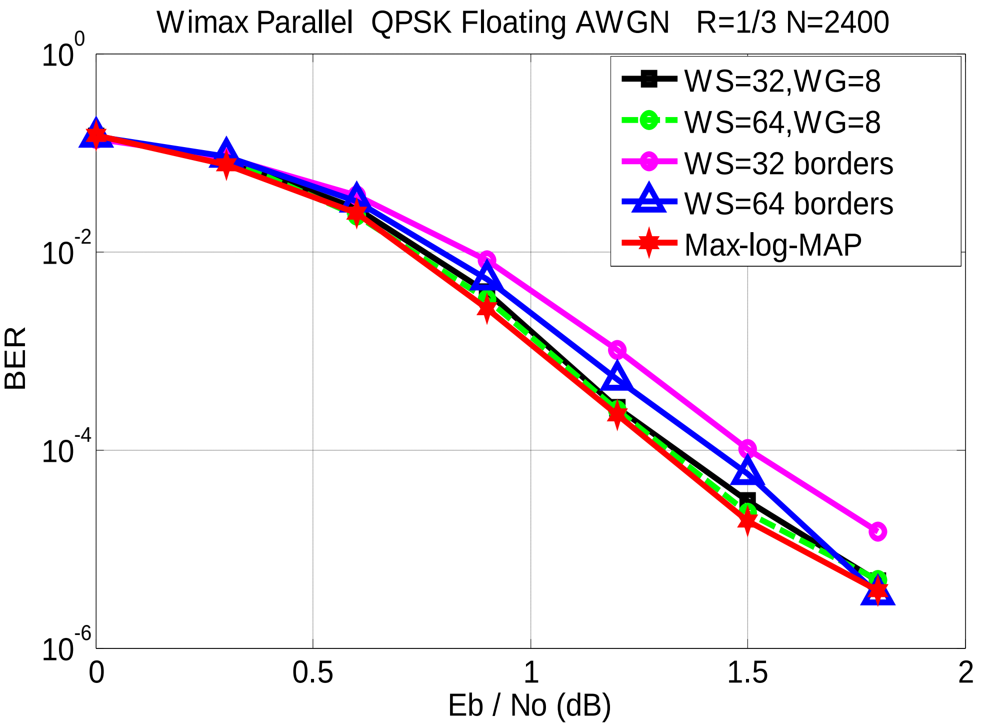
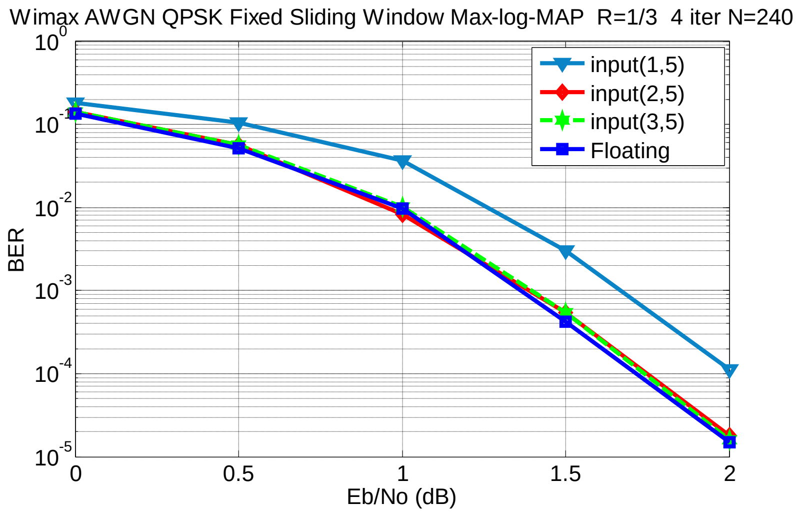
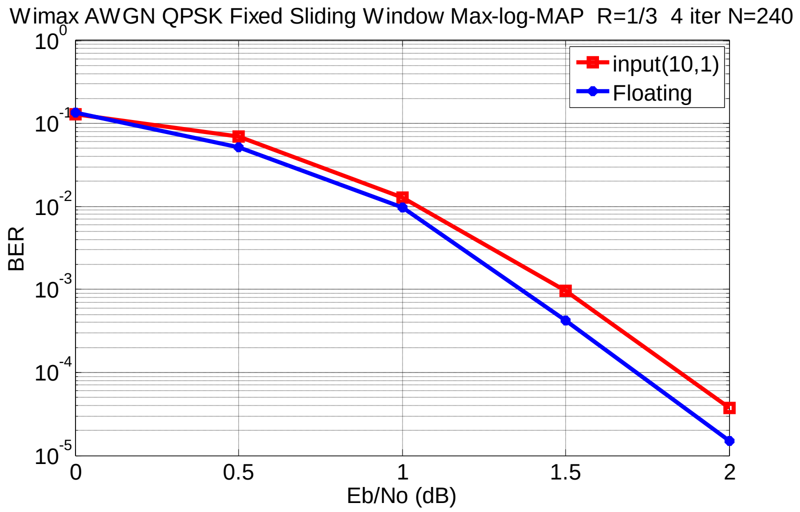


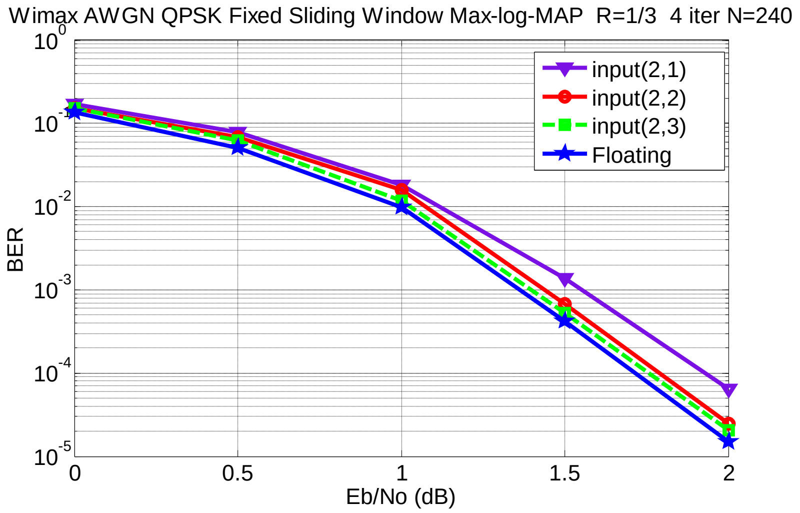
Figure 4.17 Performance between border memory and guard window technique for N=2400



(a) Input integer part quintizations

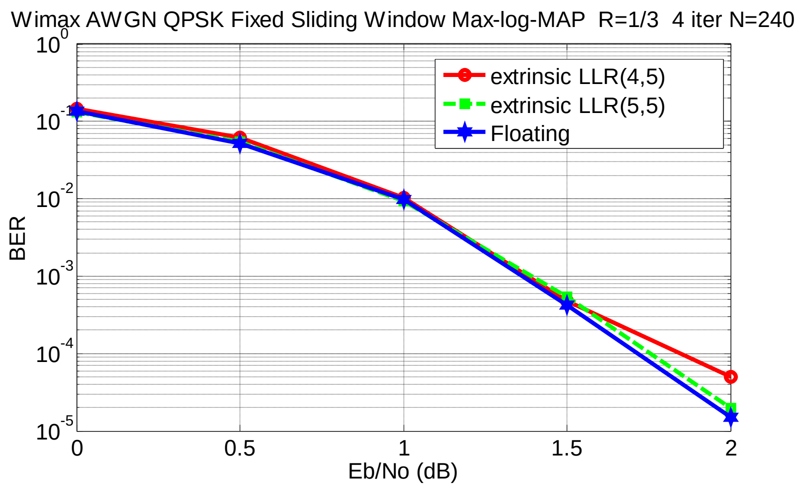


(b) Input fraction part quintizations

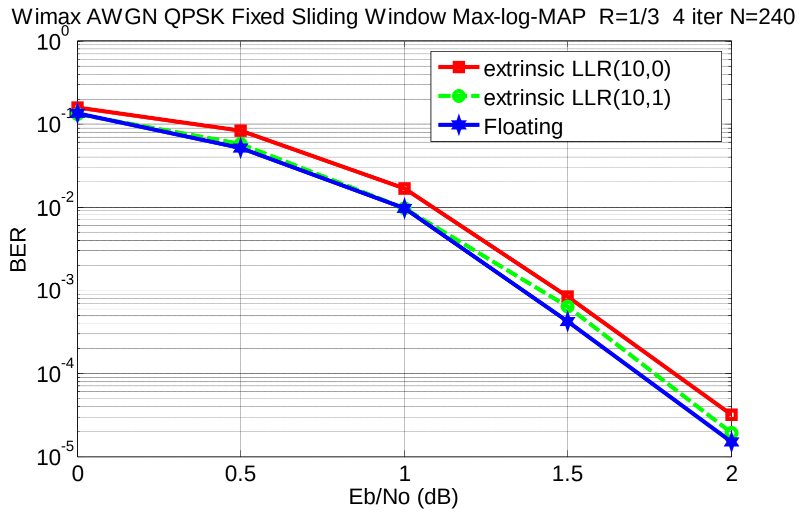


(c) both input integer and fraction parts quintizations

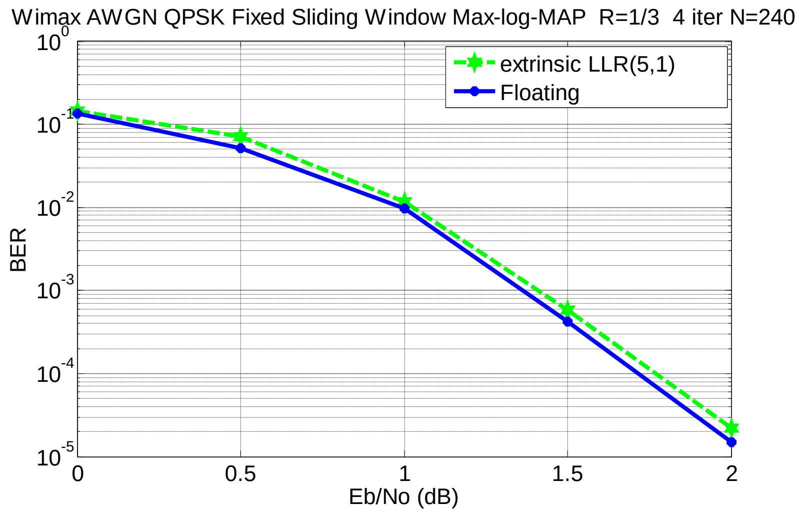
Figure 4.18 Input quintizations (a)integer part only (b)fraction part only (c)both integer and fraction parts



(a) Extrinsic LLRs integer part quintizations



(b) Extrinsic LLRs fraction part quintizations



(c) Extrinsic LLRs integer and fraction parts quintizations

Figure 4.19 Extrinsic LLRs quintizations (a)integer part only (b)fraction part only (c)both integer and fraction parts

Chapter 5

Memory Conflict Analysis

The parallel sliding window decoding leads to contention on memory due to parallel access, which causes latency and reduces the throughput. The conversion from radix-2 single binary turbo codes to radix-4 single binary turbo codes adds more conflicts as the time to write the whole data block is reduced by half.

5.1 Maximum Contention Free Interleavers

There are two types of interleavers, unconstrained interleavers and constrained interleavers. The constrained interleavers are maximum contention free (MCF) [12] which mean no conflicts happen due to parallel accesses as shown in Figure 5.1. But the MCF interleavers require that $K = M * P * W$ where K is the block length, W is the window size, P is the number of parallel windows and M is an integer as shown in Figure 5.2. Hence, the Window size W and the number of parallel windows P must be variables based on the block length K.

There are few standards that include contention-free interleavers such as WiMAX and 3GPP-LTE. Conversely, in parallel radix-4 scheme, 3GPP-LTE and WiMAX are not MCF, which means that they will face conflicts during parallel access. However, by using the even-odd memory scheme those conflicts can be avoided.

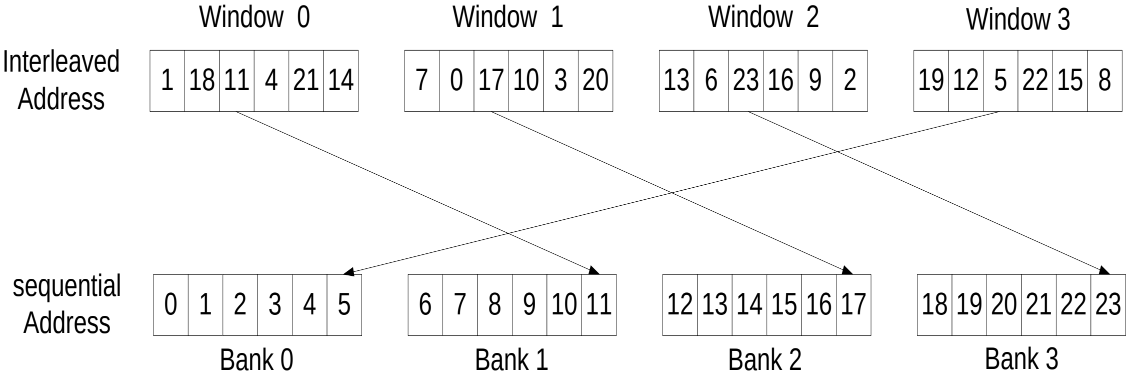


Figure 5.1 An example for no collisions for WiMAX interleaver when M=1

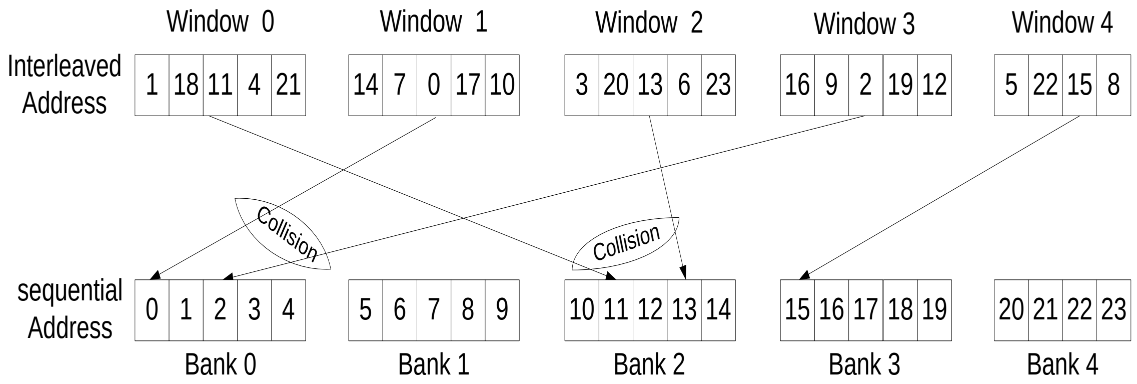


Figure 5.2 An example for collisions happen for WiMAX interleaver when M=0.96

5.2 Effect of Window Size on Memory Contentions

Buffering of the conflicting data until processed by the targeted memory, was proposed to avoid memory contentions. Computer simulations are done on Matlab to determine the buffer sizes for different interleaver patterns that are used in different standards.

Selecting the window size is an important step for reducing the size of the buffer structure as shown in Figure 5.3 and 5.4. However this leads to complexity in calculating the offset and bank number for memory addressing which require dividers. The area of the dividers is very large, and long time is needed for calculations. To avoid those drawbacks, the window size should be power of 2.

The second decoder cannot start its decoding process until the first decoder finishes writing the all values to memory. This latency is varying from standard to standard and depends on block sizes as shown in Figure 5.5. This feature makes the scheme not suitable for constrained real time systems.

5.3 The Second Scheme of Parallel Decoding Analysis

5.3.1 Decreasing the Number of Conflicts

According to the timing sequence of the parallel decoding algorithm 3.9, where each SISO decoder generates two extrinsic LLRs per clock cycle. As shown in Figure 5.6, delay buffers can be used to store one of the two LLRs which are generated by each SISO to reduce the memory conflicts and area. However, additional latency is produced because the decoder cannot start its process until the other decoder finishes all its writing operations. Although delay buffers are added the total buffer size is reduced since the total line buffers are reduced which have bigger width than the added delay buffers. Many comparisons have been done to select the efficient hardware and system requirements.

In this analysis, we assume that the memory runs at double the clock frequency of the system. The double speed of the memory clock has many benefits:

- Reduces the number of conflicts which decreases the number of buffers required to store the conflict data.

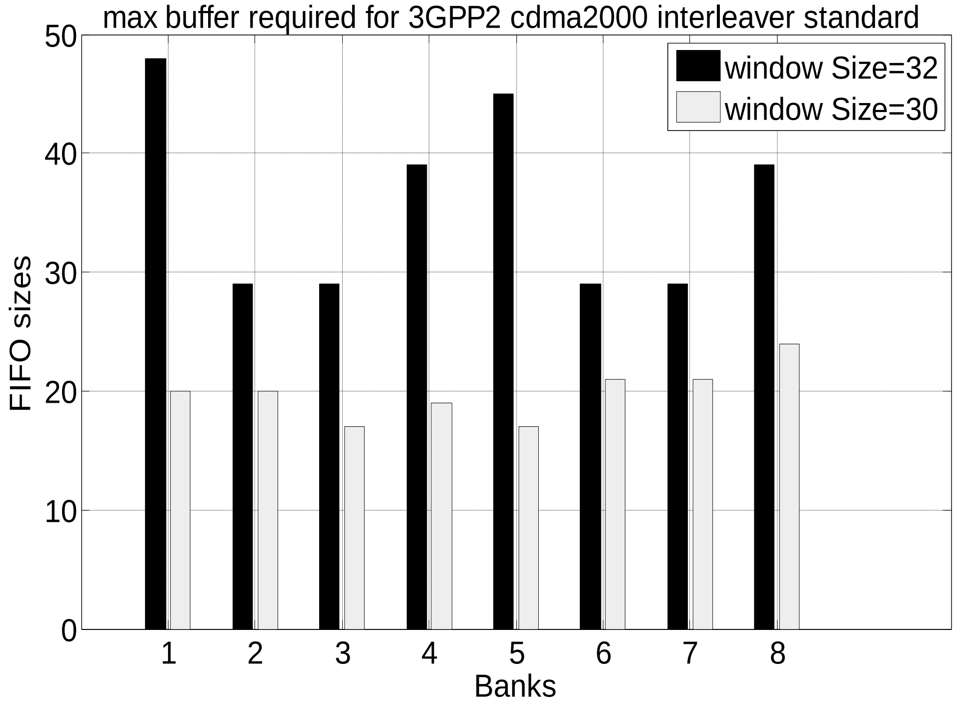


Figure 5.3 Maximum buffer sizes for 3GPP2 CDMA2K interleaver for different window sizes with 8 parallel windows

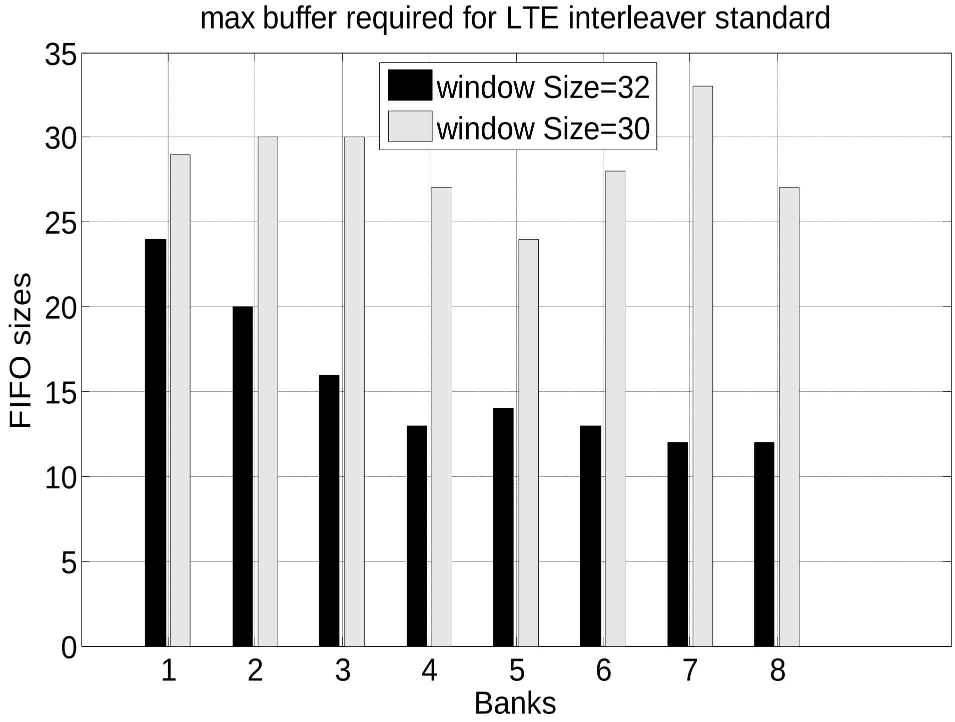


Figure 5.4 Maximum buffer sizes for LTE interleaver for different window sizes with 8 parallel windows

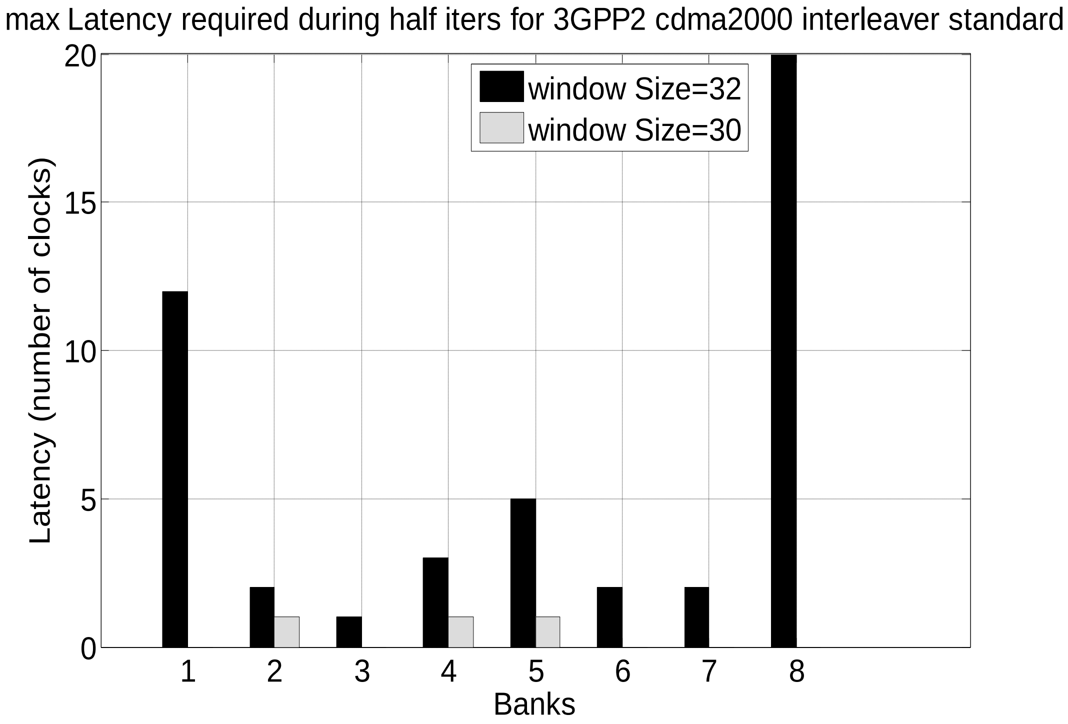


Figure 5.5 Maximum latency for 3GPP2 CDMA2K interleaver for different window sizes with 8 parallel windows

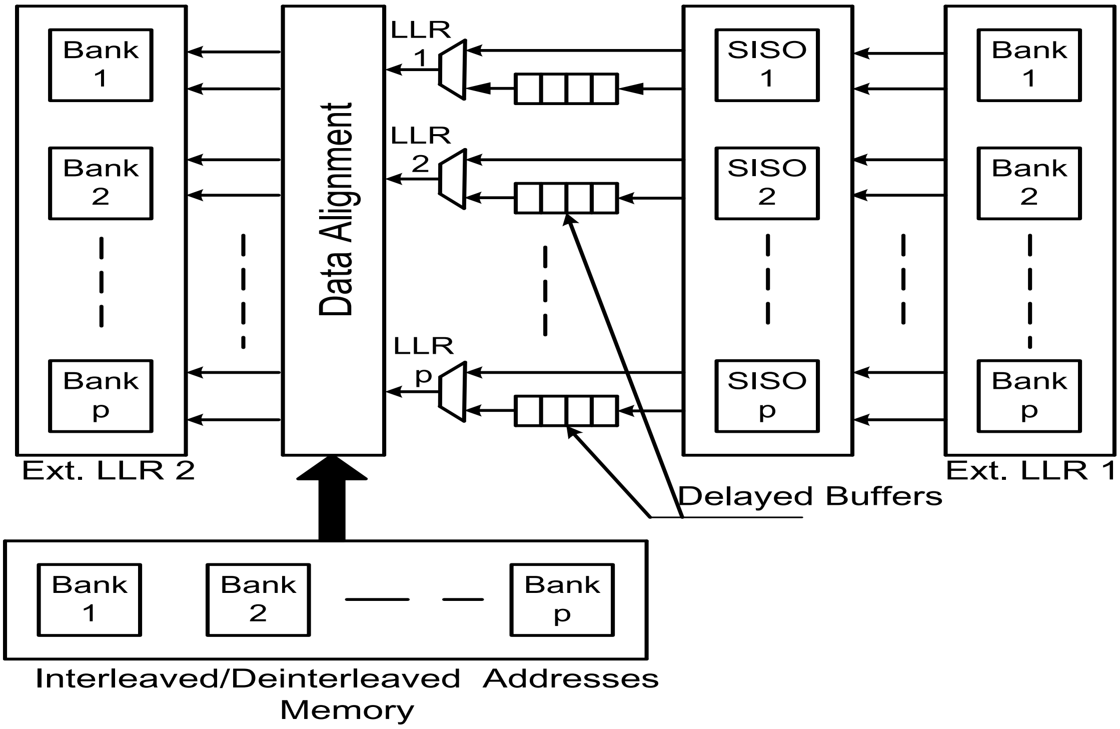


Figure 5.6 Parallel architecture with adding delayed buffers to reduce the number of concurrent values need to write by half at every clock

- Avoids variable latency. Variable latency is unsuitable for real time communications systems, through fast writing of the stored data.

- Reduces latency in case of adding delay buffer when two decoders exchange the information between them.

The use of two clocks in the design, one for the system and another for the memory, adds to the design complexity.

5.3.2 Handling of the Conflicts

As the LLR values arrive to the memory bank simultaneously, the data alignment block collects these values to prevent contentions by storing them into a buffer structure. Each LLR writes into a separate 2-dimensional array entering the intended row corresponding to the target bank as shown in Figure 5.7.

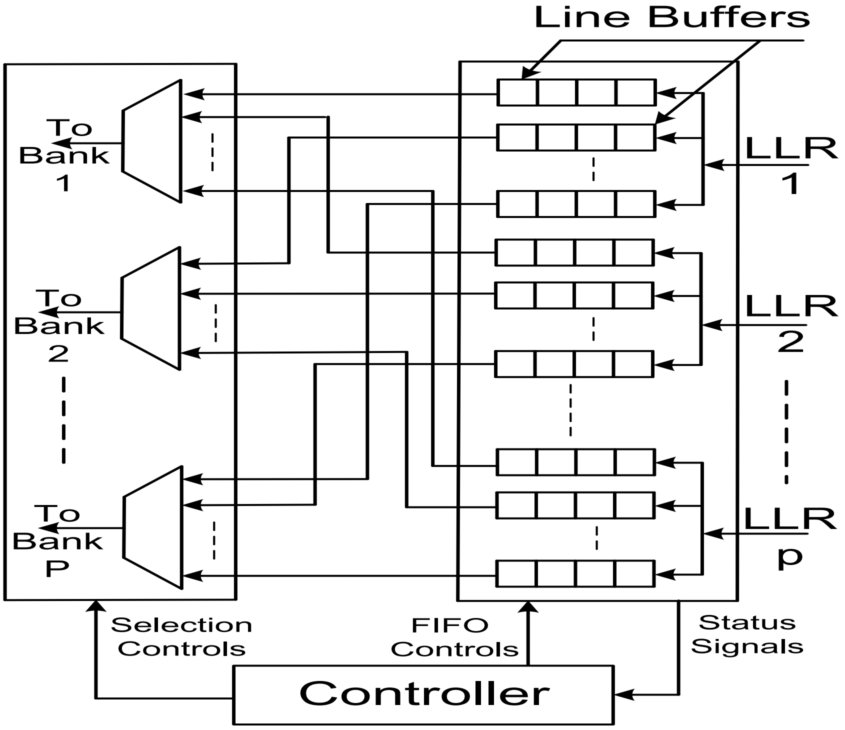


Figure 5.7 The data alignment block receives LLR values and stores them until processed by the targeted memory bank

There are two levels of buffers. The first level is the delayed buffer explained earlier. The second level is the line buffer located in the data alignment block. The line buffers store LLRs and the interleaved/deinterleaved addresses, while the delayed buffers store LLRs only. The sizes of the line buffers are determined by simulating the different standards and selecting the biggest sizes to resolve the collisions. The reduction in area due to the minimization of line buffers overcomes the increase in area due to the addition of delayed buffers. Hence, the total buffer size is reduced. Many comparisons have been done to select the efficient hardware and system requirements.

The controller selects an LLR from the stored values to be written to the target memory bank. To reduce the complexity, the control unit is divided into a number of parallel units, *p*. Each unit consists of two selectors and two row enablers connected as shown in Figure 5.8. The “selector from low” takes the request, status signal, number 1 as the highest priority, and request number n, representing the number of concurrent LLRs, as the lowest priority. The “selector from high” takes the request number n as the highest priority, and request number 1 as the lowest priority. The “row enabler 1” is active when one or more requests are asserted. The “row enabler 2” is active when two or more requests are asserted.

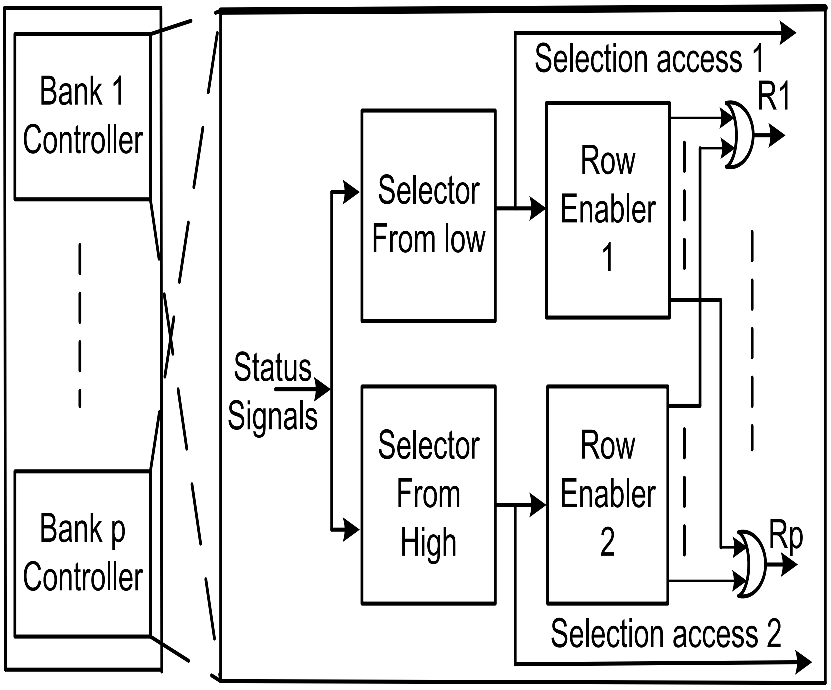


Figure 5.8 The controller of the data alignment block with divided into p small controllers

In this design, the used memories are dual port memories, which allow two concurrent memory accesses per clock cycle. Many requests may arrive simultaneously to access one of the memory banks. To determine which one or two requests will be served, a simple mechanism is applied separately for each memory bank. The upper selector scans from low to high, and selects the minimum active request index. The lower selector scans from high to low, and selects the maximum active request index. Then the two selected indices are enabled to be written to the memory bank. The outputs of the row enabler represent read enables of buffers and the outputs of the selectors represent control selections for the multiplexers for each bank.

5.3.3 Simulations Results for Memory Conflict

Table 5.1 summarizes the turbo codes used in different standards. Each standard has its own parameters such as the used code types, the possible block lengths and maximum throughput requirement. The permutation law in WiMAX, DVB-RCS and DVB-RCT is the same. Similarly, it is the same in CDMA2000 and DVB-SH. The implemented data alignment block is synthesized on Altera Stratix-III EP3SC150 FPGA with the im-

Table 5.1 main parameters in different standards

standard	codes	block sizes	number of block sizes	throughput (Mbps)
3GPP-LTE	btc	40...6144	188	100
WiMAX	dbtc	24...2400	17	70
DVB-SH	btc	1146 & 12282	2	50
HSPA+	btc	40...5114	1269	43.2
DVB-RCS	dbtc	24...864	12	31
DVB-RCT	dbtc	24...864	12	31
CDMA2000	btc	378...20736	12	2

plementation parameters shown in Table 5.2. We obtain the results indicated in Table 5.3.

From those results, the Addition of delay buffers reduces the FIFO storage sizes, simplifies the routing between the line buffers and the multiplexer network and increases the memory frequency. However, a fixed $W/4$ latency cycles are added per half iteration, so the throughput Rb is:

$$Rb = \frac{K * f_{clk}}{2 * i * (\lceil \frac{K}{F*P*W} \rceil * min(W, \frac{K}{F}) + min(\frac{W}{4}, \frac{K}{4}))} \tag{5.1}$$

where f_{clk} is system clock, i is the number of iterations, F is a factor equal to 1 for radix-2 and 2 for radix-4.

In contrast, the design without delayed buffer has one clock cycle latency per half iteration, but at the expense of lower memory clock frequency, so the throughput Rb is:

$$Rb = \frac{K * f_{clk}}{2 * i * (\lceil \frac{K}{F*P*W} \rceil * min(W, \frac{K}{F}) + 1)} \tag{5.2}$$

According to (5.1) and (5.2), the throughput for LTE standard is given in Table 5.4, with f_{clk} = memory frequency/2, W=64 symbols, P=8, i=4, F=2. The total buffers and cycle conflicts for different standards are given in Tables 5.5, 8.1. The analysis of these results indicate that the HSPA+ (using radix-4 scheme) has the biggest buffer structure between all standards. Compared to [10], our designs reduce the latency, area and conflict cycles as shown in Table 5.7.

Table 5.2 Implementation parameters

P	8	address width	15 bits
W	128 bits	delayed buffer width	6 bits
LLR Width	6 bits	line buffer width	18 bits

5.4 Memory Conflict Handling of The First Scheme

Memory conflict handling of the first scheme is similar to the second scheme. The data alignment block collects the arrived LLR values to prevent contentions by storing them

Table 5.3 Comparison between two designs for data alignment block

parameter	without delayed buffer	with delayed buffer
Number of line buffers	610	228
Number of delayed buffers	0	256
Total FIFO size (bits)	10980	5640
Number of logic cells	2064	856
Max. memory frequency(MHz)	231.48	294.9
Latency per half-iteration (clock cycles)	1	16

Table 5.4 Throughput comparison between two designs for LTE standard

Block length(bits)	without delayed buffer	with delayed buffer
40(min length)	27.5571Mbps	29.49Mbps
6144(max length)	231.48Mbps	283.104Mbps

Table 5.5 Memory analysis for radix-2 implementations (all block sizes for each standard)

standard	cycle conflicts		total buffers (bits)	
	with buffers	no buffers	with buffers	no buffers
CDMA2K	4018	9743	4422	4590
CCSDS	778	3183	2706	3006

Table 5.6 Memory analysis for radix-4 implementations (all block sizes for each standard)

standard	cycle conflicts		total buffers (bits)	
	with buffers	no buffers	with buffers	no buffers
3GPP-LTE	5669	38561	3090	7056
HSPA+	211803	495457	5640	10980
WiMAX	176	2390	3072	4716

Table 5.7 Comparison of memory conflict for HSPA+ (Radix-4 scheme) with 2 parallel SISOs

parameter	without delayed buffer	with delayed buffer	[10]
Extra Fifo Cycles	1	16	65
Conflicting Cycles(K=5114)	813	273	1037
Total Fifo Size	48	94	367

into a buffer structure. The design of the data alignment block is identical for the second scheme as shown in Figure 5.7. As the first scheme allows each SISO to write one LLR per clock cycle, there is no need to use double clock frequency for the memory. The same clock for memory and the system simplifies the design. As a result, the controller inside the data alignment block is changed to select one of the stored data in the line buffers for each bank as shown in Figure 5.9.

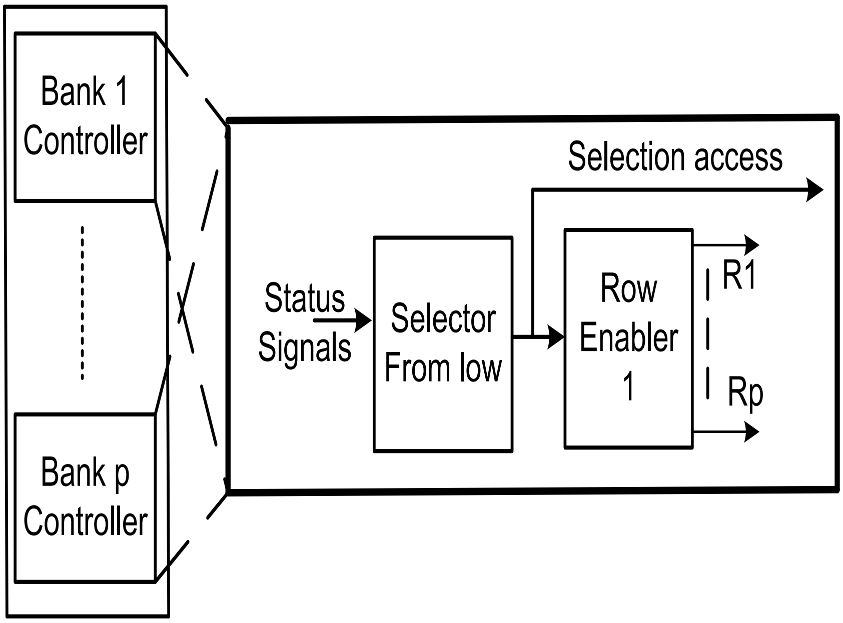


Figure 5.9 The controller of the data alignment block with divided into p small controllers

In this scheme, the used memories are single port memories to simplify the design and to reduce the power and the area of the design. So one memory access per clock cycle will be allowable for each bank. The first scheme is our timing sequence which is used in the proposed ASIP processor.

Chapter 6

ASIP Architecture

6.1 ASIP Architecture

ASIP architecture combines configurable and dedicated units through targeting certain applications. The increase of the proportion of the configurable units in the design results in more flexibility but, at the same time, it has a bad impact on the decoding throughput. So, the choice of the suitable architecture for the ASIP plays a significant role to meet the implementation requirements for different turbo decoder types.

The parallelism in turbo process is required to achieve the high throughput demand. There are two approaches to achieve the parallelism on the ASIP architecture. The first approach is to build an ASIP processor including multiple SISOs and each SISO processes independent Windows. The interfacing between multi-SISOs is controlled by the instructions of the processor. Such mechanism is fully optimized for turbo decoder architecture and avoids the waste cycles during exchanging the data [20]. The second approach is multiple ASIPs and the interfacing between them is done through communication routers to send and to receive the required data in packets format which is called network on chips (NOC) [22] [24]. The second approach produces complex interfaces and adds additional latency for the decoding time.

A pipelined processor is designed to reduce the critical path to produce high throughput. The architecture consists of nine stages: fetch, decode and execution stages as shown in Figure 6.1. The execution stages include seven stages: addresses generation, branch metric calculations, state metric calculations, three stages for LLR calculations and write back stage.

6.2 State Metric Unit

The state metric units occupy most of the design area which is around two-thirds of the hardware resources. Both forward and backward state metric values are required for calculating the LLR values. The implementation of the forward and backward state metric units is identical. To meet the throughput requirement two units are implemented, one for forward metric and the other for the backward metric, to work simultaneously. The add compare select, ACS, is the basic calculation unit for the state metric units. The feedback in the state metric unit, due to recursion, imposes certain critical path which has a big influence on the throughput. The critical path of the state metric unit, in this case represents the dominant critical path, determines how to design the pipelined stages

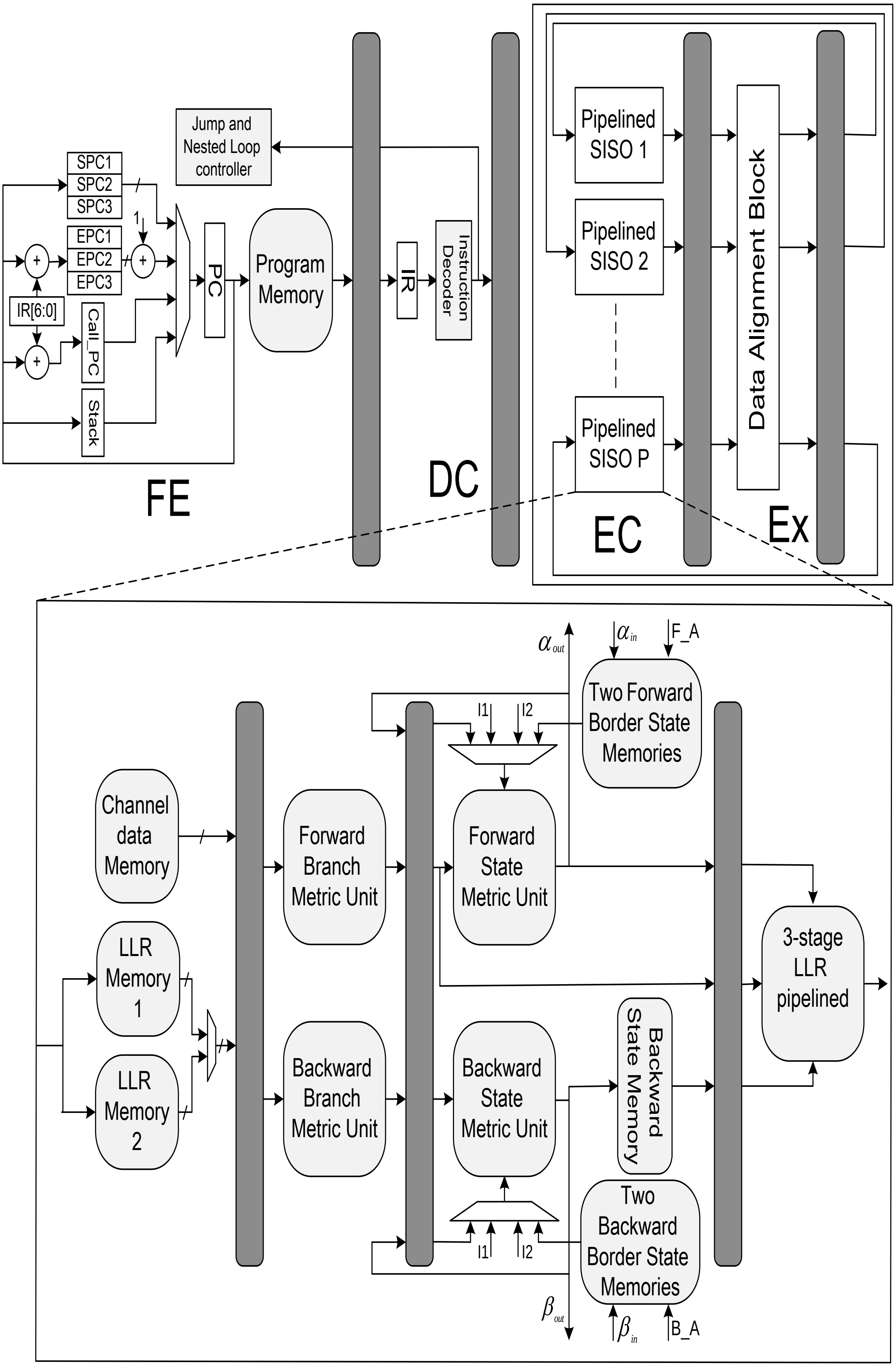


Figure 6.1 Block diagram for pipelined ASIP architecture showing the different stages

by putting registers to keep up the same length of such critical path or lesser for other blocks.

There are different ways to implement the state metric unit. Such ways are based on radix-4 calculation, radix-2 calculation or compromised way between radix-4 and radix-2. In all cases, the designed unit should be able to perform all possible calculations. Figure 6.2 shows three different architectures to build a unified block of state metric to support single and duo-binary types. For each ACS unit, the value of $\alpha(x)/\beta(x)$ is chosen by one of the inputs α_y/β_y as x takes one of values 1, 2, 3 and 4 while y takes the values from 0 to 7. These choices depend on the encoder architecture which means that it's changed from one standard to other. Each selection needs 3 bits to select one of eight possible values and there are four inputs for each ACS unit. So there is 12 bits configuration word for each ACS. The total configuration word for one state metric unit is 48 bits. These configuration words are loaded in the configuration register in the beginning of the decoding process.

The following section presents these configurations and their different impacts on the throughput, hardware utilization and area.

6.2.1 First Configuration

The first configuration is more convenient for radix-4 calculations than radix-2 as this scheme can perform the ACS between four branches in a single clock cycle. The calculation of state metric in radix-2 form has poor utilization as it uses almost half of the hardware resources. In order to increase the utilization of the hardware resources in radix-2 single binary schemes, the trellis can be compressed in time to be similar with radix-4 as proposed in section 3.4.

Such conversion has many drawbacks on the whole design. One of these drawbacks is the interleaver design. The interleaver of radix-2 single binary is based on bit level addresses which means K bit LLR values require K addresses. This conversion compresses the trellis in time, as mentioned earlier, producing K interleaved addresses in only $K/2$ clock cycles in case of radix-4 single binary. This leads to more contentions on the memory as the time to write the whole LLR values is reduced by half. This problem is not raised for radix-4 duo binary case because the interleaver design is based on symbol addresses which means $K/2$ symbol LLR values, which is equivalent to k bit LLR values, requires $K/2$ addresses.

Additionally, the conversion of 3GPP2-CDMA2000 complicates the implementation of branch metric unit. Specifically, in the decoding rate 1/5, which is used in 3GPP2-CDMA2000 standard, the radix-2 scheme calculates 8 different branch metrics while 64 different branch metrics are calculated in case of radix-4 single binary. This result in a large computational requirement and an increase in the number of memory access which requires more than one port memory leading to poor utilization. So, it is not a suitable choice for the configuration that supports both radix-2 and radix-4 and with good utilization.

In addition, in parallel radix-4 scheme, 3GPP-LTE and WiMAX are not MCF, which means that they will face conflicts during parallel access. However, it is important to note that the parallel generated addresses are even-odd patterns as shown in Figure 6.3. By dividing each memory bank into two sub-banks even and odd access banks, those conflicts can be avoided.

The Turbo decoder in the LTE (single binary) has the highest throughput. So our

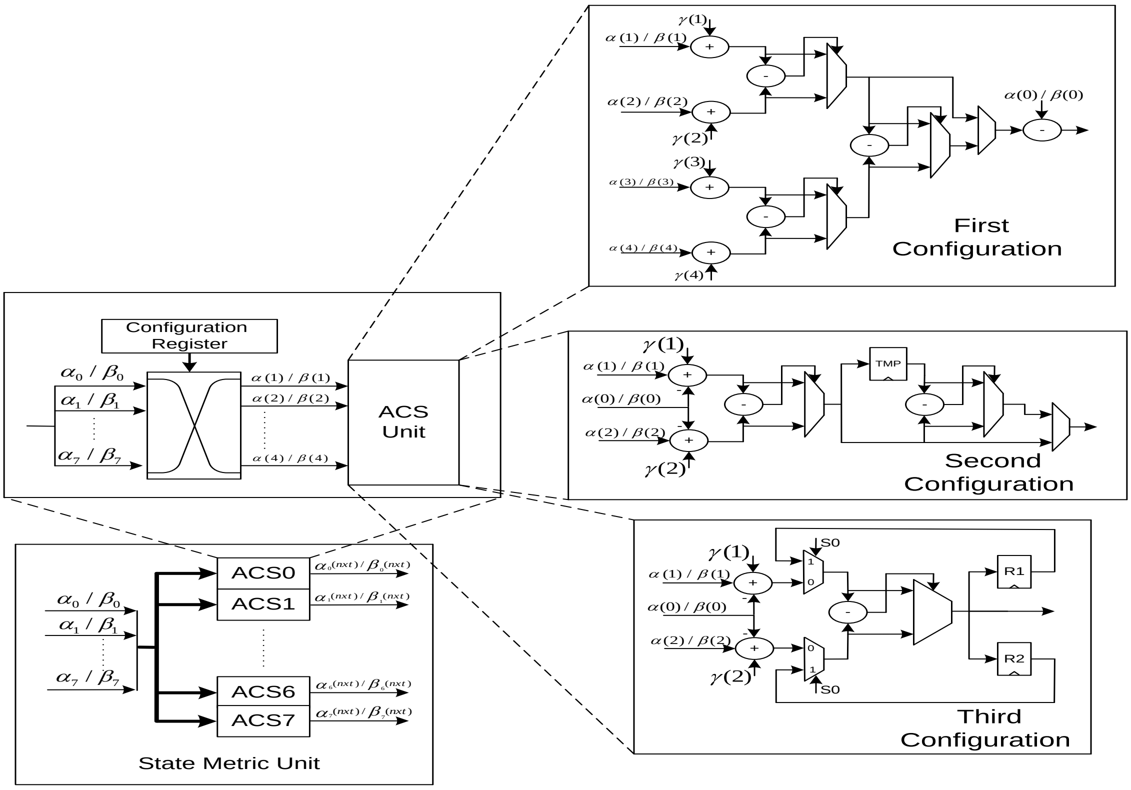


Figure 6.2 State Metric Unit

target is to speed the decoding process for single binary to achieve this requirement.

6.2.2 Second Configuration

The second configuration is based on radix-2 calculations as this scheme performs the ACS between two branches in a single clock cycle. An iterative manner is used for radix-4 calculations which produce the state metric values every two clock cycles. This scheme is suitable for all turbo code standards with good utilization for hardware resources. The avoidance of the trellis compression technique leads to less conflict, less area and less complexity in the interleaver design. The same throughput is achieved for single binary and duo-binary schemes.

The calculation of the state metrics for radix-4 duo-binary, ACS between four branches, takes two cycles. In the first cycle, the result of the ACS of two branches is stored in a temporary register. At the second clock cycle, the ACS of the other two branches is calculated, and then the output of state metric is the maximum value between the updated value and the stored value in a temporary register. The critical path for radix-2 includes two additions, one max operation and a multiplexer gate delay. On the other side, the critical path for radix-4 includes two additions, two max operations and a multiplexer gate delay.

6.2.3 Third Configuration

The third configuration is based on radix-2 calculations similar to the second configuration. In the same way, an iterative manner is used for radix-4 calculations but produce the

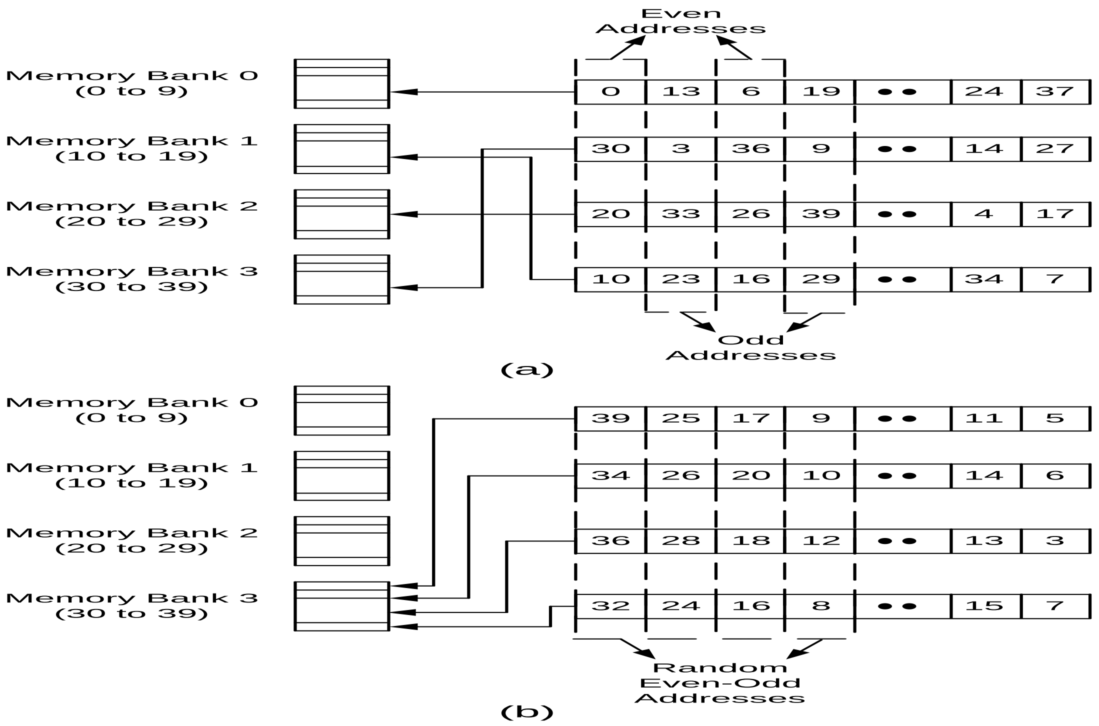


Figure 6.3 parallel access with interleaed addresses with K=40, p=4, w=10 (a) for LTE, no conflicts happen, (b) for HSPA, conflicts happen on bank 4

state metric values every three clock cycles instead of two clock cycles. This configuration is suitable for single binary calculations rather than duo-binary as it targets to speed up the radix-2 calculations by reducing its critical path at the expense of radix-4 calculations. As a result, the throughput of single binary is higher than duo-binary schemes. The increase in decoding time for duo-binary, due to taking three clock cycles for the state metric calculations, overcomes the reduction in critical path of state metric unit.

The calculation of the state metrics for radix-4 duo-binary takes three cycles. In the first cycle, the result of the ACS of two branches is stored in one of two temporary registers. In the second clock cycle, the ACS of the other two branches is calculated and stored in the other temporary register. In the third clock cycle, the output of state metric is the maximum value between the updated values in the temporary registers. The critical path for both radix-2 and radix-4 calculations are two additions, one max operation and a multiplexer gate delay.

According to the state metric implementation, the other units, the branch metric and LLR units, adapt their resources to enhance the hardware utilization. Each stage takes a fixed amount of time, so the state metric stage receives the branch metrics and sends its output to LLR stage every three cycles in case of radix-4 scheme. The implementation of the LLR and branch metric stages should have less area than the second configuration but the third configuration takes more clock cycle than second configuration in radix-4 calculations. However the area of the LLR and branch metric stages is identical to the second configuration due to the restrictions of radix-2 calculations. As a result, the branch metric and LLR calculation units exploit the hardware resources every two out of three cycles for producing their output and only the state metric units exploit all cycles. So the utilization of duo-binary scheme is poor.

One of the most important features for the third configuration is that there is no added latency cycles between the two decoders because of the conflicts on memory banks as shown in Figure 6.4. However, there are some instructions should be executed to initialize the operation of each MAP decoder when transferring from one MAP decoder to the other. Hence, there is no benefit from zero latency cycles. The zero latency cycles are suitable for dedicated architectures where the two MAP decoders are consecutively working without initialization.

Figures in 6.4, 6.5, and 6.6 show the effects of the parallel architecture of the three configurations on added latency between two decoders, memory buffers, and the probabilities of simultaneous accesses on memory banks. In addition, the different configurations are synthesized on Altera stratix-III FPGA and the results are given in Table 6.1. The table results are in case of one SISO implementation and the parallel SISOs effects are not included.

From those results, the first configuration has the highest throughput with adequate area in case one SISO implementation for both turbo code types. So, the first configuration is the best choice for the one SISO implementation. However, large latency is added for first configuration in case of the parallel architecture, as shown in Figure 6.4, because of the extra clock cycles required to resolve the memory conflicts. A large degradation of the throughput and large buffers are the result in case of using more than one SISO. As a result, such configuration should be avoided in the parallel architecture scheme.

Additionally, the third configuration has the least conflicts, the smallest area and the least number of buffers among the different configurations. In addition, the third configuration has the highest throughput per area in case of single binary scheme. However, the third configuration has the least throughput among the different configurations for duo-binary scheme.

The second configuration has an acceptable latency between the two decoders, and also not much buffers are needed to resolve the memory conflicts. The second configuration gives almost the same high throughput for both turbo code types with adequate area. The second configuration is the suitable choice amongst these configurations for parallel architecture design.

Table 6.1 Comparison of Three Configuration for one SISO

Comparison	First Config.	Second Config.	Third Config.
Max. clock frequency	125.27MHz	125.27MHz	140.88MHz
Throughput for Radix-2 LTE @6144 bit	30.8Mbps	15.5Mbps	17.4Mbps
Throughput for Radix-4 WiMAX @2400 symbol	30.1Mbps	15.2Mbps	10.17Mbps
State Metric unit	1844 LC	1035 LC	946 LC
LLR calculation unit	1948 LC	918 LC	918 LC
Branch Metric Unit	126 LC	160 LC	160 LC
One SISO unit	5888 LC	3308 LC	3130 LC

The initialization of the forward and backward states border is chosen by

- all zeros, equal probabilities for all the states, which happens at the first iteration for each MAP decoder

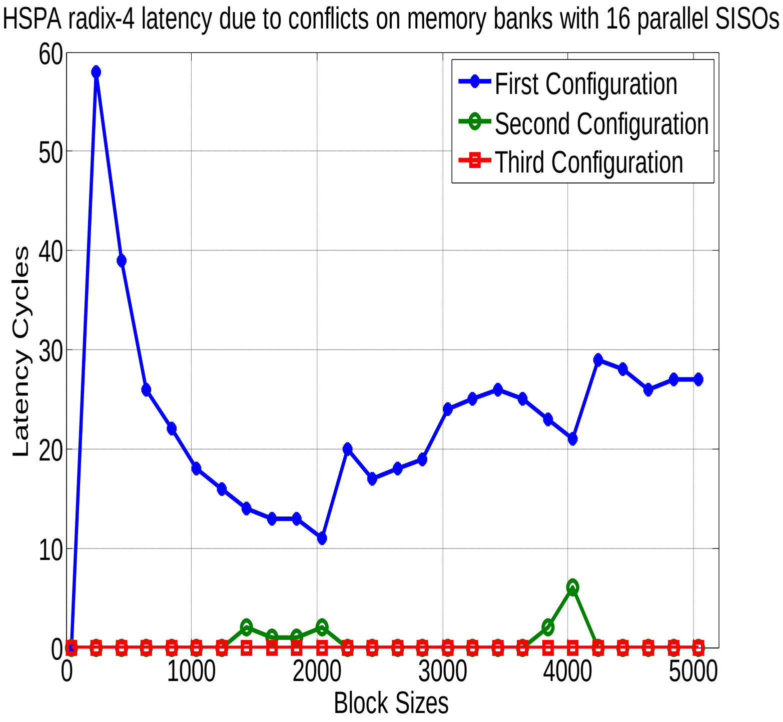


Figure 6.4 Effect of three configurations on latency between the 2 decoders due to memory conflicts with W=64, P=16, for some block sizes of HSPA+ standard

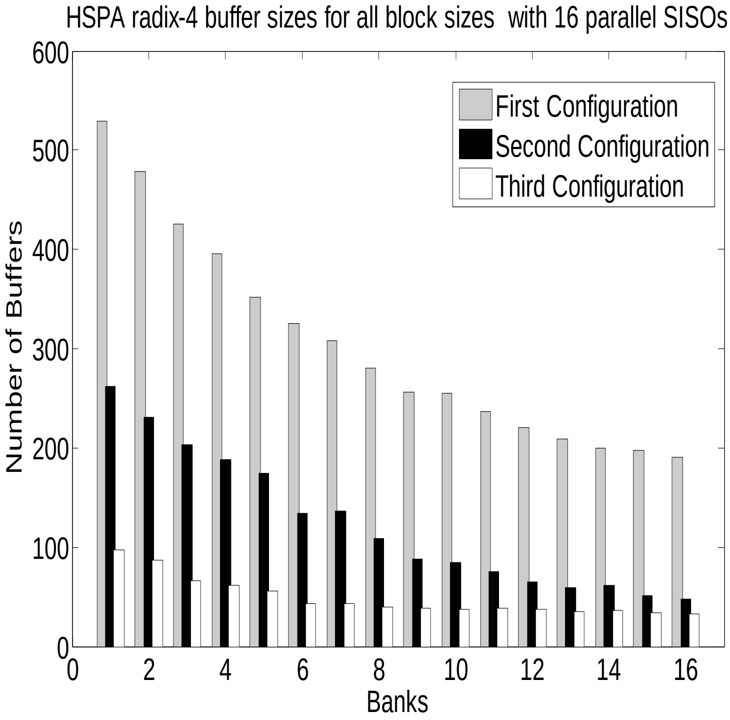


Figure 6.5 Effect of three configurations on memory buffers in the data alignment block with W=64, P=16, for all block sizes of HSPA+ standard

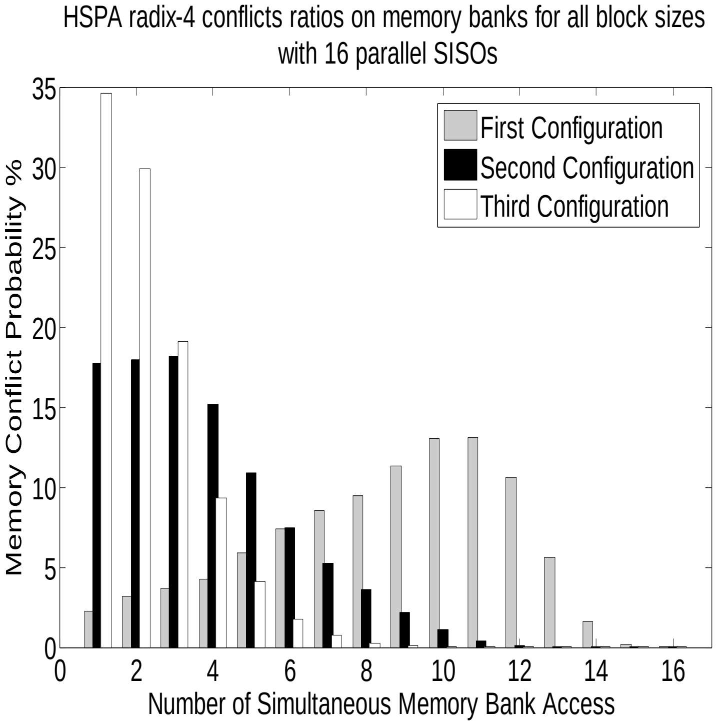


Figure 6.6 Effect of three configurations on probability of simultenous accesses on memory banks with W=64, P=16, for all block sizes of HSPA+ standard

- border values which happens at the iterations followed by the first one
- starting from the state zero S_0 or
- with the updated values for recursive operation which happens within the window calculations.

Where the starting from the state zero S_0 means giving the highest value to S0 and the lowest values to the other states.

6.3 Memory Access

To enable the processor to work in a faster manner, dedicated memories are needed to store and retrieve the manipulated data. In addition, dedicated calculation units for generating the required addresses and control signals are also needed. All these dedicated units are controlled by the instructions of the program.

Each MAP decoder for the single binary scheme has one received systematic symbol and one received parity symbol. On the other hand, each MAP decoder for duo-binary scheme has two received systematic symbols and two received parity symbols. The design of memory should be implemented to adapt with two types of turbo codes. The architecture of proposed memory consists of two sub-memories. These two sub-memories operate as two separate units in case of duo-binary and one unit in case of single binary.

There is another potential problem for the memory design: each branch metric unit may read two different values from each memory. To avoid the usage of dual port memories, the memory is divided into a number of memory banks and the size of each memory bank is W as shown in the Figure 6.7(a). According to the decoding process, the first read value from top to down of a certain bank goes to forward branch metric unit as shown in the Figure 6.7(b). While the second read from down to top of the next bank goes to backward branch metric unit. So the offset part of the address that accesses the forward bank is inverted to access the backward bank.

In addition, the LLR memory works in two modes Duo-binary and single binary modes. For single binary mode only one LLR value (LLR1) is stored while for duo-binary there are three LLR values (LLRA, LLRB and LLRC) to be stored. The maximum length of memory for single binary is around 6,500 while for duo-binary is 2400. To maximize the utilization of this block, the memory of LLRA and LLRB units as one block in single binary mode where LLRA memory is used for LLR1 even addresses and LLRB memory is used for LLR1 odd addresses. The memories of LLRA and LLRB units work as two sub blocks in duo-binary mode. The memory of LLRC values isn't used in single binary mode.

There are two memories to store the interleaver/deinterleaved addresses. These memories provide the addresses to the data alignment block to store the LLR data. Many works has used a loadable interleaver mechanism such as in [21]. All these memories are work in sequential manner, so each memory has its address counter to point to the current location. The enable signals of these addresses counters are come from the instructions of the processor.

6.4 Branch Metric Unit

The number of branch metrics changes from one standard to another according to the type of turbo codes and the coding rate. Most of the current standards have eight states. Each state in the trellis diagram has four possible branches for duo-binary scheme, where the possible combinations belong to 00, 10, 01 and 11, so there are thirty-two possible branches. However the number of calculations of the branch metric is reduced by half, as there are sixteen different values and the others are the same. In the same manner, each state in the single binary scheme has two possible branches, where the possible combinations belong to 0 and 1, so there are sixteen possible branches. However the number of calculations of the branch metrics is reduced to be four branch metrics for LTE standard and eight branch metrics for UMTS.

The proposed branch metric unit is configured to generate sixteen, eight and four branch metrics as shown in Figure 6.8. The output of the branch metric unit goes to the state metric and LLR units. As explained earlier, the calculation of the state metrics and LLR values take two clock cycles for duo-binary and one clock cycle for single binary schemes. The proposed branch metric unit produces eight branch metrics every one clock cycle. The branch metric calculations of the WiMAX standard take two phases. In the first phase, the branch metric unit produces the first eight branch metrics which belong to the pair of inputs 00 and 01. In the second phase, the branch metric unit produces the other eight branch metrics which belong to the pair of inputs 10 and 11. The branch metric calculations are done in one clock cycle in case of LTE scheme. For UMTS calculations, the upper four branch metrics are identical to the lower branch metrics which simplify the design of the state metric.

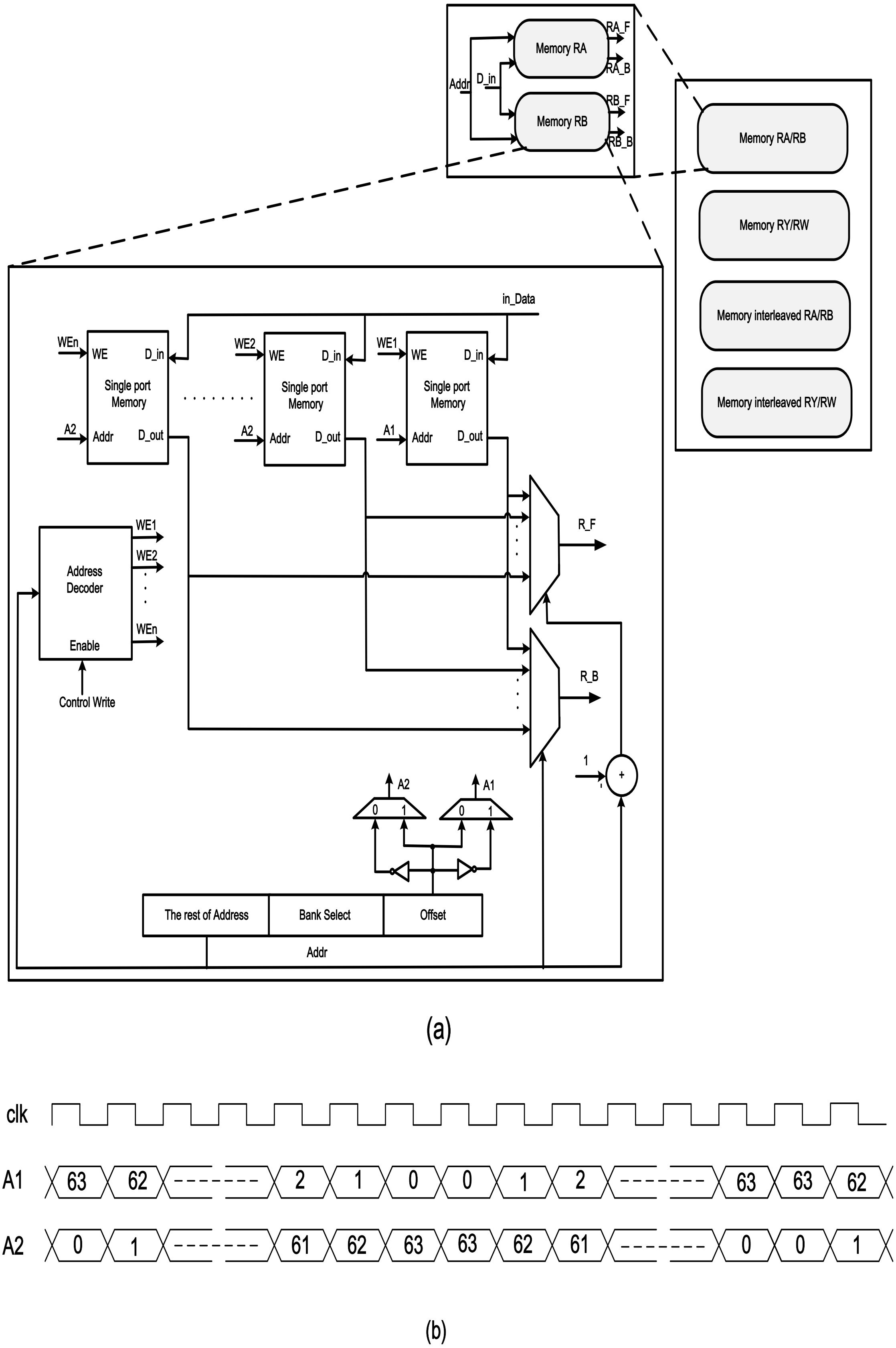


Figure 6.7 Block diagram for channel data memory

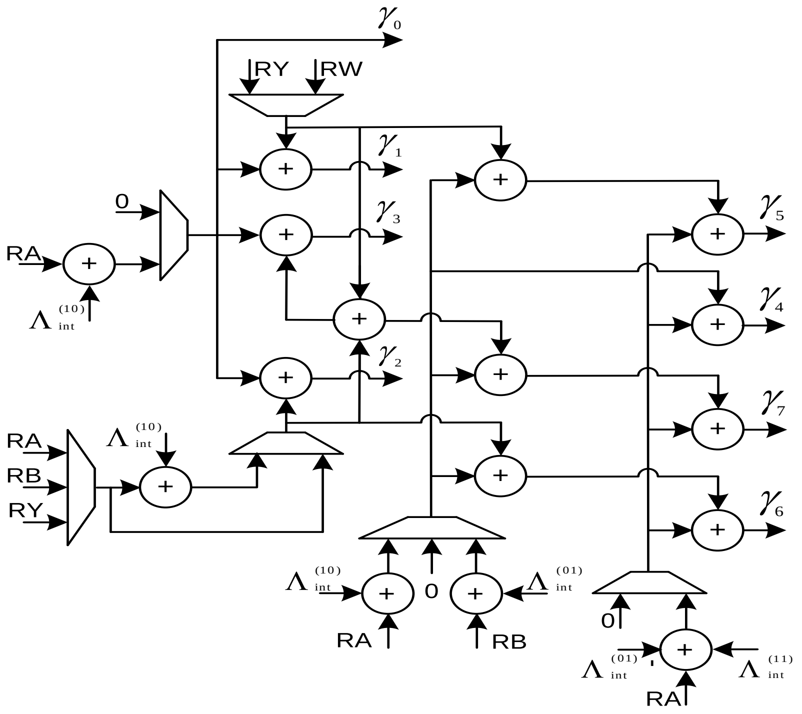


Figure 6.8 Branch Metric Unit

6.5 LLR Calculation Unit

The turbo decoder algorithm estimates the final decision of the received data based on the LLR values at the end of the pre-determined iterations. The calculation of the LLR values requires branch metrics as well as forward and backward state metrics. According to the proposed decoding process, the backward state metrics of a window are calculated in advance of the forward state calculations which results in the production of the LLR values to be delayed within the first *W* cycles. Then, the LLRs values are concurrently produced from the SISOs units and are transmitted to the LLRs memories.

The LLR calculation unit is divided into three pipelined stages in order to enhance the allowable maximum frequency as shown in Figure6.9. These stages are two 2-stage LLR calculation units and another stage is added for duo-binary calculation as shown in Figure 6.1. Each 2-stage LLR calculation unit produces one LLR value.

As mentioned earlier, the duo-binary scheme requires producing four LLR values which represent the probability of the received symbol to be one of 00, 10, 01 or 11. The LLR calculations of the duo-binary scheme take two phases. At the first phase, the LLR values of $\Lambda_{int}^{(00)}$ and $\Lambda_{int}^{(01)}$ are calculated and the difference between them, $\Lambda_{ext}^{(01)}$, is sent to the memory. In addition, the $\Lambda_{int}^{(00)}$ is stored to be used in the next clock cycle. At the second phase, the LLR values of $\Lambda_{int}^{(10)}$ and $\Lambda_{int}^{(11)}$ are calculated and subtracted from the $\Lambda_{int}^{(00)}$ to produce $\Lambda_{ext}^{(10)}$ and $\Lambda_{ext}^{(11)}$ respectively. On the other hand, the single binary scheme requires one phase for producing two LLR values, $\Lambda_{int}^{(0)}$ and $\Lambda_{int}^{(1)}$, which represent the probability of the received bit to be either 0 or 1. The difference between $\Lambda_{int}^{(1)}$ and $\Lambda_{int}^{(0)}$ is calculated, $\Lambda_{ext}^{(1)}$, and is sent to the memory.

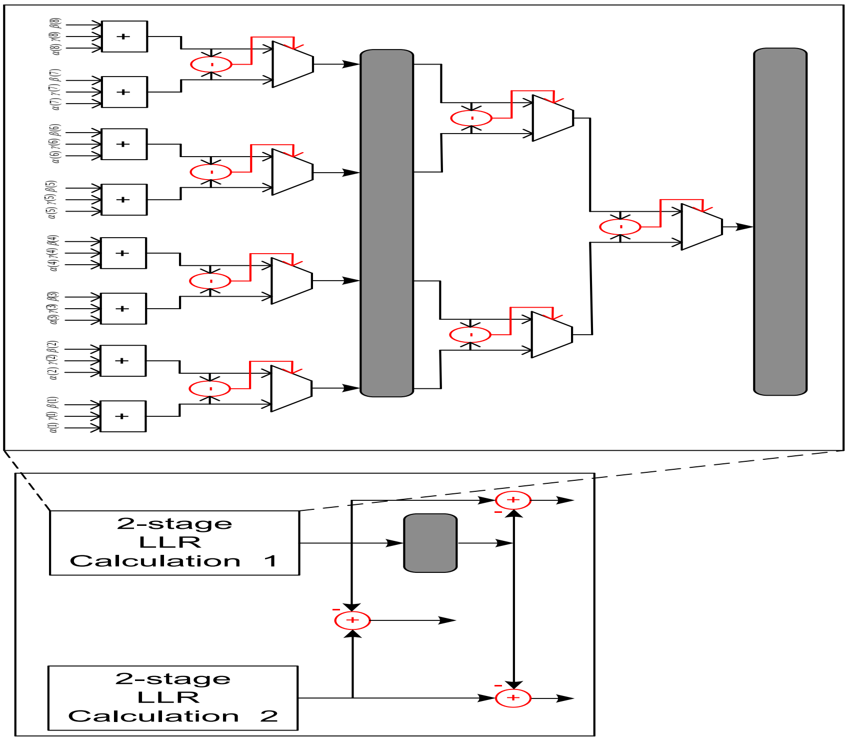


Figure 6.9 Block diagram of the LLR calculation unit

6.6 Instructions

The proposed parallel SISO processor is based on a single instruction multiple data, SIMD, instructions. The SIMD instruction performs several operations in different pipelined stages all at once. The designed instructions include two classes: control and operative instructions. The number of instructions is not required to be large as we are targeting certain applications not general programs.

The control instructions include zero overhead loop (ZOL),goto, LOOPNE, call and return instructions. The call instruction allows building one subroutine for two MAP decoders. There are three zero overhead loop (ZOL) instructions. The ZOL instruction is implemented to save the wasted cycles for initializing the loop counter register, and for avoiding the load and move instructions during the branching loop. The ZOL instructions allow the nested loop operations. The nested loop instructions are convenient to perform the iterations of the decoding process. The decoding process is composed of a certain group of instructions which are repeated numbers of times such as:

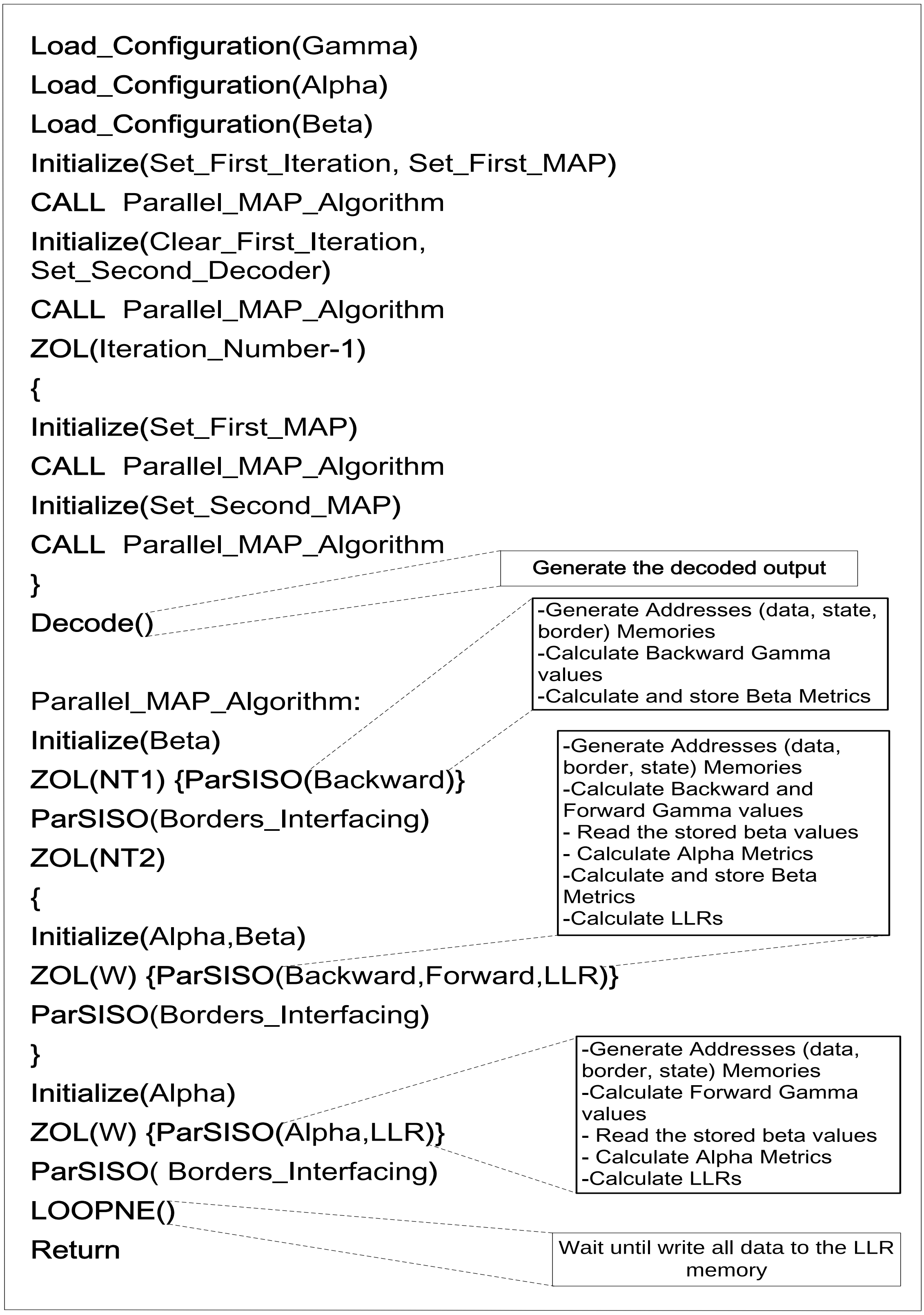
- The calling of the subroutine of the MAP decoder is repeated according to the number of iteration.
- Inside the subroutine of the MAP decoder, the window operations are repeated.

All ZOL instructions are relative jump. Our processor targets real time applications, so the predication mechanisms for jump instructions are not suitable technique. The no operation instruction Nop is used after each ZOL instruction so the ZOL instruction implies two clock cycles for the execution. There is two-level stack to hold the PC during calling the subroutine and retrieving it when the return instruction is executed. The LOOPNE, loop if not empty, instruction checks if there is any LLR data stored in the buffers of the data alignment block or not. If there is any LLR values will loop until all data is written to the memory. The call and return instructions take three cycles because of the pipeline operations. These cycles are one for execution and two for Nop instructions. These two Nop instructions are due to the fetching and the decoding of the next instructions.

Additionally, the operative instructions include parallel SISO operations (ParSISO), Move, store channel values to the memory StrData, Decode, load configuration and initialization instructions. Of course the initialization instruction is required before calling the main subroutine to set up the execution of each MAP decoder. The ParSISO instruction controls the execution of the different operations in multi-SISO units. Decoded instruction generates the Decoded output of the received data values. Figure 6.10 shows pseudo code description of Turbo decoding algorithm.

6.7 Interfacing Between SISOs

As mentioned earlier, the parallel processing is proposed to meet the high-throughput requirement. The proposed mechanism to handle the parallel processing is by dividing the incoming data block into several windows and each window is processed independently using one SISO unit. Each window operates in the forward and in the backward directions to calculate the recursion states. The NII technique is used to initialize the border values of states to avoid the degradation in the decoding performance. The border values of a certain window are calculated from the predecessor SISO for the forward states and from the successor SISO for the backward states. The interfacing between the adjacent SISOs is implemented to handle transferring the border values between the SISOs as shown in the Figure 6.11. The transferring of the border values between the SISOs happens at the last clock cycle after processing each window. The calculations of the border values are not used in the same iteration as all SISOs concurrently start which prompts to store them into the border memories. Starting from the second iteration, the border values are read from the border memories at the beginning of the window operations. The generation of the reading and writing addresses of the border memories all are done through PraSISO instruction. The circular trellis termination imposes that the first forward/backward state values to be the same as the last forward/backward state values. The SISO, in which the last state values happen, are varying and this variation depends on the block size K. So all the forward states of different SISOs direct to the first SISO to choose the proper values. By the same way, the first backward state values direct to all backward states of different SISOs.



```
Load_Configuration(Gamma)
Load_Configuration(Alpha)
Load_Configuration(Beta)
Initialize(Set_First_Iteration, Set_First_MAP)
CALL  Parallel_MAP_Algorithm
Initialize(Clear_First_Iteration,
Set_Second_Decoder)
CALL  Parallel_MAP_Algorithm
ZOL(Iteration_Number-1)
{
Initialize(Set_First_MAP)
CALL  Parallel_MAP_Algorithm
Initialize(Set_Second_MAP)
CALL  Parallel_MAP_Algorithm
}
Decode()

Parallel_MAP_Algorithm:
Initialize(Beta)
ZOL(NT1) {ParSISO(Backward)}
ParSISO(Borders_Interfacing)
ZOL(NT2)
{
Initialize(Alpha,Beta)
ZOL(W) {ParSISO(Backward,Forward,LLR)}
ParSISO(Borders_Interfacing)
}
Initialize(Alpha)
ZOL(W) {ParSISO(Alpha,LLR)}
ParSISO( Borders_Interfacing)
LOOPNE()
Return
```

Generate the decoded output

-Generate Addresses (data, state, border) Memories
-Calculate Backward Gamma values
-Calculate and store Beta Metrics

-Generate Addresses (data, border, state) Memories
-Calculate Backward and Forward Gamma values
- Read the stored beta values
- Calculate Alpha Metrics
-Calculate and store Beta Metrics
-Calculate LLRs

-Generate Addresses (data, border, state) Memories
-Calculate Forward Gamma values
- Read the stored beta values
- Calculate Alpha Metrics
-Calculate LLRs

Wait until write all data to the LLR memory

The values of NT1 and NT2 depend on the block length

Figure 6.10 Pseudo code description of Turbo decoding algorithm for single binary scheme

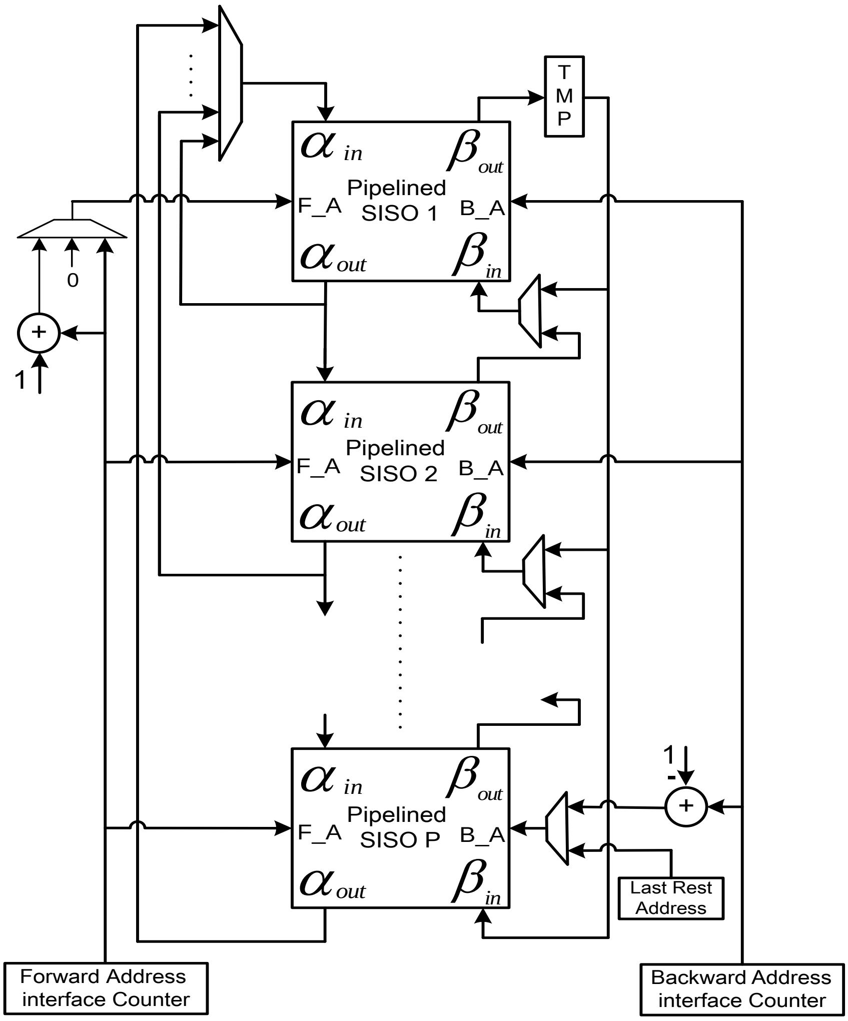


Figure 6.11 Interfacing between SISOs

Chapter 7

Inside The CPU

The architecture of the processor is divided into two main units: the data path unit (DP) and the control unit (CU) as shown in Figure 7.1. The control unit controls all parts of the DP unit, as its purpose is delivering the control signals according to the executed instruction each cycle to DP unit. While the purpose of the DP unit is manipulating the data and executing any required operation according to the control signals from CU unit. In addition, the DP unit provides the control unit with the status of some registers to enable the processor to take the decision according to executed conditional instructions. As an example, the DP unit indicates if there is an overflow resulting from a previous operation or not.

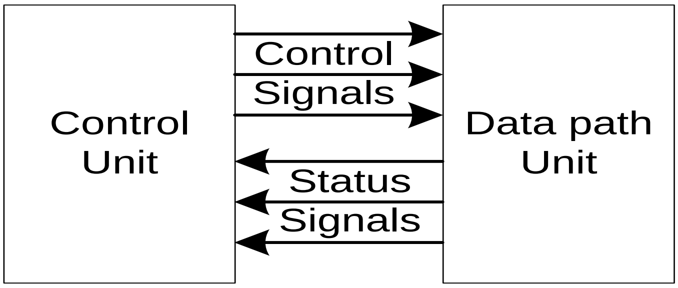


Figure 7.1 General block diagram architecture for the processor

The main task of the processor is the execution of the programs. The programs are stored in the program memory. According to the written program, the processor achieves the target task through certain instructions. In order to enable the processor to execute the instructions faster, there are some dedicated registers inside the control unit. The two main dedicated registers are instruction register IR and program counter PC. The PC register points to the next executed instruction. While the fetched instruction is loaded to IR register.

The ASIP processor executes the stored instructions in the program memory. The process of the execution of the instruction is divided into three phases: fetch, decode and execute phases. The first phase is to bring the instruction from the program memory which is pointed by the PC register. The fetched instruction is written to the IR register in the second phase. While the third phase is the execution of the instruction according to the operation code. The operation code (op-code) is a part of the instruction formatting beside the operands, if the instruction requires operands. The op-code tells the control unit which operation, such as addition or store to memory operation, is requested by the instruction. The op-code of each instruction has unique patterns. These phases may be executed sequentially as shown in Figure 7.2 which is known as fetch-decode-execute cycles.

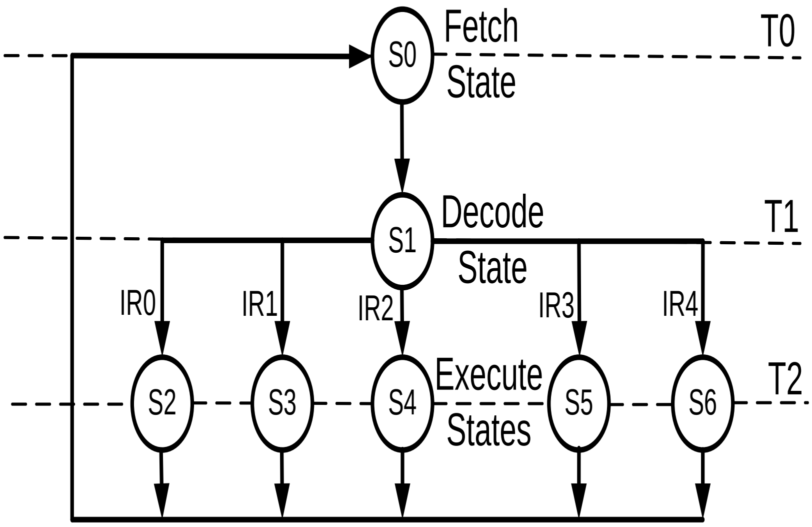


Figure 7.2 State diagram of non-pipelined processor

The processor goes between three times phases T_0 , T_1 and T_2. T_0 indicates that the processor is in the fetch state. The decode and the execute states correspond to T_1 and T_2 respectively. These signals are generated from a counter which is called a sequence counter SC as shown in Figure 7.3 . The SC is incremented every clock cycle by one until it reaches the end of the execution phase. As the SC reaches the end of the execution phase, SC goes to zero value to go to the fetch state again. The SC is an input to a binary decoder to generate the T_n signals as n is a general number.

In order to distinguish which instruction will be executed, the op-code in the IR is an input to a binary decoder to generate the q_n signals. Only one of the q_n signals is active at any one time, corresponding to the op-code value of the executed instruction.

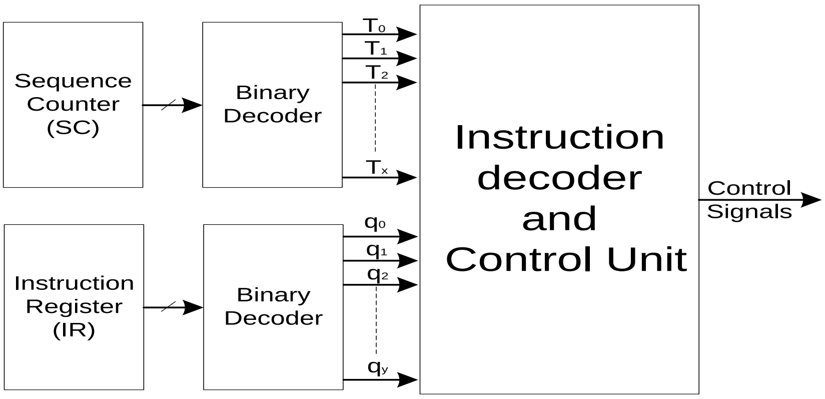
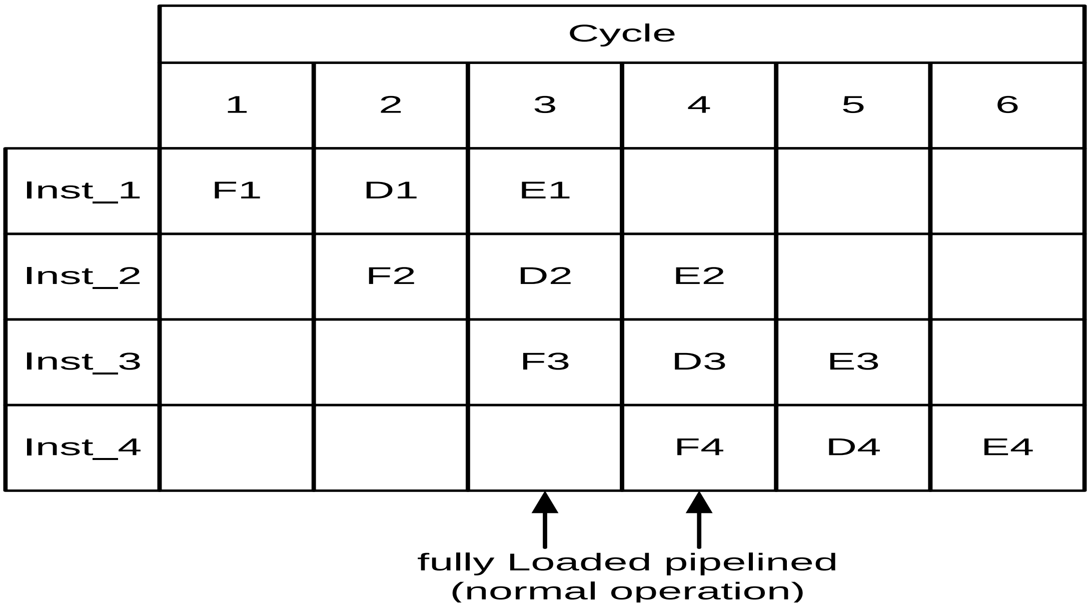


Figure 7.3 Block Diagram for generating the T_n and q_n signals

The pipelined process is used to reduce the latency time between the executed instructions which affects the overall throughput improvement. The order of the fetch-decode-execute mechanism is changed. The fetch of the first instruction happens at T_0 phase.

While at T_1 phase, decoding of the fetched instruction, the first instruction, and fetching of the next instruction, the second instruction, happen. At T_2 phase, the execution of the first instruction, decoding the second instruction and fetching a new instruction happen. The processor remains in T_2 phase for most of the time which indicates a fully loaded pipeline process (normal operation) as shown in Figure7.4. The processor may return to T_0 only when executing the branch instructions. The processor should flush the pipeline when executing any branch instructions to prevent an incorrect memory address from being loaded.



	Cycle					
	1	2	3	4	5	6
Inst_1	F1	D1	E1			
Inst_2		F2	D2	E2		
Inst_3			F3	D3	E3	
Inst_4				F4	D4	E4

fully Loaded pipelined
(normal operation)

Figure 7.4 The process of the execution of the pipelined instructions

7.1 Micro-instructions

This section describes how the instruction works in details. How the control signal is generated for different type of instructions is also presented. The following notation is used to describe the transfer of the data and control signals among different operations

$Condition : Transfer\ statements$

As the true condition is satisfied, the transfer statements occur otherwise there is no change. The op-code of the Instructions is given in Table 7.1

7.1.1 NOP Instruction

The NOP Stands for no operation instruction. the purpose of NOP instruction is inserted after some of the loop and branches instructions. The formatting of the NOP instruction is shown in Figure 7.5. The NOP instruction description is given in Table 7.2

Op-code	Reserved

Figure 7.5 The formatting of the Return instruction

Instruction	Op-Code	the active q
NOP	0000	q_0
Call	0110	q_6
Ret	0111	q_7
ZOL1	0001	q_1
ZOL2	0011	q_3
ZOL3	1001	q_9
LOOPNE	1010	q_{10}
Goto	1011	q_{11}
ParSISO	0101	q_5
Initialize	0100	q_4
Mov	1000	q_8
StrData	0010	q_2
Decode	1100	q_{12}
Config	1101	q_{13}

Table 7.1 The op-code of the Instructions

NOP	No Operation
syntax:	NOP
Operands:	None
Description:	no operation

Table 7.2 The NOP Instruction Description

7.1.2 Call Instruction

The formatting of the call instruction is shown in Figure 7.6. The call instruction description is given in Table 7.3.

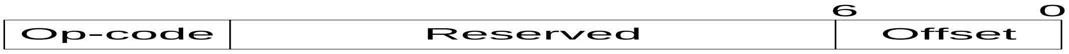


Figure 7.6 The formatting of the call instruction

CALL	Call Subroutine
syntax:	$Call$ offset
Operands:	$-64 \leq offset \leq 63$
operation:	$q_6T_2 : PC \leftarrow PC + offset$, $Stackmemory \leftarrow PC$, $SC \leftarrow 0$
Description:	saving the current PC and updating the PC with the new address

Table 7.3 The Call Instruction description

The call instruction is relatively branching instruction and is performed by adding the offset to the current PC to go to the subroutine. The current PC is stored into the stack register in order to have the ability to return back to the current position after reaching to the end of the subroutine. The calling instruction has the op-code of 0110, this means the q_6 will be active when executing the call instruction.

The clearing of the SC is required to flush the pipelined as the PC is changed and need to fill the pipeline from the branching address. As the execution only occurs at T_2 phase, there are two cycles passing from T_0 and T_1 to reach to T_2. The execution of the call instruction effectively takes three cycles.

7.1.3 Ret Instruction

The forrmating of the return instruction is shown in Figure 7.7. The ret instruction description is given in Table 7.4.

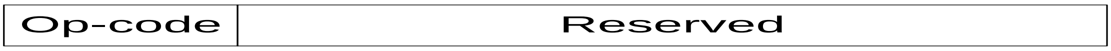


Figure 7.7 The formatting of the Return instruction

Return	Return from Subroutine
syntax:	*Ret*
Operands:	None
operation:	$q_7T_2 : PC \leftarrow Stackmemory$, $SC \leftarrow 0$
Description:	Returning the stored PC from the stack

Table 7.4 The Return Instruction description

The Ret instruction retrieves the stored PC from the stack. Updating the PC with the new value instead of the sequential increments which leads to flush the pipeline. In order to flush the pipeline, clearing SC is fulfilled during the execution phase. As mentioned earlier, the instruction that includes clearing the SC takes three cycles.

7.1.4 Goto Instruction

The formatting of the Goto instruction is shown in Figure 7.8. The Goto instruction description is given in Table 7.5.

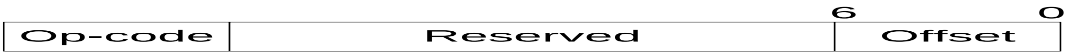


Figure 7.8 The formatting of the Goto instruction

The Goto instruction updates the PC with the new value instead of the sequential increments which leads to flushing the pipeline. In order to flush the pipeline, clearing SC is fulfilled during the execution phase. As mentioned earlier, the instruction that includes clearing the SC takes three cycles.

Goto	unconditional relative jump
syntax:	*Goto* offset
Operands:	$-64 \leq offset \leq 63$
operation:	$q_{11}T_2 : PC \leftarrow PC + offset$, $SC \leftarrow 0$
Description:	jump to the new address with updating the PC

Table 7.5 The Goto Instruction description

7.1.5 ZOL Instruction

ZOL instruction executes a set of instructions for a number of times as shown in Figure 7.9. The ZOL takes two operands NTR and NRI where NTR stands for the number of times repeated while NRI stands for the number of repeated instructions. The formatting of the ZOL instruction is shown in Figure 7.10. The ZOL instruction description is given in Table 7.6.

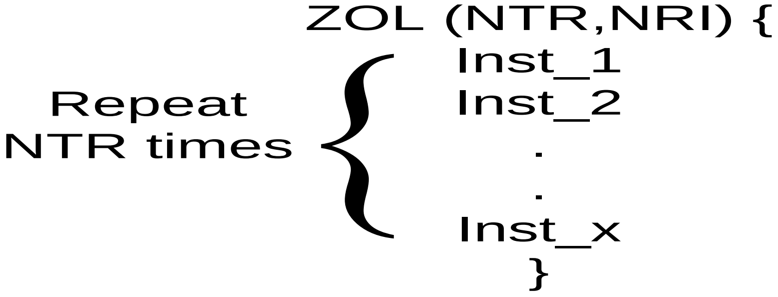


Figure 7.9 The description of how the ZOL instruction works

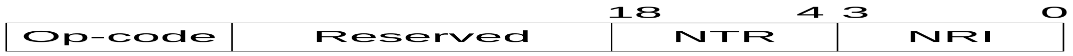


Figure 7.10 The formatting of the ZOL instruction

The purpose of using ZOL instruction is to save the wasted cycles in initializations, increment and moving cycles during the execution of the loop. In order to fulfill this task, avoiding wasted cycles, a number of dedicated registers is used as shown in Figure 7.11. The dedicated registers for ZOL instruction are SPC register, EPC register, NT register and ZOLFlag. There are two registers one to point to the first instruction in the loop which is SPC and the other to point to the last instruction in the loop which is EPC. The initialization of SPC with the current value of PC and the initialization of EPC with the resulted value of adding PC with NRI are done at the execution phase of ZOL instruction. Besides, the initializations of NT register by loading NTR value to it and loading ZOLFlag with one. The ZOLFlag indicates that the process of the ZOL instruction is running during execution of the other instructions inside the loop. A sample code using ZOL instruction with its timing sequence is given in Figure 7.12.

There are no wasted cycles during the execution of ZOL loop as updating the PC with the proper values. When the value of the PC reaches EPC, this leads the CU to decrement

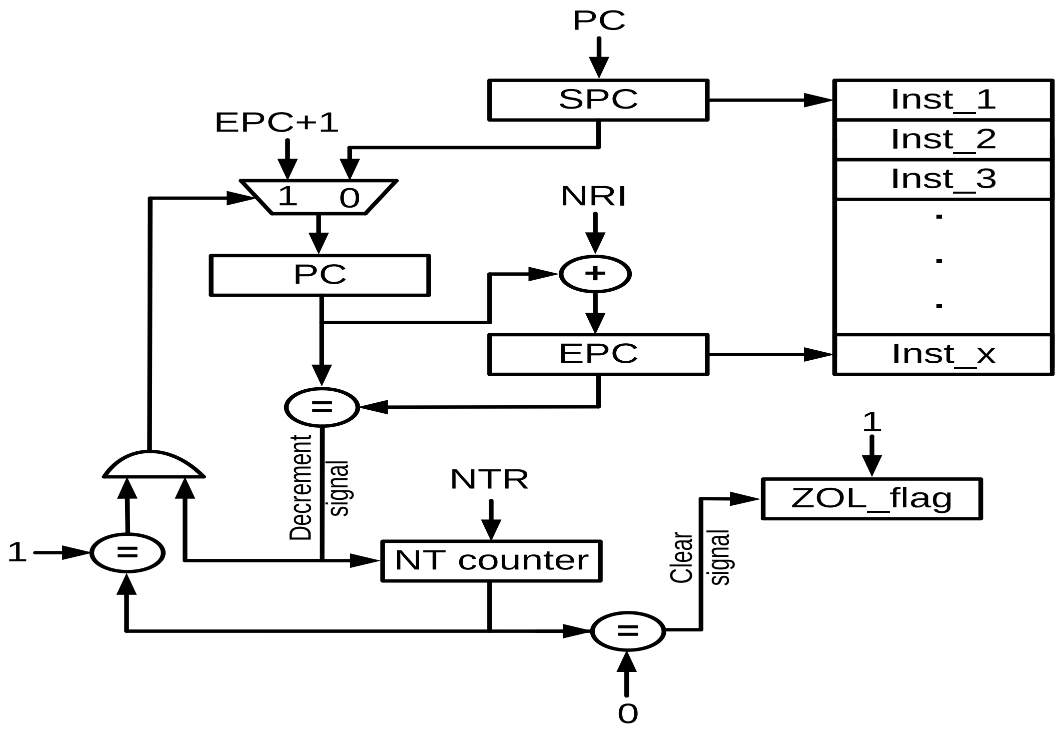


Figure 7.11 The block diagram of the ZOL instruction control parts including dedicated registers in details

the NT counter and update the PC either with SPC when NT is greater than one or with EPC+1 when NT is equals to one. As NT counter reaches to the Zero value this means the ZOL is finished and as a result ZOLFlag should be reset to zero.

All control statements of the ZOL instruction do not depend on any q except at the execution phase. This design includes three ZOL instructions: ZOL1, ZOL2 and ZOL3. Each ZOL instruction has its dedicated registers and its control parts. The nested looping is allowed. There is one restriction for using nested loop, using ZOL instruction within the loop of other ZOL, that the end of outer loop must be different of the inner loop.

In addition, the number of repeated time of the loop (NRT) depends on the value of NRI. In case of the NRI equals zero, the loop will be executed NTR+1 time. While in case of the NRI not equal to zero, the loop will be executed NTR time. For example:
$ZOL1\ 8,1$
this loop instruction contains two instructions and will be repeated eight times
$ZOL1\ 8,0$
this loop instruction contains one instruction and will be repeated nine times

7.1.6 LOOPNE Instruction

The formatting of the LOOPNE instruction is shown in Figure 7.13. The LOOPNE instruction executes a group of instructions as long as the EmptyFIFO signal is not equal to one as shown in Figure 7.14. The EmptyFIFO signal indicates the FIFO registers in the data alignment block is empty or not. As the Turbo decoder algorithm contains Two MAP decoders and each decoder exchange the information between them. Each MAP decoder cannot start its process if there is any data in FIFO registers. The LOOPNE takes one operand NRI. The LOOPNE instruction description is given in Table 7.7.

ZOL	Zero Overhead Loop
syntax:	*ZOL* NTR, NRI
Operands:	$0 \le NTR \le 32767, 0 \le NRI \le 15$
operation:	$q_1T_2 : SPC \leftarrow PC , EPC \leftarrow PC+NRI, NT \leftarrow NTR , ZOLFlag \leftarrow 1$ $(NRI == 0)q_1T_2 : PC \leftarrow PC$ $(NRI \neq 0)q_1T_2 : PC \leftarrow PC+1$ $(ZOLFlag == 1)(PC == EPC)T_2 : PC \leftarrow SPC, NT \leftarrow NT-1$ $(ZOLFlag == 1)(PC \neq EPC)T_2 : PC \leftarrow PC+1$ $(ZOLFlag == 1)(NT \neq 0)(PC == EPC)T_2 :PC \leftarrow SPC, NT \leftarrow NT-1$ $(ZOLFlag == 1)(NT \neq 0)(PC \neq EPC)T_2 :PC \leftarrow PC+1$ $(ZOLFlag == 1)(NT == 1)(PC == EPC)T_2 :PC \leftarrow EPC+1$ $(ZOLFlag == 1)(NT == 0)(PC == EPC)T_2 :PC \leftarrow PC+1 ,$ $ZOLFlag \leftarrow 0$
Description:	Conditional loop as group of instructions is executed number of times

Table 7.6 The ZOL Instruction description

LOOPNE	LOOP Not Empty
syntax:	*LOOPNE* NRI
Operands:	$0 \le NRI \le 15$
operation:	$q_{10}T_2 : SPC4 \leftarrow PC , EPC4 \leftarrow PC+NRI, LNEFlag \leftarrow 1$ $(NRI == 0)q_{10}T_2 : PC \leftarrow PC$ $(NRI \neq 0)q_{10}T_2 : PC \leftarrow PC+1$ $(LNEFlag == 1)(EmptyFIFO \neq 1)(PC == EPC4)T_2 : PC \leftarrow SPC4$ $(LNEFlag == 1)(EmptyFIFO \neq 1)(PC \neq EPC4)T_2 : PC \leftarrow PC+1$ $(LNEFlag == 1)(EmptyFIFO == 1)T_2 : PC \leftarrow PC+1 , LNEFlag \leftarrow 0$
Description:	Conditional LOOP as the EmptyFIFO signal is not eual to one execute the body of the loop

Table 7.7 The LOOPNE Instruction description

Similar mechanism for ZOL happens with LOOPNE instruction. In order to avoid the wasted cycles due to pipeline process, a number of dedicated registers is used such as SPC4, EPC4 and LNEFlag. The function of SPC4 is to point to the first instruction of the body of the loop while EPC4 points to the last instruction inside the loop. The initialization of SPC4 with the current value of PC and the initialization of EPC4 with the value of adding PC with NRI are done at the execution phase of LOOPNE instruction. Besides, the initialization of LNEFlag with one happens at the execution phase of LOOPNE instruction. The LNEFlag indicates that the process of the LOOPNE instruction is running

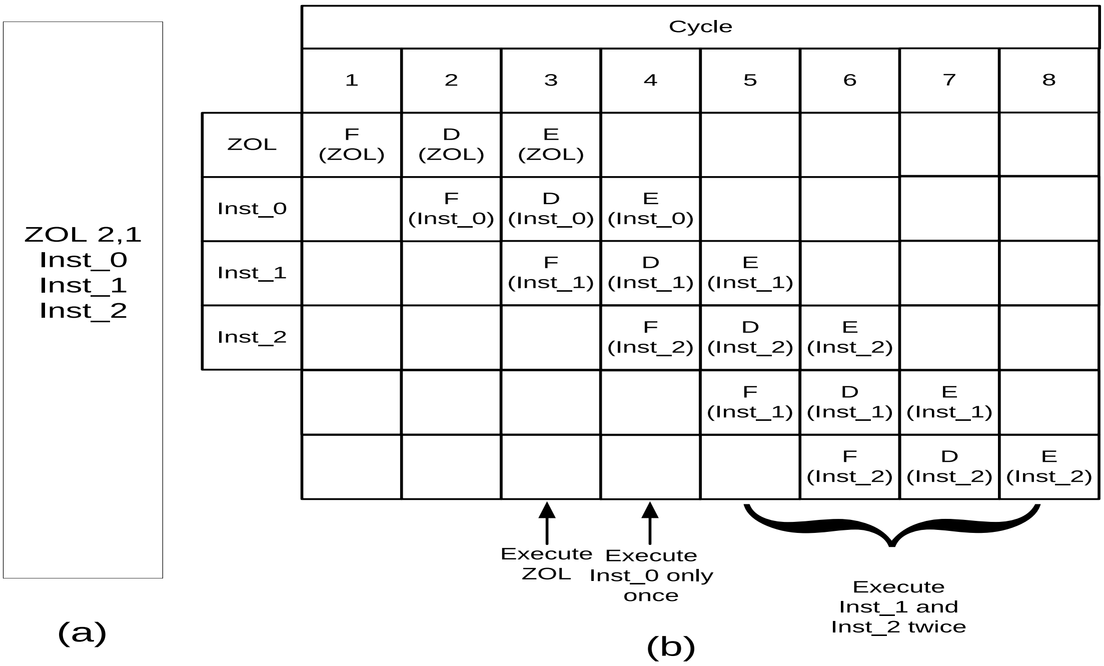


Figure 7.12 A sample code using ZOL instruction is given in (a), the timing sequence of the executions of that sample is given in (b)

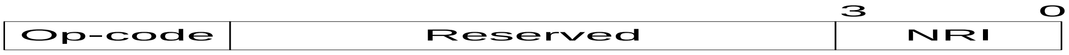


Figure 7.13 The formatting of the LOOPNE instruction

during execution of the other instructions inside the loop. The loop will be executed as long as the EmptyFIFO value is not equal to one. As the EmptyFIFO value becomes one this means the LOOPNE is finished and as a result LNEFlag should be reset to zero.

7.1.7 ParSISO Instruction

The formatting of the ParSISO instruction is shown in Figure 7.15. The ParSISO instruction description is given in Table 7.8.

ParSISO	Parallel SISO
syntax:	*ParSISO* BorderCntr, LLRCntr, FrSISOActive, ForwardCntr, BkSISOActive, BackwardCntr, PMode, AddGen
operation:	all transfers and control statements of that instruction happen at (q_5T_2)
Description:	The ParSISO instruction controls the execution of the different operations in multi-SISO units

Table 7.8 The ParSISO Instruction description

The detailed description for the operands of the ParSISO instruction is:

BorderCntr: this field is responsible for generating the enable signals to transfer the stored border values from border memories to state metrics units.

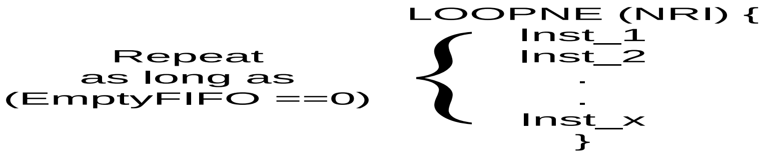
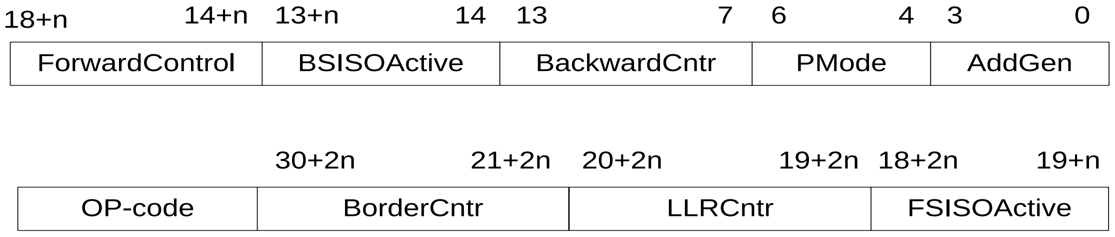


Figure 7.14 The description of how the LOOPNE instruction works



n= Log_2(Number of SISOs)

Figure 7.15 The formatting of the ParSISO instruction

LLRCntr: this field is responsible for generating the enable signals for LLR units.
FrSISOActive: Determine the number of active units that work in forward direction.
ForwardCntr: this field is responsible for generating the enable signals for forward units such as forward branch and state metrics.
BkSISOActive: Determine the number of active units that work in backward direction.
BackwardCntr: this field is responsible for generating the enable signals for backward units such as backward branch and state metrics.
PMode: Determine the phases of the operation as the Duo-binary type works in two different phases while the single binary works in single phase operation.
AddGen : Responsible for the read operation from channel memories and the read/write address generation for the State Metric memory.

The parallel SISOs are used to fulfill the high throughput demand for 4G Wireless standards. These SISO work together and according to the parallel window algorithm, each SISO contains address generation, branch metric, state metric and LLR units. These SISO work in the forward and backward directions. However, the number of SISOs work in the forward and backward direction is not the same number. The SimDecoder unit is used to control in the number of active units for each SISO. The block diagram of SimDecoder unit is shown in Figure 7.16.

The operation of the SimDecoder is similar to the binary decoder where the control inputs are the enable signal and Ind signal as shown in Figure 7.17 . When both the enable and Ind signals are ’1’, only one output signal of the SimDecoder is set corresponding to the value of the input and the other output signals are zeros. While in case of the enable signal equals to ’1’ and Ind signal equals to ’0’, group of output signal are set to ones from the index zero till the index corresponding to the input value. In addition to the output signals equal to all zeros when the enable signal equals to ’0’.

A number of SimDecoder units is used to control the parallel SISOs operations. There are SimDecoder units for forward/backward branch metric, forward/backward state metric, initialization for the state metric and LLR units. The control signals for these units are worked with ParSISO instruction. The input of SimDecoder units that control these units which work in the forward direction in each SISO is the FrSISOActive field. While the input of SimDecoder units that control in the units which work in the backward direction in each SISO is the BkSISOActive field. The control signals of the SimDecoder units, the enable signals and the Ind signals, that work in forward and backward directions are taken from the ForwardCntr and BackwardCntr fields of the ParSISO instruction respectively.

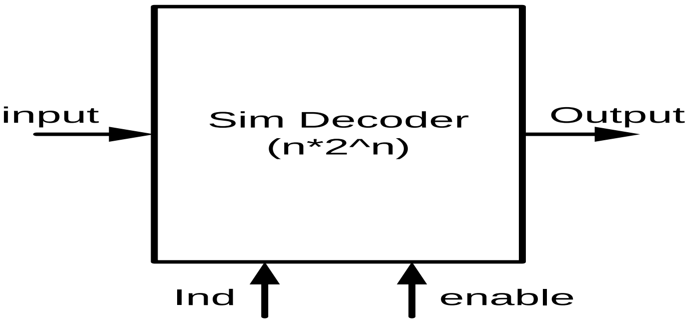


Figure 7.16 The block diagram of SimDecoder unit

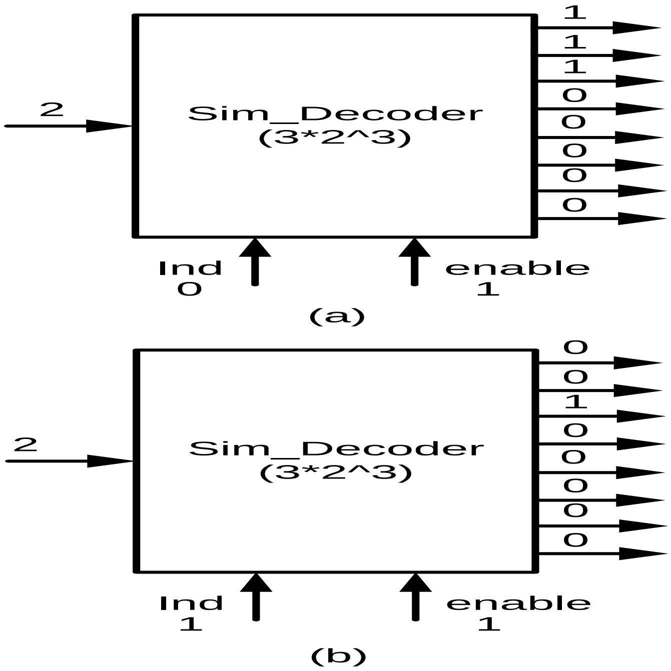


Figure 7.17 An example showing the two different modes for SimDecoder unit

7.1.8 Initialize Instruction

The formatting of the Initialize instruction is shown in Figure 7.18. The Initialize instruction description is given in Table 7.9 .

The detailed description for the operands of the Initialize instruction is:
AddInit4: for clearing the address register of the border memory.
StRAB: to interleave the systematic input to obtain the interleaved version of the systematic input.

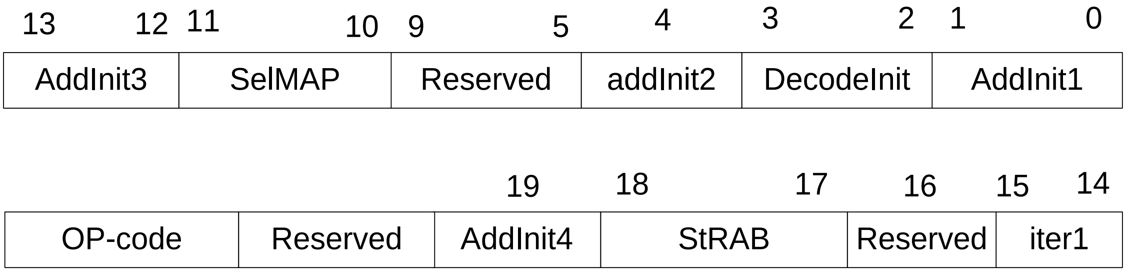


Figure 7.18 The formatting of the Initialize instruction

Initialize	Initialization
syntax:	*Initialize* AddInit4, StRAB , iter1 ,AddInit3 , SelMAP, addInit2, DecodeInit , AddInit1
operation:	all transfers and control statements of that instruction happen at $(q_4 T_2)$
Description:	The Initialize instruction initializes the registers and units of the different blocks in multi-SISO units

Table 7.9 The Initialize Instruction description

iter1: this bit should be set in the first half iteration operation. According to the decoding process, the LLR values from previous iteration should be used as this first half iteration. There is no initial values for the LLRs. After setting this bit the LLR values will be equal to zeros.
AddInit3: for clearing the address register of the LLR memory.
SelMAP: responsible for choosing one of two MAP decoders to execute the decoding process on it. Our design builds one MAP decoder and selects between two types of the input which are interleaved and deinterleaved inputs.
addInit2: select the input to beta address memory register to be either increase from zero to all ones or decrease from all ones to zeros.
DecodeInit: determine the mode of operation either in single binary or duo0bunary modes as to be used by other instruction.
AddInit1: select the input to channel data address memory register to be either increase from zero to all ones or decrease from all ones to zeros.

7.1.9 Mov Instruction

The formatting of the Mov instruction is shown in Figure 7.19. The Mov instruction description is given in Table 7.10 .

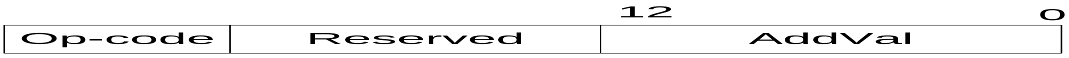


Figure 7.19 The formatting of the Move instruction

Mov	Move
syntax:	*Move* AddVal
Operands:	$0 \leq AddVal \leq 8191$
operation:	$q_8T_2 : AddReg \leftarrow AddVal$
Description:	to load address register of the channel data memory with AddVal

Table 7.10 The Mov Instruction description

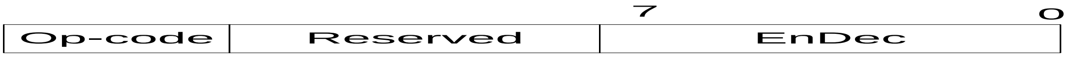


Figure 7.20 The formatting of the StrData instruction

7.1.10 StrData Instruction

The formatting of the StrData instruction is shown in Figure 7.20. The StrData instruction description is given in Table 7.11 .

The EnDecoderRx/Int is the write enable signal for the channel data memory x/Int, where x refers to one of the channel memory data such as A, B, Y, W, AInt, BInt, YInt and WInt. In each execution of that instruction, there is an increment process for the channel memory address register. The channel memory address register should be initialized to zero before using that instruction to point to the first memory location.

StrData	store channel Data
syntax:	*StrData* EnDec
Operands:	$0 \leq EnDec \leq 255$
operation:	$q_2T_2EnDec[0] : EnDecoderRA \leftarrow 1$
	$q_2T_2EnDec[1] : EnDecoderRB \leftarrow 1$
	$q_2T_2EnDec[2] : EnDecoderRY \leftarrow 1$
	$q_2T_2EnDec[3] : EnDecoderRW \leftarrow 1$
	$q_2T_2EnDec[4] : EnDecoderRAInt \leftarrow 1$
	$q_2T_2EnDec[5] : EnDecoderRBInt \leftarrow 1$
	$q_2T_2EnDec[6] : EnDecoderRYInt \leftarrow 1$
	$q_2T_2EnDec[7] : EnDecoderRWInt \leftarrow 1$
Description:	to store channel values to channel mamories

Table 7.11 The StrData Instruction description

7.1.11 Decode Instruction

The formatting of the Decode instruction is shown in Figure 7.21. The Decode instruction description is given in Table 7.12 .

When the DecodePhase signal becomes one, this indicates the valid output decoded is generated. The calculation of the decoded output is different for both types of the Turbo codes as discussed earlier. This instruction generates the decoded output depends on the initialize value of DecodeInit in the Initialize instruction.

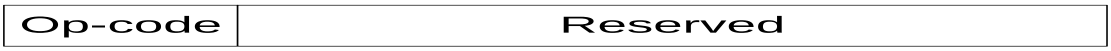


Figure 7.21 The formatting of the Decode instruction

Decode	Decode
syntax:	*Decode*
Operands:	None
operation:	$q_{12}T_2$: $DecodePhase \leftarrow 1$
Description:	to generate the Decoded output of the recieved data values

Table 7.12 The Decode Instruction description

7.1.12 Config Instruction

The formatting of the Config instruction is shown in Figure 7.22. The Config instruction description is given in Table 7.13 .

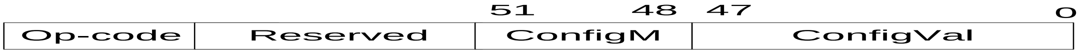


Figure 7.22 The formatting of the Config instruction

Config	Decode
syntax:	*Decode* ConfigM, ConfigVal
operation:	All transfers happen at : $q_{13}T_2$
Description:	To configure the branch metric, state metric and LLR unit with configuration words

Table 7.13 The Config Instruction description

The configuration words depend on the encoder architecture so it's constant through the decoding process. The ConfigM field determines to what unit the ConfigVal field goes.

Chapter 8

Results and Future Works

In this chapter the results of our design and the comparisons with other works are presented. In addition, the future works and conclusion are also presented.

Our design is fully scalable and all the width of input, output and internal signals are parameterized. In this design, our estimated power and area are without including the memory blocks, while the memory peripherals such as address decoders, multiplexers, and any gates related to the memory design are taken into account.

8.1 Varying of the Window Sizes & the Parallel SISOs

Table 8.1 shows the results of the power, area and the throughput due to varying the window size. Choosing large window size leads to bad impact on the throughput as shown in Figure 8.1. However, increasing the window size enhances the decoding performance.

In addition, the increase of the window size leads to large buffer requirement to handle the memory conflicts as shown in Figure 8.2.

Table 8.1 Effect of Window Size on Power, area and throughput for p=16 & p=1 SISO

	P=16 with address =15 bits			P=1 with address =13 bits		
W	Power (mW)	Area (mm^2)	Rb (Mbps)	Power (mW)	Area (mm^2)	Rb (Mbps)
32	298.4	8.681	184.6	31.8	1.0	12.42
64	236.9	7.467	171.43	22.8	0.71	12.35
128	211.7	6.749	150	18.4	0.56	12.23
256	198.8	6.462	100	16.2	0.49	11.98
512	191.2	6.246	75	15	0.45	11.52

8.2 Comparisons

The proposed implementation parameters and the memory sizes are shown in Tables 8.2 and 8.3 respectively. There are many works for Turbo decoders in different technologies. Thanks to Equation (8.1), we are able to make the comparisons between different technologies and different parameters.

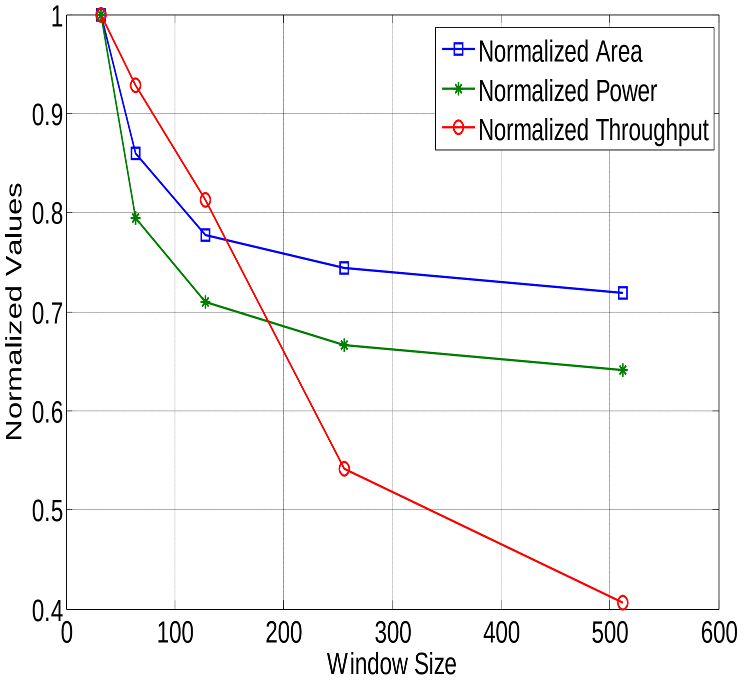


Figure 8.1 Effect of varying Window size for single binary codes with P=16, K=6144, on Area, Power and Throughput

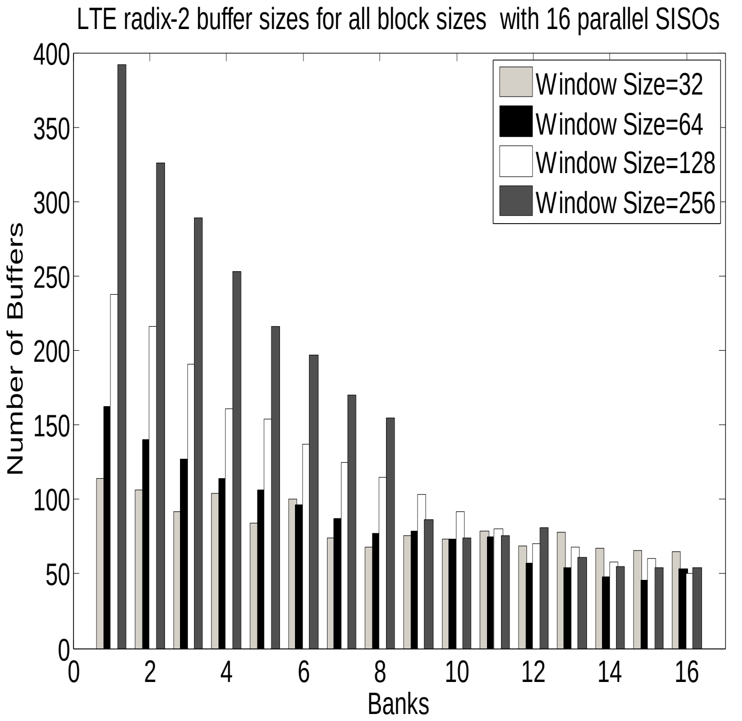


Figure 8.2 Effect of varying window sizes on buffer sizes with W=64, P=16, for all block sizes of LTE standard

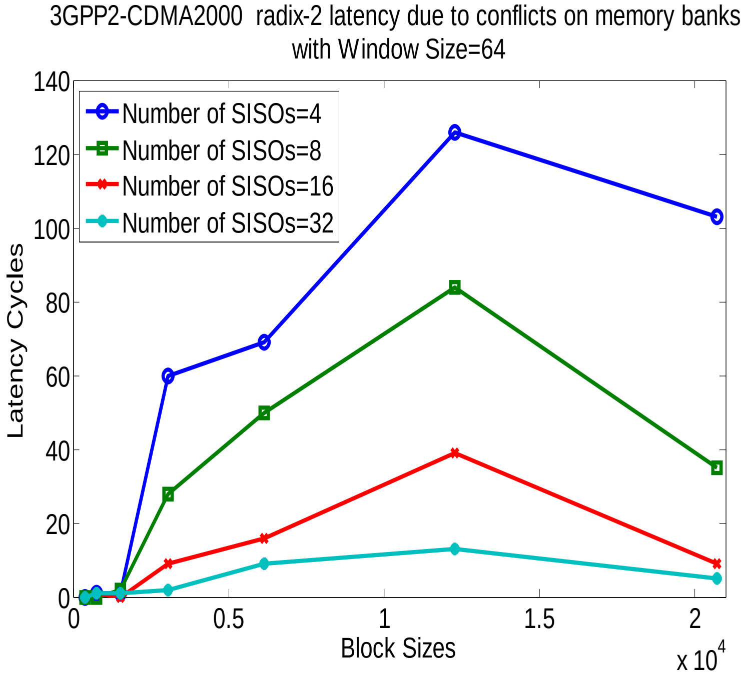


Figure 8.3 Effect of varying number of parallel SISOs on the latency between the two decoders due to memory conflicts with W=64 and for all block sizes of 3GPP2 CDMA2000 standard

$$\text{RPDP} = \left(\frac{Pr_1}{Pr_2}\right)\left(\frac{180}{\text{Tech}}\right)^2\left(\frac{100}{\text{Freq}}\right)\left(\frac{1.8}{\text{V}}\right)^2\left(\frac{\tau_1}{\tau_2}\right) \tag{8.1}$$

Where RPDP stands for relative power delay product, Pr is the power, the technology Tech is in nm , the frequency Freq is in MHz , the voltage V is in $Volt$. and τ is the time for decoding block length with K bits for one iteration. The subscript index “2” refers to our work and “1” refers to the other work. The throughput Rb is calculated from Equation (8.2) and then τ is calculated from Equation (8.3) :

$$Rb = \frac{K * f_{\text{clk}}}{2 * i * (\lceil \frac{K}{F*P*W} \rceil + 1) * W} \tag{8.2}$$

where f_{clk} is system clock, i is the number of iterations, F is a factor equal to 1 for radix-2 and 2 for radix-4.

$$\tau = \frac{K}{Rb * i} \tag{8.3}$$

Table 8.4 shows different works and compares the matric RPDP for each work. When the RPDP has a smaller value this means it has better energy efficency. Hence only the designes of [37], [38], [39], and [40] are competiting with our proposal.

The parallel SISO decoder in [37] has the smallest RPDP amongst the different designs. The proposed design in [37] supports very small packet sizes compared to the other works which leads to have very small power and area results. However, such work cannot support large block sizes which are required in the current and future standards.

The work in [39] supports only the WiMAX standard and can process the small packet sizes up to only 480 bits.

Table 8.2 Turbo Decoder Parameters

Decoding algorithm	Max-log-MAP
Quantization of channel values	4 bit
Quantization of LLRs	6 bit
Quantization of State Metric	8 bit
Window Size	64
Maximum Block Length	6144
number of iteration	4
running Clock Frequency	100 MHz
Maximum Clock Frequency	150 MHz
Technology	TSMC 180 nm CMOS technology
voltage	1.8 v
Number of Parallel SISOs	16
Power	236.9 mW
Area	7.467 mm^2
Number of Gates	4.278K

Table 8.3 Memory Sizes

Memory	Dimension	Amount	type
Interleaver	(6144*13)	2	single port
Channel values	(6144*4)	6	single port
LLR	(6144*6)	6	single port
Border	(W*8*8)	4*P	single port
State Metric	(64*8*8)	1*P	simple dual port
Program Memory	(512*(50+ $2log_2P$))	1	single port

The authors in [37] [41] introduced interleavers that allow contention-free property in the hybrid parallelism. Such interleavers are none-standard compliant and are not suitable for wireless communications standards that include none-collision-free interleavers. In addition, these designs include dedicated architectures for certain decoders.

Most works that are dedicated for a certain scheme, are none-standard compliant and have smaller RPDP compared to reconfigurable architectures such as in [38] [42] .

The ASIP design in [53] runs for the turbo decoder of the LTE standard. This design gives the highest RPDP value amongst different designs and produces 140Mbps throughput at only one iteration. One iteration is not enough for producing good decoding performance.

Compared to previous systems, our design meets the design requirements such as the high-through and low energy demands.

8.3 Future Works

One of the main future works is to build parallel interleaver address generators which will reduce the amount of the memories that used to store the interleaved addresses in the

Table 8.4 Comparison with existing Turbo decoder architectures

Ref.	gates Number	Area (mm^2)	Pr (mW)	Energy ($nJ/bit/iter$)	Pkt Size	Tech. (nm)	Volt. (V)	iter.	Freq. (MHz)	Rb ($Mbps$)	RPDP
[41]	2.67 M	17.81	275	0.22	4096	130	1.32	8	80	160	1.8473
[11]	410 K	14.5	1450	10	5114	180	1.8	6	145	24	16.7297
[37]	409 K	7.16	-	2.19	384	180	1.8	4.43	160	71.7	0.2476
[42]	-	10	2464	1.4	2048	130	1.2	5	352	352	1.6552
[13]	800 K	10.7	-	0.61	6144	130	1.2	8	250	187.5	3.0463
[43]	64.2 K	2.24	-	0.63	4800	130	1.2	8	200	24.3	3.0725
[44]	44.1 K	1.2	-	0.7	5114	130	1.2	6	246	18.6	2.9571
[45]	34.4 K	8.897	-	6.98	5114	250	2.5	6	100	5.48	4.5190
[46]	-	-	35	-	5114	180	1.8	10	50	2	8.4316
[47]	324 K	9	306	-	5114	180	1.8	10	93	2	39.6336
[48]	44.1 K	1.2	61.5	-	5114	130	1.2	5.5	246	10.8	4.3737
[49]	553 K	1.2	61.5	-	6144	130	1.2	5.5	302	390.6	1.5181
[9]	-	2.1	219	0.21	6144	90	1	8	275	129	2.728
[8]	-	2.1	300	-	6144	65	1.1	6.5	300	150	5.9761
[34]	-	16	650	-	5114	180	1.8	6	166	60	2.6203
[50]	-	3.8	-	-	5114	130	-	6	500	308	-
[38]	-	13.1	573	0.126	5120	130	1.2	6	256	758	0.512
[51]	-	0.4	230	0.315	6144	45	LV	6	333	100	$(14.42/(LV^2))$
[25]	-	-	800	-	5114	180	1.8	5	400	2.08	46.329
[52]	-	1.46	452	-	6144	40	1.1	6.5	400	350	10.3937
[53]	-	10.37	570	-	6144	65	1.2	1	320	140	63.5399
[54]	-	8.7	330	2.36	5000	180	1.8	5	285	27.6	1.9763
[55]	602 K	2.1	219	0.21	6144	90	1	8	275	130	2.8723
[56]	14.4 K	6.38	762	-	128 K	90	-	-	-	131.28	-
[39]	635 K	7.16	197.3	0.43	480	130	1.2	4	100	115.4	0.4169
[40]	-	5	265	-	5120	90	LV	5	200	930	$(0.8605/(LV^2))$
[57]	-	9.61	1356	0.12	4096	90	0.9	8	175	1400	2.135
[58]	70 K	0.7	650	0.7	6144	65	1.2	6	250	152	14.2369
[59]	-	0.62	76.8	-	6144	65	LV	5	400	18.6	$(14.8357/(LV^2))$
[21]	-	-	100	-	6144	65	1.1	5	400	17	17.48
Proposed	4.278 K	7.467	236.9	0.3452	6144	180	1.8	4	100	171.43	1

current design.

Other reconfigurable blocks should be added to support more channel codes such as LDPC codes. As the LDPC codes and Turbo codes are shared with the need of a large number of the memory blocks and some of the calculation units.

8.4 Conclusion

In this thesis, an efficient architecture is proposed to implement a scalable low-power configurable processor capable of supporting a multi-standard turbo decoder. Our ASIP offers high flexibility while maintaining the hardware requirements such as the power consumption, area and the throughput. A good technique is used to decrease the effects of contentions on the memory access besides the reduction of the hardware overheads.

Three configurations of the state metric unit are proposed. We made a comparison between these configurations. Each configuration has its features according the target architecture. The conflicts due to parallel decoding have bad effects on the design. We showed the effects on the throughput, the area, and the hardware utilizations of the different schemes.

The effects of the design parameters on the performance are included such as the window sizes and the number of parallel SISOs.

References

[1] A. Glavieux, C. Berrou, and P. Thitimasjshima., “Near shannon limit error-correcting coding and decoding: Turbo-codes,” In *IProc. IEEE Int. Conf. on Commun. (Geneva, Switzerland)*, pp. 1064Ű–1070 (1993).

[2] *Digital Video Broadcasting (DVB): Interaction cahnnel for satellite distribution systems*, European Telecommunications Standards Institute (ETSI), 2000.

[3] *Multiplexing and channel coding (FDD) (25.212 V8.4.0)*, 3GPP Technical Specification Group Radio Access Network - Evolved Universal Terrestrial Radio Access E-UTRA, 12-2008.

[4] *Multiplexing and channel coding (Release 8), 3GPP TS 36.212 v8.0.0*, 3GPP Technical Specification Group Radio Access Network - Evolved Universal Terrestrial Radio Access E-UTRA, 09-2007.

[5] *IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access SystemsŮAmendment 2: Medium Access Control Layers for Combined Fixed and Mobile Operations in Licensed Bands*, IEEE 802.16eŰ2005.

[6] *Physical Layer Standard for CDMA2000 Spread Spectrum Systems, C.S0002-C v2.0*, 3GPP2, 7-2004.

[7] J. Janhunen, O. Silvén, and M. Juntti, “COMPARISON OF THE SOFTWARE DEFINED RADIO IMPLEMENTATIONS OF THE K-BEST LIST SPHERE DETECTION,” In *European Signal Processing Conference (EUSIPCO 2009)*, pp. 2396Ű–2400 (2009).

[8] M. May, T. Ilnseher, N. Wehn, and W. Raab, “A 150Mbit/s 3GPP LTE Turbo Code Decoder,” In *Design, Automation and Test in Europe Conf. and Exhibition (DATE)*, pp. 1420–1425 (2010).

[9] C.-C. Wong, Y.-Y. Lee, and H.-C. Chang, “A 188-size 2.1mm^2 Reconfigurable Turbo Decoder Chip with Parallel Architecture for 3GPP LTE System,” In *Symp. VLSI circuits dig. tech. papers, Kyoto, Japan*, pp. 288–289 (2009).

[10] R. Asghar and D. Liu, “Towards Radix-4, Parallel Interleaver Design to Support High-Throughput Turbo Decoding for Re-Configurability,” In *33rd IEEE SARNOFF Symposium - 2010,Princeton, NJ, USA*,

[11] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, “A 24Mb/s Radix-4 LogMAP Turbo Decoder for 3GPP-HSDPA Mobile Wireless,” In *IEEE Int. Solid-State Circuits Conf.*, pp. 150Ű–151 (2003).

[12] O. Y. Takeshita, "On Maximum Contention-Free Interleavers and Permutation Polynomials over Integer Rings," In *IEEE Transactions on Information Theory*, pp. 1420 –1425 (2005).

[13] J.-H. Kim and I.-C. Park, "A Unified Parallel Radix-4 Turbo Decoder for Mobile WiMAX and 3GPP-LTE," In *IEEE Custom Intergrated Circuits Conf., San Jose, CA, USA*, pp. 487–490 (2009).

[14] Y. Sun, Y. Zhu, M. Goel, and J. R. Cavallaro, "Configurable and Scalable High Throughput Turbo Decoder Architecture for Multiple 4G Wireless Standards," In *IEEE International Conference on Application-specific System, Architectures and Processors (ASAP'08)*, pp. 209–214 (2008).

[15] A. Tarable, G. Montorsi, and S. Benedetto, "Mapping of interleaving laws to parallel turbo decoder architectures," In *Proc. 3rd Int. Symp. Turbo Codes Related Topics, Brest, France*, pp. 153–156 (2003).

[16] M. J. Thul, F. Gilbert, and N. Wehn, "Concurrent interleaving architectures for high-throughput channel coding," In *Proc. 2003 IEEE Int. Conf. Acoustics, Speech and Signal Processing, Hong Kong*,

[17] M. J. Thul, F. Gilbert, and N. Wehn, "Optimized concurrent interleaving architecture for high-throughput turbo-decoding," In *Proc. 9th Int. Conf. Electron, Circuits, Syst., vol.3*,

[18] F. Speziali and J. Zory, "Scalable and area efficient concurrent interleaver for high throughput turbo-decoders," In *Proceedings of Euromicro DSD-2004*,

[19] R. Asghar, D. Wu, J. Eilert, and D. Liu, "Memory Conflict Analysis and Implementation of a Re-configurable Interleaver Architecture Supporting Unified Parallel Turbo Decoding," Journal of Signal Processing Systems, Springer **60,** 15–29 (2010).

[20] M. Martina, M. Nicola, and G. Masera, "A Flexible UMTS-WiMax Turbo Decoder Architecture," IEEE Transactions on Circuits and Systems-II: Express Briefs **55,** 369–373 (2008).

[21] T. Vogt and N. Wehn, "A Reconfigurable ASIP for Convolutional and Turbo Decoding in an SDR Environment," IEEE Trans. Very Large Scale Integration (VLSI) Syst. **16,** 1309–1320 (2008).

[22] T. Vogt, C. Neeb, and N. Wehn, "A Reconfigurable Multi-Processor Platform for Convolutional and Turbo Decoding," In *In Proc. Reconfigurable Communication-centric SoCs*, pp. 16–23 (2006).

[23] O. Muller, A. Baghdadi, and M. Jézéquel, "From Parallelism Levels to a Multi-ASIP Architecture for Turbo Decoding," IEEE Transactions on Very Large Scale Integration (VLSI) Syst. **17,** 92Ű–102 (2009).

[24] F. Gilbert, M. J. Thul, and N. Wehn, "Communication centric architectures for turbo-decoding on embedded multiprocessors," In *Design, Automation and Test in Europe Conf. and Exhibition (DATE)*, pp. 356–361 (2003).

[25] Y. Lin, S. Mahlke, T. Mudge, C. Chakrabarti, A. Reid, and K. t'an Flautner, "Design and Implementation of Turbo Decoders for Software Defined Radio," In *Proc. IEEE Workshop on Signal Processing Systems Design and Implementation SIPS 2006*, pp. 22–26 (2006).

[26] M. C. Valenti and J. Sun, "The UMTS Turbo Code and an Efficient Decoder Implementation Suitable for Software-Defined Radios," International Journal of Wireless Information Networks **8,** 203Ű–215 .

[27] C. Berrou, M. Jézéquel, C. Douillard, and S. Kerouédan, "The Advantages of Non-Binary Turbo Codes," Information Theory Workshop ITW2001 Cairns, Australia pp. 61–63 (2001).

[28] P. H. P. Robertson and E. Villebrun., "Optimal and suboptimal maximum a posteriori algorithms suitable for turbo decoding," European Trans. On Telecommun. pp. 119–125 (1997).

[29] M. B. et al, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," IEEE Int. Solid-State Circuit Conf. (ISSCC) pp. 39–54 (2003).

[30] Y. Zhang and K. Parhi, "High-throughput radix-4 logMAP turbo decoder architecture," Asilomar conf. on Signals, Syst. and Computers pp. 1711–1715 (2006).

[31] V. J and A. Finger, "Improving the Max-Log-MAP Turbo Decoder," IEEE contributions, IEEE C802.16m-07/080r3 pp. 1937–1939 (2000).

[32] E. M. Abdel-Hamid, H. A. H. Fahmy, M. M. Khairy, and A. F. Shalash, "Memory Conflict Analysis For A Multi-standard, Reconfigurable Turbo Decoder," In *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS 2011)*, pp. 2701 – 2704 (2011).

[33] M. Marandian, J. Fridman, Z. Zvonar, and M. Salehi, "Performance Analysis of Sliding Window Turbo Decoding Algorithms for 3GPP FDD Mode," International Journal of Wireless Information Networks **9,** 39–54 (2002).

[34] M. J. Thul, F. Gilbert, T. Vogt, G. Kreiselmaier, and N. Wehn, "A SCALABLE SYSTEM ARCHITECTURE FOR HIGH-THROUGHPUT TURBO-DECODERS," Journal of VLSI Signal Processing Systems (Special Issue on Signal Processing for Broadband Communications) **39,** 63–77 (2005), springer Science and Business Media, Netherlands.

[35] S. C. Cho, J. U. Kim, J. S. Cha, and K.-R. Cho, "Performance Evaluation of Convolutional Turbo Codes in AWGN and ITU-R Channels," ICIC (2) pp. 695–703 (2005).

[36] R. Srinivasan, J. Zhuang, L. Jalloul, R. Novak, and J. Park, "Draft IEEE 802.16m Evaluation Methodology Document," IEEE contributions, IEEE C802.16m-07/080r3 (2007).

[37] B. Bougard, A. Giulietti, V. Derudder, J.-W. Weijers, S. Dupont, L. Hollevoet, F. Catthoor, L. V. der Perre, H. D. Man, and R. Lauwereins, "A Scalable 8.7nJ/bit 75.6Mb/s Parallel Concatenated Convolutional (Turbo-) CODEC," In *IEEE Int. Solid-State Circuits Conf.*,

[38] G. Prescher, T. Gemmeke, and T. G. Noll, “A PARAMETRIZABLE LOW-POWER HIGH-THROUGHPUT TURBO-DECODER,” In *Proc. 2005 IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP 2005), Philadelphia, Pennsylvania, USA*, pp. 25–28 (2005).

[39] C.-H. Lin, C.-Y. Chen, A.-Y. A. Wu, and T.-H. Tsai, “Low-Power Memory-Reduced Traceback MAP Decoding for Double-Binary Convolutional Turbo Decoder,” IEEE Tran. on Circuits and Systems I: Regular Papers **56,** 1005–1016 (2009).

[40] S. M. Karim and I. Chakrabarti, “An Improved Low-Power High-Throughput Log-MAP Turbo Decoder,” IEEE Tran. on Consumer Electronics **56,** 450–457 (2010).

[41] C.-C. Wong, M.-W. Lai, C.-C. Lin, H.-C. Chang, and C.-Y. Lee, “Turbo Decoder Using Contention-Free Interleaver and Parallel Architecture,” IEEE Journal of Solid-State Circuits **45,** 422–432 (2010).

[42] P. Urard, L. Paumier, M. Viollet, E. Lantreibecq, H. Michel, S. Muroor, B. Coates, and B. Gupta, “A Generic 350Mb/s Turbo-Codec Based on a 16-states SISO Decoder,” In *IEEE Int. Solid-State Circuits Conf.*, pp. 424–536 (2004).

[43] J.-H. Kim and I.-C. Park, “A 50Mbps Double-Binary Turbo Decoder for WiMAX Based on Bit-level Extrinsic Information Exchange,” In *IEEE Asian Solid-State Circuits Conf.*, pp. 305–308 (2008).

[44] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, “A 58mW 1.2mm^2 HSDPA Turbo Decoder ASIC in 0.13μm CMOS,” In *IEEE Int. Solid-State Circuits Conf.*, pp. 264–265 (2008).

[45] M.-C. Shin and I.-C. Park, “SIMD Processor-Based Turbo Decoder Supporting Multiple Third-Generation Wireless Standards,” IEEE Trans. Very Large Scale Integration (VLSI) Syst. **15,** 801–810 (2007).

[46] M. Bekooij, J. Dielissen, F. Harmsze, S. Sawitzki, J. Huisken, A. van der Werf, and J. van Meerbergen, “Power-Efficient Application-Specific VLIW Pocessor for Turbo Decoding,” In *IEEE Int. Solid-State Circuits Conf.*, pp. 180–181 (2001).

[47] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, BenjaminWiddup, G. Zhou, L. M. Davis, GraemeWoodward, C. Nicol, and R.-H. Yan, “A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18-μm CMOS,” IEEE Journal of Solid-State Circuits **37,** 1555–1564 (2002).

[48] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, “Design and Optimization of an HSDPA Turbo Decoder ASIC,” IEEE Journal of Solid-State Circuits **44,** 98–106 (2009).

[49] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, “Design and Implementation of a Parallel Turbo-Decoder ASIC for 3GPP-LTE,” IEEE Journal of Solid-State Circuits **46,** 8–17 (2011).

[50] M. May, C. Neeb, and N. Wehn, “Evaluation of High Throughput Turbo-Decoder Architectures,” In *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS 2007),New Orleans, USA*, pp. 2770 – 2773 (2007).

[51] F. Naessens, B. Bougard, S. Bressinck, L. Hollevoet, P. Raghavan, L. V. der Perre, and F. Catthoor, “A UNIFIED INSTRUCTION SET PROGRAMMABLE ARCHITECTURE FOR MULTISTANDARD ADVANCED FORWARD ERROR CORRECTION,” In *Proc. IEEE Workshop on Signal Processing Systems SIPS 2008*, pp. 31–36 (2008).

[52] T. Ilnseher, M. May, and N. Wehn, “A Multi-Mode 3GPP-LTE/HSDPA Turbo Decoder,” In *Int. conf. on Communication Systems (ICCS)*, pp. 336 –340 (2010).

[53] F. Naessens *et al.*, “A 10.37 mm2 675 mW reconfigurable LDPC and Turbo encoder and decoder for 802.11n, 802.16e and 3GPP-LTE,” In *Proc. of Symposium on VLSI Circuits*, pp. 213–214 (2010).

[54] S.-J. Lee, N. R. Shanbhag, and A. C. Singer, “A 285-MHz Pipelined MAP Decoder in 0.18-μm CMOS,” IEEE Journal of Solid-State Circuits **40,** 1718–1725 (2005).

[55] C.-C. Wong and H.-C. Chang, “Reconfigurable Turbo Decoder With Parallel Architecture for 3GPP LTE System,” IEEE Tran. on Circuits and Systems II: Express Briefs **57,** 566–570 (2010).

[56] I. Ahmed and C. Vithanage, “Dynamic Reconfiguration Approach for High Speed Turbo Decoding using Circular Rings,” In *Proc.of the 19th ACM Great Lakes symposium on VLSI*, pp. 475–480 (2009).

[57] C.-H. Lin, C.-Y. Chen, A.-Y. A. Wu, and T.-H. Tsai, “High-Efficiency Processing Schedule for Parallel Turbo Decoders Using QPP Interleaver,” IEEE Tran. on Circuits and Systems I: Regular Papers **58,** 1412–1420 (2011).

[58] D. Wu, R. Asghar, Y. Huang, and D. Liu, “Implementation of A High-Speed Parallel Turbo Decoder for 3GPP LTE Terminals,” In *IEEE 8th Int. Conf. on ASIC ,2009*, pp. 481 – 484 (2009).

[59] M. Alles, T. Vogt, and N. Wehn, “FlexiChaP: A Reconfigurable ASIP for Convolutional, Turbo, and LDPC Code Decoding,” In *2008 5th Int. Symposium on Turbo Codes and Related Topics*, pp. 84 – 89 (2008).

الملخص

تقدم هذه الرسالة تصميم لتنفيذ فك الاكواد التوربينية القابلة للتطبيق باستخدام معالج متخصص منخفض الطاقة متضمنا تصميم لعدد من الوحدات التى تعمل على التوازى داخل المعالج لتناسب السرعات العالية التى تعد من احد اهم المتطلبات الجيل الرابع (4G) لنظم الاتصالات اللاسلكية. لقد تم تنفيذ هذه الوحدات التى تعمل على التواز ى لفك الاكواد من نوع Soft-In/Soft-Out (SISO). كما تم ربط هذه الوحدات بطريقة قابلة للتحجيم لتناسب المتطلبات المختلفة. ولقد تم عرض لثلاثة تصميمات لتنفيذ فك الاكواد التوربينية. كما تم عرض و مقارنة نتائج لهذه التصميمات من حيت الانتاجية ، المساحة، وكفاءة الوحدات المستخدمة.

الوحدات التى تعمل على التوازى تؤدى الى نزاعات خلال التعامل مع الذاكرة. لذلك تم عمل تحليل وعرض اثار هذه النزاعات لكافة انماط المبدلات المختلفة Interleavers على التصميم المعالج لنظم متعددة. مثل هذه النزاعات لها تاثير على الانتاجية و كفاءة التصميم بشكل مؤثر. تم تصميم وحدة تحكم بسيطة لادارة النزاعات على الذاكرة لحظيا.

التصميم المقترح على 180nm technology يعطى انتاجية 170Mbps ، وبطاقة 236.9mW باستخدام 16 SISOs التى تعمل على سرعة 100Mhz.

مهندس: عيد محمد عبدالحميد عبدالعظيم

تاريخ الميلاد: 1986\8\26

الجنسية: مصرى

تاريخ التسجيل: ....\....\..........

تاريخ المنح: ....\....\..........

القسم: هندسة الالكترونيات والاتصالات الكهربية

الدرجة: ماجستير

المشرفون:

د. احمد فاروق شلش

د. حسام على حسن فهمى

الممتحنون:

د. عمادالدين محمود حجازى (الممتحن الخارجي)

أ.د محمد محمد خيرى (الممتحن الداخلي)

د. احمد فاروق شلش (المشرف الرئيسي)

د. حسام على حسن فهمى (عضو)

عنوان الرسالة:

تصميم وبناء معالج متخصص لفك الأكواد التوربينية لنظم متعددة

الكلمات الدالة:

الاكواد التوربينية ، معالج متخصص، بناء متوازى ، نزاعات الذاكرة ، الانتاجية العالية

ملخص الرسالة:

يقدم هذه الرسالة تصميم لتنفيذ فك الاكواد التوربينية القابلة للتطبيق باستخدام معالج متخصص منخفض الطاقة متضمنا تصميم لعدد من الوحدات التى تعمل على التوازى داخل المعالج لتناسب السرعات العالية. لقد تم تنفيذ هذه الوحدات التى تعمل على التوازى لفك الاكواد من نوع Soft-In/Soft-Out (SISO). ولقد تم عرض لثلاثة تصميمات لتنفيذ فك الاكواد التوربينية. كما تم عرض و مقارنة نتائج لهذه التصميمات من حيت الانتاجية ، المساحة، وكفاءة الوحدات المستخدمة. الوحدات التى تعمل على التوازى تؤدى الى نزاعات خلال التعامل مع الذاكرة. لذلك تم عمل تحليل وعرض اثار هذه النزاعات لكافة انماط المبدلات المختلفة على التصميم المعالج لنظم متعددة.

تصميم وبناء معالج متخصص لفك الأكواد التوربينية لنظم متعددة

اعداد
عيد محمد عبدالحميد عبدالعظيم

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة الماجستير
في
هندسة الالكترونيات والاتصالات الكهربية

يعتمد من لجنة الممتحنين:

الدكتور: عمادالدين محمود حجازى	الممتحن الخارجي
الأستاذ الدكتور: محمد محمد خيرى	الممتحن الداخلي
الدكتور: احمد فاروق شلش	المشرف الرئيسى
الدكتور: حسام على حسن فهمى	عضو

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية

مايو– 2013

تصميم وبناء معالج متخصص لفك الأكواد التوربينية لنظم متعددة

اعداد
عيد محمد عبدالحميد عبدالعظيم

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة الماجستير
في
هندسة الالكترونيات والاتصالات الكهربية

تحت اشراف

احمد فاروق شلش
أستاذ مساعد بكلية الهندسة - جامعة القاهرة
……………………………..

حسام على حسن فهمى
أستاذ مساعد بكلية الهندسة - جامعة القاهرة
……………………………..

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية

2013

تصميم وبناء معالج متخصص لفك الأكواد التوربينية لنظم متعددة

اعداد
عيد محمد عبدالحميد عبدالعظيم

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة الماجستير
في
هندسة الالكترونيات والاتصالات الكهربية

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية

2013