University of Thessaly

Department of Electrical and Computer Engineering

Μελέτη και Υλοποίηση ενός Παράλληλου Αποκωδικοποιητή σε Επαναπρογραμματιζόμενη Λογική για δίκτυα $4^{ης}$ γενιάς

Study and Implementation of a Parallel Turbo-Decoder on FPGA for 3GPP-LTE

Diploma Thesis by

Tsiokanos Ioannis

Supervisors:

Georgios Stamoulis	Antonios Argyriou
Professor	Assistant Professor

Volos, July 2016

University of Thessaly
Department of Electrical and Computer Engineering

"Μελέτη και Υλοποίηση ενός Παράλληλου Αποκωδικοποιητή σε Επαναπρογραμματιζόμενη Λογική για δίκτυα $4^{ης}$ γενιάς"

"Study and Implementation of a Parallel Turbo-Decoder on FPGA for 3GPP-LTE"

By

Tsiokanos Ioannis

Graduate Thesis for the degree of
Diploma of Science in Computer and Communication Engineering

Approved by the two-member inquiry committee at 8^{th} of July

.....	.....
Dr. Georgios Stamoulis	Dr. Antonios Argyriou

Declaration of Authorship

I, Tsiokanos Ioannis, declare that this thesis titled, ‘Study and Implementation of a Parallel Turbo Decoder’ and the work presented in it are my own. The research was carried out wholly or mainly while in candidature for the graduate degree of Diploma of Science in Computer and Communication Engineering, at the University of Thessaly, Department of Electrical and Computer Engineering, Volos, Greece.

……

Tsiokanos Ioannis

To my family and my friends

Acknowledgements

Upon completion of my thesis, I would like to thank my supervisor Dr,Georgios Stamoulis and my co-supervisor Dr. Antonios Argyriou for their trust and excellent corporation we had during this thesis and my studies.

I would also like to thank my friends and cooperators at VLSI and EDA Tools Laboratory and especially Ph.D candidate Charalampos Antoniadis for their assistance and guidance on this work.

Finally, I have to thank my family for their endless and invaluable moral support that offered me all those academic years.

Tsiokanos Ioannis,

Volos 2016

Contents

List of Tables ..... **iv**
List of Figures ..... **v**
List of Acronyms ..... **vi**
Abstract ..... **vii**
1 Introduction **1**
1.1 Motivation **1**
1.2 Thesis goal **2**
1.3 Thesis structure **3**
2 Turbo-Decoding for LTE **5**
2.1 Intoduction **5**
2.2 Turbo-Decoding Algorithm **6**
2.3 Radix-4 Max log BCJR Algorithm **9**
2.4 LTE Interleaver **13**
3 Parallel Turbo-Decoder architecture **17**
3.1 High-Level Architecture **17**
3.2 Memory Architecture **19**
3.3 Implementation Tradeoffs **19**
4 LTE Interleaver Architecture **21**
4.1 Contention Free Interleaving for LTE **21**
4.2 Master-Slave Batcher Network **23**
5 Radix-4 Max-Log BCJR Architecture **25**
5.1 VLSI Architecture **25**
5.2 Radix-4 ACS Units with Modulo-Normalization **27**
5.3 LLR Computation Unit **28**
6 Implementation Results **29**
6.1 Axi-4 Stream Ip **29**
6.2 Verilog Implementation **31**
6.3 Error-Rate Performance and key characteristics **35**
7 Conclusion **37**
7.1 Future work **37**
Bibliography **39**

List of Tables

Table 2.1 Matlab Simulator Profiling for SISO Receiver
Table 2.2 Turbo codes Interleaver parameters (Part 1 of 2)
Table 2.3 Turbo codes Interleaver parameters (Part 2 of 2)

Table 6.1 Signals associated with slave interface
Table 6.2 Signals associated with master interface
Table 6.3 Construction of Tdata_s

List of Figures

Figure 1.1 Evolution of wireless standards in the last two decades

Figure 1.2 LTE SISO Processing Chain

Figure 2.1 Parallel-concatenated turbo-encoder and block diagram of a turbo-decoder.

Figure 2.1 Basic Trellis Diagram

Figure 2.2 Basic structure of an iterative Turbo decoder. Iterative decoding based on MAP decoders. Forward/backward recursions on the trellis diagram

Figure 2.3 Example of the calculation of the forward and backward state metrics for radix-2 recursions

Figure 2.4 Radix-2 and radix-4 recursions

Figure 3.1 High-level architecture of the parallel turbo-decoder

Figure 4.1 Architecture of the contention-free interleaver

Figure 4.2 The Master-Slave Batcher Network architecture

Figure 5.1 Architecture of the implemented radix-4 max-log BCJR core

Figure 5.2 Radix-2 and Radix-4 architectures

Figure 6.1 ASIC/FPGA Design process

Figure 6.2 Synthesis report (part 1 of 3)

Figure 6.3 Synthesis report (part 2 of 3)

Figure 6.4 Synthesis report (part 3 of 3)

List of Acronyms

LTE	Long Term Evolution
3GPP	Third Generation Partnership Project
SISO	Soft-input Soft-output
LLR	Log-Likelihood-Ratio
SD	SISO Decoder
CE	Convolutional Encoder
QPP	Quadratic Polynomial Permutation
ARP	Almost Regular Permutation
HSDPA	High-Speed Downlink Packet Access
$\boldsymbol{a_k(s)}$	forward state metric
$\boldsymbol{\beta_k(s)}$	backward state metric
$\boldsymbol{\gamma_k(s', s)}$	branch metric
$\boldsymbol{Lk^s}$	Systematic Log-Likelihood-Ration
$\boldsymbol{Lk^{p2}}$	Parity LLR from the second CE
$\boldsymbol{Lk^{p1}}$	Parity LLR from the first CE
$\boldsymbol{Lk^A}$	A-priori LLR
$\boldsymbol{Lk^D}$	Intrinsic LLR
$\boldsymbol{Lk^E}$	Extrinsic LLR
BCJR	Bahl, Cocke, Jelinek and Ravin
ACS	Add-Compare-Select
OFDM	Orthogonal Frequency Division Multiplexing
AWGN	Additive White Gaussian Noise
FFT	Fast Fourier Transform
HDL	Hardware Description Language
AMBA	Advanced Microcontroller Bus Architecture

FPGA	Field- Programming Gate Array
IP	Internet Protocol
MAP	Maximum A Posteriori
OFDM	Orthogonal Frequency-Division Multiplexing
FFT	Fast Fourier Transform
BER	Bit Error Rate
Rx	Receiver

Abstract

The LTE (Long Term Evolution) and LTE-Advanced are the latest mobile communications standards developed by the Third Generation Partnership Project (3GPP). These standards represent a transformative change in the evolution of mobile technology. Within the present decade, the network infrastructures and mobile terminals have been designed and upgraded to support the LTE standards. As these systems are deployed in every corner of the globe, the LTE standards have finally realized the dream of providing a truly global broadband mobile access technology.

The turbo decoder is the most challenging component in a digital HSDPA receiver in terms of computation requirement and power consumption, where large block size and recursive algorithm prevent pipelining.

This thesis addresses hardware implementation aspects of parallel Turbo-Decoder on FPGA that reach more than 150 Mb/s LTE data-rate using multiple soft-input soft-output (SIS0) decoders that operate in parallel. To improve efficacy, we harness a radix-4-based 8x parallel turbo-decoder. Turbo-Decoding rate is set to 1/3.

Keywords:
LTE, mobile communication standards, HSDPA receiver, hardware implementation, parallel Turbo-Decoder, FPGA

Chapter 1

Introduction

Turbo coding was introduced in 1993 by Berrou, Glavieux, and Thitimajashima [1], [2], who reported extremely impressive results for a code with a long frame length. Since its recent invention, turbo coding has evolved at an unprecedented rate and has reached a state of maturity within just a few years due to the intensive research efforts of the turbo coding community. The excellent performance of turbo codes however, comes at the expense of significant computational complexity and consequently high power consumption at the receiver for proper decoding. Indeed, the computational burden of the turbo decoder far exceeds that of any other component in a receiver, especially for high data rates.

1.1 Motivation

In the past two decades we have seen the introduction of various mobile standards, from 2G to 3G to the present 4G, and we expect the trend to continue. The primary mandate of the 2G standards was the support of mobile telephony and voice applications. The 3G standards marked the beginning of the packet-based data revolution and the support of Internet applications such as email, Web browsing, text messaging, and other client-server services. The 4G standards will feature all-IP packet-based networks and will support the explosive demand for bandwidth-demanding applications such as mobile video-on-demand services. The rapid increase in wireless data traffic now begins to strain the network capacity and operators are looking for novel technologies enabling even higher data-rates than those in the past. The channel coding scheme for LTE [3] is Turbo coding. Turbo codes achieve close to Shannon capacity [4] and the Turbo decoder is typically one of the major blocks in a LTE wireless receiver. Turbo decoders suffer from high decoding latency due to the iterative decoding process, the forward–backward recursion in the

maximum a posteriori (MAP) decoding algorithm and the interleaving/de- interleaving between iterations

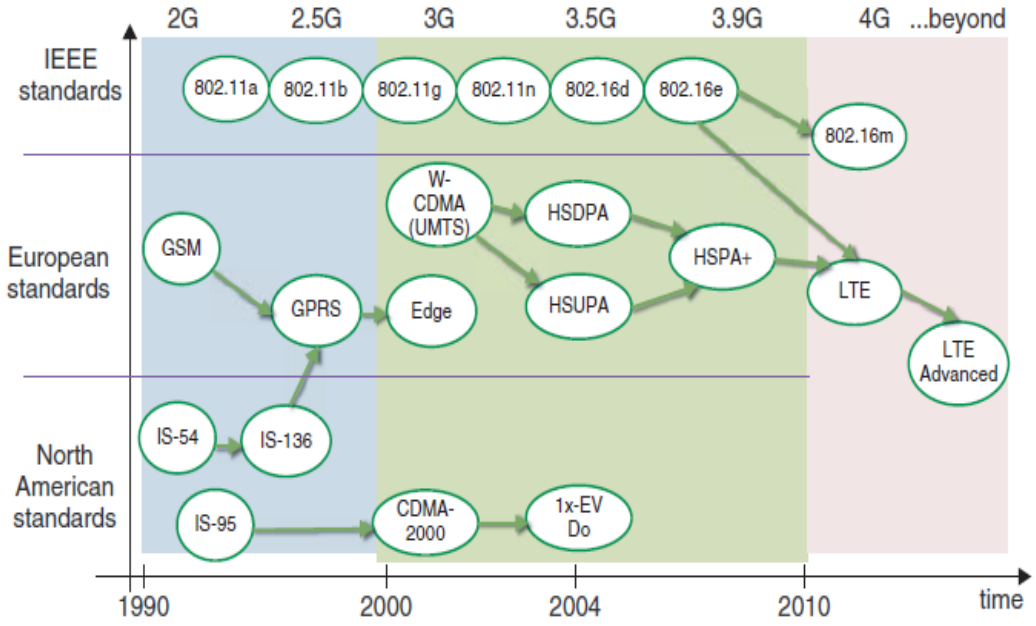


Figure 1.1 Evolution of wireless standards in the last two decades

1.2 Thesis goal

In this work, we present the implementation of a power-efficient and high throughput parallel Turbo-Decoder architecture for LTE, proposed in [5]. It is detailed an 8x parallel radix-4-based SISO Decoder. We used the Verilog Hardware Description Language (HDL) for the development of the hardware modules and we performed the verification by comparing the HDL simulation results with the corresponding from Matlab.

Another goal of this thesis is to integrate the hardware implementation of the Turbo-Decoder into a LTE compliant Single-In Single-Out model in order to accelerate the receiver's (Rx) baseband processing (Figure 2.1).

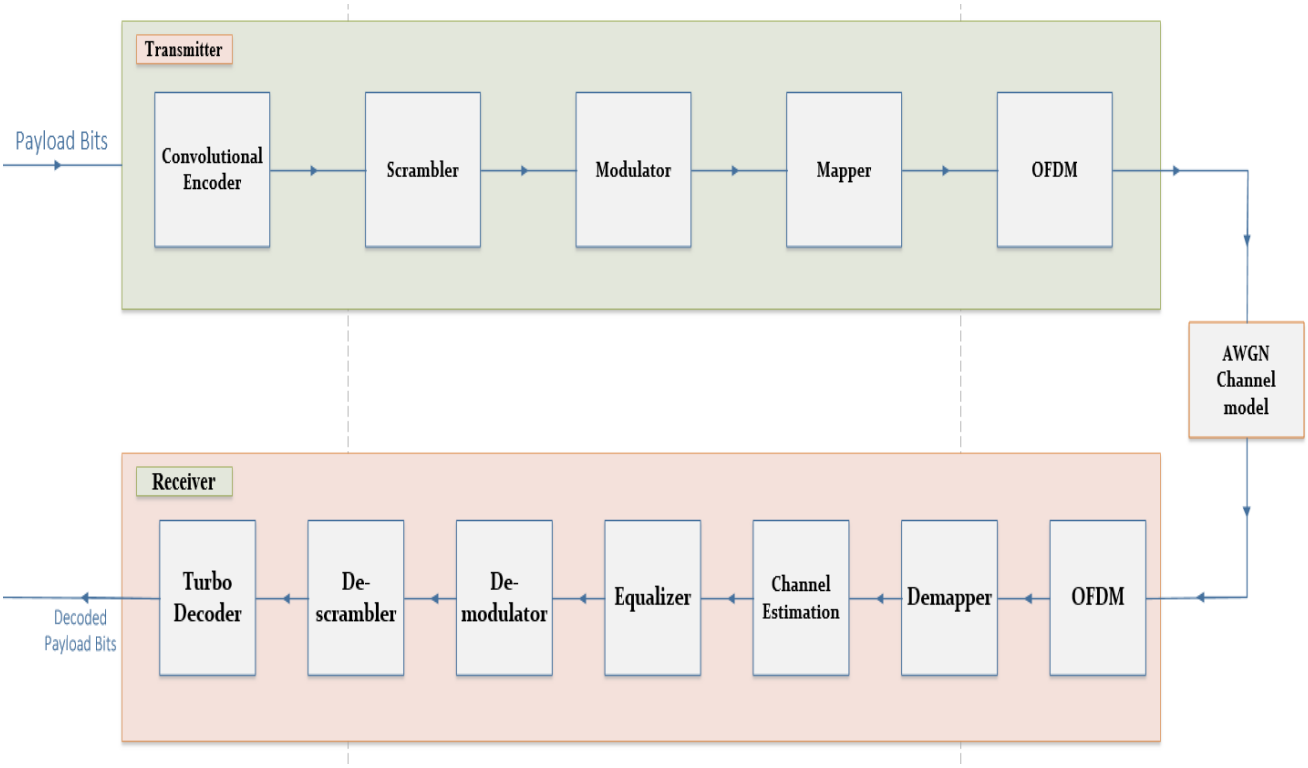


Figure 1.2 LTE SISO Processing Chain

1.3 Thesis Structure

The remainder of the thesis is organized as follows. Section 2 reviews the principle of turbo decoding and details the algorithm used for SISO decoding. The parallel turbo-decoder architecture is presented in Section 3 and the corresponding throughput/area tradeoffs are studied. The interleaver architecture is detailed in Section 4 and Section 5 describes the architecture of the SISO decoder. Section 6 provides the implementation results and we conclude in Section 7.

Chapter 2

Turbo-Decoding for LTE

2.1 Introduction

The components of the receiver [6] that is shown in figure 1.2 is shortly described below:

- **OFDM** (including Demapper)
  - Subdivides the information transmitted in the frequency domain and aligns data symbols with subcarriers
  - Cycle prefix removal
  - FFT (Fast Fourier Transform) operation to recover the received data and reference signals at each subcarrier

- **Channel Estimation and Equalizer**
  - Estimate channel frequency response based on transmitting known data or symbols
  - Recover the best estimate of the transmitting signal using a low complexity-frequency-domain equalizer

- **Demodulator**
  - Demodulate the payload symbols to the chosen constellation grid.

- **Descrambler**
  - Inverse transmitter's scrambling operation in which had encrypted the transmitted signal.

- **Turbo Decoder**
  - is used in conjunction with a Turbo Convolutional Encoder to provide an extremely effective way of transmitting data reliably over noisy data channels
  - is designed to meet the LTE specification

According to Matlab-Simulator (table 2.1) the most time consuming receiver's component is Turbo-Decoder by far.

Component	Time (sec)
OFDM	0.008512
Demapper	0.004483
Channel Estimation	0.026690
Equalizer	0.001541
Demodulator	0.055949
Descrambler	0.015564
Turbo Decoder	0.153524

Table 2.1 Matlab Simulator Profiling for SISO Receiver

2.2 Turbo-Decoding Algorithm

The turbo encoder is illustrated in the left-hand of figure 2.1. The first component encoder receives uncoded (systematic) data bits in natural order and outputs a set of parity bits. The second component encoder receives a permutation of the data bits from a block interleaver and outputs a second set of parity bits. The systematic bits and the two sets of parity bits are then transmitted over the wireless channel.

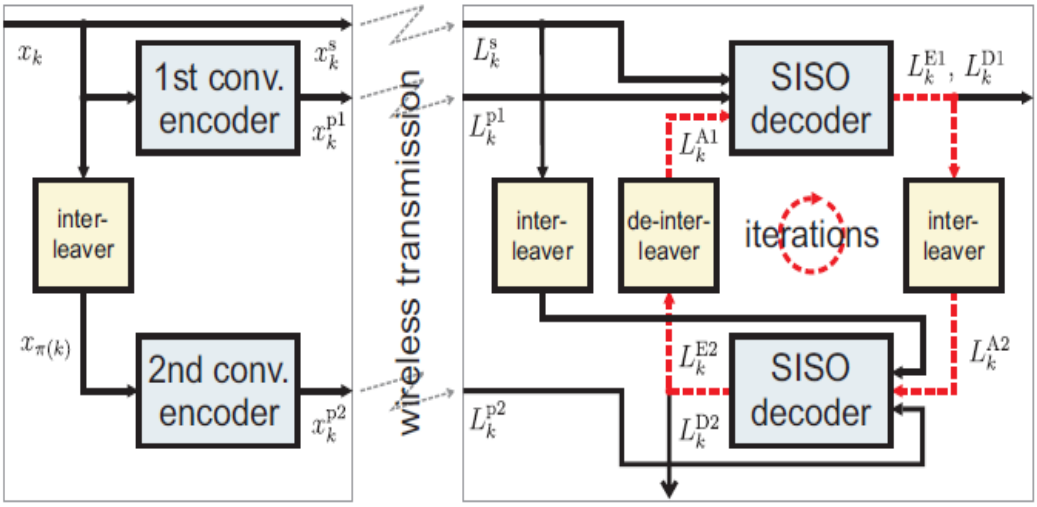


Figure 2.1 Parallel-concatenated turbo-encoder and block diagram of a turbo-decoder.

However, since this signal is usually distorted by noise and interference, the demodulator can only obtain estimates of the systematic and two sets of parity bits. These estimates are provided to the subsequent turbo decoder in the form of log-likelihood ratios (LLRs), Lk^s, Lk^{p2}, Lk^{p1}, and which express the ratio between the probabilities of the transmitted bits being 0 and being 1. The turbo decoder inverts the operations performed by the turbo encoder. A turbo decoder is based on the use of two decoders and two interleavers in a feedback loop. Figure 2.1 depicts the main idea. The first and second SD perform decoding of the convolutional code generated by the first or the second CE, respectively. One pass by both the first and the second SD is referred to as a full-iteration; the operation performed by a single SD a half-iteration. In this work is used 11 half iterations in order to produce the final decoded bits.

Each SD computes intrinsic a-posteriori LLRs Lk^{D1} and Lk^{D2}, for the transmitted bits, based on the systematic LLRs in natural Lk^s or interleaved order $L\pi(k)^s$, on the parity LLRs Lk^{p1} or Lk^{p2}, and on the so-called a-priori LLRs Lk^{A1} or Lk^{A2}. In subsequent iterations, each SD uses the extrinsic LLRs $Lk^{Ei} = Lk^{D1} - (Lk^s + Lk^{Ai})$ computed by the other SD. For the first iteration the a-priori LLRs are set to 0. Due to the interleaving used at the encoder, care must be taken to properly interleave and de-interleave the LLRs which are used to represent the soft values of the bits. Furthermore, because of the iterative nature of the decoding, care must be taken not to re-use the same information more than once at each decoding step.

A soft-in soft-out decoder is a type of soft-decision decoder used with error correcting codes. "Soft-in" refers to the fact that the incoming data may take on values other than 0 or 1, in order to indicate reliability. "Soft-out" refers to the fact that each bit in the decoded output also takes on a value indicating reliability.

The soft outputs and inputs from the component decoders are typically represented in terms of the so-called Log Likelihood Ratios(LLRs), the magnitude of which gives the sign of the bit, and the amplitude the probability of a correct decision. The LLRs are simply, as their name implies, the logarithm of the ratio of two probabilities. For example, the LLR $L(uk)$ for the value of a decoded bit uk is given by

$$L(uk) = ln\left(\frac{P(u_k=+1)}{P(u_k=-1)}\right) \qquad (2.1)$$

We summarize below what is meant by the terms *a-priori*, *a-posteriori*, and *extrinsic information*.

a-priori: The *a-priori* information about a bit is information known before decoding starts, from a source other than the received sequence or the code constraints. It is also sometimes referred to as intrinsic information to contrast with the extrinsic information described next.

extrinsic: The extrinsic information about a bit uk is the information provided by a decoder based on the received sequence and on *a-priori* information excluding the received systematic bit and the *a-priori* information for the bit. Typically, the component decoder provides this information using the constraints imposed on the transmitted sequence by the code used. It processes the received bits and *a-priori* information surrounding the systematic bit, and uses this information and the code constraints to provide information about the value of uk.

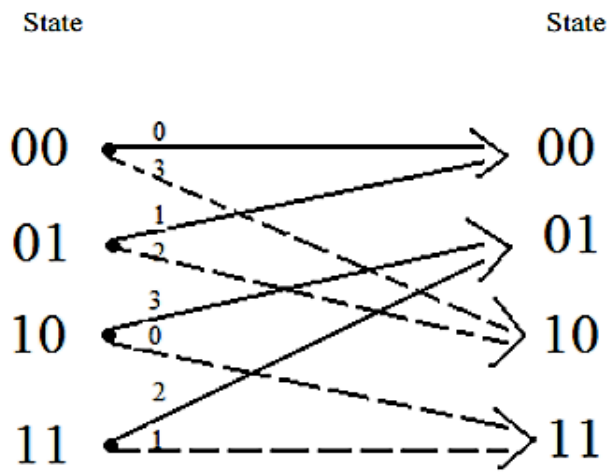
a-posterior: The *a-posteriori* information about a bit is the information that the decoder gives taking into account all available sources of information about uk. It is the *a-posteriori* LLR, that the MAP algorithm gives as its output.

2.3 Radix-4 Max-Log BCJR Algorithm

In 1974 an algorithm, known as the Maximum A-Posteriori (MAP) algorithm, was proposed by Bahl, Cocke, Jelinek and Raviv [7] for estimating the a-posteriori probabilities of the states and the transitions of an observed Markov source, when subjected to memoryless noise. This algorithm has also become known as the BCJR algorithm, named after its inventors. They showed how the algorithm could be used for decoding both algebraic and convolutional codes. The MAP algorithm examines every possible path through the convolutional decoder trellis and therefore initially seemed to be unfeasibly complex for application in most systems. Hence, it was not widely used before the discovery of turbo codes. The MAP algorithm provides not only the estimated bit sequence, but also the probabilities for each bit has been decoded correctly. This is essential for the iterative decoding of turbo codes and makes the MAP algorithm very suitable for turbo decoders

The BCJR algorithm resembles the Viterbi algorithm [8] and traverses a trellis representing the convolutional code to compute the intrinsic LLRs.

Trellis codes do not operate on independent blocks of source data, unlike the block codes. A trellis encoder maps an arbitrarily long input data stream to an arbitrarily long output code stream. Trellis codes can encode data continuously. A trellis encoder is a finite state machine. The output of the encoder depends on the inputs at that time and the current state of the encoder. The rate of encoder is k/n as in block codes because it gives n outputs for k inputs. In this trellis coded modulation method the receiver’s decision is taken depending on entire sequence of symbols rather than on symbol by symbol calculation.



State transition when input is 0

State transition when input is 1

Figure 2.1 Basic Trellis Diagram

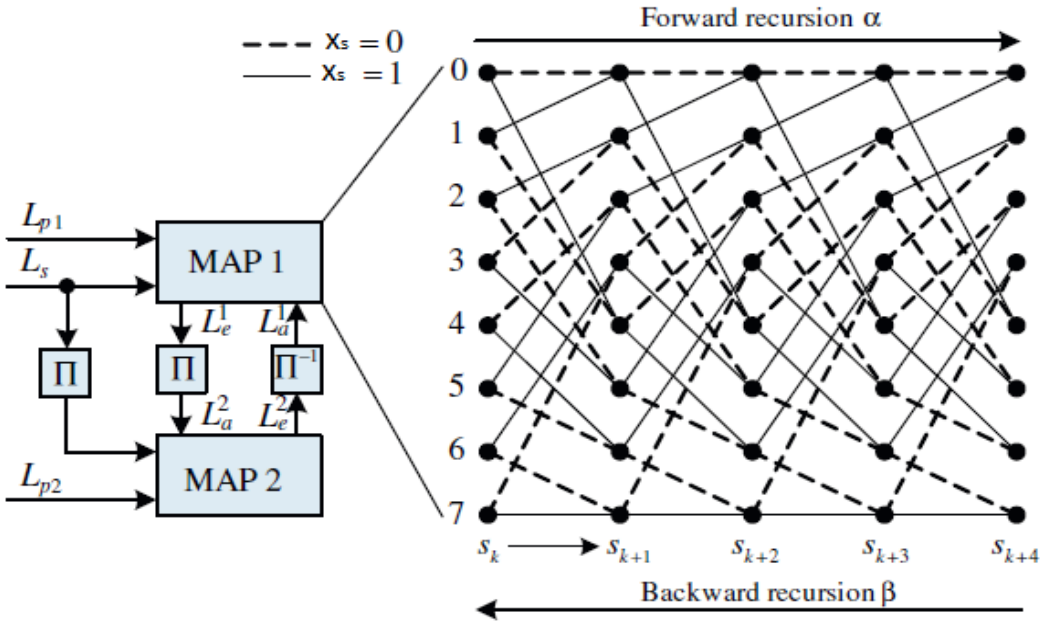


Figure 2.2 Basic structure of an iterative Turbo decoder. Iterative decoding based on MAP decoders. Forward/backward recursions on the trellis diagram.

It is applied the Max-log approximation to the forward state-metric recursions:

$$a_k(s) = \max\{a_{k-1}(s'_0) + \gamma_k(s'_0, s), \\ a_{k-1}(s'_2) + \gamma_k(s'_2, s)\} \qquad (2.2)$$

where s'_0 and s'_2 correspond to the two possible predecessor states of s (see Fig. 2). The backward state-metrics $\beta_k(s)$ are computed similarly to (2.2) in the opposite direction. Both recursions can be performed efficiently based on hardware-friendly add-compare-select (ACS) operations. The γ_k term above is the branch transition probability that depends on the trellis diagram, and is usually referred to as the branch metric (see [9] for details).

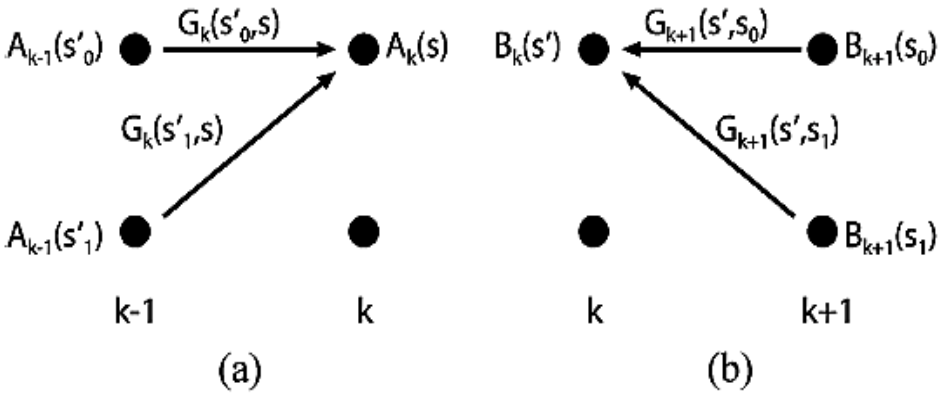


Figure 2.3 Example of the calculation of the forward and backward state metrics for radix-2 recursions

Once all a_k and β_k have been obtained, the a-posteriori output of the max-log-MAP decoder can be computed. To this end, the decoder must consider the state transitions $s' \rightarrow s$ associated with x_s= 0 and the ones associated with x_s= 1 separately and then computes:

$$L_k^{D1,D2} = \max_{(s',s):\, x_s=0} \{a_{k-1}(s') + \gamma_k(s',s) + \beta_k(s)\} - \max_{(s',s):\, x_s=1} \{a_{k-1}(s') + \gamma_k(s',s) + \beta_k(s)\}. \quad (2.3)$$

In this work, it is used a radix-4 (see figure 2.4) Max-Log turbo decoder in order to enhance the throughput. The Log-MAP core processes two received symbols per clock cycle using a radix-4 architecture, doubling the throughput for a given clock rate over a similar radix-2 architecture. Specifically, the radix-4 forward state metrics (figure 2.4) are computed on the basis of its four admissible predecessor states s'_0, s'_1, s'_2 and s'_3 (at step k-2) as follows:

$$a_k(s) = \max\{a_{k-2}(s''_0) + \gamma_k(s''_0,s), + a_{k-2}(s''_1) + \gamma_k(s''_1,s), a_{k-2}(s''_2) + \gamma_k(s''_2,s), + a_{k-2}(s''_3) + \gamma_k(s''_3,s)\}. \quad (2.4)$$

For the first trellis step (k=0) we initialize $a_k(s_0) = 1, a_k(s_1) = 0$, $a_k(s_2) = 0$ *and* $a_k(s_3) = 0$. The radix-4 branch metrics required in (2.4) are computing according to:

$$\gamma_k(s''_i,s) = \gamma_k(s''_i,s'_j) + \gamma_k(s'_j,s) \quad (2.5),$$

using the six branch metrics associated with the trellis step k and k-1 required in the radix-2 recursion.

Since the backward recursion progresses from the end of trellis diagram to its beginning for every step we initially set $\beta_k(s)$= 1/N, where N is the number of states in the turbo encoder. Then we use the radix-2 recursion (2.2) to calculate $\beta_{k-1}(s')$.

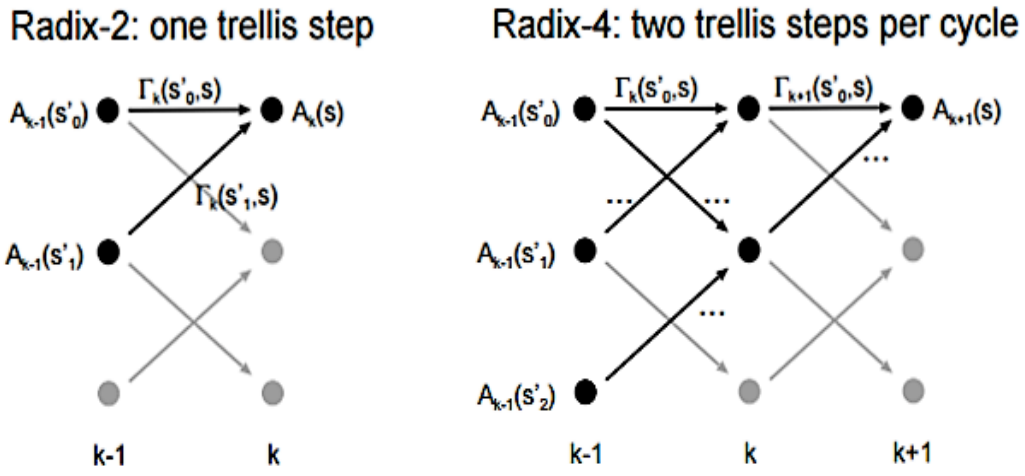


Figure 2.4 Radix-2 and radix-4 recursions

2.4 LTE Interleaver

Interleavers for turbo codes scramble the data in a pseudo-random order to minimize the correlation between the outputs of component encoders. Interleaver is an essential part and is also responsible for an excellent Bit Error Rate (BER) performance of turbo code. Although parallelism can be obtained using multiple hardware instances of a single decoder, this solution increases the memory requirements (each decoder requires separate memory) and also incurring a long latency. Recognizing these deficiencies, the LTE working group decided upon an approach that enables internal parallelism within a fast serial decoder.

Generally, the task of an interleaver is to permute the soft values generated by the MAP decoder and write them into random or pseudo-random positions. Interleaver architectures are well studied in literature [10], [11] and the recent wireless communication standards

like LTE have incorporated QPP and ARP interleavers [12] respectively.

In this work, contention free QPP interleaver architecture is used in the turbo decoder design. The recursive architecture of QPP interleaver has a simplified design and it can be easily used in the parallel architecture of turbo decoder to achieve higher throughput. Subsequently, QPP interleaver can be configured to calculate interleaved addresses for any value of block length (K). For example, 3GPP-LTE wireless standard uses 188 different values of K, ranging from 40 bits to 6144 bits. Specifically, address-computation for QPP interleavers is carried out from:

$$\pi_k(k) = (f1_k + f2_k{}^2)\ mod\ K \tag{2.6}$$

Where f1 and f2 are suitably chosen interleaver parameters that depend on the code-block length K.

i	K_i	f_1	f_2	i	K_i	f_1	f_2	i	K_i	f_1	f_2	i	K_i	f_1	f_2
1	40	3	10	**48**	416	25	52	**95**	1120	67	140	**142**	3200	111	240
2	48	7	12	**49**	424	51	106	**96**	1152	35	72	**143**	3264	443	204
3	56	19	42	**50**	432	47	72	**97**	1184	19	74	**144**	3328	51	104
4	64	7	16	**51**	440	91	110	**98**	1216	39	76	**145**	3392	51	212
5	72	7	18	**52**	448	29	168	**99**	1248	19	78	**146**	3456	451	192
6	80	11	20	**53**	456	29	114	**100**	1280	199	240	**147**	3520	257	220
7	88	5	22	**54**	464	247	58	**101**	1312	21	82	**148**	3584	57	336
8	96	11	24	**55**	472	29	118	**102**	1344	211	252	**149**	3648	313	228
9	104	7	26	**56**	480	89	180	**103**	1376	21	86	**150**	3712	271	232
10	112	41	84	**57**	488	91	122	**104**	1408	43	88	**151**	3776	179	236
11	120	103	90	**58**	496	157	62	**105**	1440	149	60	**152**	3840	331	120
12	128	15	32	**59**	504	55	84	**106**	1472	45	92	**153**	3904	363	244
13	136	9	34	**60**	512	31	64	**107**	1504	49	846	**154**	3968	375	248
14	144	17	108	**61**	528	17	66	**108**	1536	71	48	**155**	4032	127	168
15	152	9	38	**62**	544	35	68	**109**	1568	13	28	**156**	4096	31	64
16	160	21	120	**63**	560	227	420	**110**	1600	17	80	**157**	4160	33	130
17	168	101	84	**64**	576	65	96	**111**	1632	25	102	**158**	4224	43	264
18	176	21	44	**65**	592	19	74	**112**	1664	183	104	**159**	4288	33	134
19	184	57	46	**66**	608	37	76	**113**	1696	55	954	**160**	4352	477	408
20	192	23	48	**67**	624	41	234	**114**	1728	127	96	**161**	4416	35	138
21	200	13	50	**68**	640	39	80	**115**	1760	27	110	**162**	4480	233	280
22	208	27	52	**69**	656	185	82	**116**	1792	29	112	**163**	4544	357	142
23	216	11	36	**70**	672	43	252	**117**	1824	29	114	**164**	4608	337	480
24	224	27	56	**71**	688	21	86	**118**	1856	57	116	**165**	4672	37	146
25	232	85	58	**72**	704	155	44	**119**	1888	45	354	**166**	4736	71	444
26	240	29	60	**73**	720	79	120	**120**	1920	31	120	**167**	4800	71	120
27	248	33	62	**74**	736	139	92	**121**	1952	59	610	**168**	4864	37	152
28	256	15	32	**75**	752	23	94	**122**	1984	185	124	**169**	4928	39	462
29	264	17	198	**76**	768	217	48	**123**	2016	113	420	**170**	4992	127	234
30	272	33	68	**77**	784	25	98	**124**	2048	31	64	**171**	5056	39	158
31	280	103	210	**78**	800	17	80	**125**	2112	17	66	**172**	5120	39	80
32	288	19	36	**79**	816	127	102	**126**	2176	171	136	**173**	5184	31	96
33	296	19	74	**80**	832	25	52	**127**	2240	209	420	**174**	5248	113	902
34	304	37	76	**81**	848	239	106	**128**	2304	253	216	**175**	5312	41	166
35	312	19	78	**82**	864	17	48	**129**	2368	367	444	**176**	5376	251	336
36	320	21	120	**83**	880	137	110	**130**	2432	265	456	**177**	5440	43	170

Table 2.2 Turbo codes Interleaver parameters (Part 1 of 2)

i	K_i	f_1	f_2	i	K_i	f_1	f_2	i	K_i	f_1	f_2	i	K_i	f_1	f_2
37	328	21	82	**84**	896	215	112	**131**	2496	181	468	**178**	5504	21	86
38	336	115	84	**85**	912	29	114	**132**	2560	39	80	**179**	5568	43	174
39	344	193	86	**86**	928	15	58	**133**	2624	27	164	**180**	5632	45	176
40	352	21	44	**87**	944	147	118	**134**	2688	127	504	**181**	5696	45	178
41	360	133	90	**88**	960	29	60	**135**	2752	143	172	**182**	5760	161	120
42	368	81	46	**89**	976	59	122	**136**	2816	43	88	**183**	5824	89	182
43	376	45	94	**90**	992	65	124	**137**	2880	29	300	**184**	5888	323	184
44	384	23	48	**91**	1008	55	84	**138**	2944	45	92	**185**	5952	47	186
45	392	243	98	**92**	1024	31	64	**139**	3008	157	188	**186**	6016	23	94
46	400	151	40	**93**	1056	17	66	**140**	3072	47	96	**187**	6080	47	190
47	408	155	102	**94**	1088	171	204	**141**	3136	13	28	**188**	6144	263	480

Table 2.3 Turbo codes Interleaver parameters (Part 2 of 2)

QPP interleaver can be configured to produce contention-free interleaved addresses for any of these values by changing the values of f1 and f2 in the expression (2.6). The expression (2.6) can be implemented efficiently in hardware because only addition, multiply and modulo-operations are involved. Furthermore, QPP interleavers map even addresses to even addresses and odd to odd.

Chapter 3

Parallel Turbo-Decoder Architecture

In the conventional BCJR algorithm (non-parallel), computations of forward-state, backward-state and branch metrics for entire trellis stages result in huge memory requirement and impose large decoding delay. Major steps involving in these parallel Turbo-decoding relating to state metrics are presented as follow.

Initialization: Assuming that the encoder is reset, the forward state metrics are initialized as $a_{k=0}(s_i) = 1 \ \forall \ i=0$ and $a_{k=0}(s_i) = 0 \ \forall \ i\neq 0$.

Forward recursion: During this process, the forward state metric of each states for successive trellis stages are computed as in (2.4).

Backward-recursion and estimation of backward state metrics: If N represents total number of states in each trellis stage, the backward state metrics are initialized as $\beta_k(s_i) = 1/N \ \forall \ i \in N$ (N is the number of trellis states) and during the backward recursion it is used the radix-2 recursion as in (2.2) in order to carry out $\beta_{k-1}(s_i)$.

In order to increase throughput, a promising solution is to instantiate N-BCJR units and to perform N-fold parallel decoding of trellis. This approach increases the turbo-decoding throughput by a factor of N compared to a non-parallel turbo-decoder.

3.1 High-Level Architecture

This work contains N=16 max-log BCJR instances, input memories for the storage of systematic and parity LLRs and one intermediate memory for the storage of the extrinsic LLRs. Radix-4 technique is used therefore two trellis steps are processed per clock cycle. It is noteworthy that the use of radix-4 recursions entails 2x increased memory-bandwidth, since the LLRs associated with even and odd numbered trellis steps are required per clock.

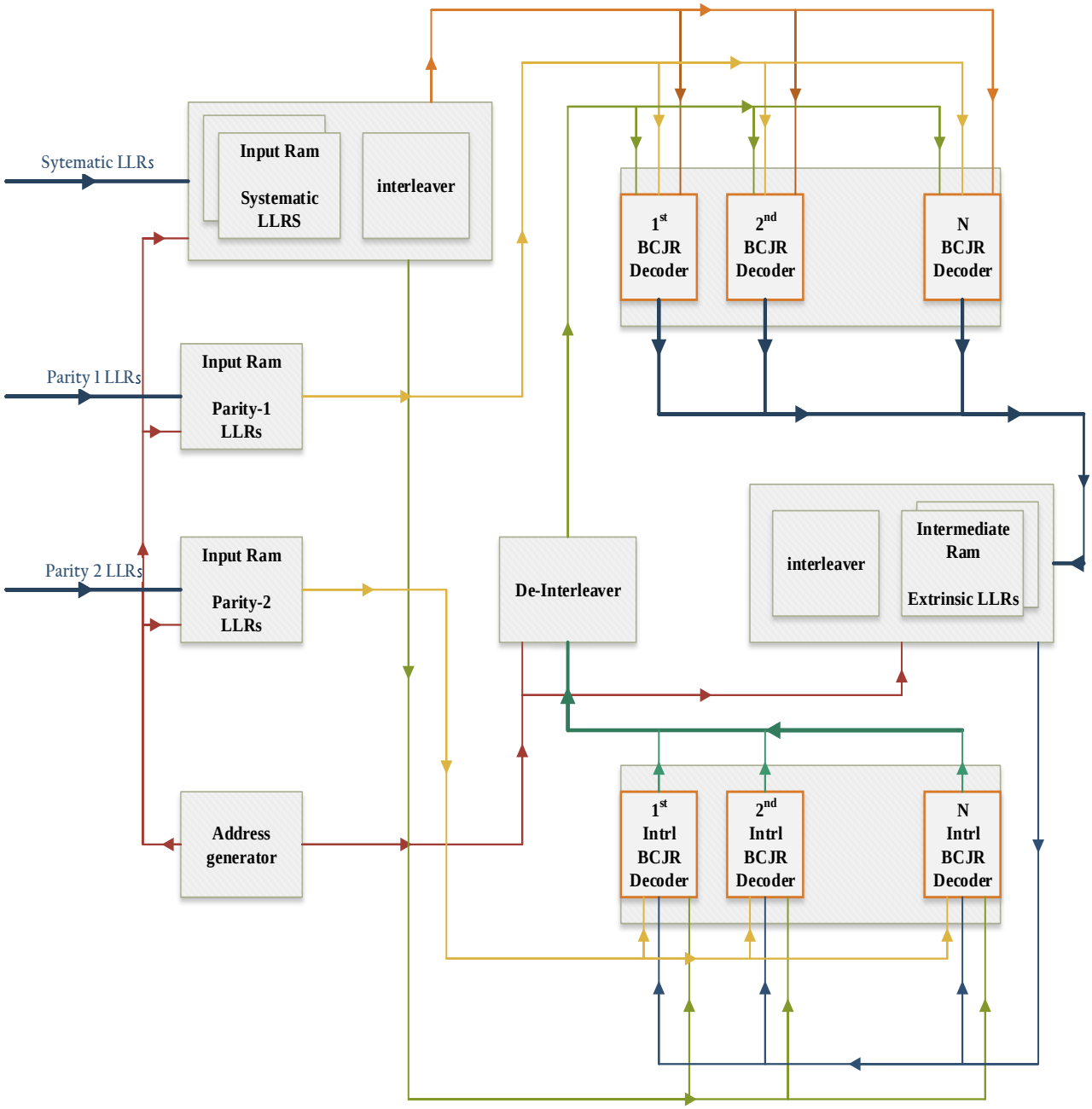


Figure 3.1 High-level architecture of the parallel turbo-decoder

3.2 Memory Architecture

With low power and big throughput in mind, turbo-decoder is based on the architecture in figure 3.1. In this design and taking into consideration the fact that radix-4 recursion is used, 4 block-rams store one block of the LLRs of the systematic and both sets of parity bits and 2. Two input block rams are associated to the systematic LLRs, one stores the systematic LLRs relating to the even numbered and the other for the odd-numbered trellis steps. In addition, 2 input block rams are used to store parity 1 and parity 2 LLRs. Furthermore, 2 block rams store the intermediate extrinsic LLRs, one for the odd and one for the even trellis-steps and 2 block ram in for the de-interleave unit. Totally, 8 block rams are used and the 4 block rams for systematic and extrinsic require half the amount of storage in contrast with the parities block rams. Each memory contains N LLR-values per address. This partitioning enables 2xN (N is the number of the parallel decoders) LLRs to be read per clock cycle.

3.3 Implementation Tradeoffs

Typically, the throughput of digital circuits can be increased by architectural and circuit-level transformations such as pipelining or parallel processing. For turbo decoders, the applicability of pipelining is limited due to the presence of feedback loops and the accompanying extra registers increase the energy consumption.

Comparative study of BER performances has shown that the parallel turbo decoder achieves an adequate BER performance. Recently, the VLSI implementations of parallel turbo decoders with N=8 [13], N=16 [14], N=32 [15] and N=64 [16] have been reported for higher data-rate applications. One of the key aspects of this work is the use of radix-4 recursions in order to achieve high throughput. Despite the fact that the use of radix-4 increases the area that BCJR decoders occupied, the area of the rather large, input and intermediate, memories remains the same. Clearly, the throughput improvement has to be paid for by a complexity increase.

Chapter 4

LTE Interleaver Architecture

Interleaving means the permutation of the order of the data bits in a code block. Turbo codes require specific interleavers which minimize the correlation between the SISO decoder inputs of subsequent half-iterations to achieve best decoding error rate performance. However, the rules for the generation of the interleaved pattern are highly complex.

In turbo decoder implementations the interleaver is a sub-block of the address generator, which generates the addresses for the memories in natural or permuted order. Thus, depending on the turbo decoder half-iteration, the SISO decoder inputs can be read from the input and from the intermediate memory in natural or interleaved order. After decoding, the LLR outputs of the SISO decoder block are written back in natural or interleaved order to the same address in the intermediate memory, depending on the specific turbo decoder half-iteration.

For most interleavers, parallel and interleaved memory access leads to an interleaver bottleneck which is caused by access-contentions. Thus, an Interleaver that alleviates the interleave bottleneck is of primary importance for parallel turbo decoding.

4.1 Contention-Free Interleaving for LTE

This LTE Interleaver exhibits two approaches to in order to have access to the memories in interleaved and natural order. The first approach to solve the memory access contention problem is to constrain the interleaver to be contention-free. Contention-free interleavers [17] allow instant access and trivial mapping for LLRs values that are required for the N parallel SISO decoders. For example, if K is the block length and N divides the K without remainder, the interleaved or natural order LLRs values can be always read from N memories. The second property is that the interleaver is maximally vectorizable [18], the address-distance

between each of the N interleaved addresses is always an integer multiple of the trellis-segment length S.

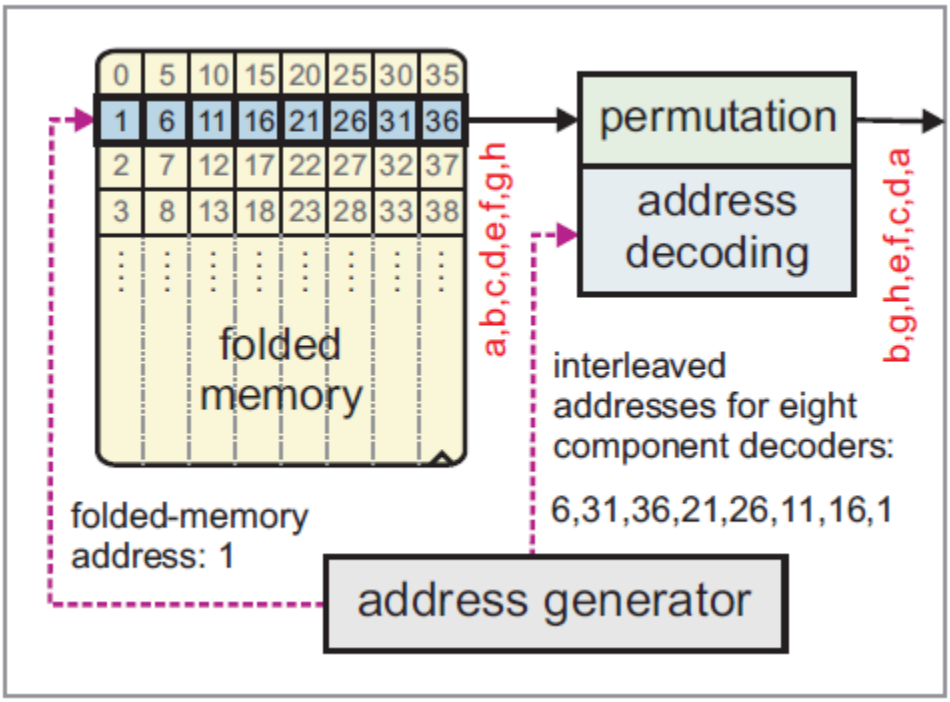


Figure 4.1 Architecture of the contention-free interleaver

As it is said in this work, radix-4 is used and therefore even and odd-numbered systematic and extrinsic LLRs are stored in separate RAMs with S/2 addresses. Figure 4.1 indicates the storage of K LLRs relating to one code-block (with length K) in a folded memory. Folded memory has S addresses and each address contains N LLRs. Therefore, K = NxS LLR values can be stored. In figure 4.1 it is used N=8 and S=5. LLRs are written column-wise and each column corresponds to an SISO decoder. As is illustrated in figure 4.1 the address-distance between each of the N LLRs in the same row is an integer multiple of 5 (trellis-segment S) and this is due to the maximally-vectorizable interleaver.

In the natural order phase, starting from the folded memory address 0 in increment way, the straightforward N LLrs located to the N BCJR instances. The value of nth corresponds to the nth BCJR.

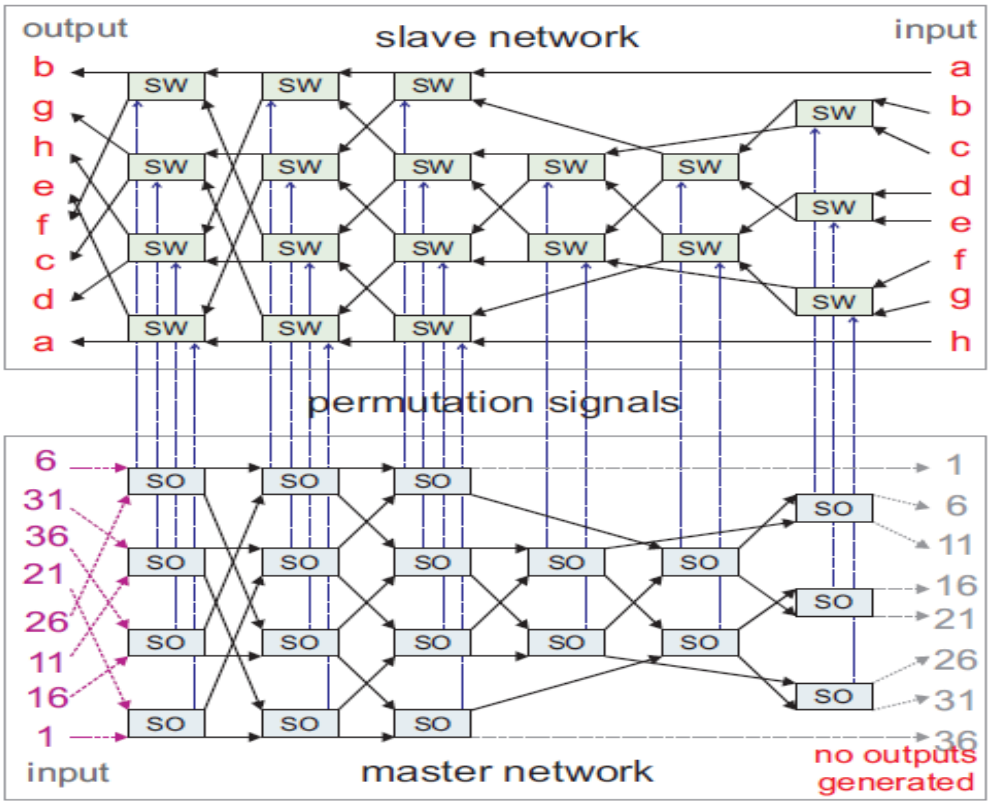
Since LTE interleaver is maximally-vectorizable, the N interleaved addresses always point out at the same row in the folded memory. As illustrated in figure 4.1 the 8 interleaved addresses (6,31,36,21,26,11,16,1) relevant to address 1 in the folded memory point out in the same row. In the interleaved phase, address-decoding

generates the sorting order that is required to assign the LLRs from the folded memory to the corresponding SISO decoders and a permutation according to the extracted sorting-order is applied to the N LLR values, which are then passed to the corresponding BCJR instances. This enables N-fold parallel access to the folded memory.

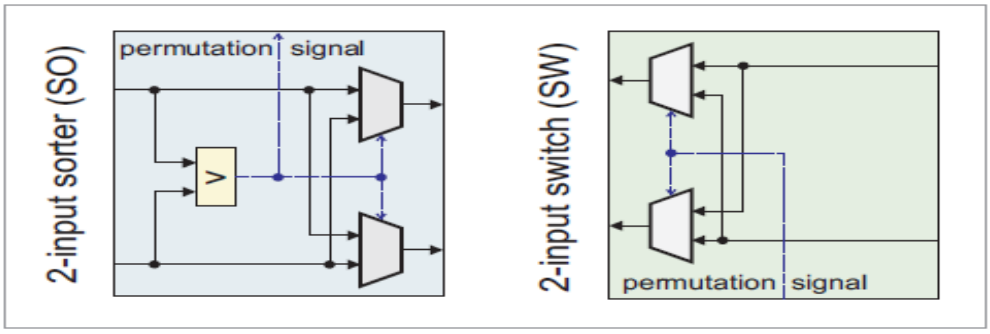
4.2 Master-Slave Batcher Network

Address decoding and permutation for maximally-vectorizable contention-free interleaver based on [5] is depicted in figure Master-Slave Batcher Network.

Address-decoding that it is reffered in 4.1 chapter is carried out in the master network and the slave network performs the permutation by applying the inverse-sorting order to the N LLRs. The master network consists of a number of 2-input sorter (SO) units and the slave network of a 2-input switches (SW). The permutational signals from the master networks control the switches in the slave network.



(a) MS Batcher network for $N = 8$.



(b) 2-input sorter (SO) and 2-input switch (SW) units.

Figure 4.2 The Master-Slave Batcher Network architecture

This network is a hardware efficient way to perform address-decoding and permutation because only Multiplexers (MUXs) with 2 inputs and 1 output are required. LTE interleaver is of primary significance for parallel turbo-decoders.

Chapter 5

Radix-4 Max-Log BCJR Architecture

In this design, Radix-4 Max-log BCJR with 16 instances dominate the circuit area and the power consumption. Consequently, is very significant an area-power efficient implementation of radix-4 max-log BCJR.

5.1 VLSI Architecture

The architecture of the radix-4 max-log BCJR is presented in figure 5.1.

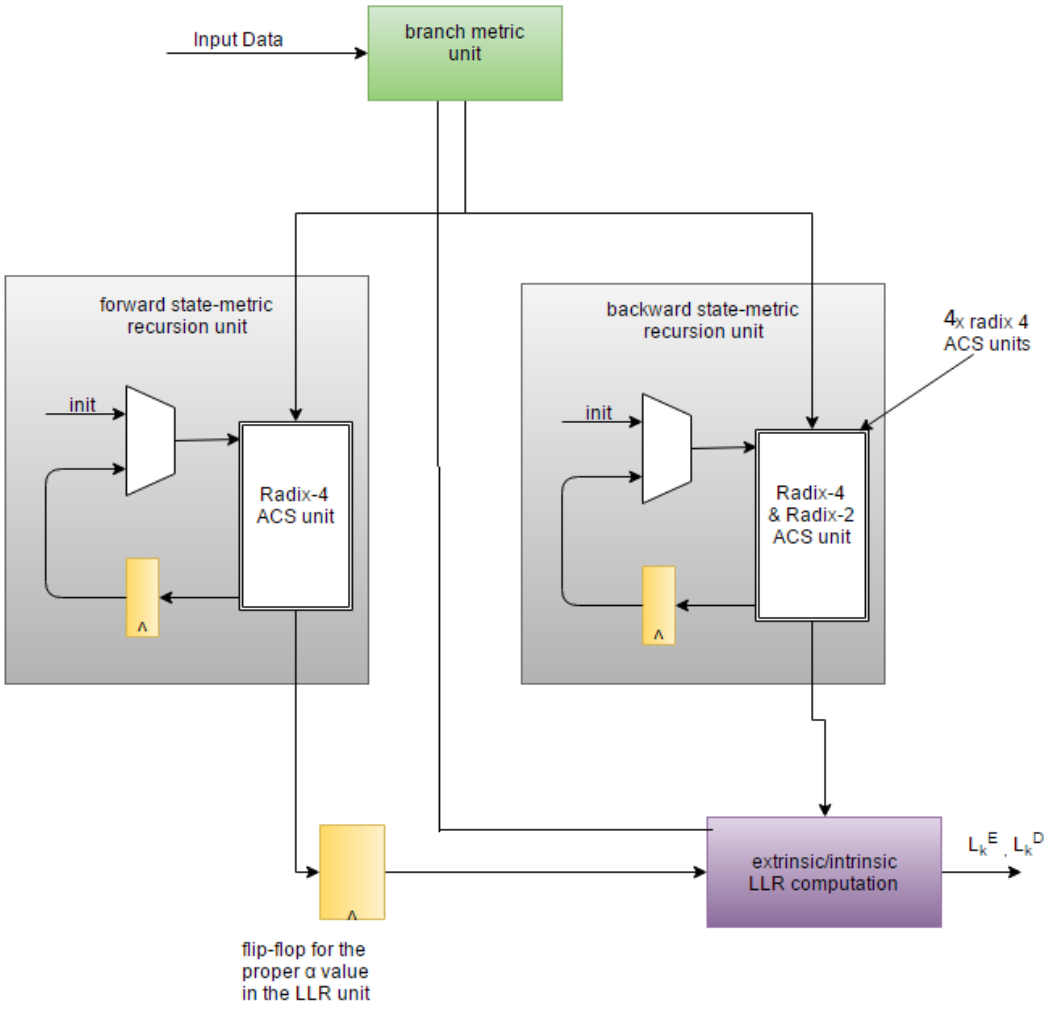


Figure 5.1 Architecture of the implemented radix-4 max-log BCJR core

In this design, two trellis steps are computed per clock cycle. This computation is performed using 2 parallel units, the forward state-metric recursion unit and the backward state-metric recursion unit. The problem of this approach is the unknown backward (or forward) state metrics which are required in the beginning of the backward (or forward) recursion. In the very first iteration, uniform state metrics can be used for initialization. The forward state metrics are initialized as $a_{k=0}(s_i) = 1 \ \forall \ i=0$ and $a_{k=0}(s_i) = 0 \ \forall \ i\neq 0$ and in every clock cycle (2.4) is used to compute the forward state metrics for this trellis step. In the backward state-metric recursion unit in every step the backward metrics are initialize $\beta_k(s_i) = 1/N \ \forall \ i \in N$, where N is the number of trellis-states (in this work we have 4 states).

The branch metrics unit first work out the radix-2 branch metrics and then compute the radix-4 branch metrics according to (2.5). The

results of the forward state-metric recursion unit are passed from flip-flops before used to compute the intrinsic LLRs. This occurs because we want to delay a cycle the results from forward state-metric recursion unit because in the LLR computation unit we need the forward metrics from the previous cycle. For example, for the computation of $L_{k-1}{}^{D}$ we need to know a_{k-2} , see (2.3).

5.2 Radix-4 ACS Units with Modulo-Normalization

The recursive state metric computation cannot be pipelined or parallelized due to the presence of the feedback loop. Hence, we shall focus on measures for reducing the complexity of the state metric recursions to shorten the critical path and to reduce area and power consumption. The normalization technique used in this thesis is focused to achieve high-speed performance of turbo decoder from an implementation perspective. In addition, radix-2 and radix-4 ACS that is depicted in figure 5.2 are hardware friendly.

The comparison (CMP circuit for modulo-normalization [19] achieves the renormalization with a controlled overflow in the data path and requires only a 3-input XOR gate. In the parallel radix-4 ACS is utilized 4 adders, 6 CMP circuits and a 4-1 MUX (4 inputs,1 output). The selection signal is carried out by the six parallel CMP followed by Karnaugh-map minimization. Radix-2 ACS requires only 2 adders a CMP circuit and a MUX with select signal the output from CMP circuit.

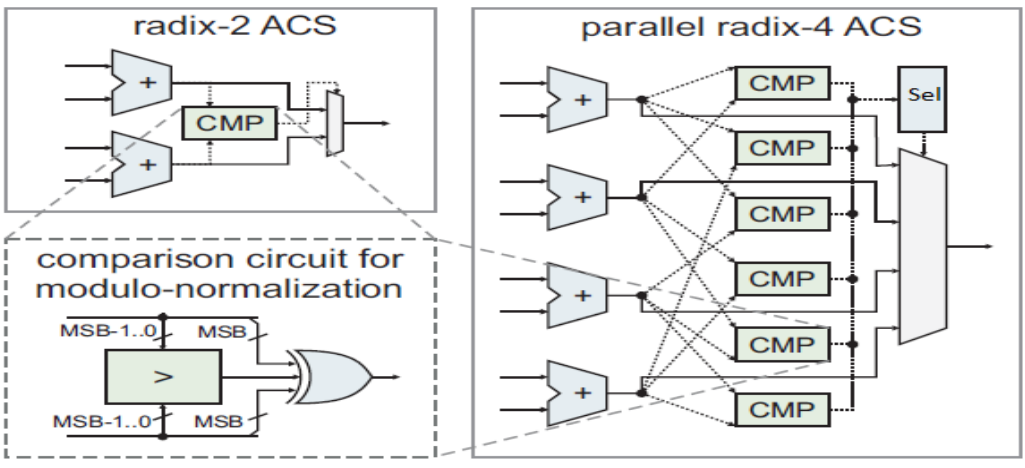


Figure 5.2 Radix-2 and Radix-4 architectures

5.3 LLR Computation Unit

The LLR computation unit that is presented in figure 5.1 calculate the intrinsic and extrinsic LLRs for the trellis step k-1 and k in each clock cycle. Hence for the computation of the intrinsic and extrinsic LLRs for step k-1, a_{k-2}, β_{k-1} and γ_{k-1} are required. a_{k-2} came from the flip flop after the forward state metric-recursion unit, γ_{k-1} came from branch metric unit and β_{k-1} from backward state metric-recursion unit. a_{k-1}, β_k and γ_k are required for the calculation of the intrinsic and extrinsic LLRs for the step k. With aid of radix-2 ACS a_{k-1} stem from a_{k-2} , γ_k came from branch metric unit and β_k from backward state metric-recursion unit.

Now for the computation of the intrinsic LLRs (2.3), the max of $a_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)$ relating to a state transitions $s' \rightarrow s$ associated with x_s= 0 and the ones associated with x_s= 1 must be calculated. In order to compute this is used a design similar with radix-4 ACS with the difference that adders have 3 inputs (α, β, γ).

Chapter 6

Implementation Results

In this chapter, it is shown simulation and synthesis results and it is summarized the key points of the 8x parallel implemented Tutbo-Decoder.

6.1 Axi-4 Stream Ip

AXI4-Stream is a subset of Advanced Microcontroller Bus Architecture (AMBA) AXI4 protocol. It is designed for high-speed streaming data. To simplify interoperability, Xilinx IP requiring streaming interfaces use a strict subset of the AXI4-Stream protocol. An AXI4-Stream Ip is easy to use, flexible and is a high performance IP. This Turbo Decoder Ip must have a master and a slave interface because it is requirement to receive and send data. Table 6.1 and Table 6.2 show the signal names of the slave and master interface and define them.

Pin	Direction	Port width (bits)	Description
Aclk	Input	1	Clock: Sample on the rising edge
Arst	Input	1	Reset: Active low reset. When asserted low the decoder is reset.

En	Input	1	Enable: Clock enable signal
Tvalid_s	Input	1	Tvalid: indicates that the master is driving a valid transfer. A transfer takes place when both Tvalid and Tready are asserted
Tready_s	Output	1	Tready: indicates that the slave can accept a transfer in the current cycle.
Tlast_s	Input	1	Tlast: indicates the boundary of a packet.
Tdata_s	Input	32	Tdata: is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.

Table 6.1 Signals associated with slave interface

Pin	Direction	Port width (bits)	Description
Aclk	Input	1	Clock: Sample on the rising edge
Arst	Input	1	Reset: Active low reset. When asserted low the decoder is reset.
En	Input	1	Enable: Clock enable signal
Tvalid_m	Output	1	Tvalid: indicates that the master is driving a valid transfer. A transfer takes place when both Tvalid and Tready are asserted
Tready_m	Input	1	Tready: indicates that the slave can accept a transfer in the current cycle.

Tlast_m	Output	1	Tlast: indicates the boundary of a packet.
Tdata_m	Output	64	Tdata: is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.

Table 6.2 Signals associated with master interface

Tdata_s is the primary input for this work and its length is 8 Byte. This size contains the systematic LLR, parity 1 and 2, the number of iterations the decoder must implement and code block and is organized as follows:

31 19	18 15	14 10	9 5	4 0
Block length	iterations	Systematic LLR	Parity 1 LLR	Parity 2 LLR

Table 6.3 Construction of Tdata_s

6.2 Verilog Implementation

It is shown that Max-Log BCJR algorithm is totally suitable between the implantation complexity and the decoding performance. Now it is investigated how to implement the turbo decoder into a Field-Programming Gate Array (FPGA). Verilog is used as the Hardware Design Language for design entry and behavioral simulation. A basic Application Specific Integration Circuit (ASIC) / FPGA design process is depicted in figure 6.1.

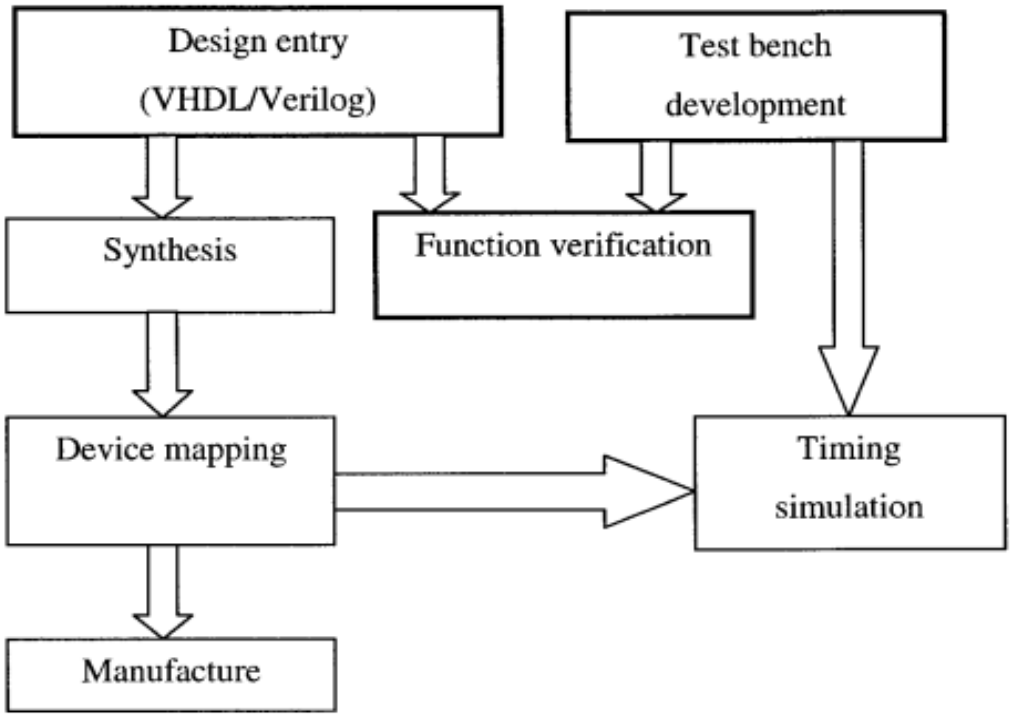


Figure 6.1 ASIC/FPGA Design process.

1) Design entry
In this step system interfaces and functionalities are defined. The detailed design is captured in Verilog, which provides useful programming features for structured design techniques. With these techniques, a complex design can be analyzed into simpler implementation modules. Each module has its own definition of functionality and interface.

2) Test bench development
The functionality of Verilog Design must be verified before going further in synthesis. Test benches is developed with this purpose, which is also programmed in Verilog that provides design entity with the stimulus and verifies the outputs.

3)Functionality verification
In this step, combinations of inputs (stimulus) are fed into the design entity and the outputs are verified. Usually the stimulus and results are generated and saved into files before the Verilog simulation. The test bench will read in the stimulus, feed them into the design entity, obtain the outputs of the design entity and compare these outputs to the outputs that should be obtained. A properly design verification program should be take into account the mathematic limitations in a

realistic hardware design including the finite resolution and limited dynamic range of the date representation.

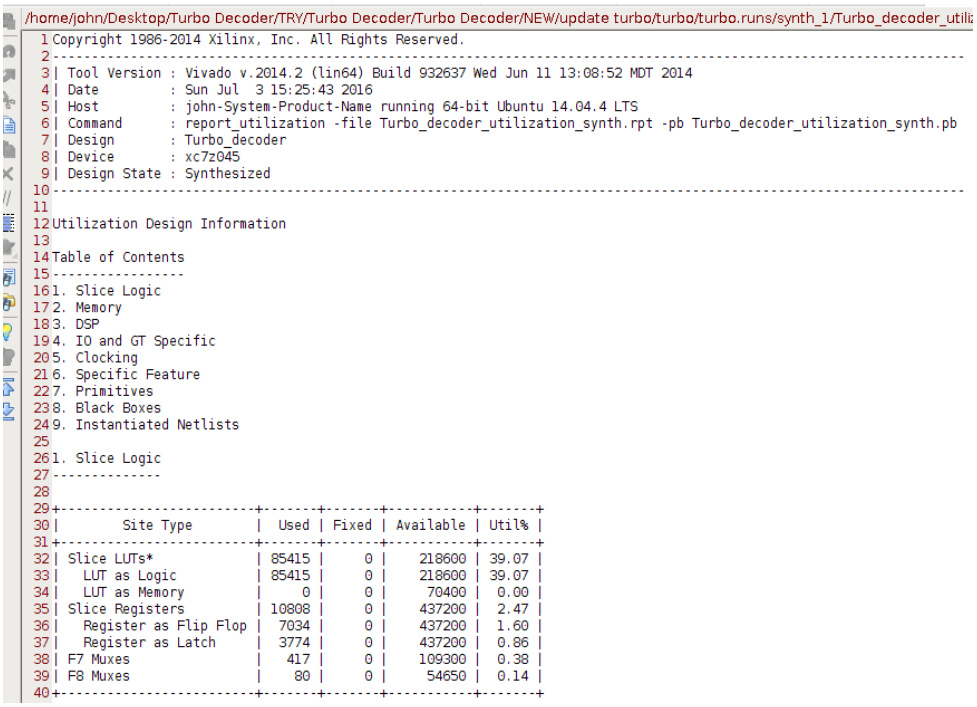
4) Synthesis
Synthesis is a process of transforming a design specification into an implementation, i.e converting an abstract design description into a hardware abstract. This process is performed using the synthesis tools based on certain synthesis technology library provided by FPGA manufactures.

5) Device mapping
This process tries to find proper devices from a library based on synthesis result. In this phase, a timing model generation program provided by a device vendor or third part simulation model supplier could be used to generate the accurate timing model of the design.

6) Timing Simulation
The timing model generated during the device mapping is combined into the test bench and the verification is performed again. When the design is performed correctly with the timing model, is ready to be manufactured. However, if the design fails with this timing model, the designer has to go back to the first step, modify the design and go through all the steps again until the design passes the timing simulation.

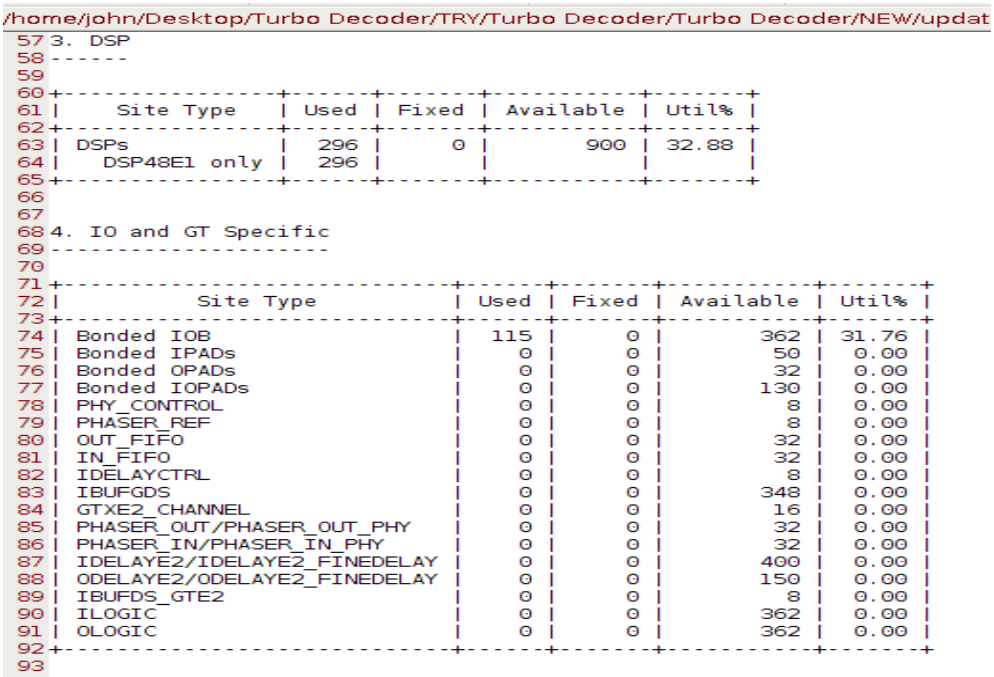
In this thesis, it is implemented a parallel turbo decoder and the corresponding Verilog test bench in Verilog. The functional verification is performed by comparing the decoding performance of Verilog implementation with a Matlab-simulation. The parallel Turbo-Decoder for LTE at Register-Transfer-Level (RTL) and the design description follows a proper coding style to make it synthesizable. For implementation, simulation and synthesis is used a Xillinx tool, Vivado and the test platform was ZYNQ-7 ZC706 [20].

Synthesis results are presented below:



```
/home/john/Desktop/Turbo Decoder/TRY/Turbo Decoder/Turbo Decoder/NEW/update turbo/turbo/turbo.runs/synth_1/Turbo_decoder_utili
Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2014.2 (lin64) Build 932637 Wed Jun 11 13:08:52 MDT 2014
| Date         : Sun Jul  3 15:25:43 2016
| Host         : john-System-Product-Name running 64-bit Ubuntu 14.04.4 LTS
| Command      : report_utilization -file Turbo_decoder_utilization_synth.rpt -pb Turbo_decoder_utilization_synth.pb
| Design       : Turbo_decoder
| Device       : xc7z045
| Design State : Synthesized
------------------------------------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        |  Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             | 85415 |     0 |    218600 | 39.07 |
|   LUT as Logic          | 85415 |     0 |    218600 | 39.07 |
|   LUT as Memory         |     0 |     0 |     70400 |  0.00 |
| Slice Registers         | 10808 |     0 |    437200 |  2.47 |
|   Register as Flip Flop |  7034 |     0 |    437200 |  1.60 |
|   Register as Latch     |  3774 |     0 |    437200 |  0.86 |
| F7 Muxes                |   417 |     0 |    109300 |  0.38 |
| F8 Muxes                |    80 |     0 |     54650 |  0.14 |
+-------------------------+------+-------+-----------+-------+
```

Figure 6.2 Synthesis report (part 1 of 3)



```
/home/john/Desktop/Turbo Decoder/TRY/Turbo Decoder/Turbo Decoder/NEW/updat
3. DSP
------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| DSPs           |  296 |     0 |       900 | 32.88 |
|   DSP48E1 only |  296 |       |           |       |
+----------------+------+-------+-----------+-------+


4. IO and GT Specific
---------------------

+-------------------------------+------+-------+-----------+-------+
|           Site Type           | Used | Fixed | Available | Util% |
+-------------------------------+------+-------+-----------+-------+
| Bonded IOB                    |  115 |     0 |       362 | 31.76 |
| Bonded IPADs                  |    0 |     0 |        50 |  0.00 |
| Bonded OPADs                  |    0 |     0 |        32 |  0.00 |
| Bonded IOPADs                 |    0 |     0 |       130 |  0.00 |
| PHY_CONTROL                   |    0 |     0 |         8 |  0.00 |
| PHASER_REF                    |    0 |     0 |         8 |  0.00 |
| OUT_FIFO                      |    0 |     0 |        32 |  0.00 |
| IN_FIFO                       |    0 |     0 |        32 |  0.00 |
| IDELAYCTRL                    |    0 |     0 |         8 |  0.00 |
| IBUFGDS                       |    0 |     0 |       348 |  0.00 |
| GTXE2_CHANNEL                 |    0 |     0 |        16 |  0.00 |
| PHASER_OUT/PHASER_OUT_PHY     |    0 |     0 |        32 |  0.00 |
| PHASER_IN/PHASER_IN_PHY       |    0 |     0 |        32 |  0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY   |    0 |     0 |       400 |  0.00 |
| ODELAYE2/ODELAYE2_FINEDELAY   |    0 |     0 |       150 |  0.00 |
| IBUFDS_GTE2                   |    0 |     0 |         8 |  0.00 |
| ILOGIC                        |    0 |     0 |       362 |  0.00 |
| OLOGIC                        |    0 |     0 |       362 |  0.00 |
+-------------------------------+------+-------+-----------+-------+
```

Figure 6.3 Synthesis report (part 2 of 3)

```
/home/john/Desktop/Turbo Decoder/TRY/Turbo Decoder/
131
132 +----------+-------+----------------------+
133 | Ref Name |  Used | Functional Category |
134 +----------+-------+----------------------+
135 | LUT2     | 50709 |                  LUT |
136 | CARRY4   | 20374 |           CarryLogic |
137 | LUT4     | 14193 |                  LUT |
138 | LUT3     | 11651 |                  LUT |
139 | LUT6     | 10355 |                  LUT |
140 | FDRE     |  6664 |         Flop & Latch |
141 | LUT5     |  5986 |                  LUT |
142 | LDCE     |  3774 |         Flop & Latch |
143 | LUT1     |  3566 |                  LUT |
144 | MUXF7    |   417 |                MuxFx |
145 | DSP48E1  |   296 |     Block Arithmetic |
146 | FDPE     |   211 |         Flop & Latch |
147 | FDCE     |   127 |         Flop & Latch |
148 | OBUF     |    96 |                   IO |
149 | MUXF8    |    80 |                MuxFx |
150 | FDSE     |    32 |         Flop & Latch |
151 | IBUF     |    19 |                   IO |
152 | BUFG     |     4 |                Clock |
153 +----------+-------+----------------------+
154
155
156 8. Black Boxes
157 --------------
158
159 +----------+------+
160 | Ref Name | Used |
161 +----------+------+
162
163
164 9. Instantiated Netlists
165 ------------------------
166
167 +----------+------+
168 | Ref Name | Used |
169 +----------+------+
170
```

Figure 6.4 Synthesis report (part 3 of 3)

6.3 Error-Rate Performance and key characteristics

To achieve a good error-rate performance, the input LLRs are quantized to 5 bit, the extrinsic LLRs to 6 bit and all state metrics in the radix-4 ACS units require 10 bits. This Turbo-Decoder implements 5.5 full iterations to carry out the decoded bits.

The majority of chip area is occupied by the BCJR instances. The maximum measured clock frequency is 300 MHz, at which a throughput of 200 Mb/s has measured.

Chapter 7

Conclusions

In the recent years, high-throughput design and implementation have become dominating requirement in the field of VLSI design of wireless-communication systems. There has been a rapid surge in data-rate for next-generation wireless-communication and this will lead to more complex algorithms and VLSI architectures in next few decades. Based on this scenario, I have aggregated the study of turbo-code and the implementation of high-throughput parallel-turbo decoder on FPGA in this thesis. In this work it is detailed a parallel turbo decoder for the 3GPP-LTE standard. The use of radix-4 in combination with 8 parallel SISO decoders is of paramount importance in order to achieve high throughput and an area efficient design.

7.1 Future Work

For the future work, proposed VLSI-architecture of high-throughput parallel-turbo decoder can be re-designed into area-efficient architecture. Similarly, power-reduction techniques could be incorporated to conceive high-throughput architecture for low-power applications. Possible extensions in this project may be the following:

- *Windowing*
  To significantly reduce the large memory requirements, windowing can be employed. In this approach the trellis is processed in small windows.

- *Early termination*
  Decoders for turbo codes are iterative in nature. There are techniques that can be used to reduce the average number of iterations. There are stopping rules based on comparing a metric on bit reliabilities (soft bit decisions) with a threshold. If the metric is smaller than the threshold, the decoder continues with a new iteration; otherwise, it stops.

Bibliography

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding. Turbo codes", in Proc, Int. Conf. Communication, May 1993, pp 1064-1070.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near optimum error correcting coding and decoding. Turbo-codes," IEEE Trans. Commun., vol.44, no 10, pp. 1261- 1271, 1996.

[3] 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (EUTRA); Multiplexing and channel coding (Release 9), 3GPP Organizational Partners TS 36.212, Rev. 8.3.0, May 2008.

[4] C. E. Shannon and W. Weaver, The Mathematical Theory of Communication. Urbana, IL: Univ. Illinois Press, 1949.

[5] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "Design and implementation of a parallel turbo-decoder asic for 3gpp-lte," Solid-State Circuits, IEEE Journal of, vol. 46, no. 1, pp. 8 –17, jan. 2011.

[6] H.Zarrinkoub, Understanding LTE with MATLAB From Mathematical Modeling to Simulation and Prototyping, United Kingdom: John Wiley & Sons Ltd, 2014

[7] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. Inf. Th., vol. 20, no. 2, pp. 284–287, Mar. 1974.

[8] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," IEEE Trans. Inf. Th., vol. 13, no. 2, pp. 260–269, Apr. 1967.

[9] J. P. Woodard and L. Hanzo, "Comparative study of turbo decoding techniques: an overview," IEEE Trans. Vehicular Tech., vol. 49, no. 6, pp. 2208–2233, Nov. 2000.

[10] S. Vafi and T. Wysocki, "Performance of convolutional interleavers with different spacing parameters in turbo codes," Proceedings of Australian Communication Theory Workshop, pp. 8-12, 2005.

[11] S. Lee, C. Wang and W. Sheen, "Architecture Design of QPP Interleaver for Parallel Turbo Decoding," Proceedings of IEEE Vehicular Technology Conference (VTC), pp. 1-5, 2010.

[12] A. Nimbalker, Y. Blankenship, B. Classon, and T. K. Blankenship, “ARP and QPP interleavers for LTE turbo coding,” in Proc. IEEE WCNC, Las Vegas, NV, USA, Mar. 2008, pp. 1032–1037.

[13] C-C. Wong and H-C. Chang, “Reconfigurable Turbo Decoder With Parallel Architecture for 3GPP LTE System,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 57, pp. 566-570, July-2010.

[14] 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, vol. 45, no. 2, pp. 422-432, February-2010.

[15] S. M. Karim and I. Chakrabarti, “High Throughput Turbo Decoder Using Pipelined Parallel Architecture and Collision Free Interleaver,” IET Communications, vol. 6, pp. 1416-1424, 2012.

[16] Y. Sun and J. R. Cavallaro, “Efficient Hardware Implementation of a Highly-Parallel 3GPP LTE/LTE-Advance Turbo Decoder,” INTEGRATION, the VLSI Journal, vol. 44, pp. 305-315, 2011.

[17] O. Y. Takeshita, “On maximum contention-free interleavers and permutation polynomials over integer rings,” IEEE Trans. Inf. Th., vol. 52, no. 3, pp. 1249–1253, Mar. 2006.

[18] J. Sun and O. Y. Takeshita, “Interleavers for turbo codes using permutation polynomials over integer rings,” IEEE Trans. Inf. Th., vol. 51, no. 1, pp. 101–119, Jan. 2005.

[19] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, “VLSI architectures for metric normalization in the Viterbi algorithm,” in Proc. IEEE ICC, vol. 4, Atlanta, GA, USA, Apr. 1990, pp. 1723–1728.

[20] ZC706 Evaluation Board for the Zynq-7000 XC7Z045 All Programmable SoC User Guide.