Politecnico di Torino

Master's Degree Thesis

High Throughput Turbo-Decoders

Concurrent-PMAP and High-Radix Exploration

Supervisors:

Prof. Guido MASERA
Prof. Maurizio MARTINA

Candidate:

Simone FAVERO

Master's Degree in Electronic Engineering

Abstract

In digital communications, the throughput demand is rapidly increasing through years. Since modern standards have already reached throughputs above 1 Gb/s, it is expected, in the following years, to reach requirements up to 1 Tb/s.

Channel-Coding is a fundamental technique employed in communication links, in order to identify and correct errors. Therefore, it should be adapted aiming to support the near future throughput requirements. Turbo-Codes are a promising choice in this scenario, due to their capability to work close to the Shannon's Limit. Therefore, the suitability of Turbo-Decoder architectures is object of study.

Focusing on Turbo-Decoders, this work is proving how high degrees of parallelization are necessary to satisfy the discussed data-rate requirements. Therefore, a detailed study on the Concurrent-PMAP architecture, implemented on different radix-orders, is proposed. The aim is to highlight guidelines to select area efficient implementations, capable to push throughputs toward the next-future demand.

The results are showing how the presented architecture is a valid candidate for high-throughputs, considering radix-2 and radix-4 implementations. Moreover, a general approach to develop a comparison model between architectures is detailed, aiming to speed-up the selection process among new proposed solutions.

Contents

Acronyms **7**

1 Introduction **18**

1.1 The High-Throughput Challenge 19

1.2 Organization and Objectives 20

2 Channel Coding **22**

2.1 Digital Communication Link 22

2.2 Noise and Errors 24

3 Convolutional Codes **26**

3.1 Codes Classification 26

3.2 Encoding 27

3.2.1 LTE/UMTS Standard Code 28

3.2.2 Trellis-Diagram 29

3.2.3 Hamming Distance 31

3.2.4 Code Termination 32

3.2.5 Puncturing and High Code-Rate 32

3.3 Decoding 33
3.3.1 The Log-MAP Algorithm 34
3.3.2 The Max-Log-MAP Algorithm 37

4 Turbo-Codes **39**
4.1 Turbo-Encoding 39
4.1.1 Interleaver 40
4.2 Turbo-Decoding 41
4.3 Algorithms Comparison 43
4.4 SISO-Decoder Building Blocks 44

5 Parallelization **46**
5.1 Key Performance Indicators 46
5.2 Baseline Architecture 48
5.3 Algorithm Parallelism 49
5.3.1 Windowing 50
5.4 Trellis Parallelism 53
5.5 SISO Parallelism 55
5.6 Decoder Parallelism 58
5.7 Iteration Parallelism 58

6 BER Performance **60**
6.1 BER Analysis 60
6.1.1 Window Size 61

6.1.2 Sub-Frame Size . . . 62
6.1.3 Channel-LLR Bitwidth . . . 62
6.1.4 Permutation Law . . . 63
6.1.5 Max* Correction . . . 64
6.2 Quantization and Bitwidths . . . 64

7 State of the Art Architectures 69
7.1 PMAP . . . 69
7.2 XMAP . . . 70
7.3 FPMAP . . . 72
7.4 UXMAP . . . 73
7.5 Conclusions . . . 74

8 High-Radix PMAP Exploration 76
8.1 Methodology . . . 77
8.2 Assumptions . . . 77
8.3 Logic Analysis . . . 79
8.3.1 BMU . . . 79
8.3.2 PMU . . . 80
8.3.3 SOU . . . 84
8.3.4 Logic Units Summary . . . 85
8.3.5 GE Model . . . 87
8.4 Memory Analysis . . . 89
8.4.1 Storage Requirements . . . 90

8.4.2 Memory Conflicts 90
8.4.3 Access Policies 92
8.4.4 GE Model 97
8.5 Throughput 101
8.6 Uncertainty 103
8.7 Full Architecture 104

9 Model Results **107**
9.1 Tool Organization 107
9.2 Solutions Space Exploration 108
9.3 Efficient Solutions 116
9.3.1 Architecture 1 116
9.3.2 Architecture 2 119
9.4 High-Radix Limitation 121
9.5 Very High Throughputs 123

10 Model Validation **128**
10.1 Logic Units Validation 128
10.2 Architectures Validation 130
10.2.1 Architecture 1 131
10.2.2 Architecture 2 132
10.2.3 Conclusions 134

11 Conclusions and Future Works **135**

11.1 Conclusions 135

11.2 Future Works 136

A Recursive Trellis Exploration **138**

B Fast CS Operators **140**

B.1 Look-Ahead Logic CS2 140

B.2 Multiple Inputs CS 142

C Radix and Scheduling Aware Memory-Mapping **146**

Acronyms

ACS Add-Compare-Select. 44

ALPHA-MEM Alpha-Memory. 89–91, 95, 110, 117–120, 132

APP A-Posteriori-Probability. 34, 36, 38, 84–87

ARP Almost Regular Permutation. 41, 63, 93

AWGN Additive-White-Gaussian-Noise. 24

BER Bit-Error-Rate. 24, 25, 31, 37, 43, 47, 60, 63, 64, 73, 127, 135

BMU Branch Metric Unit. 44, 48–51, 70, 71, 79, 80, 83–87, 128, 139

BPSK Binary-Phase-Shift-Keying. 23, 24

CPMAP Concurrent-PMAP. 1, 20, 21, 75, 76, 78, 123, 124, 135–137

CS2 Compare-and-Select. 81–84, 140

ECC Error Correcting Code. 18, 19, 26

ESF Extrinsic-Scaling-Factor. 38, 43, 44, 64, 78, 85, 130, 135

EXTR-INF-MEM Extrinsic-Information-Memory. 12, 89–95, 118, 120, 131, 132

FB Forward-Backward. 49, 50, 52, 69, 70, 78, 79, 89–91, 110, 147, 148

FEC Forward Error Correction. 18

FER Frame-Error-Rate. 24

FI Full-Iteration. 42, 47

FIR Finite Impulse Response. 28

FPMAP Fully-Parallel-MAP. 69, 72, 75

FSM Finite-State-Machine. 29

GE Gate-Equivalent. 87, 89, 97

HI Half-Iteration. 42, 48, 101

IIR Infinite Impulse Response. 28

IN-FR-MEM Input-Frame-Memory. 12, 89–92, 117, 118, 120, 131, 132

KPI Key Performance Indicators. 19, 46, 47, 74, 123

LDPC Low-Density-Parity-Check. 19

LLR Log-Likelihood-Ratio. 23, 24, 33–35, 41, 44, 62, 64, 65, 71, 79, 89, 128

LM Log-Map. 20, 37, 43, 44, 64

LTE Long Term Evolution. 20, 26, 28, 34, 41, 60, 63, 67, 78, 135, 138

LUT Look-Up-Table. 35, 37, 64, 103, 143, 145

MAP Maximum-A-Posteriori. 34

MLM Max-Log-Map. 43, 44, 48, 49, 51, 53, 60, 64, 65, 67, 78, 135

MSB Most-Significant-Bit. 140, 141

NII Next-Iteration-Initialization. 51, 52, 70, 78

NII-MEM NII-Memory. 12, 89–91, 96, 97, 110, 118, 120

PMAP Parallel-MAP. 20, 69, 71, 75, 78, 79, 101, 108, 114, 136, 146

PMU Path Metric Unit. 44, 45, 48–51, 70, 79, 81, 83–87, 95, 101, 108, 111, 121, 122, 128, 139, 140, 142

QPP Quadratic Permutation Polynomial. 41, 63, 93

RCA Ripple-Carry-Adder. 88

RCS Ripple-Carry-Subtractor. 88

RSC Recursive Convolutional Encoder. 39

SIHO Soft-Input-Hard-Output. 33

SISO Soft-Input-Soft-Output. 33, 36, 41

SNR Signal-to-Noise-Ratio. 24

SOU Soft Output Unit. 44, 45, 48–52, 56, 70, 71, 79, 84–87, 92, 95, 128, 139

SOVA Soft-Output-Viterbi-Algorithm. 43

STG State Transition Graph. 29

UMTS Universal Mobile Telecommunications System. 20, 26, 28, 34, 41, 60, 63, 67, 78, 135, 138

UXMAP Unrolled-X-MAP. 20, 69, 73, 75, 124, 125, 127, 133, 135, 136

XMAP X-MAP. 69–71, 73, 75, 133

List of Figures

2.1 Point-to-Point communication link block scheme. 22

2.2 IQ constellation diagram for the BPSK modulation format. 23

2.3 BER as function of E_b/N_0 considering different code-rates R. 25

3.1 Convolutional code block scheme, considering $v = 2$, $k = 2$, $n = 3$. 27

3.2 Encoding block scheme following the example polynomial generator matrix. 28

3.3 Encoding block scheme considering the LTE/UMTS standard code. 29

3.4 Trellis-Diagram for the LTE/UMTS standard code, considering a single Trellis-section. 30

3.5 Trellis-Diagram for the LTE/UMTS standard code, considering K Trellis-sections. 31

3.6 A representation of the Log-Map algorithm on the Trellis-Diagram. 37

4.1 Turbo-encoder block scheme including two parallel RSC. 40

4.2 Turbo-decoder block scheme including two SISO modules. 42

4.3 BER as function of E_b/N_0, considering different algorithms applied in a Turbo-decoder architecture. 43

4.4 SISO-decoder block scheme example. 44

4.5 Internal structure of the main SISO-decoder logic units. 45

5.1 Baseline Turbo-decoder architecture, obtained without exploiting parallel computations. 48

5.2 Forward-Backward scheduling - graphical representation in time. 49

5.3 Butterfly scheduling - graphical representation in time. 50

5.4 Windowing applied to the Forward-Backward and Butterfly scheduling policies. 51

5.5 Acquisition and NII - graphical representation in time. 52

5.6 Radix-4 benefits against a Radix-2 architecture. 53

5.7 Radix-4 approach represented on a Trellis-Diagram. 54

5.8 SISO concurrency, considering FB scheduling and windowing. 55

5.9 Concurrent SISO half-iteration block scheme. 56

5.10 Sequential SISO considering FB scheduling and windowing. 56

5.11 Sequential SISO half-iteration block scheme. 57

5.12 Concurrent and sequential Decoder-parallelism block schemes, considering a half-iteration. 58

5.13 Iteration parallelism applied to Decoder-parallelized architectures. 59

6.1 BER variation as function of the window size WS. 61

6.2 BER variation as function of the sub-frame size Kp. 62

6.3 BER variation as function of the channel LLR bitwidth. 63

6.4 BER variation as function of the permutation law. 64

6.5 BER variation as function of the max* correction approach. 65

6.6 Modulo normalization condition graphically represented. 67

7.1 Scheduling and block scheme organization for the PMAP architecture. . . . 70

7.2 Scheduling and block scheme organization for the XMAP architecture. . . 71

7.3 Block scheme organization for the FPMAP architecture. 72

7.4 Block scheme organization for the UXMAP architecture. 73

8.1 BMU-N block scheme. 80

8.2 Amount of paths to be discriminated during a forward propagation, considering a radix-2 and a radix-4 implementation. 81

8.3 CS2 (Left) and CS4 (Right) block scheme. 81

8.4 CS4-fast block scheme (Left) and CS8 implementation including a CS4-fast operator (Right). 82

8.5 Parallel paths example considering a radix-16 Trellis-section. 83

8.6 Minimum CS2 SOU block scheme, considering a radix-16 approach. 85

8.7 Parallel accesses visually represented in a generic time instant. 91

8.8 IN-FR-MEM and EXTR-INF-MEM organization in order to manage the access to the systematic information. 92

8.9 Mapping-matrix example, highlighting with different colors the parallel accesses during both natural and interleaved processing. 94

8.10 Block scheme representing the EXTR-INF-MEM accesses controlled by crossbar and memories, which content is derived from the mapping-matrix. 95

8.11 Alpha memory access policy example, considering two windows with 8 Trellis-sections each. 96

8.12 NII-MEM access policy block scheme. 97

8.13 GE/bit variation model for library memories, as function of the number of words. 99

8.14 GE/bit complete model, considering both library and synthesized memories.101

8.15 Logic units located on the critical path considering different radix-orders. . 102

8.16 Radix-2 SISO block scheme, showing connections between logic units and memories. 105

8.17 Concurrent-PMAP block scheme, with shared elements and including unique global memories. 106

9.1 Analysis tool block scheme, representing the main function groups included. 108

9.2 Area-Efficiency, Throughput variation (Top) and Most Efficient Radix-Order (Bottom) as function of the Frame-Size K. 109

9.3 Area-Efficiency, Throughput variation (Top) and Most Efficient Radix-Order (Bottom) as function of the Sub-Frame-Size Kp. 110

9.4 Area-Efficiency, Throughput variation (Top) and Most Efficient Radix-Order (Bottom) as function of the Window-Size WS. 111

9.5 Area-Efficiency, Throughput variation (Top) and Most Efficient Radix-Order (Bottom) as function of the Channel LLR bitwidth w. 112

9.6 Most efficient radix-orders as function of K, Kp and WS. 113

9.7 Area efficiency as function of K, Kp and WS, using the radix-orders indicated in 9.6. 114

9.8 Area efficiency as function of K, Kp and WS - Parametric plot. 114

9.9 Throughput as function of K, Kp and WS. 115

9.10 Maximum throughput with parallel Turbo-decoders on a given area as function of K, Kp and WS. Two areas are considered: 4 mm^2 (Left) and 16 mm^2 (Right). 115

9.11 Most efficient radix-orders (Left) and area efficiency (Right) as function of Kp and WS, considering $K = 6144$. 116

9.12 Reliability analysis results considering *Architecture 1*. 118

9.13 Reliability analysis results considering *Architecture 2*. 120

9.14 Most efficient radix-orders as function of K, Kp and WS, assuming the same working clock frequency for all the radix-orders. 121

9.15 Maximum throughput with concurrent PMAPs on a 10 mm^2 area. Technology: 65 nm. 123

9.16 Area efficiency distribution as function of K, Kp and WS. Technology: 28 nm (Scaled). 125

9.17 Maximum throughput with concurrent PMAPs on a 16.5 mm^2 area. Technology: 28 nm (Scaled). 125

9.18 Area efficiency distribution. Technology: 7 nm (Scaled). 126

9.19 Maximum throughput with concurrent PMAPs on a 10 mm^2 area. Technology: 7 nm (Scaled). 126

10.1 Testbench block scheme for logic units. 129

A.1 Trellis-Diagram recursive exploration algorithm - block scheme. 139

B.1 Look-Ahead based CS2. 141

B.2 Proposed steps to compute the LUT content - block scheme. 144

B.3 CSN-fast, proposed implementation block scheme. 145

C.1 Natural and interleaved parallel accesses in time - Radix 2, no scheduling policy. 147

C.2 Natural parallel accesses in time - Radix 4, no scheduling policy. 147

C.3 Natural parallel accesses in time - Radix 2, Forward-Backward scheduling. 148

C.4 Natural parallel accesses in time - Radix 4, Forward-Backward scheduling. 148

List of Tables

5.1 EPIC KPI requirements considering different use cases. 47

5.2 Comparison table on windowing applied with Forward-Backward and Butterfly scheduling. 51

6.1 Reference C-model parameters for the BER analysis. 61

6.2 Branch metrics bitwidths as function of w, considering different radix orders. 66

6.3 Δ_{Γ}^{max} as function of w considering different radix orders. 68

6.4 Fundamental bitwidths summarized for different radix orders, considering $w = 6$ bits. 68

7.1 Comparison table between different implemented architectures, considering the PMAP, XMAP, FPMAP and UXMAP approaches. 74

8.1 Number of Branch-Metrics requested to be computed by a radix-N BMU, including their bitwidths. 79

8.2 BMU components analysis up to radix-16 implementation, including the total amount of required adders. 80

8.3 Compare-and-Select operators analysis up to radix-16 implementation. . . 82

8.4 Max() operators analysis up to a radix-16 implementation. 84

8.5 Radix-2 required operators. 86

8.6 Radix-4 required operators. 86

8.7 Radix-8 required operators (CS4-fast usage). 87

8.8 Radix-16 required operators (CS4-fast usage). 87

8.9 Area synthesis results on the considered operators, considering a bitwidth equal to 8-bits and three available technologies. 88

8.10 Memory storage requirements considering a radix-2 architecture. 90

8.11 Required parallel memory accesses on different radix orders. 91

8.12 Crossbar area synthesis results on different technologies. 97

8.13 GE/bit data on different library memory models, considering the 65 nm technology. 98

8.14 GE/bit variation on synthesized memories, considering the 65 nm technology. 100

8.15 Critical path synthesis data on different technologies, considering different radix-orders and data expressed on 11 bits. 102

8.16 Working clock frequencies included in the developed model considering different radix-orders, with a bitwidth equal to 11 bits. 103

9.1 Parameters default values and variation ranges for the proposed analysis. . 108

9.2 K, Kp and WS proposed variation ranges for the global analysis. 113

9.3 Comparison indicators for *Architecture 1* considering different radix-orders. 117

9.4 Logic units area distribution in a SISO-decoder, considering *Architecture 1* and a radix-4 implementation. 117

9.5 Memory sizes, partitioning and area distribution considering *Architecture 1.* 118

9.6 Considered relative errors for the reliability analysis on *Architecture 1.* . . . 118

9.7 Comparison indicators for *Architecture 2* considering different radix-orders. 119

9.8 Logic units area distribution in a SISO-decoder, considering *Architecture 2* and a radix-2 implementation. 119

9.9 Memory sizes, partitioning and area distribution considering *Architecture 2.* 120

9.10 CS8 model comparisons on radix-8 and radix-16 architectures. 122

9.11 Area scaling factors from the 65 nm technology to other technological nodes. 124

10.1 Logic units area comparison between model results and synthesis results. . 129

10.2 Critical path synthesis data and comparison with the increment factors employed in the developed model. 130

10.3 Library memory models for *Architecture 1*. 131

10.4 *Architecture 1*, Radix-2 - model and synthesis data comparison. 131

10.5 *Architecture 1*, Radix-4 - model and synthesis data comparison. 132

10.6 Library memory models for *Architecture 2*. 132

10.7 *Architecture 2*, Radix-2 - model and synthesis data comparison. 133

10.8 *Architecture 2*, Radix-4 - model and synthesis data comparison. 133

B.1 Area and critical path comparisons between CS2 models based on subtractors and look-ahead logic. 142

B.2 Logic results from the set of 6 comparisons included in a CS4-fast operator. 143

B.3 Area and critical path comparisons between the two available approaches to realize a CS4-fast operator. 145

Chapter 1

Introduction

In digital communications, the capability to detect and correct errors is fundamental to guarantee the validity of the received information. Another primary parameter is the amount of useful data that can be transmitted and received per unit time. The two previous concepts are at the basis of modern wireless communication standards, which are aiming to manage the transferring of information in a fast and reliable way.

During the previous decades, the demand for high-speed communication services, especially wireless ones, has increased, setting higher standards for the required throughput. It is enough to consider that the fifth generation standard for cellular networks, 5G, is able to achieve throughputs above 1 Gb/s. A projection of this increasing demand in the following years could potentially lead to throughput requirements in the order of hundreds of Gb/s, up to 1 Tb/s.

As mentioned before, a remarkable importance is assigned to the error correction capability in a digital communication system, which is required to be adapted considering the high-throughput requirements.

Given a noisy channel, errors can be detected and corrected employing the *Forward Error Correction (FEC)* or *channel coding* theory. The key idea is to add redundant information to the original information to be transmitted, employing it to improve the reliability of the received data, by identifying and correcting errors.

An *Error Correcting Code (ECC)* can be described as an algorithm to be applied on the original information to be transmitted, in order to properly select the redundant information. Different ECC can be found in literature, with the common aim of reducing the error-rate at the receiver side. The maximum boundary for the data rate, considering an error-free transmission on a noisy channel, is expressed by the *Shannon's Theorem*, developed by Claude Shannon in 1948 [1]. The content of the theorem is briefly summarized below.

Given a noisy channel, characterized by a maximum transmission rate (Capacity C), if considering a generic transmission rate for useful information $R < C$, it is possible to establish an ECC which is capable to reduce the error probability at the receiver to an arbitrary low value.

As an immediate consequence of the theorem, the usage of error correction could potentially lead to an error-free communication, considering a rate really close to the channel capacity. This powerful result is one of the main reasons why ECC have been object of study in the past, as well as in the present.

A fundamental step in coding theory is the introduction of *Turbo-Codes*, carried out by C. Berrou in 1991 [2]. This specific category of codes is able to work close to the limit imposed by the Shannon's Theorem, which makes them optimal when considering rigid requirements on the information rate. Following these considerations, Turbo-Codes are one of the main choices for communication standards. Therefore, they will be considered as a reference ECC in this work.

1.1 The High-Throughput Challenge

Moving toward high throughput requirements, channel coding is expected to be technologically adapted in order to process the information properly. Following this purpose, the European project *EPIC* delivered, during 2020, important contributions to the design of ECC technologies, considering the expected requirements for different use cases in the near future.

The project highlights the importance of searching for new algorithmic and architectural solutions, since the improvements in technology, controlled by the *Moore's Law*, are not expected to completely cover the future requirements [3].

The EPIC project considered three different types of codes as feasible candidates: Turbo-Codes, Low-Density-Parity-Check (LDPC) codes and polar codes. With the aim of comparing the different solutions and outline guidelines for the use cases under analysis, different *Key Performance Indicators (KPI)* have been established. The KPI list is including the *throughput* definition. Moreover, also the *code flexibility* is considered as a fundamental property. Part of the KPI will be discussed in the following chapters and used as metrics to compare different solutions.

1.2 Organization and Objectives

This work aims to explore very high-throughput Turbo-decoder architectures, with a particular accent on the usage of different degrees of parallelization, including high-radix approaches. The main objectives are presented.

- A classification of the fundamental parallelization degrees, employed in modern Turbo-decoders architectures, will be detailed, highlighting the main advantages and disadvantages for each technique.
- A generic approach to model comparisons among different architectural solutions will be presented, aiming to easily explore large architectural spaces and define the guidelines to select the best design choices, given a set of requirements. In particular, the presented model will be adapted to analyze PMAP-based architectures, aiming to explicit which design choices allow the introduction of high-throughput solutions.
- The use of different radix-orders will be studied on PMAP-based architectures, in order to explicit the suitable choices in terms of high-throughput and area efficiency. Fundamental solutions will be proposed, aiming to cover the issues presented by high-radix designs. Moreover, it will be proved how radix-2 and radix-4 architectures can be considered solid choices, depending on the specifications. Furthermore, guidelines in order to open the possibility for efficient radix-8 and radix-16 implementations will be discussed.
- The Concurrent-PMAP architecture will be pushed toward high degrees of parallelism, aiming to maximize the achievable throughput. The objective is to prove the suitability of this Turbo-decoder implementation, especially if compared to the UXMAP architecture, highlighting the necessity for a more detailed comparison between PMAP and XMAP based solutions, especially when high degrees of parallelization are employed.

The structure and the content of this work are briefly summarized, considering the included chapters.

- **Chapter 2 - Channel Coding**: This chapter is introducing the concept of channel coding, considering a generic digital communication link. Important definitions and metrics are also mentioned, highlighting the coding benefits on the error-correction performances.
- **Chapter 3 - Convolutional Codes**: This chapter is presenting the concept of convolutional codes. First, a focus on the encoding operation is included, introducing the LTE/UMTS standard code. Then, decoding algorithms are covered, with a particular accent on the Log-Map algorithm.

- **Chapter 4 - Turbo Codes**: This chapter is introducing the definition of Turbo-Codes, highlighting how encoding and decoding operations are handled. Moreover, important architectural building blocks are presented.
- **Chapter 5 - Parallelization**: This chapter is providing a classification of the fundamental parallelization degrees, suitable for Turbo-decoder architectures. The main benefits and drawbacks are discussed for each technique.
- **Chapter 6 - BER Performance**: This chapter is presenting quantitative considerations about the effect of some architectural choices on the error-correction performances. Therefore, guidelines to introduce an acceptable BER are highlighted. Moreover, quantization techniques employed to represent algorithm parameters in the architecture are discussed.
- **Chapter 7 - State-of-the-Art Architectures**: This chapter is reviewing fundamental State-of-the-Art architectures, highlighting their main advantages and disadvantages and comparing some practical implementations. The importance of introducing high degrees of parallelism is remarked.
- **Chapter 8 - High-Radix PMAP Exploration**: This chapter is introducing the guidelines to build a comparison model among different architectures. The presented steps are employed to explore the use of high-radix approaches in PMAP-based Turbo-decoders. The development of the comparison tool is detailed. Logic and memory organizations are discussed in-depth, highlighting the main challenges and introducing specific solutions.
- **Chapter 9 - Model Results**: This chapter is presenting the results obtained with the developed comparison model. The suitability of high-radix solutions in a PMAP-based architecture is discussed, covering the limitations of radix-orders higher than 4. Two detailed examples of efficient radix choices are included. Moreover, very high-throughput solutions are analyzed, highlighting the suitability of the Concurrent-PMAP architecture.
- **Chapter 10 - Model Validation**: This chapter is validating the results provided by the comparison model. Synthesis operations are performed in order to collect area and critical path estimations. Model and synthesis results are compared, showing the relative errors and introducing key-points to further improve the model accuracy.
- **Chapter 11 - Conclusions and Future Works**: This chapter is summarizing the results obtained in this work, introducing possible steps for future studies.

Chapter 2

Channel Coding

In this chapter, some basic concepts about channel coding will be covered, considering a generic communication link. Meanwhile, some important notation will be introduced, as well as the definition of several significant metrics. In the last part of the chapter, a review of the channel coding benefits on the error-rate is presented.

2.1 Digital Communication Link

A typical point-to-point digital communication link is summarized by the block scheme presented in figure 2.1.

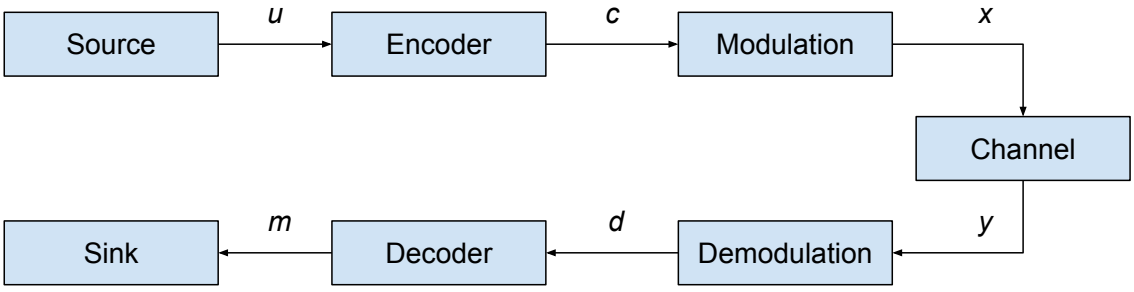

Figure 2.1: Point-to-Point communication link block scheme.

The **Source** is producing the original information u, which is a K-length sequence of bits. The **Encoder** is generating the codeword c, a N-length bits sequence, which is the encoded information obtained starting from u. Since the **Encoder** is expected to add

redundant information, $N > K$. An important metric to express the amount of redundancy is the *code-rate*, evaluated as follows.

$$R = \frac{K}{N}$$

During the **Modulation**, the analog signal employed to transmit the information is properly modified following a specific modulation policy, in order to embed the codeword information in it. The obtained signal x can then be transmitted on the **Channel**. Different modulation schemes are available, introducing trade-offs between bit-rate and error-rate. In this work, the *Binary-Phase-Shift-Keying (BPSK)* modulation format is considered, which is working on the phase of the analog signal. The constellation is composed by only 2 symbols, which are selected according to the value of each codeword bit, following the constellation diagram in figure 2.2. The subscript k in the figure is referring to a single bit/symbol information in the bit-sequence.

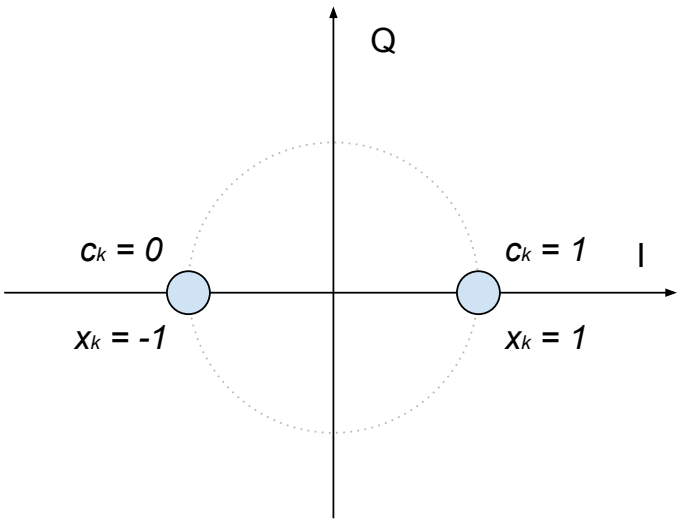

Figure 2.2: IQ constellation diagram for the BPSK modulation format.

The **Channel** is expected to add noise to the signal, as well as other effects like distortion and interference. The **Demodulation** operation is supposed to retrieve information on the received codewords. Due to the **Channel** effects, each symbol is correctly demodulated within a given probability. The demodulated sequence of symbols d is expressed relying on the definition of *Log-Likelihood-Ratio (LLR)*, a specific metric capable to express information on the reliability of each demodulated symbol. The LLR definition will be detailed in the following section.

The last important step to be highlighted is the **Decoding** operation, which is able, starting from the LLR information, to generate the estimated received message m. If m is not affected by errors, it is expected to be equal to u. In a simulation environment, errors can be estimated by directly comparing u and m.

2.2 Noise and Errors

As mentioned in the previous section, the channel is typically introducing some noise superimposed to the modulated signal, generating a source for possible errors at the receiver side. In this work, the channel will be considered as a *Additive-White-Gaussian-Noise (AWGN)* one. As a consequence, a received symbol can be written using the following notation.

$$y_k = x_k + n_k$$
$$n_k \approx \mathcal{N}(0, \sigma^2)$$

The superimposed noise n_k is represented as a *Normal distribution* with average equal to 0 and variance equal to σ^2. Starting from this specification, it is possible to write the conditional probability of the received symbol y_k.

$$P(y_k|x_k) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(y_k - x_k)^2}{2\sigma^2}\right)$$

The *Likelihood-Ratio* for a received symbol, considering a BPSK modulation, can be derived as follows.

$$\frac{P(y_k|x_k = 1)}{P(y_k|x_k = -1)} = exp\left(\frac{2y_k}{\sigma^2}\right)$$

If a logarithm is applied to the final result, the Log-Likelihood-Ratio (LLR) is defined.

$$L(y_k) = \frac{2y_k}{\sigma^2}$$

The LLR can be either positive or negative, its sign is embedding the information about the received symbol y_k. The magnitude of this quantity is instead related to the reliability of the received symbol. For instance a positive and large LLR value is stating that x_k is expected to be equal to 1, with a reliable decision by the receiver.

The overall robustness of the system against errors can be evaluated through the *Bit-Error-Rate (BER)*, representing the amount of wrong received bits per unit time. Another useful metric in this scenario is the *Frame-Error-Rate (FER)*, which is considering the amount of wrong information frames (K-length bits sequences) per unit time. Typically, those two quantities are strongly related to the *Signal-to-Noise-Ratio (SNR)* of the system, which is the ratio between the signal power and the noise power in the channel. However, in digital systems, it is often employed a normalized version of the SNR, defined as the ratio between the bit-energy E_b and the noise power density N_0. The higher this ratio, the lower the impact of the AWGN introduced by the channel, resulting in a lower BER.

A complete view on the error-correction performances of a generic digital communication system is given by graphically representing the BER as function of E_b/N_0. A typical graph is reported in figure 2.3, extracted from [4].

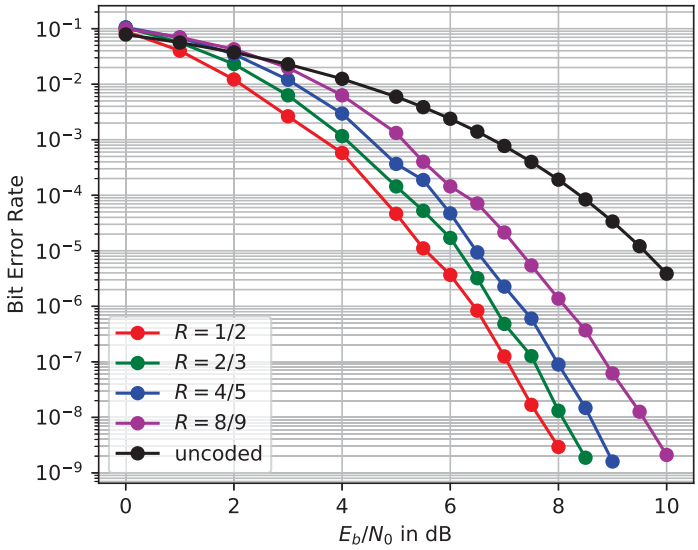

Figure 2.3: BER as function of E_b/N_0 considering different code-rates R.

As expected, considering all the represented coderates R, the curves are presenting lower BER values for higher E_b/N_0. The higher the amount of introduced redundant information (lower R), the better the error-correction capability.

The BER curve can be generally subdivided in 3 regions.

- **Low** E_b/N_0: In this region, the error-correction capabilities are limited, therefore the uncoded communication is the best option in terms of BER.
- **Medium** E_b/N_0 **(Waterfall region)**: In this region, the best BER reduction benefits from the use of codes are found.
- **High** E_b/N_0 **(Error floor region)**: In this region, the BER reduction slope is decreasing if compared to the waterfall region one, tending approximately to reach the uncoded slope. This last region is not visible in the graph above.

Codes are typically compared in terms of *coding gain*, which is the difference, usually expressed in dB, between two E_b/N_0 points, given a specific BER.

Chapter 3

Convolutional Codes

This chapter aims to introduce the class of convolutional codes, since they are representing the basic building block for Turbo-Codes. Then, encoding on convolutional codes will be covered, introducing fundamental definitions and classification metrics. In this section, the standard LTE/UMTS convolutional code will be presented. A specific decoding algorithm, the Log-Map, will be discussed in detail.

3.1 Codes Classification

Two main classes of ECC can be defined.

- **Block codes**: In this class, each message u, with length K, is mapped to a codeword c of length N, independently of the other messages. Since each message is considered separately, this type of codes are *memory-less*. The encoding operation can be mathematically described employing a (K x N) matrix.
- **Convolutional codes**: In this class, each message u, with length K, is subdivided in M sub-messages of length k, so that $M \cdot k = K$. Each k information bits are encoded on n bits. As a consequence, the final codeword length is $N = M \cdot n$. The encoding operation over k bits is taking into account the k bits to be encoded and some information on the previous encoding operations, stored in v memory steps. An example block scheme for a convolutional code is presented in figure 3.1.

A convolutional code can be described by 3 parameters (v, k, n). The code presented in the example can be characterized as (2, 2, 3). The code-rate R can be evaluated in two different ways, as K/N or k/n.

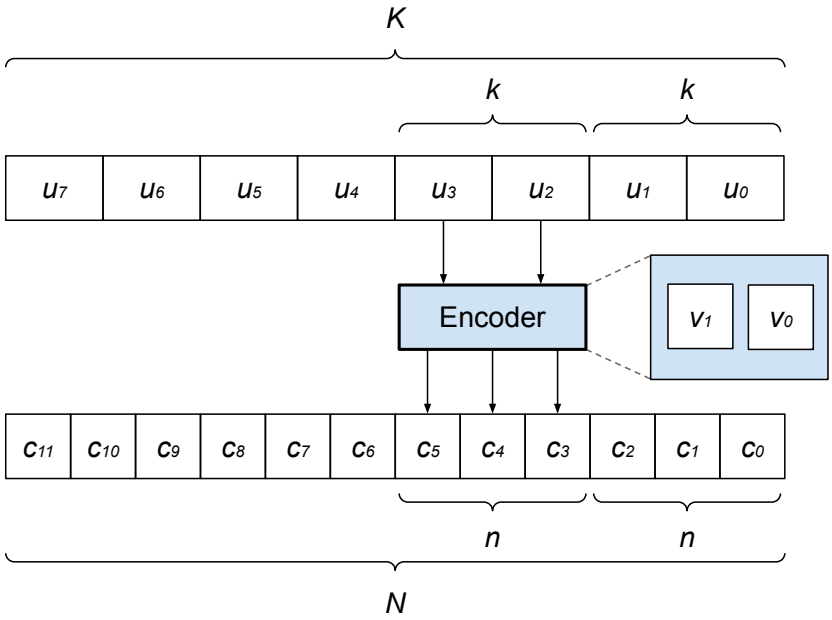

Figure 3.1: Convolutional code block scheme, considering $v = 2$, $k = 2$, $n = 3$.

3.2 Encoding

Given the parameters (v, k, n), the encoding operation can be described through a *polynomial generator matrix*, which is including (k x n) polynomials. An example of this matrix is presented below, for a code described by the set of parameters (2, 1, 2).

$$G(Z) = (1 + Z^{-1} + Z^{-2} \quad 1 + Z^{-2})$$

The polynomials included in the matrix are representing a set of digital filters, described by their transfer functions in Z domain. In this specific case, two codeword bits ($n = 2$) are produced starting from a single bit from the original message ($k = 1$), by employing the indicated filter functions. The maximum memory depth of the filters is fixed by the v parameter. A block scheme for this particular encoder function is represented in figure 3.2.

Starting from the characteristics of the filters included in the matrix, a classification for convolutional encoders can be easily derived.

- **Systematic**: In this type of encoders, part of the n output bits produced are an exact copy of the original message bits k, provided at the encoder input. In other words, the original information is part of the codeword. As a consequence, at least one filter function included in the matrix is simply represented with a 1.

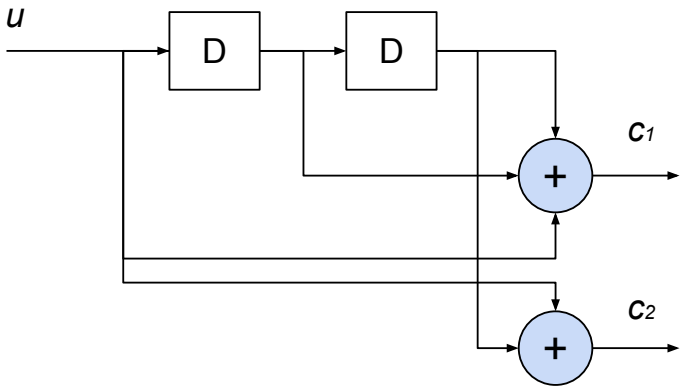

Figure 3.2: Encoding block scheme following the example polynomial generator matrix.

- **Non-Systematic**: This type of encoders are not embedding the original message bits in the final codeword.
- **Recursive**: In this type of encoders, at least one of the filter functions in the polynomial matrix is including a feedback loop. Equivalently, at least one filter is characterized by an *Infinite Impulse Response (IIR)*.
- **Non-Recursive**: The polynomial generator matrix for this type of encoders is including only filters characterized by a *Finite Impulse Response (FIR)*. Consequently, no feedback loops are found in the implementation.

3.2.1 LTE/UMTS Standard Code

Universal Mobile Telecommunications System (UMTS) [5] and *Long Term Evolution (LTE)* [6] recent standards are declaring, in the Turbo-Code section, the usage of a specific convolutional code, characterized by the parameters (3, 1, 2). The specific polynomial generator matrix is reported below, as well as the encoder block scheme, represented in figure 3.3.

$$G(Z) = \begin{pmatrix} 1 & \dfrac{1 + Z^{-1} + Z^{-3}}{1 + Z^{-2} + Z^{-3}} \end{pmatrix}$$

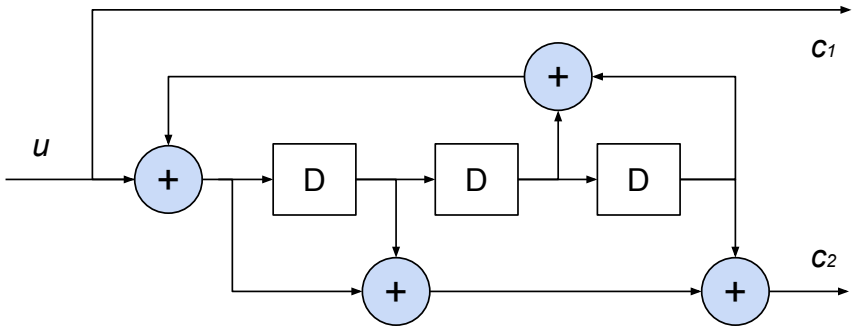

Figure 3.3: Encoding block scheme considering the LTE/UMTS standard code.

As noticeable from the polynomial equations and the block-scheme, this code is both *systematic* and *recursive*. This standard code will be considered as a reference choice in this work.

3.2.2 Trellis-Diagram

Since each output bit produced by the encoder is a linear combination between the encoder input and the registers' state, the overall encoding process can be summarized using a *Finite-State-Machine (FSM)*. The total number of states is 2^v. For instance, the reference code is characterized by $2^3 = 8$ states. Given a present state, the transition to the next state is decided by the encoder input and the present state itself. Typically, FSM are graphically represented employing a *State Transition Graph (STG)*, which is capable to explicit the transitions between states in a compact way.

However, when dealing with codes, the usage of a *Trellis-Diagram* representation is suggested. The key idea is to unroll the state transitions on a timeline, in order to explicit all the possible paths from a state to another. With the aim of better understanding this representation, the Trellis-Diagram for the reference code is reported in figure 3.4, just considering a generic state transition from S_k to S_{k+1}.

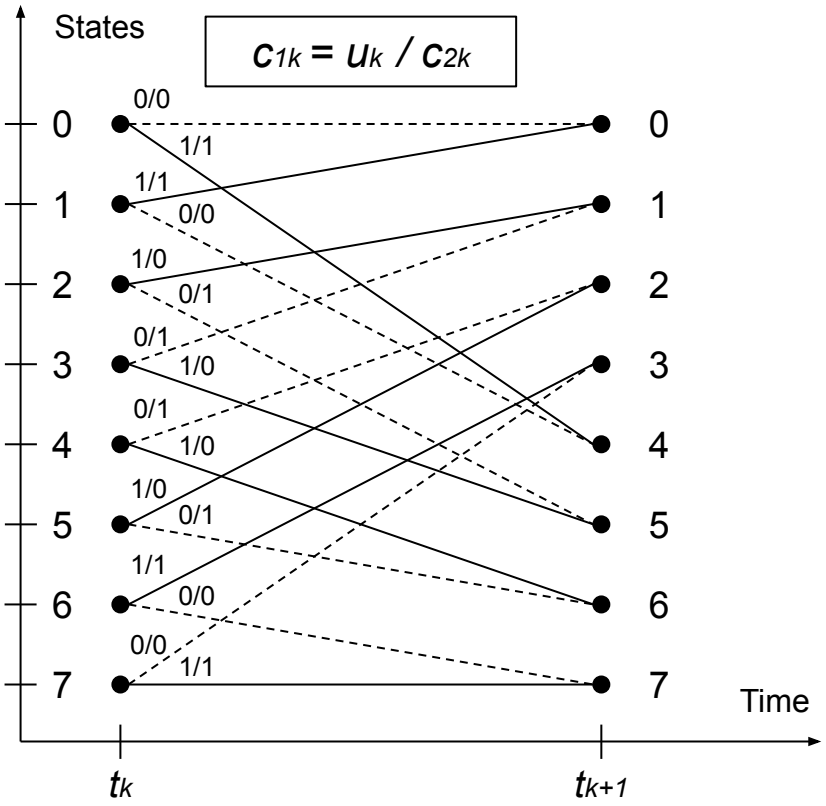

Figure 3.4: Trellis-Diagram for the LTE/UMTS standard code, considering a single Trellis-section.

Given the exact starting state S_k and the encoder input u_k, which is also representing the systematic information, it is possible to define the arrival state. Moreover, information about the calculated codeword bit c_{2k} is provided in the representation.

The lines connecting the states are called **branches**. As a convention, contiguous lines are related to branches with a systematic bit equal to 1, while dashed lines are representing branches with a systematic bit equal to 0. The portion of a Trellis-Diagram included between two generic time instants is defined as a **Trellis-section**.

Considering a K-bits source information u to be encoded, K adjacent Trellis-sections can be used to describe all the possible state transitions that could occur during the encoding operation.

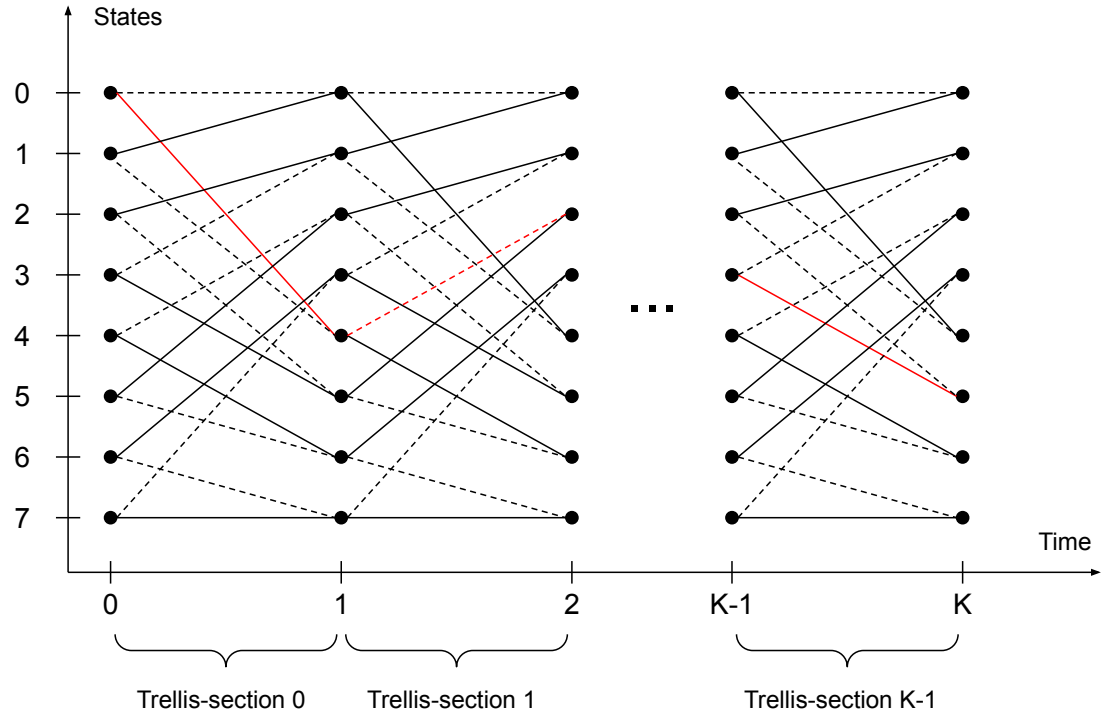

Figure 3.5: Trellis-Diagram for the LTE/UMTS standard code, considering K Trellis-sections.

A combination of branches from a generic starting state to a generic arrival state is considered a **path** inside the Trellis-Diagram. The red branches in figure 3.5 are defining an example path.

3.2.3 Hamming Distance

The Hamming distance between two codewords is defined as the number of positional bits that are different in the analyzed codewords. Given the set of codewords associated to a specific block of source information, it is possible to define the *minimum Hamming distance*, by comparing each codeword with all the others ones. The same approach can be carried out considering the set of all the possible codewords, called *codespace*. An example of Hamming distance calculation is reported below.

$$H_d(0010, 1000) = 2$$

The minimum Hamming distance has a strong connection with the error-correction capabilities of the code: the higher the distance value, the lower the expected BER, due to the "higher separation" in the considered set of codewords.

3.2.4 Code Termination

Typically, when encoding a K-bits information frame u, the initial state of the encoder is known. For instance, the internal memory elements can be initialized in order to start from the state $S_0 = 0$. The same assumption cannot be directly considered for the ending state S_{K-1}. As discussed in the next section, the information on the ending state can potentially increase the error-correction capabilities. For this reason, a brief review of the code termination techniques is presented.

- **Direct truncation**: The final state is completely decided by the frame, without any information about it. As a positive aspect, no additional complexity is added for the encoding/decoding operations. On the other hand, the error correction capabilities are expected to be subjected to a degradation.
- **Return-to-zero (tail-bits)**: At the end of the encoded frame, v additional bits are added to the encoded information, in order to force the final state to return to 0. Those bits are also known as *tail-bits*. Following this approach, the final state is known, with drawbacks on the encoder complexity and the code-rate.
- **Circular codes**: Following this solution, the initial state S_0 is initialized to a particular known state S_c. The code is organized such that, after encoding the K-bits source information, the ending state will still be S_c. As an advantage, the final state is known without adding any additional bits. On the negative side, it is necessary to ensure that the employed code is supporting the circular sequence of states.

3.2.5 Puncturing and High Code-Rate

Reducing the amount of redundant information increases the code-rate, introducing a benefit on the useful information bit-rate. When high-throughput requirements are involved, it is fundamental to guarantee the capability to work with high code-rates.

The immediate solution is the direct use of a specific code intrinsically characterized by a high-rate. However, in a scenario in which it is desired to work with different code-rates, this solution is inefficient, since a direct high-rate code is requiring the usage of a specific encoder and decoder. Therefore, flexibility against different code-rates is not guaranteed.

A more flexible solution consists in puncturing bits/symbols from an original low-rate code. A punctured bit/symbol will not be transmitted on the channel, increasing the equivalent rate of the code. Typically, the puncturing operation is performed following a *puncturing pattern*, expressed by an n x p matrix, where p is representing the *puncturing period*. A puncturing matrix example is reported below, converting an initial rate

$R_i = 1/2$ to a final rate $R_p = 4/5$.

$$P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The puncturing pattern choice is aiming to introduce an optimal trade-off between a high code-rate and an acceptable error-correction performance.

At the receiver side, the decoder is aware of the puncturing pattern, and it is performing a *de-puncturing* operation. As mentioned before, puncturing is a flexible solution, since both encoder and decoder are working with the same low-rate mother-code. As a consequence, different puncturing patterns can be applied, still employing the same encoder/decoder.

3.3 Decoding

Starting from a given encoding algorithm, it is possible to explicit how the decoding operation will be performed. Depending on the use cases, decoding algorithms can work with *hard-information* or *soft-information*. The former is obtained by representing any information without giving details about its reliability, while the latter is capable to provide a metric for the information robustness. For instance, if considering a binary information bit b, the hard-information is expressed by simply specifying the bit value. On the other hand, the soft-information may be represented employing a signed integer, whose sign is representing the bit value, while the magnitude its reliability. As another example, the *Log-Likelihood-Ratio (LLR)*, produced by after the demodulation, can be considered as soft-information.

The decoding algorithms can then be subdivided in two main classes.

- **Soft-Input-Hard-Output (SIHO)**: In this class, the input information to be decoded is required to be represented as soft-information. On the other hand, the decoded message at the output is provided as hard-information.
- **Soft-Input-Soft-Output (SISO)**: In this class, both the information to be decoded and the decoded message are required to be represented as soft-information. As discussed in the next chapter, this category of decoders is especially convenient if *code concatenation* is applied.

A list of fundamental decoding algorithms is including the Viterbi algorithm (SIHO), the Soft-Output Viterbi algorithm (SISO), the BCJR algorithm (SISO) and its logarithmic implementation, the Log-MAP algorithm (SISO).

The focus in this work is centered around the Log-MAP algorithm, largely used in Turbo-decoders architectures. Therefore, a review of this algorithm is presented, considering its application on the reference LTE/UMTS standard code.

3.3.1 The Log-MAP Algorithm

The Log-MAP algorithm is a Maximum-A-Posteriori (MAP) based algorithm, directly derived from the BCJR algorithm. The latter was first proposed in 1974 by Bahl, Cocke, Jelinek and Raviv [7], which are also giving the name to the algorithm.

The BCJR algorithm aims to estimate the A-Posteriori-Probability (APP) for each possible received message symbol, considering the full sequence of LLRs information produced after the demodulation. Then, the most likely received message symbol, given by the Maximum APP, can be used to extract the hard-information.

As an example, considering the reference code and a BPSK modulation, each message bit u_k is associated with two codeword bits/symbols c_{1k} and c_{2k}. The demodulation is producing the LLR information L_{1k} and L_{2k}. Starting from the LLR and other fundamental metrics, the algorithm is evaluating two APP, first assuming $u_k = 0$ and then with $u_k = 1$. The maximum APP will include the information about the most likely received message bit.

It is remarkable how, since MAP-based algorithms are willing to find the most likely transmitted symbols one by one, the overall sequence of symbols generated after the decoding operation may not correspond to a valid path inside the Trellis-Diagram.

Since the BCJR algorithm is fully derived in probability domain, many multiplications between parameters are required. Multiply operators are not particularly suitable for hardware implementation, especially in terms of complexity. As a consequence, the algorithm can be rewritten in logarithmic domain, exploiting the logarithm property to convert all the products in additions. Therefore, the Log-MAP algorithm is performing the same operations stated by the BCJR algorithm, but employing simpler addition operators.

The application of the Log-MAP algorithm is strongly relying on the usage of the Trellis-Diagram. Considering the reference code, the evaluation of the Maximum APP is performed on each Trellis-section. As a matter of fact, given a generic path in the diagram, a Trellis-section k is containing the information related to the transmitted bit u_k, expressed under the form of a codeword.

The soft-information computation for a generic message bit u_k is requiring 4 steps: the *branch-metrics computation*, the *forward-propagation*, the *backward-propagation* and the *soft-output computation*.

Branch-Metrics Computation

Given a Trellis-section k, including the transitions between states S_k and S_{k+1}, it is possible to associate a branch-metric Γ to each branch, calculated as follows.

$$\Gamma(S_k, S_{k+1}) = u_k(S_k, S_{k+1}) \cdot L_a(u_k) + \sum_{i=1}^{n} c_{ik}(S_k, S_{k+1}) \cdot L_{ik}$$

L_{ik} is representing the LLR for the codeword bit c_{ik}, while $L_a(u_k)$ is representing the *a-priori information*, which is an additional information on the bit u_k that can be handled by the decoder. The usage of L_a is exploited by Turbo-Codes, in order to improve the error-correction capabilities of the decoding algorithm.

For instance, observing the Trellis-Diagram in figure 3.4, it is possible to evaluate Γ considering $S_k = 2$ and $S_{k+1} = 5$. In this case, $u_k = c_{1k} = 0$ and $c_{2k} = 1$.

$$\Gamma(2, 5) = 0 \cdot La_k + 0 \cdot L_{1k} + 1 \cdot L_{2k} = L_{2k}$$

Before proceeding to the next step, all the branch metrics values for each Trellis-section should be computed, with k between 0 and $K - 1$.

Forward-Propagation

The Trellis-Diagram is explored from left to right, recursively calculating the *forward state-metrics* $A(S_k)$. Also in this case, a generic state transition from S_k to S_{k+1} can be considered. Since the algorithm for the state metrics computation is recursive, $A(S_{k+1})$ can be directly derived starting from $A(S_k)$ and $\Gamma(S_k, S_{k+1})$ values.

$$A(S_{k+1}) = ln\left(\sum_{S_k} e^{A(S_k)+\Gamma(S_k,S_{k+1})}\right)$$

The formula, derived from the BCJR algorithm, is not suitable for a hardware implementation due to its complexity. However, it is possible to rearrange the equation employing the *Jacobian logarithm*, as reported in [8].

$$ln(e^{\delta_1} + e^{\delta_2}) = max^*(\delta_1, \delta_2) = max(\delta_1, \delta_2) + ln(1 + e^{-|\delta_1 - \delta_2|})$$

It is remarkable how the max^* operator is including a simple maximum evaluation between the involved parameters and a correction term, that typically is pre-computed and stored in a Look-Up-Table (LUT).

After applying the Jacobian logarithm, the state-metrics computation is indicated as follows.

$$A(S_{k+1}) = \underset{S_k}{max^*}(A(S_k) + \Gamma(S_k, S_{k+1}))$$

When considering the Trellis-section k, S_{k+1} is representing the arrival state, for which it is desired to compute the state metric, while S_k are all the possible starting states connected with S_{k+1} with a branch $\Gamma(S_k, S_{k+1})$.

For instance, always considering the figure 3.4, if $S_{k+1} = 5$, the two possible starting states are $S_k = 2$ and $S_k = 3$. Therefore, $A_{k+1}(5)$ can be computed as follows.

$$A_{k+1}(5) = max^*(A_k(2) + \Gamma_k(2,5), A_k(3) + \Gamma_k(3,5))$$

Backward-Propagation

The Trellis-diagram is now explored from right to left, computing the *backward state-metrics* $B(S_k)$. The equation presents a strong similarity if compared to the forward-propagation one.

$$B(S_k) = \underset{S_{k+1}}{max^*}(B(S_{k+1}) + \Gamma(S_k, S_{k+1}))$$

In this case S_{k+1} is the starting state in the diagram, while S_k is the arrival one.

Soft-Output Computation

The maximum A-Posteriori-Probability (APP) for a generic bit u_k, considering the k^{th} Trellis-section, can be estimated as follows.

$$\underset{(S_k,S_{k+1})|u_k}{max^*}(A(S_k) + \Gamma(S_k, S_{k+1}) + B(S_{k+1}))$$

Typically, two maximum APPs are computed, considering $u_k = 0$ and $u_k = 1$. The hard-decision is derived from the maximum APP between the two computed values. Referencing the Trellis-Diagram, the branches transitioning from S_k to S_{k+1} can be divided in two groups, half of them related to $u_k = 0$, the other half to $u_k = 1$.

Since the algorithm is classified as SISO, the soft-information for the bit u_k is computed employing the following equation.

$$\begin{aligned} L(u_k) = &\underset{(S_k,S_{k+1})|u_k=1}{max^*}(A(S_k) + \Gamma(S_k, S_{k+1}) + B(S_{k+1})) - \\ &- \underset{(S_k,S_{k+1})|u_k=0}{max^*}(A(S_k) + \Gamma(S_k, S_{k+1}) + B(S_{k+1})) \end{aligned}$$

The hard-information is computed as follows.

$$m_k = \begin{cases} 0 & if\ L(u_k) < 0 \\ 1 & if\ L(u_k) \geq 0 \end{cases}$$

A schematic view of the LM algorithm on the Trellis-Diagram is presented in figure 3.6.

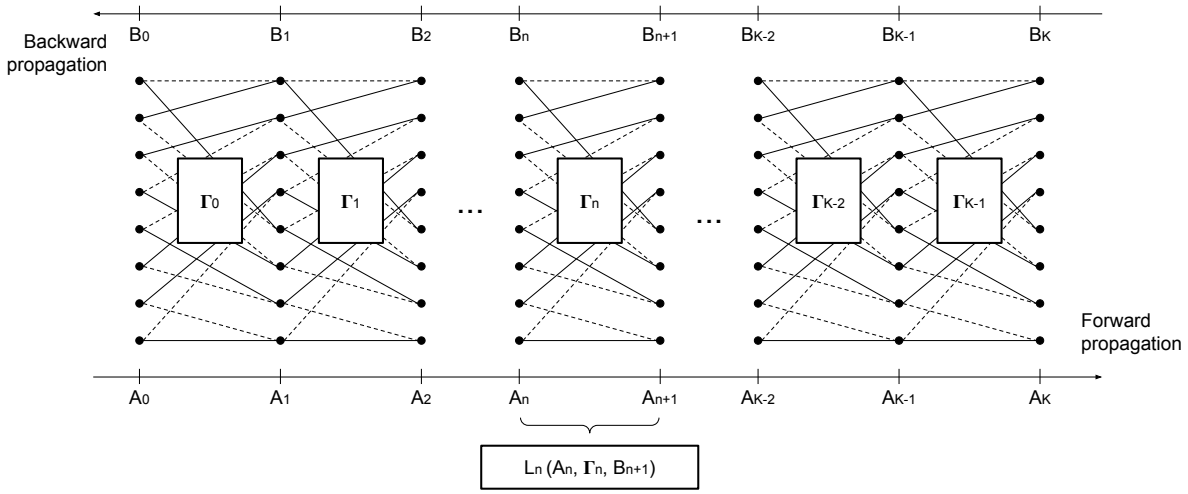

Figure 3.6: A representation of the Log-Map algorithm on the Trellis-Diagram.

An important aspect to be discussed is the state metrics initialization. Since $A(S_k)$ and $B(S_k)$ are evaluated recursively, the algorithm starts from the assumption that A_0 and B_K are known. For this reason, it is fundamental to collect information about the starting and the arrival states in the Trellis-Diagram. If those states are known, their state metrics should be initialized to 0, considering the metrics for the other states initialized with the minimum possible value (ideally $-\infty$). For instance, if the starting state is $S_0 = 0$, then $A_0(0) = 0$, while $A_0(1...7) = -\infty$. Non-initialized state metrics could lead to a BER degradation.

3.3.2 The Max-Log-MAP Algorithm

A sub-optimal version of the Log-MAP algorithm can be derived by approximating the max^* operator with a simple max operator. The correction term included in the Jacobian logarithm is neglected.

$$max^*(\delta_1, \delta_2) \approx max(\delta_1, \delta_2)$$

A first direct advantage is found on the hardware architecture, which is not including any correction factor LUT. Furthermore, the estimated noise variance σ completely disappears

from the soft-output computation. As a consequence, the algorithm is more reliable against the noise variance estimation.

A clear negative impact is found instead on the APP estimation, which is affected by an error. However, a compensation factor, called Extrinsic-Scaling-Factor (ESF), can be considered on the soft-output values, in order to lower the impact of this error [9]. A typical value for the ESF is 0.75.

Chapter 4

Turbo-Codes

This chapter aims to first introduce the concept of concatenated codes, the category to which Turbo-Codes are belonging, highlighting their main advantages. Then, Turbo-encoding will be covered, introducing the concept of interleaving. Proceeding in the chapter, the basic block scheme of a Turbo-decoder will be discussed, explaining the main organization. In the same section, the concept of extrinsic information will be introduced. Moreover, a comparison among different decoding algorithms in a Turbo-decoder architecture is presented, focusing on their error-correction capabilities. The last section is including a review of the basic building blocks included in a SISO-decoder architecture.

4.1 Turbo-Encoding

In order to improve error-correction performances, it is possible to encode the same information source relying on multiple codes. Turbo-Codes are based on this approach. More specifically, a Turbo-Code is including multiple *convolutional* codes, which can be organized in order to follow a *serial* or a *parallel* configuration.

The example shown in figure 4.1 is considering two *recursive convolutional encoders* operating in parallel, both employing the reference code presented in Chapter 3. The message bits u are directly connected at the input of the first encoder RSC_1. On the other hand, before entering the second encoder RSC_2, the message bits are processed by an interleaver block, which is applying a scrambling operation, producing a different sequence of bits u^{π}. More details about the interleaving process will be given proceeding through this section.

Each RSC is producing two codeword bits c^s and c^p per each message bit. c^s is representing the *systematic information*, while c^p the *parity information*. Since the systematic information generated by the two RSC is the same, it will be considered only once in

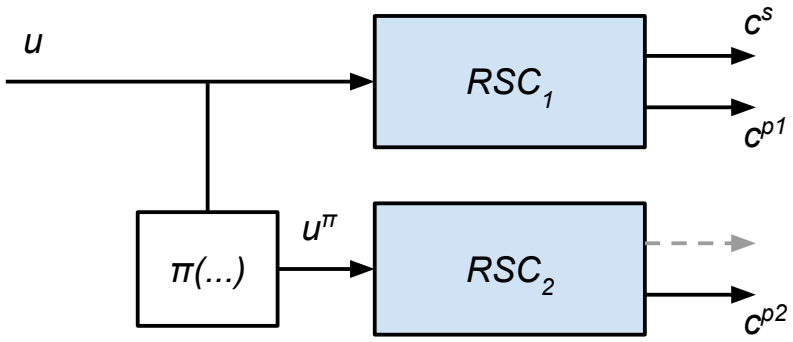

Figure 4.1: Turbo-encoder block scheme including two parallel RSC.

the final codeword. Therefore, the Turbo-codeword c generated per each message bit is including three contributions: 2 parity bits and 1 systematic bit.

$$c = [c^s, c^{p1}, c^{p2}]$$

Given the code-rates R_1 and R_2, characterizing the convolutional codes involved, the equivalent *Turbo-code-rate* can be estimated as follows.

$$R = \frac{R_1 R_2}{R_1 + R_2 - R_1 R_2}$$

Moreover, if $R_1 = R_2 = R^{'}$, the equivalent code-rate formula can be further simplified.

$$R = \frac{R^{'}}{2 - R^{'}}$$

For instance, considering the Turbo-Code presented in the example, since $R^{'} = 1/2$, the resulting code-rate R is equal to $1/3$.

4.1.1 Interleaver

As previously mentioned, the interleaver function is scrambling the message bits, applying a specific *interleaving law* or *permutation law*, generally expressed as $j = \pi(i)$. The scrambled message u^π is generated as expressed below.

$$u = [u_0, u_1, ..., u_{K-1}] \quad \xrightarrow{\pi} \quad u^\pi = [u_{\pi(0)}, u_{\pi(1)}, ..., u_{\pi(K-1)}]$$

Interleaving laws are invertible, introducing a de-interleaving function $i = \pi^{-1}(j)$.

Scrambling the message bits before feeding them to the second encoder serves two main purposes.

- Errors are typically generated in *bursts* by the communication channel, which is concentrating them in specific parts of the information stream. Interleaving data is helping to spread the errors, improving the error-correction capabilities.
- The *minimum Hamming-distance* characterizing the Turbo-Code is depending on the permutation law. Interleaving functions can be specifically derived in order to maximize this distance and improve the error-correction performances.

A further aspect detailed in this work is the impact of the permutation law on the decoder hardware architecture, discussed in the next chapters.

Two main classes of interleavers can be defined.

1. **Random based interleavers**: In this class, interleavers are deriving the permutation laws by employing random or pseudo-random methodologies. Since the generation is random based, interleaving data must be stored in a proper memory.
2. **Structured interleavers**: In this class, interleavers are based on algebraic equations, from which the permutation laws are derived. The direct advantage is the possibility to generate interleaving data by implementing in hardware the specific algorithm, avoiding the memory storage. Common interleavers belonging to this class are the *Quadratic Permutation Polynomial (QPP)* and the *Almost Regular Permutation (ARP)* interleavers, respectively indicated in the LTE [6] and UMTS [5] standards for Turbo-Codes.

4.2 Turbo-Decoding

The Turbo-decoding process is summarized by the block scheme in figure 4.2.

Two *SISO decoders* are applying a specific decoding algorithm, considering the LLR information extracted from the received symbols. The first module $SISO_1$ is considering the systematic information L_s and the first parity information L_{p1}. L_s is also employed, after a scrambling process, by the $SISO_2$ module, which is also considering the second parity information L_{p2}.

It is remarkable how both the SISO decoders are including a third input for the *a-priori information* L_a. Each SISO module, starting from the soft-output information L, is able to estimate the extrinsic information L_e, which is considered as a-priori information by

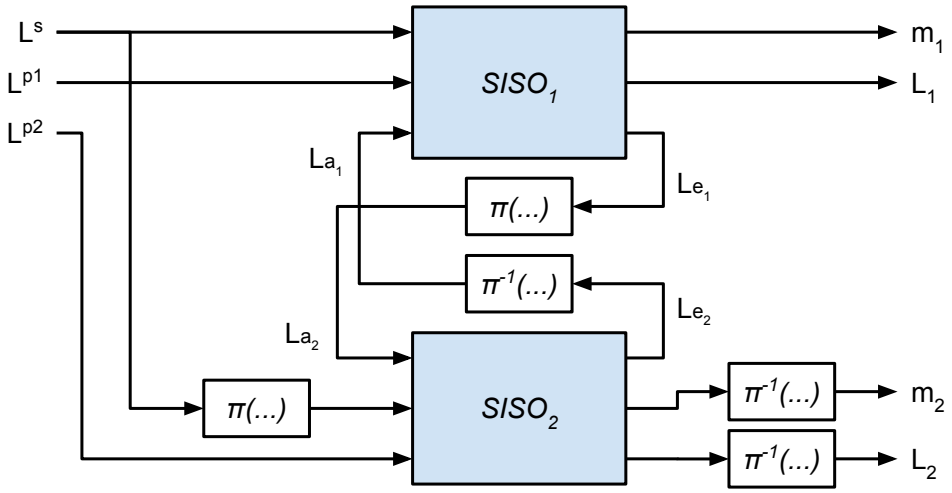

Figure 4.2: Turbo-decoder block scheme including two SISO modules.

the other SISO decoder. As noticeable, proper interleaving and de-interleaving functions are required to correctly perform the exchange of information.

The decoding process can be repeated iteratively, until the two SISO modules are producing the same received message. The continuous exchange of extrinsic information during the different iterations can be visualized as a turbine movement, from which the name Turbo-Codes.

The extrinsic information L_e is computed starting from the soft-output value L, also known as *a-posteriori information*, and subtracting to it the systematic and a-priori information.

$$L_k^e = L_k - (L_k^s + L_k^a)$$

The performed subtractions are necessary to avoid feedback loop related issues.

It is noticeable how the two SISO decoders are performing the same operations on different data. Therefore, it is possible to alternate the natural and interleaved decoding processes on a single SISO module. In this case, each performed processing is called *Half-Iteration (HI)*. A *Full-Iteration (FI)* is composed by two consecutive Half-Iterations.

The number of Full-Iterations to be performed depends on the desired error-correction capabilities. Two different approaches can be used to set this number. The first solution consists in fixing it to a specific value that ensures the BER requirements. Despite this method is not introducing any further complexity in the hardware architecture, it is lacking in terms of flexibility. As a matter of fact, the number of Full-Iterations can be dynamically modified according to the information to be decoded, if a *stop criterion* is introduced. Different rules, typically based on observations on the produced LLR values,

are available to define a stop condition [10]. Introducing a stop criterion is potentially increasing the amount of decoded bits per unit time. On the negative side, its implementation is expected to require further hardware resources.

4.3 Algorithms Comparison

It is now possible to compare the error-correction performances of different algorithms on a Turbo-decoder architecture, based on the reference convolutional code. Different BER curves, extracted from [4] and represented in figure 4.3, are employed to compare the following algorithms.

- Soft-Output-Viterbi-Algorithm (SOVA)
- Max-Log-Map (MLM)
- Max-Log-Map (MLM) with ESF = 0.75
- Log-Map (LM)

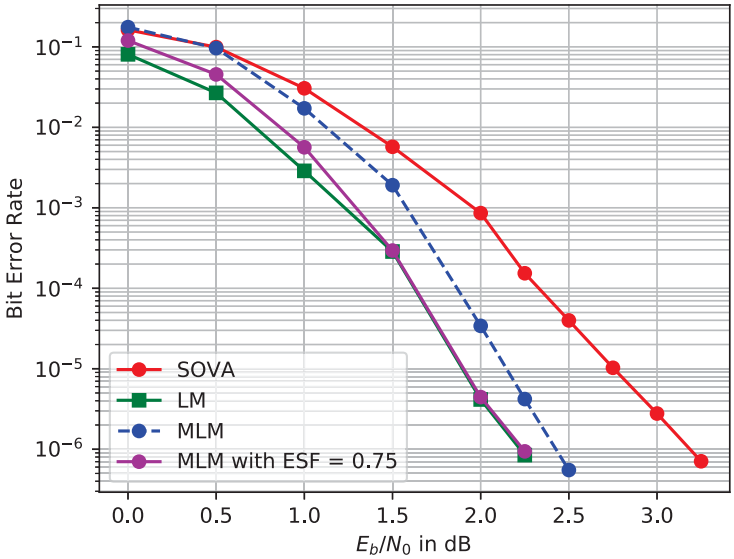

Figure 4.3: BER as function of E_b/N_0, considering different algorithms applied in a Turbo-decoder architecture.

As visible, the LM and the MLM algorithms are outperforming the SOVA, especially for higher E_b/N_0 values. The approximation introduced by the pure Max-Log-Map (MLM) algorithm is strongly affecting the achievable BER, especially if compared to the Log-Map (LM) application. The introduction of the Extrinsic-Scaling-Factor (ESF) is locating the

MLM algorithm in a sweet spot: the additional complexity introduced by the ESF is expected to be small, and the BER difference compared to the LM algorithm is noticeable only for small E_b/N_0 values.

For the reasons mentioned above, the Max-Log-Map (MLM) algorithm, including the usage of the ESF, is expected to introduce an optimal trade-off between architectural complexity and error-correction performances. Therefore, it will be considered as a reference algorithm in the next chapters.

4.4 SISO-Decoder Building Blocks

A SISO-decoder architecture working with the Max-Log-Map (MLM) algorithm is including three fundamental computational units, listed below. Each of the presented units is meant to work on the Trellis-Diagram, considering a Trellis-section at a time.

- **Branch Metric Unit (BMU)**: In this unit, channel and a-priori LLR from a specific Trellis-section k are employed to evaluate the branch metrics Γ_k.
- **Path Metric Unit (PMU)**: This unit, starting from the results provided by the BMU, is capable to recursively perform the forward or backward state-metrics propagation. Following this purpose, it is including several *Add-Compare-Select (ACS)* units, able to apply the *max* operator between different candidates for the new state-metrics values.
- **Soft Output Unit (SOU)**: In this unit, starting from the data produced by the BMU and the PMU, the a-posteriori information L_k is computed. Then, if necessary, hard and extrinsic informations can be derived.

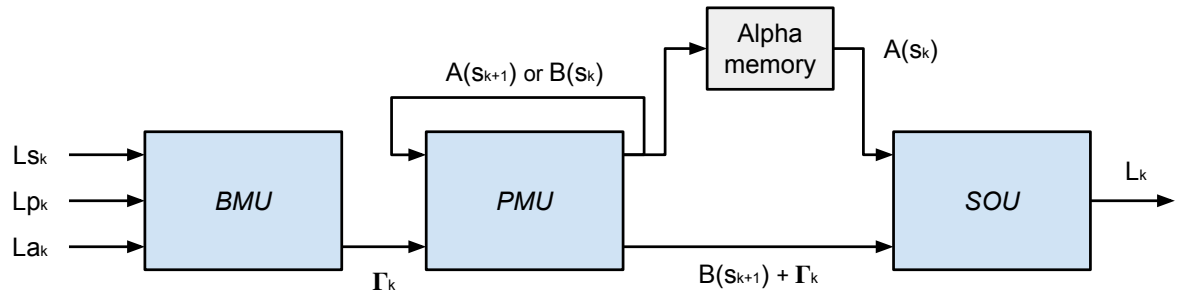

Figure 4.4: SISO-decoder block scheme example.

A basic connection scheme for the listed logic units is depicted in figure 4.4. Since the Path Metric Unit (PMU) is meant to perform both the forward and backward propagations, a memory is required to store the forward propagation metrics *alpha*. Moreover, since

the PMU processing is recursive, a feedback loop is necessary. As discussed later in this work, this loop is imposing an important limitation on the critical path of the hardware architecture.

A single half-iteration is performed by first computing the *alpha-metrics*, employing the generated Γ values. Those metrics are saved one by one in the alpha memory. Then, the PMU is recursively evaluating the *beta-metrics*, while feeding the SOU proper additions between the generated beta values and the branch metrics Γ. Meanwhile, the SOU is loading the alpha-metrics values stored in memory, in order to employ them to perform the soft-output evaluation.

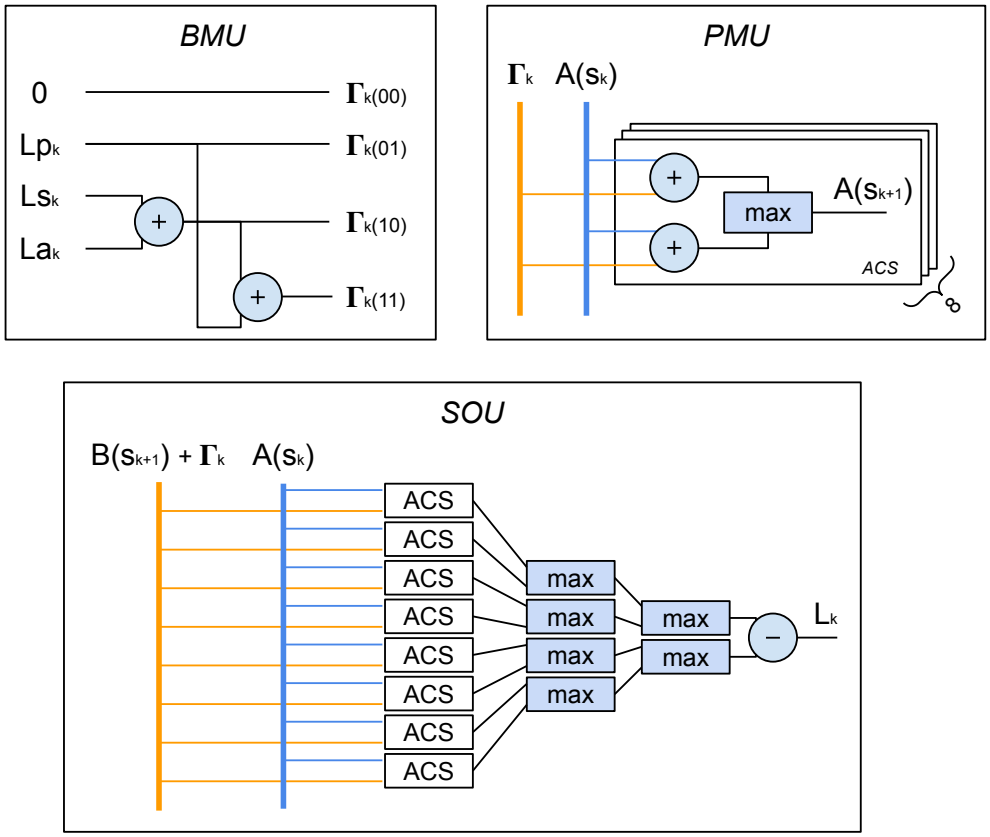

Figure 4.5: Internal structure of the main SISO-decoder logic units.

In figure 4.5, an overview of the internal structure for the different computational units is given. The represented *max* operator is meant to be implemented by means of a subtractor and a multiplexer. It is remarkable how each unit is including many instances of simple logic operators, such as adders or subtractors.

Chapter 5

Parallelization

Parallel computation is a fundamental property to be exploited in order to move architectures toward high-throughputs. Therefore, this chapter aims to summarize and classify fundamental parallelization techniques, widely used in modern architectures, giving an overview on their main characteristics. The first section is introducing some important Key Performance Indicators (KPI), including the definition of throughput. Then, different levels of parallelization will be discussed one by one. *Algorithm-parallelism* will be analyzed, introducing the concepts of scheduling and windowing. Then, *Trellis-parallelism* or *high-order radix* will be discussed. Proceeding in the chapter, *SISO-parallelism* will be introduced and then *Decoder-parallelism*, considering both the concurrent and sequential approaches. In the last section, *Iteration-parallelism* will be covered. Quantitative considerations about the consequences on error-correction performances will be provided in the next chapter.

5.1 Key Performance Indicators

As mentioned in the introduction, different KPI were introduced by the EPIC project, in order to characterize decoders architectures. Part of those metrics will be employed in this work to perform comparisons among different solutions.

- **Latency**: It is defined as the total amount of time required by a Turbo-decoder to fully process an entire information frame.
- **Throughput**: It is defined as the number of decoded information bits per unit time.
- **Area**: It is representing the total area of the Turbo-decoder architecture, typically expressed in mm^2. The area can be used to evaluate the *complexity* of the archi-

tecture. An *algorithmic complexity* estimation can be found instead by counting the number of elementary operators involved and considering the total amount of required memory.

- **Area Efficiency**: It is computed as the ratio between the throughput achievable by the Turbo-decoder and its total area. The area efficiency is expressing how the area is well employed in order to reach high-throughputs. As discussed later in this work, this metric is fundamental when high degrees of parallelization are meant to be introduced.
- **ECC performance**: It is expressing the error-correction capabilities of the Turbo-decoder, measured by the BER as function of E_b/N_0.
- **Code Flexibility**: It is measuring the adaptability of the architecture against different frame-sizes K and different code-rates R. A further degree of flexibility is the capability to apply a *stop-criterion* to the Full-Iteration number. As discussed in chapter 2, employing *puncturing* could fully cover the code-rate flexibility requirement.

A summary of the required KPI is included in table 5.1, for the different use cases considered by the EPIC project [11]. It is noticeable how area efficiency requirements may be different depending on the considered technology. The throughput requirements for all the use cases are in the order of hundreds of Gb/s. Latency is widely ranging between nanoseconds and milliseconds. Moreover, also the BER is presenting important variations from case to case, ranging from 10^{-15} up to 10^{-6}.

Technology				28 nm	7 nm
Use Case	**BER**	**Latency**	**Throughput**	**Area Efficiency**	
			[Gb/s]	[Gb/s/mm^2]	
Data Kiosk	10^{-12} - 10^{-14}	0.5 ms	1000	100	220
Virtual Reality	10^{-6}	0.5 ms	500	50	54
Intra-Device Com.	10^{-12}	100 ns	500	50	50
Fronthaul	10^{-13}	1 us	1000	100	100
Back haul	10^{-8}	1 us	250	25	25
Data Center	10^{-12} - 10^{-15}	100 ns	1000	100	162
Hybrid Wireless Fiber	10^{-12}	200 ns	1000	100	120
High Throughput Sat.	10^{-10}	10 ms	100-1000	100	n/a

Table 5.1: EPIC KPI requirements considering different use cases.

5.2 Baseline Architecture

Before proceeding with the introduction of several parallelization degrees, a basic Turbo-decoder architecture is presented, without assuming any parallel computation. The 4 main steps involved in the application of the Max-Log-Map algorithm are performed one by one in sequence, relying on the three fundamental logic units presented in Chapter 4. Moreover, multiple memories are required to store the several computed metrics, as illustrated in figure 5.1.

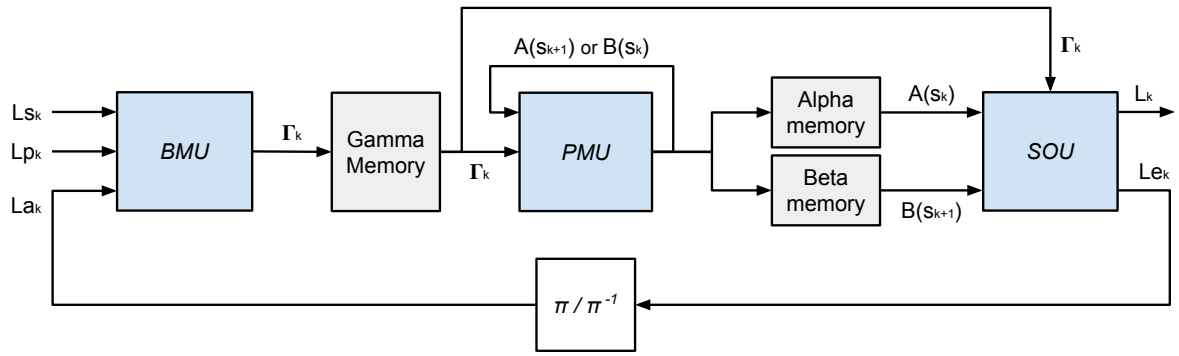

Figure 5.1: Baseline Turbo-decoder architecture, obtained without exploiting parallel computations.

First, the BMU is employed to compute K branch-metrics, processing each single Trellis-section. The results, stored in a proper memory, are then employed by the PMU, which is recursively computing *Alpha* and *Beta* state-metrics, performing two consecutive processings on the Trellis-Diagram. Also in this case, results are stored in given memories. As a last step, the SOU, starting from the gathered branch and state metrics, is estimating K soft-informations, one per each Trellis-section. Moreover, also extrinsic information values are produced.

Different Half-Iterations are performed by re-employing the same architecture, considering the required data permutation, especially for the a-priori information.

As noticeable, this approach is dramatically affecting latency and throughput achievable by the Turbo-decoder. Indeed, 4 separate processings on K Trellis-sections are required to be performed. Furthermore, memory organization is poorly optimized, introducing high storage requirements, since it is necessary to store K branch-metrics and $2K$ state-metrics. In order to face these issues, several computations can be parallelized, starting from the presented basic implementation. The analyzed degrees of parallelism are detailed in the next sections.

5.3 Algorithm Parallelism

It is possible to organize the algorithmic steps in a given order, called *scheduling* policy. Relying on some algorithm properties, parallelism between sequences of operations can be introduced. Two different types of scheduling choices are presented in this work, the *Forward-Backward (FB)* scheduling and the *Butterfly* scheduling, applied to the Max-Log-Map algorithm. Therefore, the steps to be organized are the branch metrics computation, the forward recursion, the backward recursion and the soft-output computation.

Forward-Backward Scheduling

The block scheme depicted in figure 4.4, introduced while discussing the SISO building blocks, is a classic example of FB scheduling. First, the Forward propagation is fully completed, saving the alpha metrics in the proper memory. Then, the backward propagation is performed and, meanwhile, the soft-output values are computed. The branch metrics computation is always performed during both the propagations.

A graphical representation of the FB scheduling is depicted in figure 5.2. The algorithmic steps are represented in time, during the processing of a K-bits information frame. The gray area is highlighting the required storage memory, in this case for the *alpha* metrics. As visible, the full latency to process the frame is equal to $2K$ time-instants or clock cycles. The hardware requirements for a SISO-decoder operating with this scheduling are 1 BMU, 1 PMU, 1 SOU and a K-capacity memory.

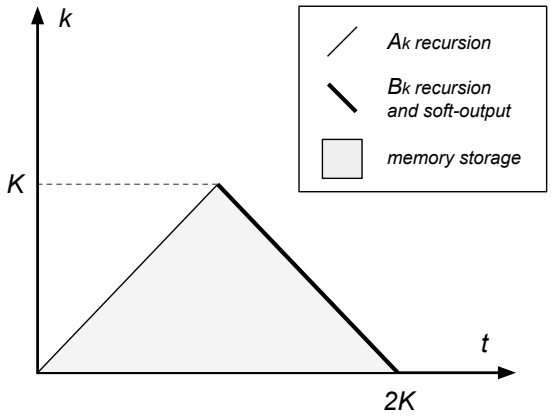

Figure 5.2: Forward-Backward scheduling - graphical representation in time.

Butterfly Scheduling

The forward and backward recursions can potentially be executed in parallel, implementing a *Butterfly* scheduling. Once both the recursions are halfway through the frame, the soft-output computation can start, since both *alpha* and *beta* metrics are available for

each Trellis-section. The scheduling is visually represented in figure 5.3.

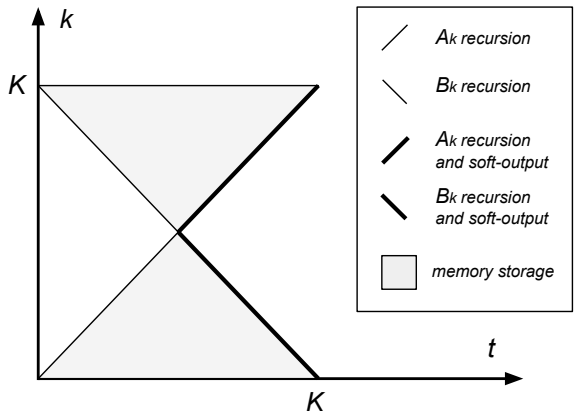

Figure 5.3: Butterfly scheduling - graphical representation in time.

As noticeable, the total amount of time necessary to process the entire frame is half the Forward-Backward scheduling one, giving to the Butterfly scheduling an advantage in terms of latency and throughput. However, the architectural complexity is expected to be higher, since the application of this scheduling is requiring 2 BMU, 2 PMU and 2 SOU. The required memory storage is constant if compared to the FB scheduling one.

A clear throughput-complexity trade-off is present in the choice between the presented scheduling algorithms. Moreover, other interesting scheduling approaches are available, as detailed in [12].

5.3.1 Windowing

In order to reduce the memory requirements for the state metrics storage, the frame to be processed can be divided in different *windows* of length WS. The specific scheduling algorithm is applied on each single window, instead of considering the full frame, drastically reducing the amount of required storage. Figure 5.4 is representing the application of *windowing* to both the considered scheduling policies.

The total amount of required memory has been reduced to WS. While the hardware complexity for the butterfly scheduling is the same, the Forward-Backward solution is requiring an additional BMU and PMU. Table 5.2 is summarizing some fundamental parameters for comparison purposes.

The Forward-Backward scheduling appears to be an optimal solution, assuming that the windows size WS is small enough if compared to the frame size K. Under this assumption, K+WS $\approx K$. Therefore, the Forward-Backward solution is preferred, since it is requiring

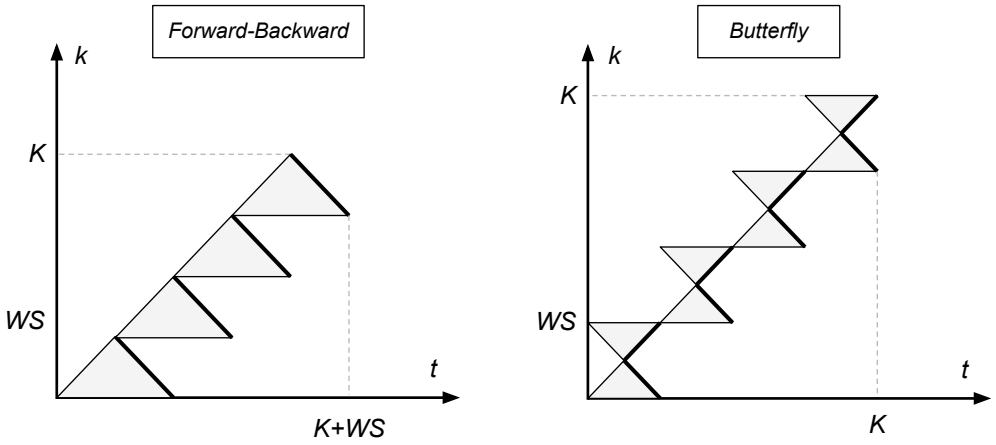

Figure 5.4: Windowing applied to the Forward-Backward and Butterfly scheduling policies.

Quantity	**Forward-Backward**	**Butterfly**
Latency	K+WS	K
Memory size	WS	WS
BMU number	2	2
PMU number	2	2
SOU number	1	2

Table 5.2: Comparison table on windowing applied with Forward-Backward and Butterfly scheduling.

only one SOU. On the contrary, if K and WS are comparable and high-throughputs are desired, the Butterfly solution is to be considered.

Observing figure 5.4, the alpha metrics recursion is continuously performed without any interruption, as originally stated by the Max-Log-Map algorithm. On the contrary, the beta metrics propagation is not continuous, generating a degradation on the error-correction performances. In order to partially solve this issue, the beta propagation must be initialized with reliable values per each window. There are two main available techniques that can be used to fulfill this purpose: the *Acquisition* and the *Next-Iteration-Initialization (NII)*, both represented in figure 5.5.

The *Acquisition* is starting the recursive beta-metrics calculation before the beginning of a given window, as represented by the dashed lines. Therefore, the reliability of the beta initialization values is increased, with a drawback on the hardware complexity, since a further BMU and PMU are required.

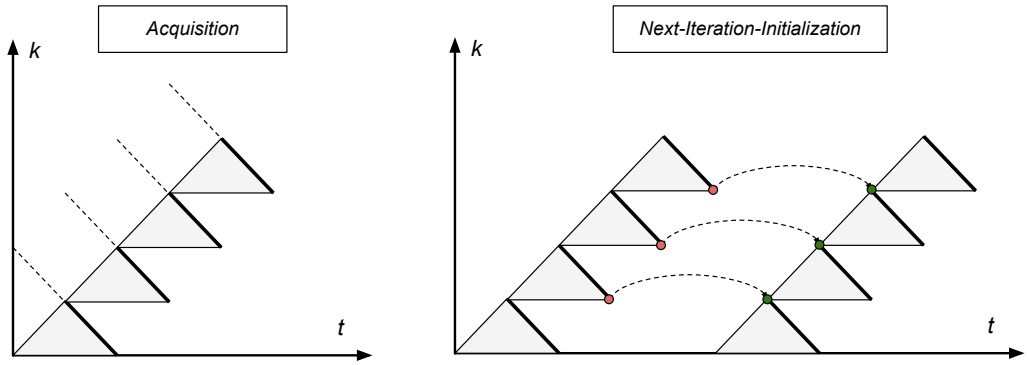

Figure 5.5: Acquisition and NII - graphical representation in time.

The *Next-Iteration-Initialization* is storing, for each window, the final beta metrics from a generic iteration, in order to use them as initialization values for the next iteration. Also in this case, the degradation on the error-correction performances is partially solved, with a drawback on the required memory storage, which is now including a memory contribution for the beta metrics.

In order to explicit the best solution from case to case, it is necessary to compare the impact of the additional logic against the additional memory, considering the BER benefits from the two approaches.

Moreover, if the improvements from a single solution are not enough, it is possible to combine them. Observing the NII case in figure 5.4, the beta metrics in the first iteration are not correctly initialized. Therefore, *Acquisition* could be used to solve this issue and provide a reliable initialization.

In this work, the *Forward-Backward* scheduling will be considered as a reference choice, since an information frame is generally expected to include a large amount of windows. Moreover, the usage of the computational units is efficient, since, during the central part of the processing, they are always performing computations. On the other hand, the SOU in the *Butterfly* scheduling are not performing any useful computation for half the processing time.

Regarding the initialization choice, the *Next-Iteration-Initialization* will be considered as a reference. The required amount of additional memory is not expected to be large. Moreover, the *Acquisition* is increasing the amount of frame data required to be available at the same time, which is negatively impacting on the memory organization.

5.4 Trellis Parallelism

As mentioned before, the pure Max-Log-Map algorithm is expected to work with single Trellis-sections, when performing the forward/backward propagation and when evaluating the soft-output values. In other words, each algorithmic step is involving a single Trellis-section, defining a radix-2 approach. It is possible to process multiple adjacent Trellis-sections at the same time, considering *higher-order radix* approaches. A *radix-N* decoder is meant to process $log_2(N)$ Trellis-sections at the same time.

For instance, as depicted in figure 5.6, given a certain amount of time instants, a radix-4 architecture is able to process double the Trellis-sections if compared to a radix-2 architecture, potentially doubling the throughput.

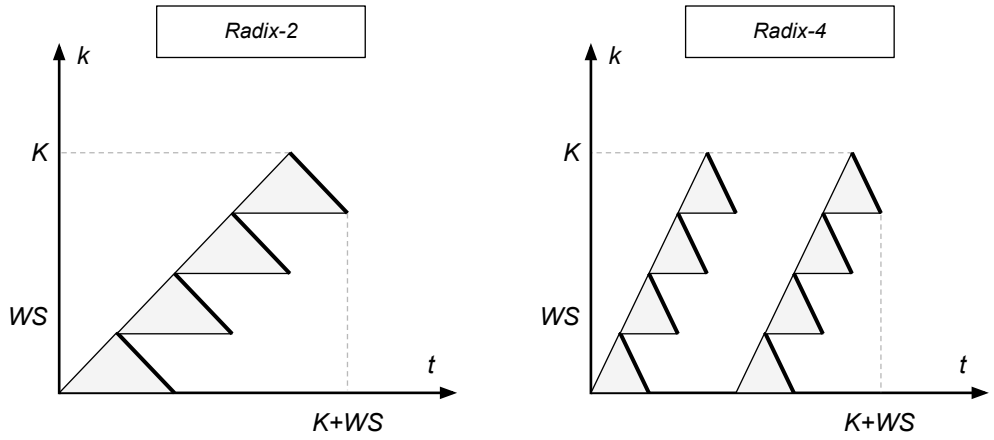

Figure 5.6: Radix-4 benefits against a Radix-2 architecture.

A schematic view of the radix-4 approach in a Trellis-Diagram is illustrated in figure 5.7. It is noticeable how the operations performed by the algorithm are the same, but considering aggregated Trellis-sections, characterized by *aggregated branch metrics*. For instance, a radix-4 branch metric is containing informations about two consecutive Trellis-sections. Given a combined branch, which is highlighting a *path* composed by 2 basic branches, the equivalent branch metric value is given by the sum between all the radix-2 branch metrics in the path. Therefore, considering all the combinations, a larger amount of branch metrics is involved in a radix-4 architecture.

$$\Gamma(S_k, S_{k+2}) = \Gamma(S_k, S_{k+1}) + \Gamma(S_{k+1}, S_{k+2})$$

Consequently, the complexity of the forward/backward propagations is increased, since they need to work with a larger amount of paths to be discriminated. The equations for a radix-4 architecture are reported.

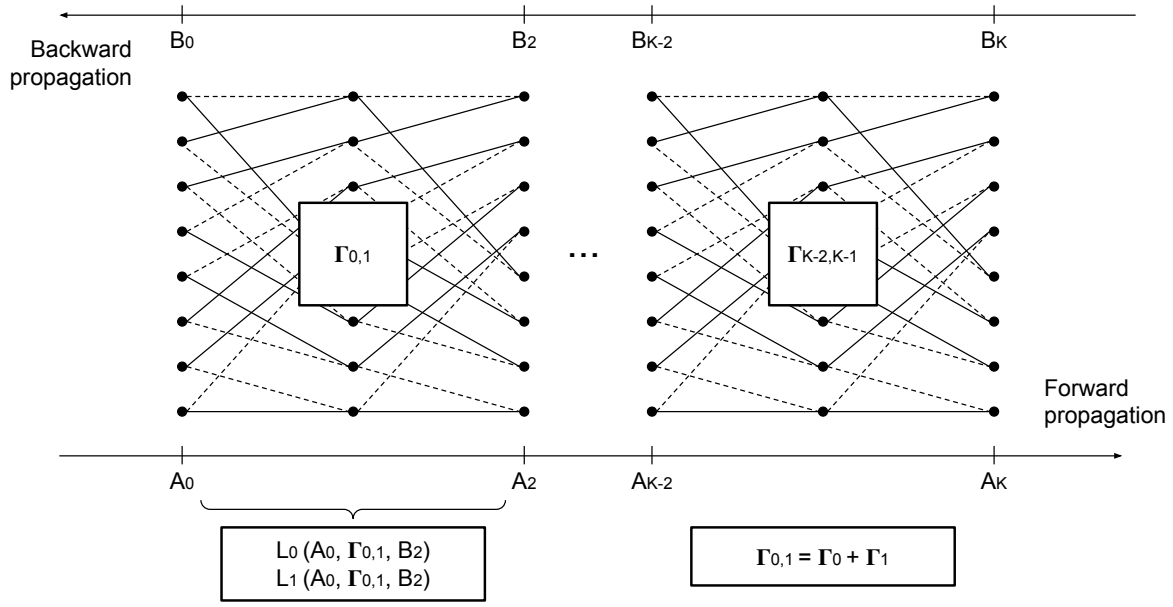

Figure 5.7: Radix-4 approach represented on a Trellis-Diagram.

$$A(S_{k+2}) = \underset{S_k}{max^*}(A(S_k) + \Gamma(S_k, S_{k+2}))$$
$$B(S_k) = \underset{S_{k+2}}{max^*}(B(S_{k+2}) + \Gamma(S_k, S_{k+2}))$$

Finally, also the soft-output computation is more complex, since multiple soft-informations are meant to be evaluated at the same time. In the radix-4 case, it is necessary to simultaneously perform similar computations for two message bits, with a higher amount of involved paths.

$$L(u_k) = \underset{(S_k,S_{k+2})|u_k=1}{max^*}(A(S_k) + \Gamma(S_k, S_{k+2}) + B(S_{k+2})) - \\ - \underset{(S_k,S_{k+2})|u_k=0}{max^*}(A(S_k) + \Gamma(S_k, S_{k+2}) + B(S_{k+2}))$$

$$L(u_{k+1}) = \underset{(S_k,S_{k+2})|u_{k+1}=1}{max^*}(A(S_k) + \Gamma(S_k, S_{k+2}) + B(S_{k+2})) - \\ - \underset{(S_k,S_{k+2})|u_{k+1}=0}{max^*}(A(S_k) + \Gamma(S_k, S_{k+2}) + B(S_{k+2}))$$

Also in this case, a trade-off between better throughput and higher-complexity is present. The larger complexity is potentially affecting the area efficiency and the critical path of the decoder architecture. In order to understand which rules could guide to the best choice in terms of radix order, a more detailed analysis on this approach is presented in this work.

It is remarkable how the use of high radix orders is not compromising the error-correction capabilities of the Turbo-decoder.

5.5 SISO Parallelism

A further level of parallelism is introduced by splitting the K-bits information frame into N Kp-bits *sub-frames*. Each sub-frame can be processed in parallel by different SISO modules, consequently increasing the throughput of the architecture. Different approaches are available.

Concurrent SISO

As shown in figure 5.8, the key idea is to include multiple SISO modules operating in parallel on different sub-frames. The total latency has been reduced from $K + WS$ to $Kp + WS$ for a single half-iteration. Consequently, the throughput is also receiving benefits from this configuration.

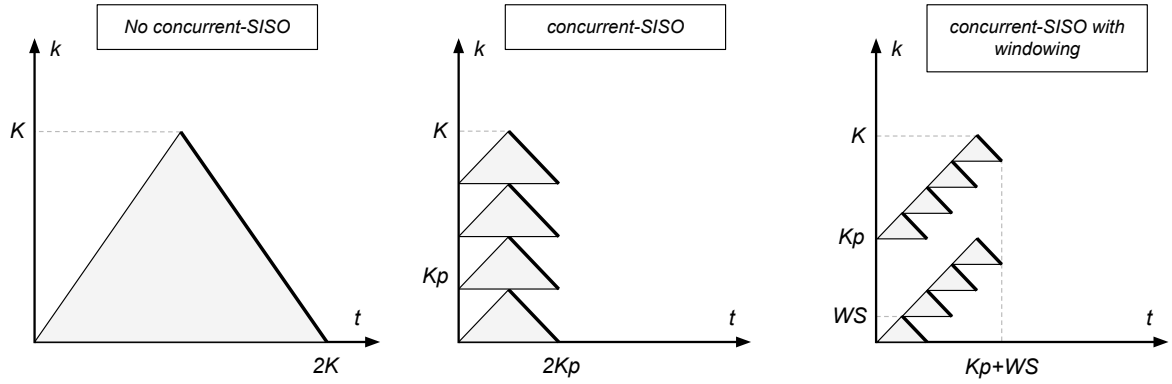

Figure 5.8: SISO concurrency, considering FB scheduling and windowing.

A block scheme for the SISO organization is depicted in figure 5.9, considering a single half-iteration. As visible, N SISO decoders are required.

Sequential SISO

This approach is organizing Kp SISO modules in a pipeline, in order to have the capability to process different sub-frames at the same time. Each of the Kp steps necessary to process the sub-frame is associated to its own pipeline segment. As a consequence, it is possible to enter a new sub-frame in the architecture each clock cycle. The resulting processing is shown in figure 5.10.

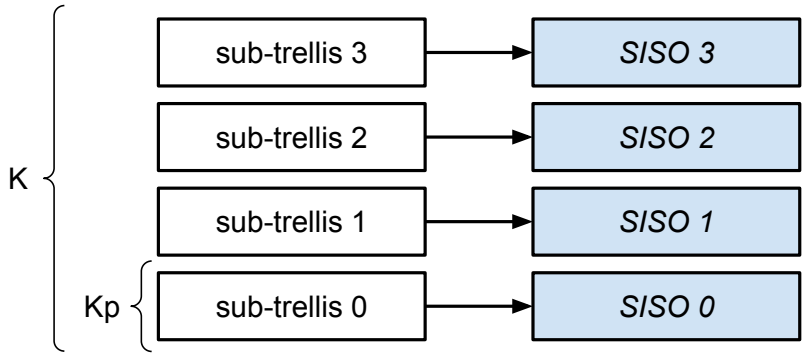

Figure 5.9: Concurrent SISO half-iteration block scheme.

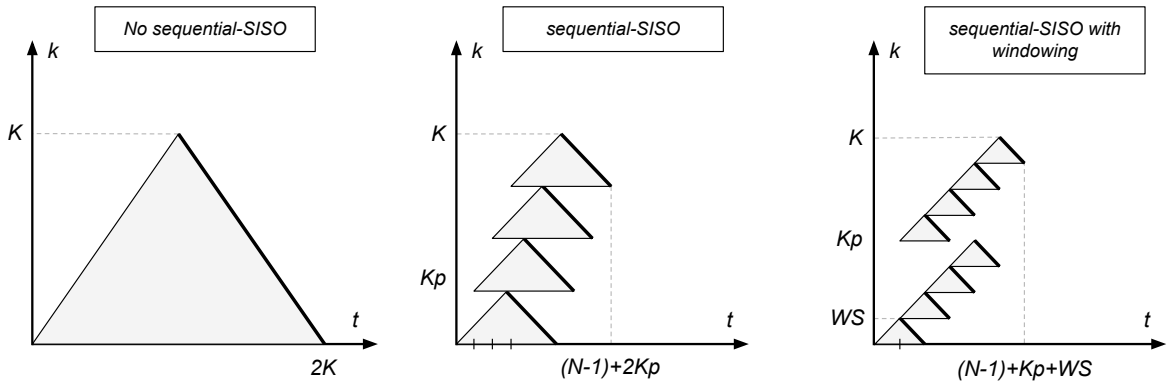

Figure 5.10: Sequential SISO considering FB scheduling and windowing.

Also in this case, an important speed-up factor is present, which is improving latency and throughput. The SISO organization block scheme is depicted in figure 5.11. Since the soft-output computation starts halfway through the sub-frame, the first $Kp/2$ SISO modules are not requiring to include a SOU. Indeed, the name SISO module has been employed for sake of simplicity, even if the unit is including only a part of the computational logic units.

Each SISO module is forwarding the useful informations to the next modules by employing pipelines, in order to perform the algorithm. Consequently, memories to store alpha/beta metrics are not required.

A fundamental drawback to be discussed is the degradation of the error-correction performances, when SISO-level parallelism is employed. Similarly to the windowing concept, also in this case the original Trellis is subdivided, partially losing the continuity of the metrics computation for both alpha and beta metrics. A quantitative analysis of this degradation is proposed in the next chapter.

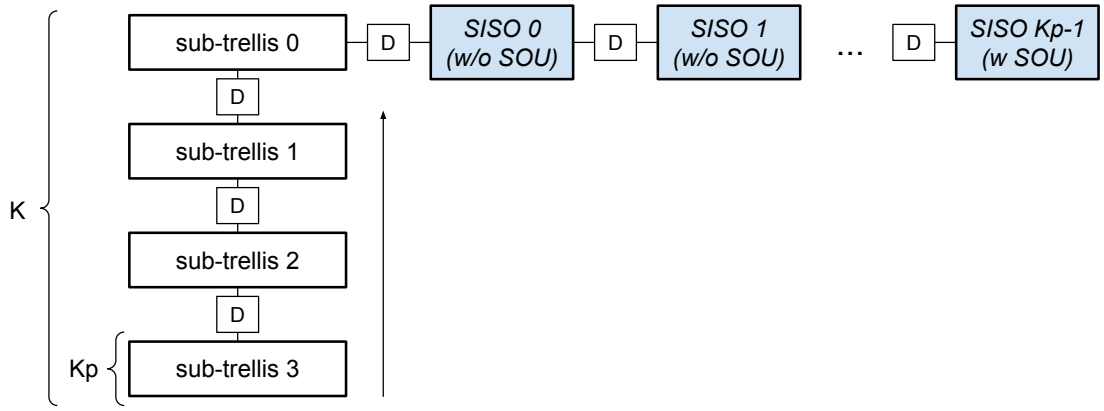

Figure 5.11: Sequential SISO half-iteration block scheme.

However, employing sub-frames is introducing an important advantage in terms of flexibility, giving the possibility to adapt the decoding process to frame sizes which are multiple of *Kp*. In order to ensure this property, a larger complexity is expected on the permutation related hardware, since it is necessary to adapt the interleaving function to the frame dimension. Typically, two solutions are available: the first one consists in involving multiple smaller interleavers that are capable to work on Kp-bits sub-frames. This approach has a negative impact on the error-correction performances, especially when frames larger than *Kp* are processed. The second solution consists in giving full control on the interleaving law, employing flexible hardware (generally crossbars). The BER is not affected in this case, since the permutation is completely customizable. However, a large impact on the hardware complexity is expected. Hybrid solutions are also available, looking for a trade-off between error-correction capabilities and complexity, as suggested in [13].

Both the concurrent and sequential approaches are valid solutions to improve the throughput. Moreover, as pointed out in Chapter 7, important State-of-the-Art architectures are derived employing SISO-level parallelism. In this work, the concurrent SISO parallelism will be considered as a reference choice, since it is maintaining intact the original recursive nature of the SISO processor. However, the best option typically depends on the Turbo-decoder specifications, such as frame size and sub-frame size, and consequently it should be properly selected from case to case.

Shuffled SISO

As presented in the first Turbo-decoder example, in figure 4.2, there is the possibility to perform the natural and the interleaved processing in parallel, if the total number of SISO modules is doubled. Following this approach, two half-iterations can be performed at the same time, reducing the latency and improving the throughput. The main drawback is on the hardware complexity.

5.6 Decoder Parallelism

A further degree of parallelization can be introduced by including different Turbo-decoders processing F frames at the same time. Also in this case, considering a half-iteration, it is possible to organize the Turbo-decoder instances in a concurrent fashion or in a sequential one, as shown in figure 5.12.

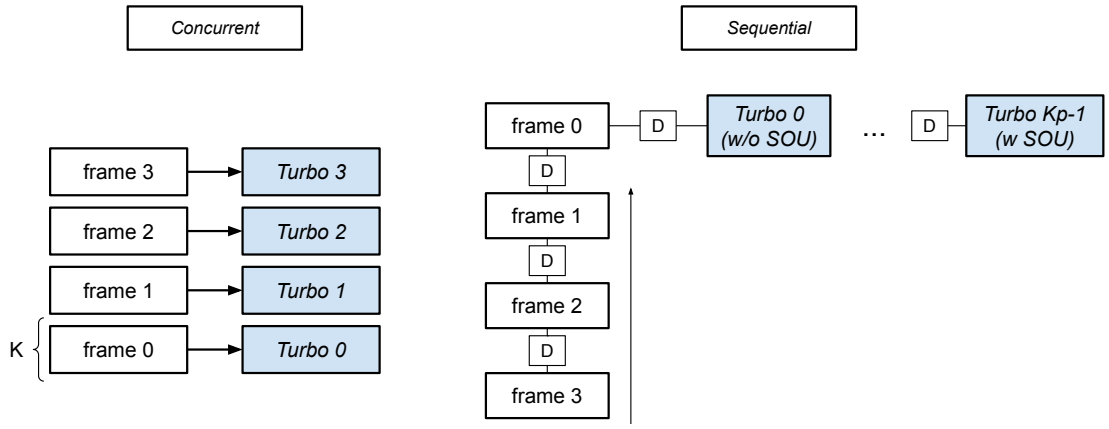

Figure 5.12: Concurrent and sequential Decoder-parallelism block schemes, considering a half-iteration.

As noticeable, the organization is similar if compared to the SISO-level parallelism. More specifically, the Turbo modules shown in the sequential approach are composed by many sequential-SISO units operating in parallel on different sub-frames. In this work, the choice is focused on the concurrent approach: since all the Turbo-decoders are performing the same algorithmic steps in parallel, some elements in the architecture, like the control signals or the memory access addresses, can be shared among the different instances, reducing the overall complexity. Moreover, it is expected for the different layers of pipeline registers included in the sequential approach to have a non-negligible impact on the area.

5.7 Iteration Parallelism

A last parallelization level can be explored, considering multiple iterations performed on different frames at the same time. The Decoder-level parallelism, presented in the previous section, is meant to process the same half-iteration on multiple frames. As a consequence, in order to fully complete the decoding process, it is necessary to reuse the architecture multiple times until the correct number of iterations is performed.

The architectures presented in figure 5.12 can be spatially parallelized, interposing interleaver and de-interleaver functions between them, as shown in figure 5.13. Iteration

parallelism can be also applied on plain Turbo-decoders architectures, without considering any Decoder-level parallelization.

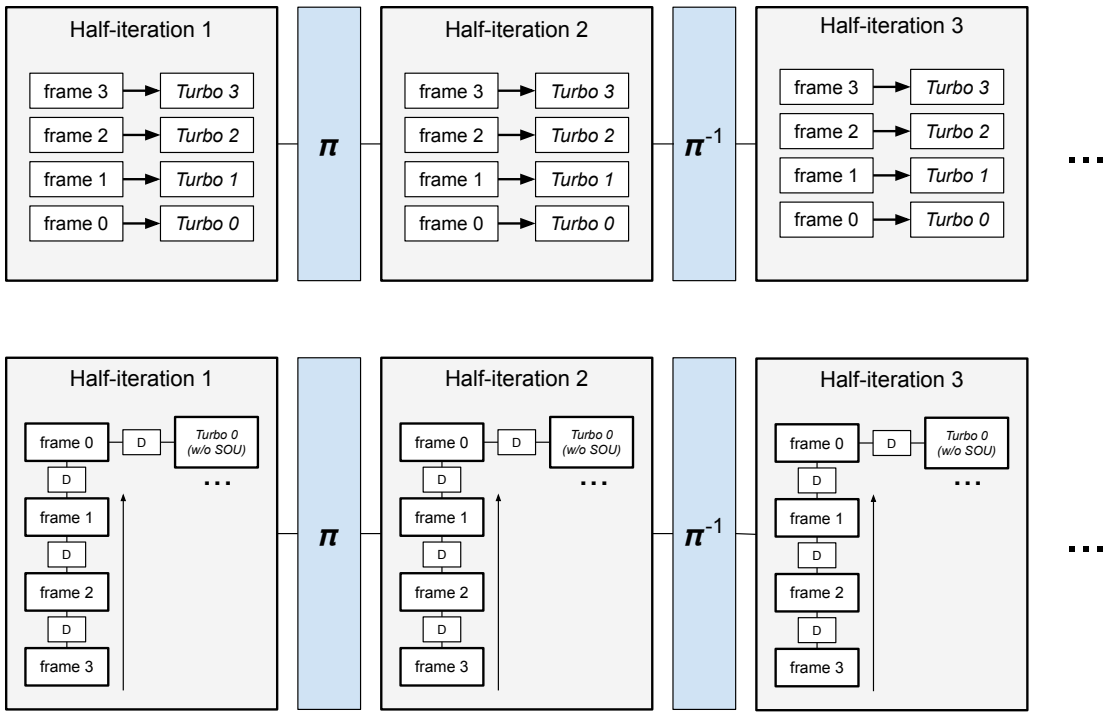

Figure 5.13: Iteration parallelism applied to Decoder-parallelized architectures.

The powerful advantage, for the cases shown in figure 5.13, is the possibility to introduce new frames in the architecture avoiding to wait for a specific amount of iterations to be performed.

The throughput benefits come with some drawbacks: the complexity is increasing proportionally to the half-iterations number, due to the additional Turbo-decoders and interleaver blocks. Moreover, the total amount of half-iterations is assumed to be fixed, which is denying the possibility to easily exploit the usage of a *stop criterion*, an important technique to increase the throughput.

Chapter 6

BER Performance

This chapter aims to analyze the quantitative impact of some architectural choices on the error-correction performances, giving the guidelines on how to select Turbo-decoders specifications, in order to maintain an acceptable Bit-Error-Rate. All the presented estimations are performed employing a Turbo-decoder model written in C-language [1], from which BER data can be extracted and then graphically represented. The last section is fully dedicated to the quantized representation of parameters in a Turbo-decoder architecture, considering the Max-Log-Map algorithm.

6.1 BER Analysis

In this section, the BER variation will be analyzed as function of the *window* size, the *sub-frame* dimension, the *channel LLR* bitwidth, the employed *permutation law* and the max^* operator correction approach. The C-model is capable to emulate a concurrent SISO architecture, including the usage of windowing.

The parameters indicated in table 6.1 will be considered as default values, when not under analysis. As a unique exception, when comparing the permutation laws, the frame-size is lowered to 4096 bits, for compatibility purposes to both the LTE/UMTS standards.

[1] Turbo-Decoder model available at the VLSI group - Politecnico di Torino

Parameter	Symbol	Default Value
Frame-size	K	6144 bits
Sub-Frame-size	Kp	256 bits
Window-size	WS	32 bits
Channel LLR bits	w	6 bits
Iterations	n_I	4
Max^* correction	-	ESF
Permutation law	-	LTE standard

Table 6.1: Reference C-model parameters for the BER analysis.

6.1.1 Window Size

As mentioned during the introduction of windowing, it is expected for smaller windows to have a negative impact on the error-correction. Different sizes WS have been tested on the reference model, tracking the BER as function of E_b/N_0. The plot in figure 6.1 is summarizing the results.

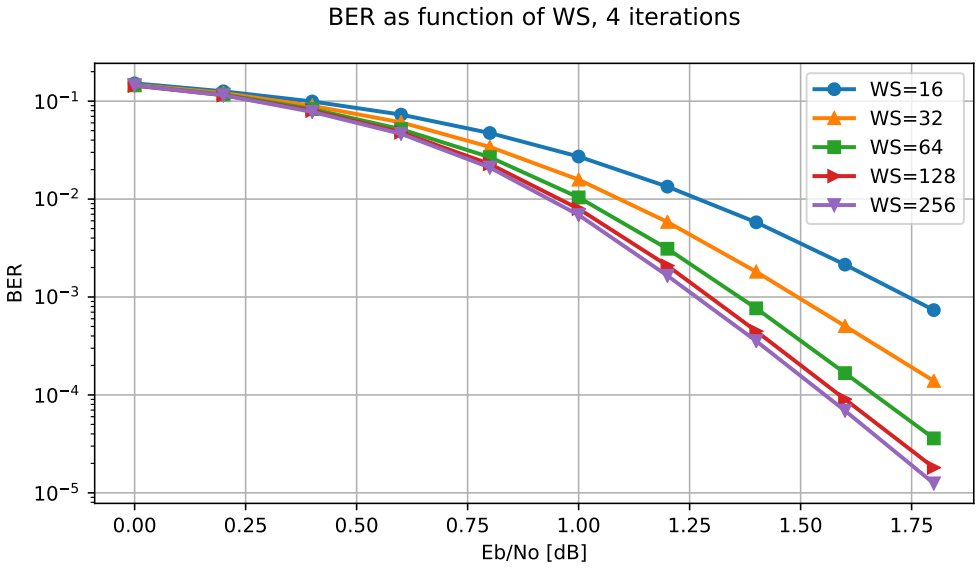

Figure 6.1: BER variation as function of the window size WS.

As visible, the major benefits from increasing the window size are found around 32 bits or 64 bits. The impact of larger windows on the BER is significantly smaller. For this reason, 32 bits can be considered as an optimal choice, introducing an acceptable trade-off between memory storage requirements and error-correction performances.

6.1.2 Sub-Frame Size

Similarly to windows, also introducing sub-frames is producing a BER degradation, as shown in figure 6.2.

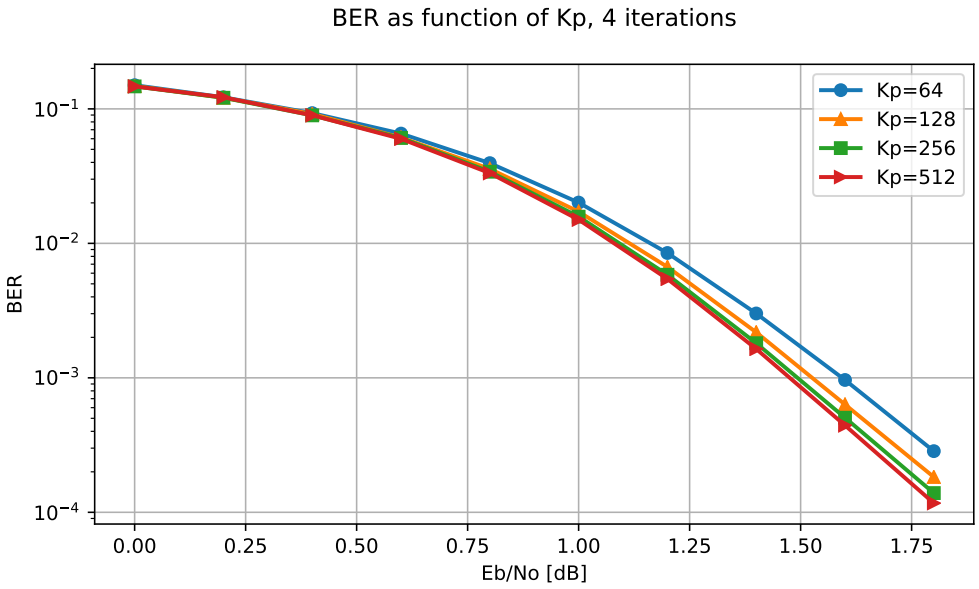

Figure 6.2: BER variation as function of the sub-frame size Kp.

Since sub-frames are larger than windows, their impact is less noticeable. From the reported graph, it is possible to observe that above sizes of 128 bits or 256 bits the BER improvements are almost not noticeable. Therefore, considering the scenario under analysis, an acceptable reference value for the sub-frame size could be around 256 bits.

6.1.3 Channel-LLR Bitwidth

As detailed in the next section, the amount of bits over which the channel LLRs are represented is affecting the accuracy of other fundamental quantities, such as the branch metrics or the state metrics. A larger bitwidth is positively affecting the BER, introducing higher precision in the representation. On the other hand, both logic and the memory areas are expected to proportionally increase with larger bitwidths.

As shown in figure 6.3, working below 5 bits is significantly affecting the BER. On the contrary, above 6 bits, the improvements are not noticeable from the graph, since the curves are superimposed. A slight difference is found between 5 bits and 6 bits, especially for high E_b/N_0 values.

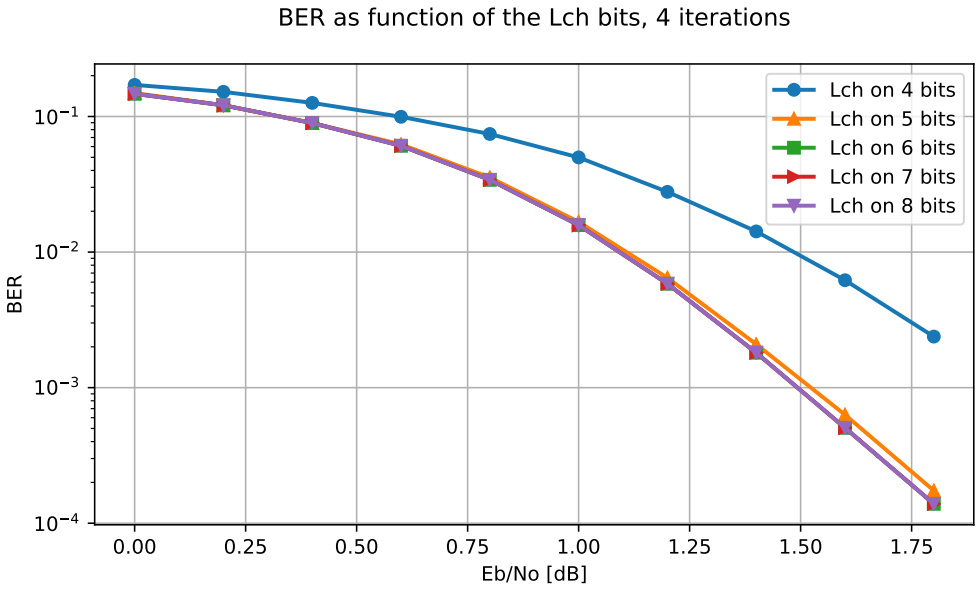

Figure 6.3: BER variation as function of the channel LLR bitwidth.

Representing the channel LLR information on 6 bits is an optimal choice, avoiding to introduce BER degradation, while maintaining an acceptable complexity.

6.1.4 Permutation Law

The choice of the permutation law can significantly affect the achievable BER. Therefore, the LTE and UMTS standard permutations have been compared. Respectively, they are considering a Quadratic Permutation Polynomial (QPP) interleaver and a Almost Regular Permutation (ARP) interleaver.

Observing figure 6.4, it is noticeable how the two curves are superimposed, considering the default parameters. Therefore, choosing a specific permutation law between the presented ones is not expected to introduce any significant improvement.

As mentioned after introducing the interleaving concept, permutation laws are object of study, mainly due to their impact on the error-correction capabilities. As a consequence, it is important to consider the possibility for new standards to be introduced in the next future.

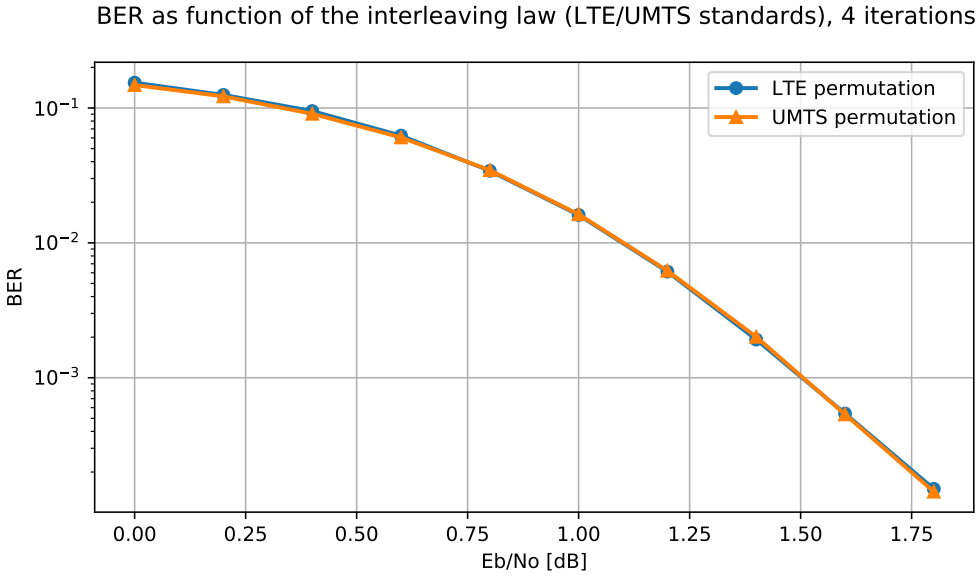

Figure 6.4: BER variation as function of the permutation law.

6.1.5 Max* Correction

The *max* operator employed in the Max-Log-Map algorithm is derived from the *max** operator, by neglecting the Jacobian logarithm correction factor, which is included by the Log-Map algorithm employing the usage of a LUT. In order to partially overcome the BER degradation, an Extrinsic-Scaling-Factor (ESF) can be employed in the MLM algorithm, in order to compensate the approximation. Three possible cases are compared in figure 6.5.

As expected, the ESF is moving the BER curve toward the Log-Map one, which is employing the LUT correction factor. The ESF will be considered as a reference choice in this work, in order to improve the BER, while maintaining the usage of the MLM algorithm. Moreover, the cost in terms of additional complexity is expected to be limited, since it is simply necessary to scale the soft-output values.

6.2 Quantization and Bitwidths

It is fundamental to define guidelines to determine the amount of bits over which quantities are represented in the hardware architecture, also called bitwidths, considering their impact on both the BER and the complexity.

The starting point are the LLRs derived from the channel. As a first step, a *quantization* operation needs to be considered, in order to represent the LLRs on two's complement

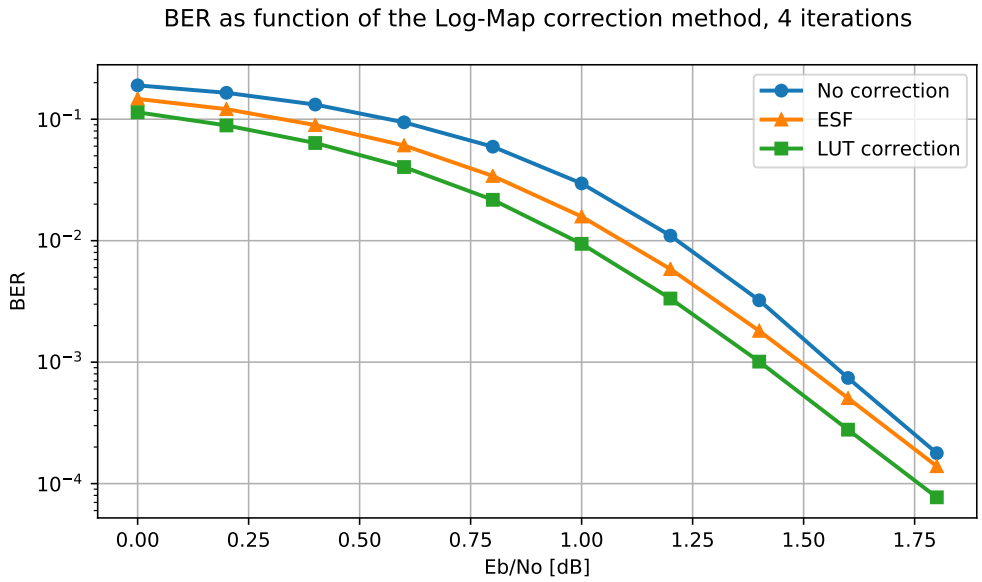

Figure 6.5: BER variation as function of the max* correction approach.

fixed-point numbers, within a given amount of bits. Then, if necessary, quantized values are required to be *saturated*. The Log-Likelihood-Ratio equation is recalled below.

$$L(y_k) = \frac{2y_k}{\sigma^2}$$

Since the MLM algorithm is not affected by the variance estimation, the quantization will be applied on y_k, which is expected to range in a generic interval $[-A, A)$. As proposed in [10], a typical value for A, considering a Turbo code-rate equal to 1/2, is 1.2. As the rate increases, the A parameter, known as *interval of quantization*, is expected to increase. The quantization and saturation functions are reported below, considering w as the total amount of selected bits for the representation.

$$y_k^Q = sat\left(\left\lfloor y_k \frac{2^{w-1}-1}{A} + 0.5 \right\rfloor, 2^{w-1}\right)$$

$$sat(a,b) = \begin{cases} a & if\ a \in [-b,\ b-1] \\ b-1 & if\ a > b-1 \\ -b & if\ a < -b \end{cases}$$

Typical w values are ranging from 3 to 6 bits [10]. As mentioned in the previous section, the reference value considered in this work is $w = 6$ bits. The extrinsic information L_e is typically quantized in the range [-2A, 2A), as reported in literature [10]. Therefore, starting from w, L_e will be quantized over $w + 1$ bits.

Knowing the quantization for the channel LLR and the extrinsic information, the branch metrics bitwidth can be derived. Considering the radix-2 case, the worst-case branch metric is given by the sum between L_s, L_p and L_a, respectively expressed on w, w and $w+1$ bits. Therefore, the correct bitwidth for the branch metrics is $w+2$. When higher radix orders are considered, the quantization is expected to be performed on larger intervals. For instance, a radix-4 branch metric is requiring $w+3$ bits, since it is defined as the sum between two radix-2 branches. The table 6.2 is reporting the expected bitwidths up to a radix-16 architecture.

Radix order	w_Γ
2	w+2
4	w+3
8	w+4
16	w+5

Table 6.2: Branch metrics bitwidths as function of w, considering different radix orders.

Regarding the state metrics quantization, two aspects need to be considered. First, due to the recursive propagations, their value is expected to increase proceeding through the Trellis-Diagram. Second, the bitwidth should be set in order to preserve the information about the differences computed when evaluating the next state metrics or the soft-output values.

A possible solution to this problem is found by employing the *hardware modulo normalization* [14]. Given a set of fixed-point numbers, expressed on w bits, the information about their difference Δ is preserved if the following condition is met.

$$\Delta_{max} \leq 2^{w-1} - 1$$

As shown in figure 6.6, where the full range of values representable on w bits is reported on a circle, the required condition is met if all the numbers in the given set are falling inside a precise half-circumference. In the examples, two set of values are represented, respectively in green and red. The former is respecting the given condition, as a consequence w bits are enough to preserve the difference between those numbers. On the other hand, considering the red set, w bits are not sufficient to correctly maintain the information on the difference.

If the given condition is met, the continuous increment of the metrics during the propagation is correctly handled, since adders will automatically perform a modulo normalization over a fixed amount of bits, when the overflow condition is reached. This solution is optimal also from the complexity point of view, since it is not requiring any large additional hardware component to be included.

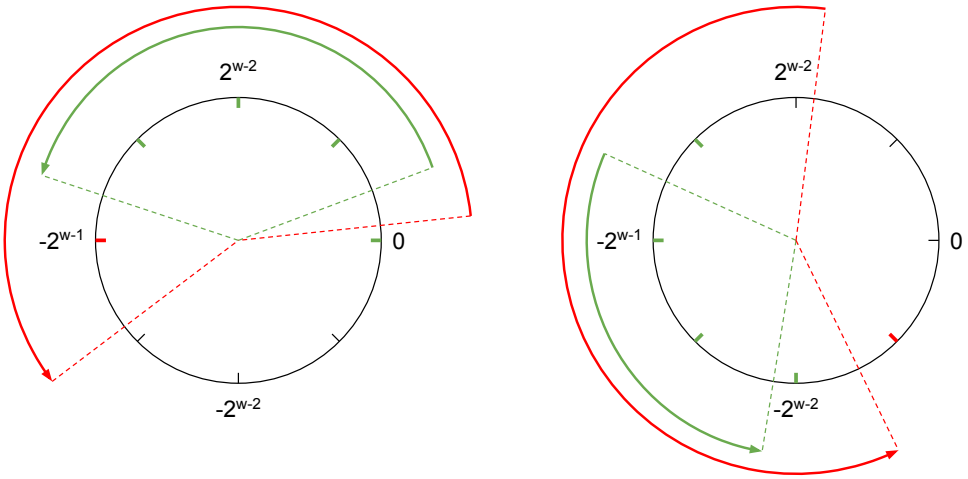

Figure 6.6: Modulo normalization condition graphically represented.

It is now necessary to select the bitwidth in order to fulfill the condition. During the application of the Max-Log-Map algorithm, the worst-case difference Δ_{max} is found when evaluating the soft-output values, as reported in the following equation.

$$\begin{aligned} L(u_k) = & \underset{(S_k,S_{k+1})|u_k=1}{max^*}(A(S_k) + \Gamma(S_k, S_{k+1}) + B(S_{k+1})) - \\ & - \underset{(S_k,S_{k+1})|u_k=0}{max^*}(A(S_k) + \Gamma(S_k, S_{k+1}) + B(S_{k+1})) \end{aligned}$$

The computation is considering two times the difference between two state metrics Δ_{SM} and a difference between two branch metrics Δ_Γ. Therefore, Δ_{max} can be estimated as follows.

$$\Delta_{max} = 2\Delta_{SM}^{max} + \Delta_\Gamma^{max}$$

The maximum branch metrics difference is found considering a first metric computed with all the codeword bits set to 1 and a second metric obtained from a all-zeros codeword. The table 6.3 is reporting Δ_Γ^{max} considering different radix orders, as function of the channel information bitwidth w.

Regarding the maximum state metrics difference, it is possible to prove that Δ_{SM} is bounded when performing the metrics propagation in the Trellis-Diagram [14]. A possible way to derive this boundary is by simulating the forward or backward recursion, considering a zero initial condition for the state metrics [12]. The simulation is carried on until a steady state is reached. Then, Δ_{SM}^{max} is corresponding to the maximum observable difference. Employing this technique produced a result equal to $4 \cdot 2^w$, considering the standard LTE/UMTS code [4].

Radix Order	Δ_{Γ}^{max}
2	2^{w+1}
4	$2 \cdot 2^{w+1}$
8	$3 \cdot 2^{w+1}$
16	$4 \cdot 2^{w+1}$

Table 6.3: Δ_{Γ}^{max} as function of w considering different radix orders.

It is now possible to apply the modulo normalization condition in order to compute the state metrics bitwidth w_{SM}.

$$w_{SM} = \lceil log_2(2\Delta_{SM}^{max} + \Delta_{\Gamma}^{max} + 1) + 1 \rceil$$

All the obtained results will be employed in this work in order to set the bitwidths for all the involved quantities, starting from w and considering different radix orders. The results for $w = 6$ bits are summarized in table 6.4.

Radix Order	w_{ext}	w_{Γ}	Δ_{Γ}^{max}	Δ_{SM}^{max}	Δ_{max}	w_{SM}
2	7 bits	8 bits	128	256	640	11 bits
4	7 bits	9 bits	256	256	768	11 bits
8	7 bits	10 bits	384	256	896	11 bits
16	7 bits	11 bits	512	256	1024	12 bits

Table 6.4: Fundamental bitwidths summarized for different radix orders, considering $w = 6$ bits.

Chapter 7

State of the Art Architectures

This chapter aims to review some fundamental modern Turbo-decoders architectures, focusing on the employed parallelization degrees and pointing out their main characteristics. The analyzed architectures are the Parallel-MAP (PMAP), the X-MAP (XMAP), the Fully-Parallel-MAP (FPMAP) and the Unrolled-X-MAP (UXMAP). The architectures will be compared in terms of throughput, latency and logic complexity. The latter is estimated considering the total amount of logic units found in the architecture. As highlighted in this work, this kind of complexity estimation is not fully descriptive, since memory organization plays an important role in Turbo-decoders.

7.1 PMAP

The Parallel-MAP (PMAP) architecture [15] is mainly focused on the usage of *concurrent-SISO* parallelism. The original K-bits frame is subdivided in Kp-bits sub-frames. The architecture is meant to process those sub-frames in parallel employing $N = K/Kp$ SISO-modules. Moreover, algorithmic level parallelism (scheduling) and windowing are meant to be employed in this type of architecture.

The block scheme in figure 7.1 is including a total of 4 PMAP-cores. Moreover, a Forward-Backward scheduling policy is employed, including 2 windows per each sub-block. A first approximation for latency, throughput and logic complexity is given by the following equations, considering the working clock frequency f, the clock period $T = 1/f$ and a generic amount of half-iterations n_{HI}.

$$\text{Throughput (PMAP)} = \frac{K \cdot f}{(Kp + WS) \cdot n_{HI}}$$

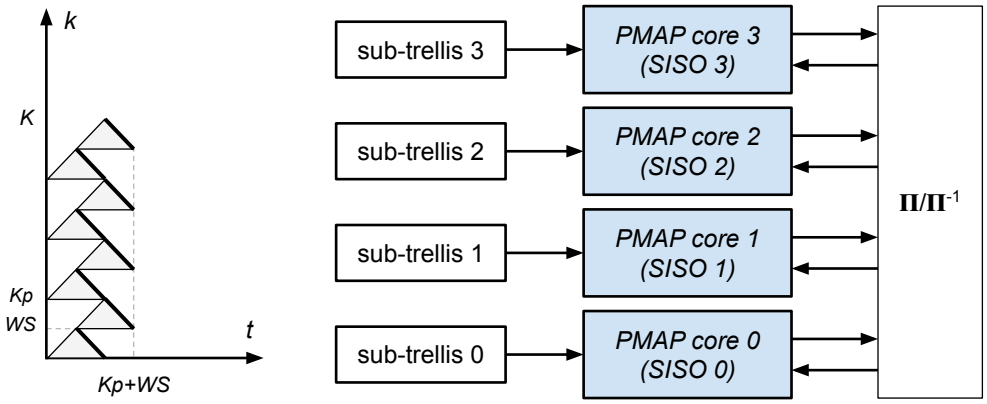

Figure 7.1: Scheduling and block scheme organization for the PMAP architecture.

$$\text{Latency (PMAP)} = (Kp + WS) \cdot n_{HI} \cdot T$$

$$\text{Logic Complexity (PMAP)} \propto K/Kp \cdot (2 \cdot BMU + 2 \cdot PMU + SOU)$$

It is noticeable how increasing the number of sub-frames is generating a trade-off between better throughput and latency and worse complexity. On the other hand, reducing the window size is apparently increasing the throughput without affecting the complexity. However, WS is affecting the internal architecture of a SISO processor, in particular the memories for the NII and the alpha metrics, as well as the achievable error-correction performances. For this reason, the presented equations are to be intended as a rule of thumb to understand the general effect of the parameters on the SISO-level organization.

Following the Forward-Backward scheduling, it is expected for the architecture to include $2N$ BMU, $2N$ PMU and N SOU. In case Butterfly scheduling is employed, N additional SOU would be required.

Focusing on flexibility, the architecture can be adapted to work on frames with multiple sizes, as explained in Chapter 5. Moreover, puncturing is ensuring the adaptability on multiple code-rates. As a last aspect, also the half-iterations number is customizable by introducing a stop-criterion.

7.2 XMAP

The X-MAP (XMAP) architecture [16] is exploiting a *sequential-SISO* parallelism, in addition with a Butterfly scheduling. Also in this case, Kp-bits sub-frames are defined,

which are processed by the XMAP-core, depicted in figure 7.2.

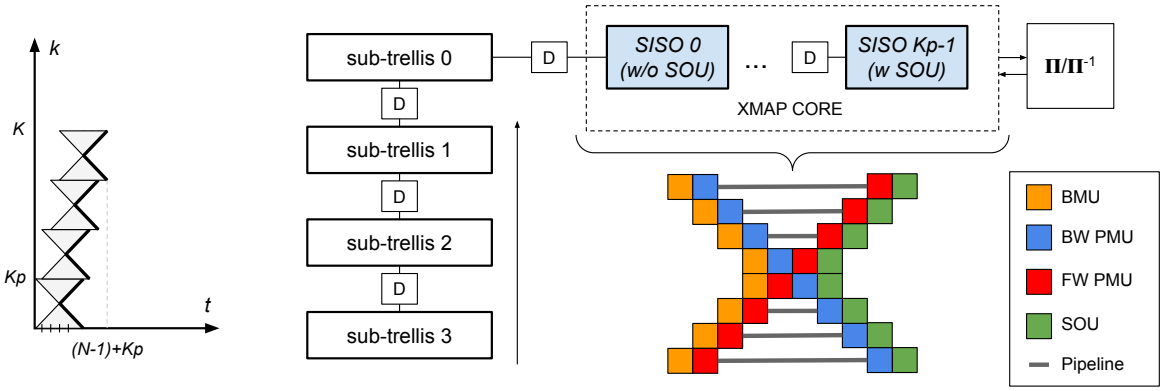

Figure 7.2: Scheduling and block scheme organization for the XMAP architecture.

As visible, it is possible to start decoding a new sub-frame each clock cycle. As a difference with respect to the reference sequential-SISO parallelism, presented in Chapter 5, the XMAP core is including half the expected amount of BMU. The branch metrics evaluated during the first part of the processing (orange squares) are forwarded to the second scheduling region, where soft-output information is computed, employing pipelines. Since also the state metrics are required to be forwarded, additional pipelines are included. Following this approach, the SOU can access the required information to estimate the a-posteriori LLRs. Throughput, latency and logic complexity can be first approximated considering the equations below.

$$\text{Throughput (XMAP)} = \frac{K \cdot f}{(Kp + N - 1) \cdot n_{HI}}$$

$$\text{Latency (XMAP)} = (Kp + N - 1) \cdot n_{HI} \cdot T$$

$$\text{Logic Complexity (XMAP)} \propto Kp \cdot (BMU + 2 \cdot PMU + SOU)$$

Increasing the amount of sub-frames is not always improving latency and throughput. The main reason is the dependency from the total number of sub-frames N, found in the latency and throughput estimations. The complexity is now proportional to Kp, in opposition with the inverse proportionality found for the PMAP architecture.

As noticeable from the equations, the throughputs achievable by the PMAP and the XMAP architectures are expected to be similar. As a rule of thumb, if Kp $\gg$ N, the PMAP architecture is expected to introduce a lower complexity in terms of logic units. On the contrary, if Kp $\ll$ N, the XMAP is introducing less computational units.

In terms of flexibility, also this architecture can be adapted to handle multiple frame-sizes and different code-rates. Moreover, also a stop-criterion can be introduced.

7.3 FPMAP

The Fully-Parallel-MAP (FPMAP) architecture [17] can be considered as an extreme application of the *concurrent-SISO* parallelism, considering a sub-frame size Kp equal to 1. Moreover, this architecture is also exploiting the usage of *shuffled-SISO* decoders in order to further improve the throughput. As a consequence, given the frame size K, the total number of SISO processors is $2K$.

Each SISO decoder has the capability, considering a single Trellis-section, to compute the branch metrics, perform the alpha/beta propagations and compute the extrinsic information. More specifically, the evaluated state metrics are exchanged to the neighboring SISO modules, as shown in figure 7.3.

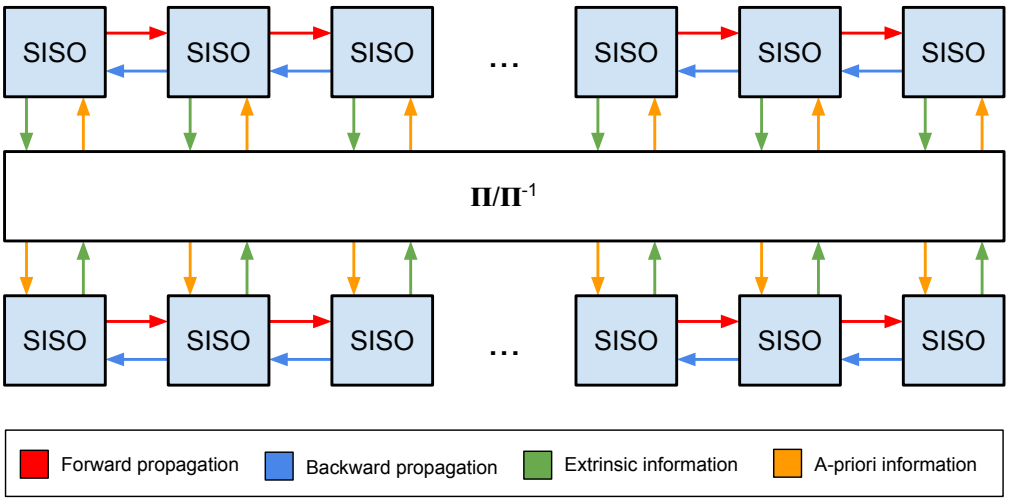

Figure 7.3: Block scheme organization for the FPMAP architecture.

This specific SISO organization is expected to complete a single iteration in a clock cycle, considering the use of shuffled decoding and the dimension 1 sub-frames. Latency, throughput and logic complexity estimations are reported below.

$$\text{Throughput (FPMAP)} = \frac{K \cdot f}{n_I}$$

$$\text{Latency (FPMAP)} = n_I \cdot T$$

$$\text{Logic Complexity (FPMAP)} \propto 2K \cdot (BMU + 2 \cdot PMU + SOU)$$

As noticeable, the logic complexity is now depending on the frame size. Therefore, the required amount of logic units is expected to be higher if compared to the two previous architectures. The throughput is bounded by the number of iterations, which is expected to be dramatically increased, due to the BER degradation introduced by the small sub-frame size. This last issue is drastically limiting the area efficiency of the architecture.

The FPMAP approach is presenting less flexibility in terms of frame size, due to the architecture's organization. However, puncturing and stop-criterions are still available.

7.4 UXMAP

The Unrolled-X-MAP (UXMAP) architecture, reviewed in [18], is an evolution of the XMAP approach, in which a *sequential decoder-level* parallelization is employed, by including multiple XMAP cores in parallel in order to process multiple frames. Moreover, also *iteration parallelism* is considered, by spatially replicating the overall structure and including interleaving/de-interleaving functions, as depicted in figure 7.4.

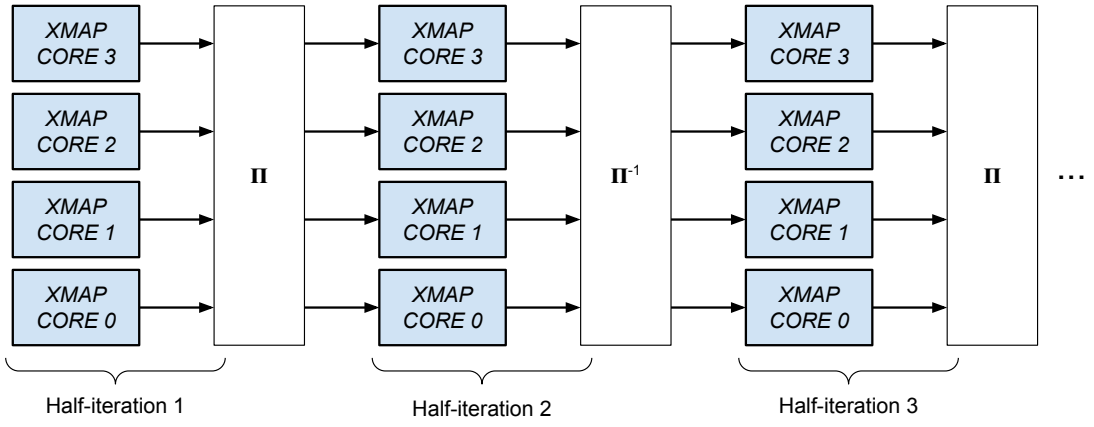

Figure 7.4: Block scheme organization for the UXMAP architecture.

This architecture is capable to start decoding a new frame each clock cycle, reaching the maximum throughput among the presented State-of-the-Art architectures. Throughput, latency and logic complexity can be estimated as follows.

$$\text{Throughput (UXMAP)} = K \cdot f$$

$$\text{Latency (UXMAP)} = n_{HI} \cdot Kp \cdot T$$

$$\text{Logic Complexity (UXMAP)} \propto n_{HI} \cdot N \cdot Kp \cdot (BMU + 2 \cdot PMU + SOU)$$

As visible, the logic complexity is negatively affected by this architectural approach. Moreover, interleaving and de-interleaving functions are typically *hardwired* between the different stages, affecting the flexibility on the frame-size. In order to partially solve this last issue, hybrid interleaving connections, including a degree of flexibility, can be employed [13]. Furthermore, since the number of half-iterations is meant to be fixed, stop-criterions are not expected to be applied in this scenario. On the other hand, the architecture is fully flexible on the code-rate, since puncturing can be employed.

7.5 Conclusions

A quantitative comparison between practically implemented architectures is presented in table 7.1, which is including the evaluation of some fundamental KPI.

Reference	[19]	[20]	[21]	[22]
Architecture	PMAP	XMAP	FPMAP	UXMAP
Frame-size	6144	6144	6144	128
Sub-Frame-size	384	32	1	32
Window-size	32	32	-	-
Radix	4	4	2	4
Max iterations	5.5	7	39	4
Technology [nm]	65	28	28	28
Frequency [MHz]	410	625	252	800
Throughput [Gb/s]	1.0	1.1	39.86	102.4
Area [mm^2]	2.5	0.49	24.09	16.54
Area efficiency [$Gb/s/mm^2$]	0.41	2.3	1.65	6.20

Table 7.1: Comparison table between different implemented architectures, considering the PMAP, XMAP, FPMAP and UXMAP approaches.

A first noticeable difference is found in the throughputs achievable by the FPMAP and the UXMAP approaches, far higher if compared to the PMAP and the XMAP ones. This aspect is highlighting the importance of improving the degrees of parallelization, in case very high-throughputs are required.

It is remarkable how the XMAP-based architectures are limiting the sub-frame size, in order to receive benefits from it, in terms of low complexity. For the same reason, the UXMAP implementation is also working with a much shorter frame, if compared to the other solutions. Typically, the PMAP and the XMAP architectures are expected to reach similar throughput performances, around 1 or 2 Gb/s [23]. It is important to consider that the throughput estimation is strongly related to the working clock frequency involved and, as a consequence, on the critical path, which is depending on the selected technology.

As expected, considering the FPMAP example, throughput and area efficiency are strongly limited by the large required number of half-iterations, introduced with the aim to reach an acceptable error-correction performance. The UXMAP architecture is outperforming the other presented solutions in terms of throughput, mainly due to the exploitation of several degrees of parallelization, which are generating an important drawback on the area.

In order to directly compare the area occupation, it is necessary to consider architectures implemented on the same technology. If this condition is not met, a scaling factor could be applied, aiming to introduce a first approximation. As a consequence, also the area efficiency is affected by the technological node.

High-radix approaches can be applied to all the architectures presented in this chapter. As visible from the examples in table 7.1, the radix-4 approach is typically considered as an optimal choice, due to the limited increment in the logic complexity, as well as the throughput benefit.

Studies on the UXMAP architecture have shown that the overhead in logic complexity introduced by radix-8 and radix-16 approaches is exceeding the saved area due to the lower amount of pipelines, resulting in a less efficient choice [18]. As a consequence, radix-orders higher than 4 should be automatically discarded.

As stated in the introduction, this work aims at finding if the same consideration can be extended on PMAP-based architectures or if higher-radix orders are available. Moreover, the suitability of a Concurrent-PMAP architecture is investigated, including multiple PMAP decoders working in parallel.

Chapter 8

High-Radix PMAP Exploration

As mentioned in the previous chapters, introducing high degrees of parallelism is a key-point to move Turbo-decoder architectures toward high throughputs. When parallelization is employed, especially at SISO or Decoder level, area efficiency is the reference indicator to compare different solutions. For instance, if SISO-level parallelization is employed, the most efficient SISO architecture allows to maximize the amount of modules in a given area, consequently maximizing the achievable throughput. The same consideration can be extended to multiple Turbo-decoders.

Important informations can be derived by comparing State-of-the-Art implementations. However, this comparison could not provide a full picture, since the solutions under analysis could be based on different specifications and realized on different technologies. Following these considerations, it could be difficult to study the effect of the employed algorithmic or architectural choices separately. In order to introduce a fair comparison and explore the solutions space, a generalized approach is proposed in this work. The aim of this chapter is to develop an analysis tool capable to accomplish specific comparisons among different architectures, given their specifications. The objective is to explicit the guidelines to select the best design choices in order to maximize throughput and area efficiency.

The first section will introduce a possible methodology to define a comparison model. The presented steps will be employed in the successive sections to explore area efficiency and throughput on different radix orders, including SISO and Decoder level parallelism. Assumptions on the developed model will be declared, as well as the included degrees of freedom. Proceeding in the chapter, logic area, memory area and throughput will be discussed in separated sections, introducing proper estimation models for each quantity. Then, uncertainty estimation techniques will be discussed. In the last part of the chapter, some details on the complete architectural model will be presented, including the advantages of a Concurrent-PMAP architecture.

8.1 Methodology

As a first step, it is necessary to list the indicators that will be employed to compare different architectures. Considering each indicator, a mathematical model is derived. Each parameter contained in those models is required to be estimated by the analysis tool.

The next step consists in identifying the solutions space, highlighting the full set of architectures to be compared. As detailed in chapter 5, there are many available choices just considering the available parallelization degrees. Many others aspects can be studied, such as the usage of different computational blocks or different memory access policies. Furthermore, also different decoding algorithms can be selected. Each analyzed solution needs to be detailed enough in order to explicit all the parameters required for the indicators' computation.

With the aim of maximizing the model generalization, it is useful, if possible, to estimate the indicators independently of the employed technology. For instance, areas can be measured in terms of equivalent number of elementary gates, avoiding to express them in mm^2. During this step, synthesis operations on small architectural blocks can be useful in order to improve the accuracy of some indicators, as well as analyze their variation as function of given parameters.

Once every indicator has been modeled, the analysis tool should be able to detail specifications on single architectures, given the set of input parameters. Moreover, it is useful to introduce the capability to analyze how indicators are modified while exploring the solutions space, understanding which architectures best answer to given requirements.

A last fundamental aspect to be included is the uncertainty model, since the analysis tool is based on estimations. Each major quantity should be associated to an error, typically expressed as a percentage of the nominal value. Those errors can be propagated in the mathematical models employed for the indicators, by applying the *deterministic approach*. A further possibility is to observe the indicators' sensitivity as a function of specific quantities. The uncertainty study is fundamental to prove the robustness of the obtained results.

8.2 Assumptions

In this work, two indicators will be observed: throughput and area efficiency. The latter is expressed considering the equation reported below. A total of three contributions are required to be estimated: *throughput*, *logic area* and *memory area*. It is noticeable how the study on the area efficiency is automatically including a throughput analysis.

$$A_{eff} = \frac{Throughput}{Area(logic) + Area(memory)} = \frac{Th}{A_L + A_M}$$

Since this study is centered around the use of high-radix orders, it is expected for the logic area to exponentially increase with the radix. However, the impact on the area efficiency strongly depends on how the area is distributed between logic and memory. If the Turbo-decoder area is dominated by memories, this could potentially open the possibility to introduce efficient high radix-order solutions.

As expressed in the previous section, each quantity under analysis requires an estimation model, which is desired to be flexible and adaptable for all the considered solutions under test. Therefore, the full set of explored architectures is defined, considering the following assumptions.

- The **LTE/UMTS standard code** is employed as a reference convolutional code.
- The **Max-Log-Map algorithm** is adopted, including the usage of the **Extrinsic-Scaling-Factor**.
- Each architecture is considering a **Forward-Backward scheduling** policy.
- The **Next-Iteration-Initialization** is the reference initialization method.
- All the architectures are **PMAP-based**, including the possibility for concurrent Decoder-level parallelism (**Concurrent-PMAP**).

On the other hand, the following degrees of freedom will be considered by the model.

- Variable **Frame size** (K).
- Variable **Sub-frame size** (Kp).
- Variable **Window size** (WS).
- Variable **Channel LLR bitwidth** (w).
- Variable **Radix order (2, 4, 8 and 16 available)**.

Other important aspects regarding specific choices in the model will be discussed further in this chapter. More specifically, the next sections are including a detailed analysis on logic, memory and throughput.

8.3 Logic Analysis

Since the PMAP architecture is based on spatially replicating instances of the same SISO decoder, it is then necessary to focus the analysis on the logic units included in a single SISO module, analyzing their variation as a function of the radix order and the data parallelism. The analyzed units are the Branch Metric Unit (BMU), the Path Metric Unit (PMU) and the Soft Output Unit (SOU). Moreover, considering the Forward-Backward scheduling, each SISO entity is including 2 BMU, 2 PMU and 1 SOU.

8.3.1 BMU

This unit is expected, starting from the channel LLR, to evaluate all the required branch metrics, necessary for each algorithmic step. As a first measure of complexity, considering different radix-orders, it is useful to analyze how many branches are required to be computed in a radix-N BMU, recalling that a radix-N branch is given by the sum among all the radix-2 branches in a given path. Table 8.1 is reporting the number of required branches up to a radix-16 architecture, indicating their bitwidths as function of w.

Radix	BM number	w_Γ
2	16	w+2
4	32	w+3
8	64	w+4
16	128	w+5

Table 8.1: Number of Branch-Metrics requested to be computed by a radix-N BMU, including their bitwidths.

As visible, the amount of required branch metrics is significantly growing with the radix-order, as well as the complexity of the involved adders, since the bitwidth is increasing. A fundamental optimization to be performed in this scenario consists in introducing the minimum amount of required adders, avoiding redundant calculations. For instance, considering the radix-2 case, 2 adders are enough to explicit all the 16 branch metrics in a single Trellis-section, as indicated in figure 4.5.

Multiple radix-2 BMUs (BMU2) are at the basis of higher-order BMUs. For instance, in a radix-4 architecture, two BMU2 units can be employed in parallel to explicit all the radix-2 branches, which can be combined to form radix-4 branch metrics. The block scheme for a generic BMU-N architecture is presented in figure 8.1.

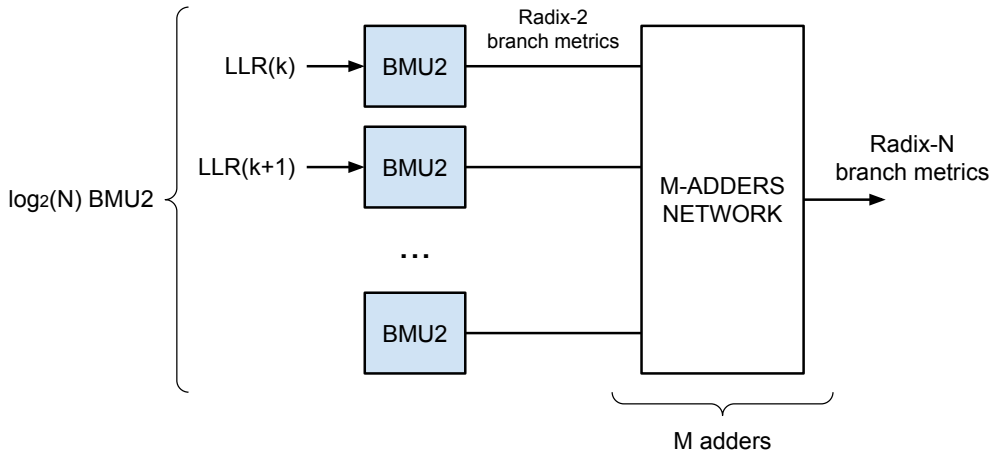

Figure 8.1: BMU-N block scheme.

As visible, the challenge for a radix-N BMU is to find the best adders network to properly combine the radix-2 branch metrics, including the minimum amount of adders. As reported in Appendix A, given a specific code, it is possible to state which additions are to be performed by recursively exploring the paths in the Trellis-Diagram. As a result, the required minimum amount of adders up to a radix-16 BMU is reported in table 8.2.

Radix	**BMU2 number**	**M-adders**	**Total adders**
2	1	-	2
4	2	9	13
8	3	54	60
16	4	149	157

Table 8.2: BMU components analysis up to radix-16 implementation, including the total amount of required adders.

Despite the adders minimization, the architectural complexity is still significantly increasing with the radix-order. Moreover, also a critical path increment is expected, due to the higher complexity found in the M-adders network. However, since the BMU is a feed-forward logic block, pipeline can be applied, aiming to reduce the critical path, with a drawback on the total latency.

8.3.2 PMU

The amount of paths to be discriminated while performing the forward/backward propagations is increasing with the radix order, as shown in figure 8.2. As a consequence, the

max operators are expected to work with a larger amount of input operands.

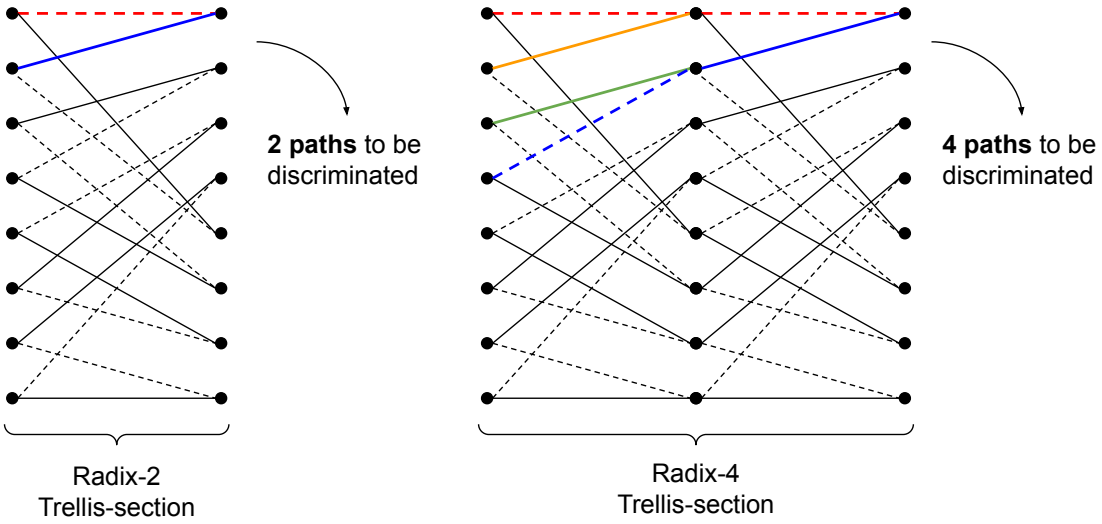

Figure 8.2: Amount of paths to be discriminated during a forward propagation, considering a radix-2 and a radix-4 implementation.

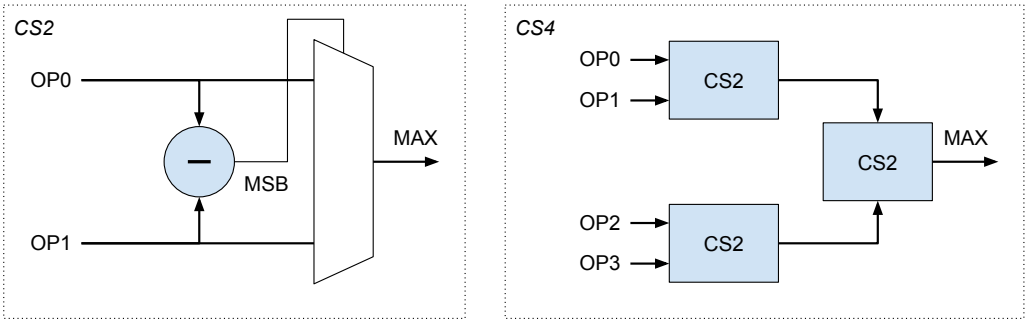

Figure 8.3: CS2 (Left) and CS4 (Right) block scheme.

The basic building block for the PMU is the Compare-and-Select (CS2) operator, implementing a maximum operation between two operands, as depicted in figure 8.3. Moreover, a layer of adders is also required to fully compute the next state metrics. If more than two paths are to be compared, a tree-like disposition of CS2 operators is capable to satisfy the request. This approach can be followed in order to determine Compare-and-Select operators working with an arbitrary number of operands. An example of a CS4 operator is shown in figure 8.3.

It is fundamental to consider the presence of the feedback loop included in the PMU, required to perform the recursive propagation. Due to this loop, the critical path is bounded, since the use of pipeline is not available. Therefore, the cascade of logic operators included in the PMU is potentially defining the *maximum working clock frequency* for the

entire Turbo-decoder architecture. Table 8.3 is reporting the expected amount of CS2 operators up to a radix-16 architecture, specifying how many units are found on the critical path. It is noticeable how the radix-4 critical path is expected to be double the radix-2 one, nullifying the throughput improvements obtained by processing multiple Trellis-sections at the same time.

Radix	Compared paths	CS-N	CS2 on the critical path
2	2	CS2	1
4	4	CS4 (3 CS2)	2
8	8	CS8 (7 CS2)	3
16	16	CS16 (15 CS2)	4

Table 8.3: Compare-and-Select operators analysis up to radix-16 implementation.

In order to solve this issue, the Compare-and-Select units for radix-orders higher than 2 can be modified, aiming to reduce their critical path, and accepting a drawback on their complexity. As suggested in [24], it is possible to implement a faster CS4 architecture by performing multiple comparisons in parallel, employing their results to drive a selection multiplexer, as shown in figure 8.4.

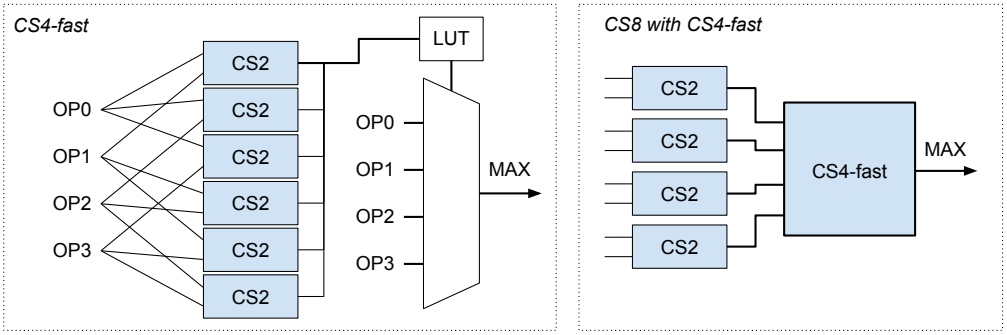

Figure 8.4: CS4-fast block scheme (Left) and CS8 implementation including a CS4-fast operator (Right).

The LUT is describing a logic function capable to interpret the CS2 results and produce the correct selection signal for the multiplexer. It is noticeable how the total amount of CS2 operators is higher if compared to the architecture presented in figure 8.3. However, the CS2 represented in 8.4 are not requiring any selection multiplexer, since only the logic result of the comparisons is required.

As an advantage, the amount of CS2 units on the critical path is now half if compared to the original implementation. Moreover, the new obtained CS4-fast architecture can be employed as a building block for radix-8 and radix-16 Compare-and-Select units, as shown

in figure 8.4. Additionally, in this work, the parallel CS2 approach presented in [24] has been also extended to a radix-8 Compare-and-Select unit, obtaining a CS8-fast operator. As detailed in Appendix B, the latter is including 28 parallel CS2 units, which results are employed to control an 8-inputs multiplexer. In the analysis model, a degree of freedom over which CS8 architecture will be employed is guaranteed, in order to explicit the most efficient choice. Further details on this last operator and how to derive the content of the LUTs are included in Appendix B. Furthermore, a possible approach to reduce the CS2 operator critical path is presented, relying on look-ahead logic.

Additional critical path reductions can be carried out considering radix-orders higher or equal than 16, exploiting the concept of *parallel paths* [25]. Observing a radix-16 Trellis-Diagram, composed by 4 consecutive Trellis-sections, if the paths converging to a specific state are highlighted, several couples of parallel paths can be identified. Two or more paths are considered parallel if they are sharing the same starting and destination states.

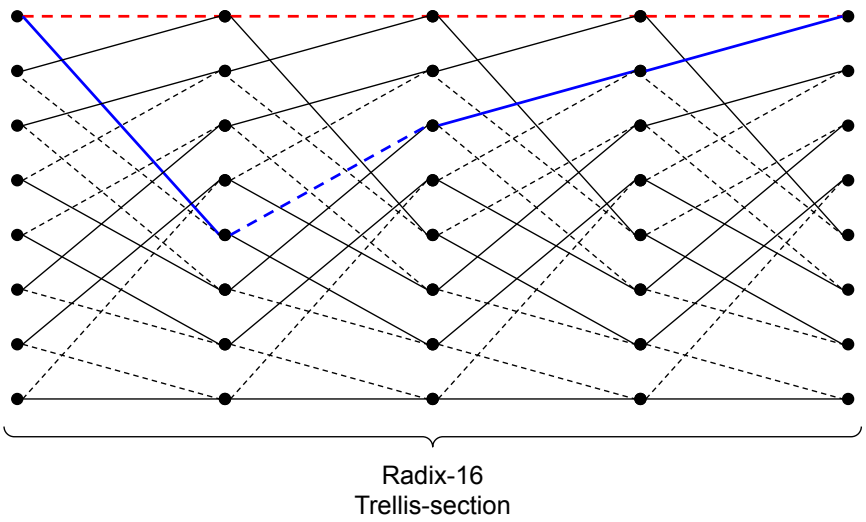

Figure 8.5: Parallel paths example considering a radix-16 Trellis-section.

While performing forward and backward propagations, parallel paths can be discriminated before the PMU, since the starting state metrics are not required to evaluate the maximum. If this operation is performed, the amount of paths to be compared inside the PMU is lowered to 8, reducing the critical path. More specifically, if parallel paths are eliminated, the PMU can be always implemented following a radix-8 implementation, regardless of the employed radix-order.

The parallel paths' discrimination can be located, as a last computational step, in the BMU and it is not representing a boundary for the working clock frequency, since pipeline stages can be added. The additional complexity found in the BMU depends on the amount of parallel paths to be compared, which is increasing with the radix-order.

In this work, the radix-16 BMU will be modified in order to include this feature. As a consequence, the radix-16 PMU is following the same implementation of the radix-8 one. It is remarkable how discarded parallel paths are still required for the soft-output computation, therefore completely neglecting those paths, as described in [25], is affecting the error-correction performances. In this work, this further possibility has not been considered, since it is desired to maintain the error-correction capabilities independent from the employed radix order.

8.3.3 SOU

The amount of APPs to be compared, indicated in table 8.4, is increasing with the radix order. Moreover, multiple message bits are meant to be decoded at the same time. Also in this case, the basic building blocks are adders and Compare-and-Select units.

Radix	$(A + \Gamma + B)$ number	max() operators
2	16	2 max() over 8 inputs
4	32	4 max() over 16 inputs
8	64	6 max() over 32 inputs
16	128	8 max() over 64 inputs

Table 8.4: Max() operators analysis up to a radix-16 implementation.

Since pipelining is available for the SOU, the critical path is not representing an issue. On the other hand, the accent is posed on the minimization of the included amount of CS2 operators. As a matter of fact, the SOU complexity would increase exponentially if the approach presented for the radix-2 case, in figure 4.5, is extended for higher-radix orders.

In [26] a minimum complexity radix-16 SOU architecture is presented, illustrated by the block scheme in figure 8.6. The key point is to group the APPs characterized by the same sequences of systematic bits, expressed by the subscripts in the figure, and compare them. After finding the maximum for each systematic sequence, other comparisons are performed, aiming to isolate specific systematic bits in the sequence. In this work, the same principle has been extended and adapted to radix-4 and radix-8 SOU architectures.

It is remarkable how, in this case, Compare-and-Select operators working with more than two operands are not required to follow a fast implementation, since the SOU can be pipelined. Therefore, the CS2 tree-like structure will be considered as a reference, in order to limit the complexity.

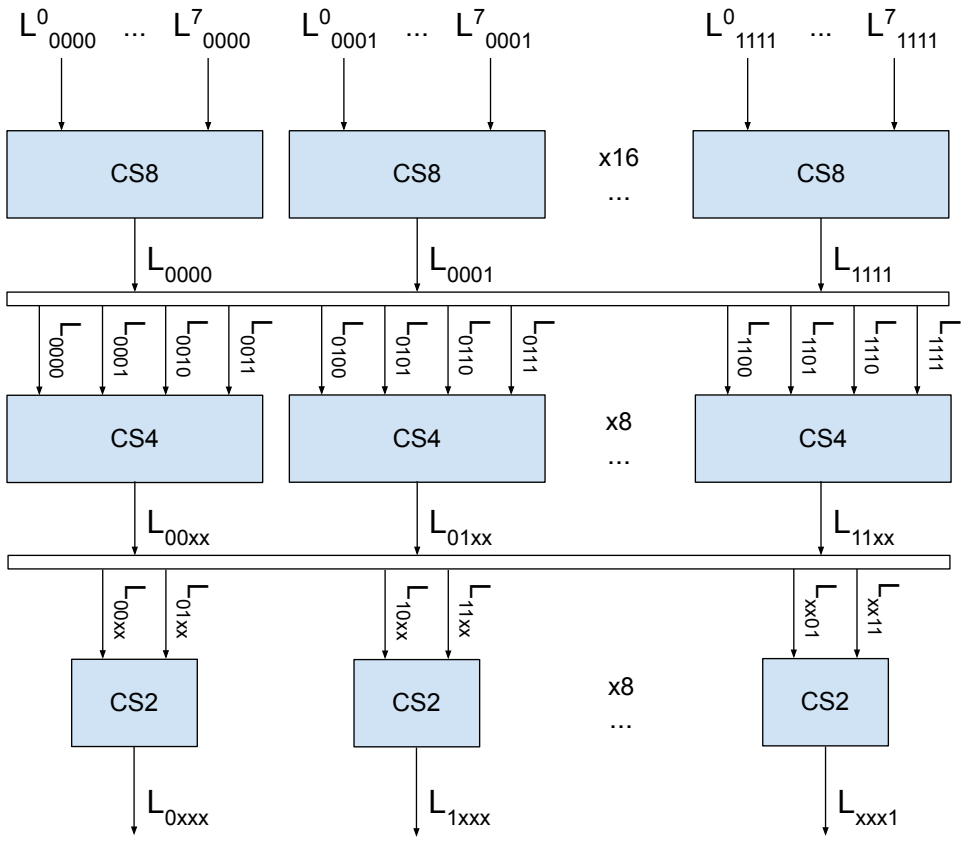

Figure 8.6: Minimum CS2 SOU block scheme, considering a radix-16 approach.

8.3.4 Logic Units Summary

As a summary, the logic components included in the analysis model are listed below, subdivided per each radix-order. Moreover, the contribution of the Extrinsic-Scaling-Factor unit is neglected, since it is not expected to dramatically affect the logic area.

Radix-2

- **BMU**: It is including 2 adders (ADD) to evaluate all the required 16 branch metrics.
- **PMU**: It is composed by 8 comparison units, each containing 2 adders (ADD) and 1 CS2 operator.
- **SOU**: It is including 16 adders (ADD) to fully evaluate the APPs. Moreover, a cascade of 14 CS2 operators is employed. A unique subtractor (SUB) is required to compute the soft-information.

Unit	ADD	CS2	SUB
BMU	2	-	-
PMU	16	8	-
SOU	16	14	1

Table 8.5: Radix-2 required operators.

Radix-4

- **BMU**: It is including 2 BMU2 units to evaluate the radix-2 branch metrics. Then a 9-adders (ADD) network is required to fully compute the composed branch metrics.
- **PMU**: It is composed by 8 comparison units, each containing 4 adders (ADD) and 1 CS4-fast operator.
- **SOU**: It is employing a 32-adders (ADD) layer, required to compute the APPs. Then, 4 CS8 and 4 CS2 units are employed to discriminate the maximum APPs. 2 subtractors (SUB) are included to compute the soft-informations.

Unit	ADD	CS2	CS4-fast	SUB
BMU	13	-	-	-
PMU	32	-	8	-
SOU	32	32	-	2

Table 8.6: Radix-4 required operators.

Radix-8

- **BMU**: It is including 3 BMU2 units to evaluate the radix-2 branch metrics. Then, a 54-adders (ADD) network is required to fully compute the composed branch metrics.
- **PMU**: It is composed by 8 comparison units, whose architecture can be defined following two available options: the first one consists in employing 4 CS2 and 1 CS4-fast, the second solution is employing the usage of a single CS8-fast unit. In both the solutions, 8 adders per comparison unit are required to compute the state-metrics to be compared.
- **SOU**: It is employing a first 64-adders (ADD) layer, required to compute the APPs. Then, 8 CS8 and 6 CS4 are employed to discriminate the maximum APPs. 3 subtractors (SUB) are included to compute the soft-informations.

Unit	ADD	CS2	CS4-fast	SUB
BMU	60	-	-	-
PMU	64	32	8	-
SOU	64	74	-	3

Table 8.7: Radix-8 required operators (CS4-fast usage).

Radix-16

- **BMU**: It is including 4 BMU2 units to evaluate the radix-2 branch metrics. Then, a 149-adders (ADD) network is required to fully compute the composed branch metrics. Since parallel paths are expected to be discriminated before the PMU, 64 CS2 units are included.
- **PMU**: It is composed by 8 comparison units. Their structure is following the same approach described for the radix-8 PMU. Moreover, 64 additional adders (ADD) are required to fully compute the sum between Γ and beta-metrics values to be forwarded to the SOU.
- **SOU**: It is employing a first 128-adders (ADD) layer, required to compute the APPs. Then, 16 CS8, 8 CS4 and 8 CS2 units are employed to discriminate the maximum APPs. 4 subtractors (SUB) are included to compute the soft-informations.

Unit	ADD	CS2	CS4-fast	SUB
BMU	157	64	-	-
PMU	128	32	8	-
SOU	128	144	-	4

Table 8.8: Radix-16 required operators (CS4-fast usage).

8.3.5 GE Model

Given the operators included in a specific solution, it is necessary to estimate the logic area A_L. Simply counting the total amount of operators is not enough, since, as visible from the area efficiency equation, logic and memory areas are required to be expressed on compatible measurement units in order to be added. Moreover, a generalization with respect to the employed technology is also desired.

Following these considerations, the selected unit for the area measurements is the Gate-Equivalent (GE), expressing the equivalent area of a reference gate, which is typically

selected as a NAND-2 gate with minimum strength. By employing the GE, similarities between areas estimated on different technologies are expected.

Since all the considered logic units are sharing a common set of basic operators, it is enough to characterize their areas, expressed in GE. The higher the accuracy in this estimation, the more reliable the total logic area A_L. With this purpose, synthesis on simple operators have been performed in this work, considering three different technologies (45 nm, 65 nm and 90 nm).

Before proceeding, since most of the components are including adders or subtractors, it is necessary to fix a common architectural model for their implementation. Since they are meant to work with a limited amount of bits, an acceptable trade-off between delay and complexity is introduced by employing Ripple-Carry-Adders (RCA) or Ripple-Carry-Subtractors (RCS). Therefore, they have been considered as a reference choice.

	Technology		
	65 nm	90 nm	45 nm
Ripple-Carry-Adder (ADD)			
Area [um^2]	58.68	127.00	32.45
Area [GE]	40.75	45.03	40.66
Ripple-Carry-Subtractor (SUB)			
Area [um^2]	66.96	145.35	36.97
Area [GE]	46.50	51.50	46.30
Compare-and-Select-2 (CS2)			
Area [um^2]	67.32	121.36	38.84
Area [GE]	46.75	43.00	48.67
Compare-and-Select-4 (CS4-fast)			
Area [um^2]	322.20	582.83	194.18
Area [GE]	223.75	206.53	243.33
Compare-and-Select-8 (CS8-fast)			
Area [um^2]	1296.36	2364.47	796.67
Area [GE]	900.25	837.87	998.33

Table 8.9: Area synthesis results on the considered operators, considering a bitwidth equal to 8-bits and three available technologies.

Moreover, since a dependency is found between the area and the bitwidth, a fixed parallelism of 8-bits has been employed for all the operators reported in table 8.9. Then, considering each operator, the area variation has been studied as a function of its bitwidth. In every case, a proportionality between area and bitwidth has been noticed. Therefore, the results reported in the table can be easily adapted to different parallelisms by including a scaling factor.

As expected, expressing area results in terms of GE is moving the estimations toward a generalization against different technologies. In this work, the 65 nm technology has been considered as reference. However, the declared results can be extended to other technological nodes, considering a limited uncertainty. Referencing the 65 nm node, other technologies are presenting maximum percentage variation around 10% on the GE values.

8.4 Memory Analysis

A total of 4 memory classes are required to be included in the explored architectures.

- **Input-Frame-Memory (IN-FR-MEM)**: This memory is meant to store the channel LLR values, including the systematic information L_s and the two parity informations L_{p1} and L_{p2}. Moreover, it is required to be simultaneously accessed by all the SISO instances.
- **Extrinsic-Information-Memory (EXTR-INF-MEM)**: This memory is storing the extrinsic information values L_e. Also in this case, the access is guaranteed to all the SISO-modules in the architecture.
- **Alpha-Memory (ALPHA-MEM)**: This memory is necessary in order to correctly implement the Forward-Backward scheduling. Each SISO-decoder is including a specific Alpha-Memory, which is storing the alpha-metrics values $A(S_k)$.
- **NII-Memory (NII-MEM)**: This memory is required to correctly introduce the beta metrics initialization. Also in this case, each SISO-module will include a specific NII-Memory, which is meant to store the initial beta metrics $B(S_k)$ for each window.

As a first step, storage requirements for each considered memory will be covered, understanding the effect of high-radix orders. Then, the *memory conflicts* issue will be introduced, discussing possible solutions. Moreover, informations on the implemented memory access policies will be provided. Once the memory organization has been defined, a GE area model will be derived.

8.4.1 Storage Requirements

Considering the frame-size K, the sub-frame size Kp and the window size WS, the storage requirements, indicated in table 8.10, are derived, considering a radix-2 architecture.

Memory class	**N. instances**	**Words**	**Bits per word**
IN-FR-MEM	1	K	$3 \cdot w$
EXTR-INF-MEM	1	K	w_{EXT}
ALPHA-MEM	K/Kp	WS	$8 \cdot w_{SM}$
NII-MEM	K/Kp	Kp/WS	$8 \cdot w_{SM}$

Table 8.10: Memory storage requirements considering a radix-2 architecture.

Radix-orders higher than 2 are potentially affecting the state-metrics bitwidth w_{SM} and the total amount of alpha-metrics to be computed. For instance, considering a radix-4 architecture, the required amount of alpha metrics to be saved is $WS/2$, since each algorithmic step is involving two adjacent Trellis-sections. As a consequence, the alpha memory is reduced in size, since employing a radix-N approach is requiring a total of $WS/log_2(N)$ state-metrics. Observing table 8.10, no other storage contributions are affected by the employed radix-order.

8.4.2 Memory Conflicts

As described while introducing memory classes, stored data are required to be accessible at the same time by multiple logic blocks. Indeed, Global memories (IN-FR-MEM/EXTR-INF-MEM) are shared among different SISO-decoders in parallel. Moreover, further multiple accesses are generated by the Forward-Backward scheduling, since Forward and Backward logic units are simultaneously accessing Global and Local memories.

Memory conflicts are generated when multiple data are required to be accessed in parallel, but the memory architecture is not capable to handle the request. Two main solutions are available: the first one consists in using *multi-port* memories, able to handle multiple reading/writing operations on different addresses. The second option consists in *partitioning* the original *single-port* memory in smaller memories, with a number of partitions corresponding to the amount of requested parallel accesses. Since multi-port memories have a significant impact on the area, especially if compared to single-port ones, partitioning has been considered the reference technique to avoid conflicts.

By increasing the radix-order, the amount of required parallel accesses from Global memories is incremented, since multiple Trellis-sections are processed at the same time. Therefore, a negative impact is found on the total number of required partitions. The estimated amount of parallel accesses is reported in table 8.11, considering different radix-orders.

Memory Class	**Parallel Accesses**			
	Radix-2	Radix-4	Radix-8	Radix-16
IN-FR-MEM	$2 \cdot K/Kp$	$4 \cdot K/Kp$	$6 \cdot K/Kp$	$8 \cdot K/Kp$
EXTR-INF-MEM	$3 \cdot K/Kp$	$6 \cdot K/Kp$	$9 \cdot K/Kp$	$12 \cdot K/Kp$
ALPHA-MEM	2	2	2	2
NII-MEM	2	2	2	2

Table 8.11: Required parallel memory accesses on different radix orders.

Figure 8.7 is visually representing how multiple accesses are generated during a generic time instant. As visible, considering a single SISO-decoder processing a sub-frame, the FB scheduling is generating two parallel accesses on the Input-Frame-Memory. On the other hand, three parallel accesses are generated for the Extrinsic-Information-Memory, since a further access is required to write the extrinsic-information.

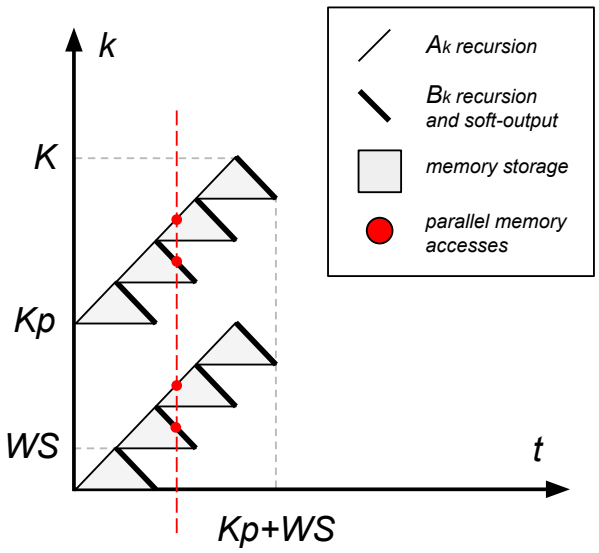

Figure 8.7: Parallel accesses visually represented in a generic time instant.

8.4.3 Access Policies

After properly partitioning the memories, it is necessary to understand how the partitions or *memory banks* can be correctly accessed, in order to perform the decoding algorithm. Also in this case, the accesses should be managed in order to avoid potential conflicts on single memory banks.

Input-Frame-Memory

A fundamental aspect to be discussed is the access to the systematic information L_s, which is requested during both natural and interleaved half-iterations. As a consequence, conflicts should be avoided considering both the access orders. Since it is desired to avoid increasing the access policy complexity, it is possible to assume that the systematic information is requested from the Input-Frame-Memory only during the natural processing, as represented in figure 8.8.

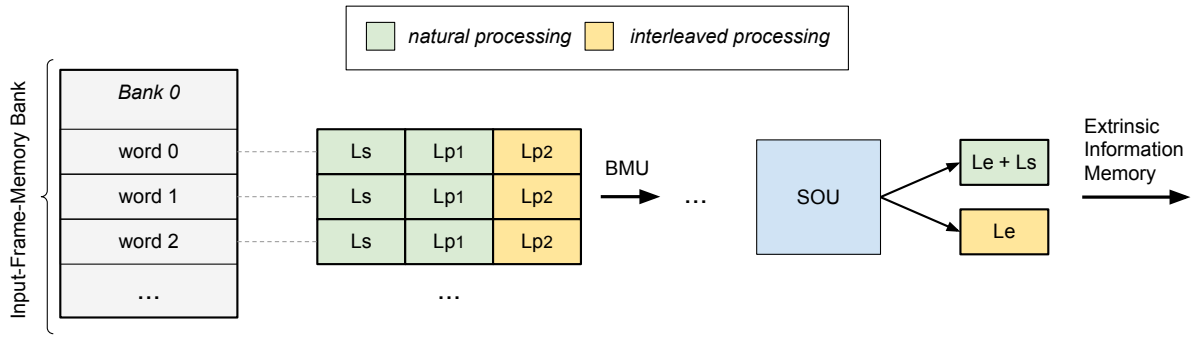

Figure 8.8: IN-FR-MEM and EXTR-INF-MEM organization in order to manage the access to the systematic information.

As visible, the systematic and parity informations are assumed to be saved following the arrival order inside the Input-Frame-Memory. During the natural half-iterations, informations on L_s and L_{p1} are extracted from the memory. Once the SOU has computed the extrinsic information L_e, the systematic one L_s is re-added, before saving the result in the Extrinsic-Information-Memory. As a consequence, during the interleaved processing, data extracted from the Extrinsic-Information-Memory are automatically including the sum between the a-priori information L_a and the systematic one, avoiding the interleaved access for the Input-Frame-Memory. Therefore, during the interleaved half-iterations, only the second parity information is required from the frame data.

The described policy has a small impact on the decoder complexity, since the additional logic is expected to be limited.

Extrinsic-Information-Memory

Also in this case, it is mandatory to guarantee conflict-free accesses during both natural and interleaved processing. Two approaches are available.

- The first solution consists in employing specific permutation laws capable to guarantee conflict-free accesses. As an advantage, the problem is solved with the minimum amount of hardware resources and a simple memory mapping. However, this approach could limit the choice for the permutation law, since both interleaving and de-interleaving functions are required to satisfy the *contention-free* property [27], considering a radix-2 architecture. It has been proved that a small fraction of interleavers are meeting this requirement [27]. It is mentionable how the ARP and QPP based interleavers are contention-free. If higher radix orders are employed, additional restrictions are required on the permutation law, since the amount of memory banks is expected to increase.
- The second solution consists in finding a conflict-free memory mapping and a specific access policy given a certain permutation law. Therefore, in opposition with the first proposed solution, the hardware is adapted to the selected interleaving law. This approach, presented in [28], has been proved to be valid for any employed permutation law. Therefore, it is ensuring maximum flexibility on the interleaver choice. On the negative side, hardware complexity is expected to be higher if compared to the first solution, due to the introduction of additional memories and logic elements.

This work is considering the second approach as a reference choice for all the explored architectures, in order to guarantee the maximum flexibility against new possible standards for permutation laws. The presence of flexible interleaving hardware is also an advantage if multiple frame-sizes are desired to be processed by the Turbo-decoder.

As described in [28], the memory access organization is based on a *mapping-matrix*. Following a radix-2 approach, each row is representing a processed sub-frame in natural order, containing Kp elements. Therefore, a single column is containing a set of $N = K/Kp$ parallel accessed data from different sub-frames.

Figure 8.9 is including a mapping-matrix example, considering 5 sub-frames and 5 elements per sub-frame. Since $N = 5$, a total of 5 memory partitions are needed. Therefore, the cells in the matrix are indicating in which memory banks the required data are stored. During the natural processing, parallel accessed data are represented by the matrix columns. As noticeable from the example, the conflict-free property is guaranteed, since each partition is found at most once per column.

Since conflicts should be avoided also during the interleaved processing, parallel accesses can be highlighted by employing a *tiles-matrix*, which is indicating with letters the simultaneously accessed data after applying the permutation on the original sequence. By

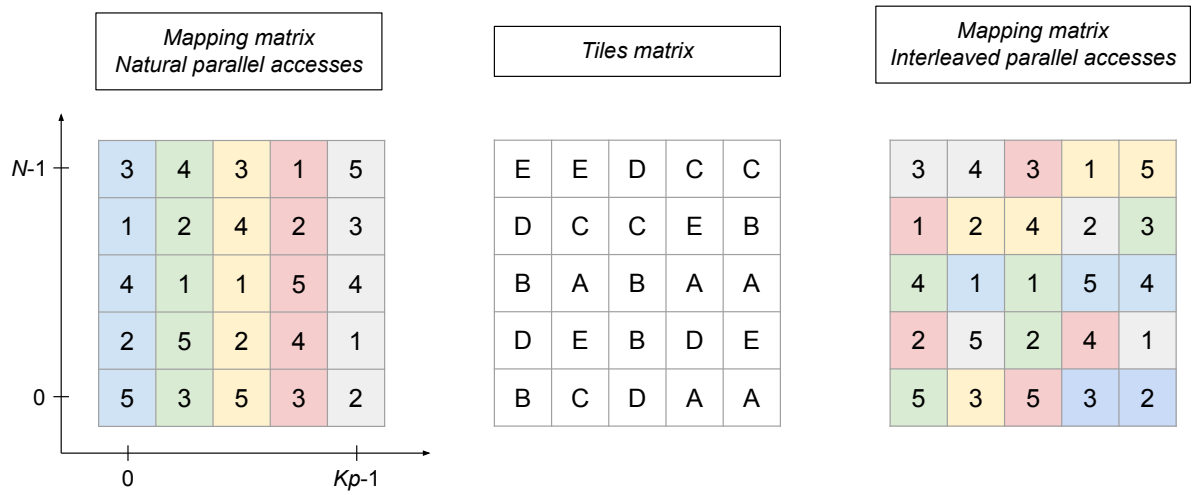

Figure 8.9: Mapping-matrix example, highlighting with different colors the parallel accesses during both natural and interleaved processing.

highlighting the tiles on the original mapping-matrix, it is possible to observe that the conflict-free property is still satisfied.

If the expressed property is verified on both columns and tiles, the mapping-matrix can be considered valid and, therefore, it can be used to map the information in the Extrinsic-Information-Memory. In [28], the steps required to derive a mapping-matrix, given a generic permutation law, are detailed.

As mentioned, additional hardware is required in order to apply the spatial permutations in time, following the content of the mapping-matrix. With this purpose, a *crossbar* is introduced, allowing to fully customize the data permutations. Moreover, specific memories are required to store the permutation data employed to control the crossbar and access the memory-banks in specific locations. These additional hardware requirements, depicted in figure 8.10, are representing the main drawback of this approach, since they are expected to have a non-negligible impact on the Turbo-decoder complexity.

The approach presented in [28] is assuming to work with a radix-2 architecture, without considering a specific scheduling policy. Therefore, a possible extension of the proposed technique has been studied in this work, with the aim of adapting it to the considered set of architectures under analysis. More details are included in Appendix C.

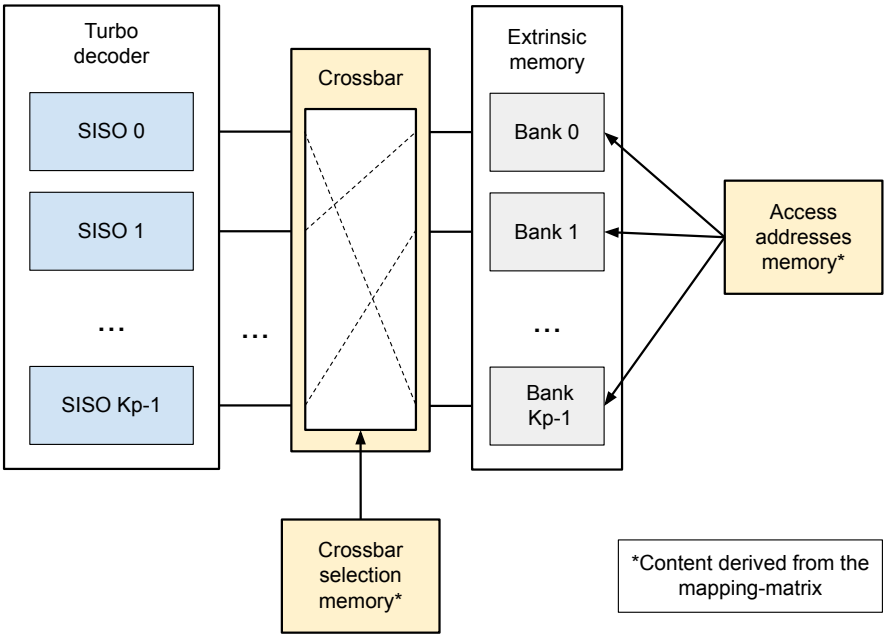

Figure 8.10: Block scheme representing the EXTR-INF-MEM accesses controlled by crossbar and memories, which content is derived from the mapping-matrix.

Alpha-Memory

As indicated in table 8.11, each Alpha-Memory is expected to be split in two partitions, aiming to ensure the concurrent access for the forward/backward units. The access policy should be organized in order to avoid conflicts. Moreover, it is desired to prevent the introduction of unnecessary memory locations.

In order to satisfy this last requirement, it is necessary to replace old alpha metrics, employed by the SOU, with new ones computed by the forward PMU, without wasting memory space. Therefore, an alternated access to the two memory partitions is proposed, as illustrated in figure 8.11.

In the proposed example, the forward state metrics are saved by interleaving the access to the two alpha memory partitions. Then, the soft-output computation is performed in backward order. Once a branch metric data has been read by the SOU, it is replaced by new state metrics, belonging to a new window.

As visible, following this policy, the two memory banks are accessed without conflicts. It is noticeable how the forward accesses are performed in the same locations accessed by the SOU during the previous clock cycle, opening the possibility to re-use the generated addresses.

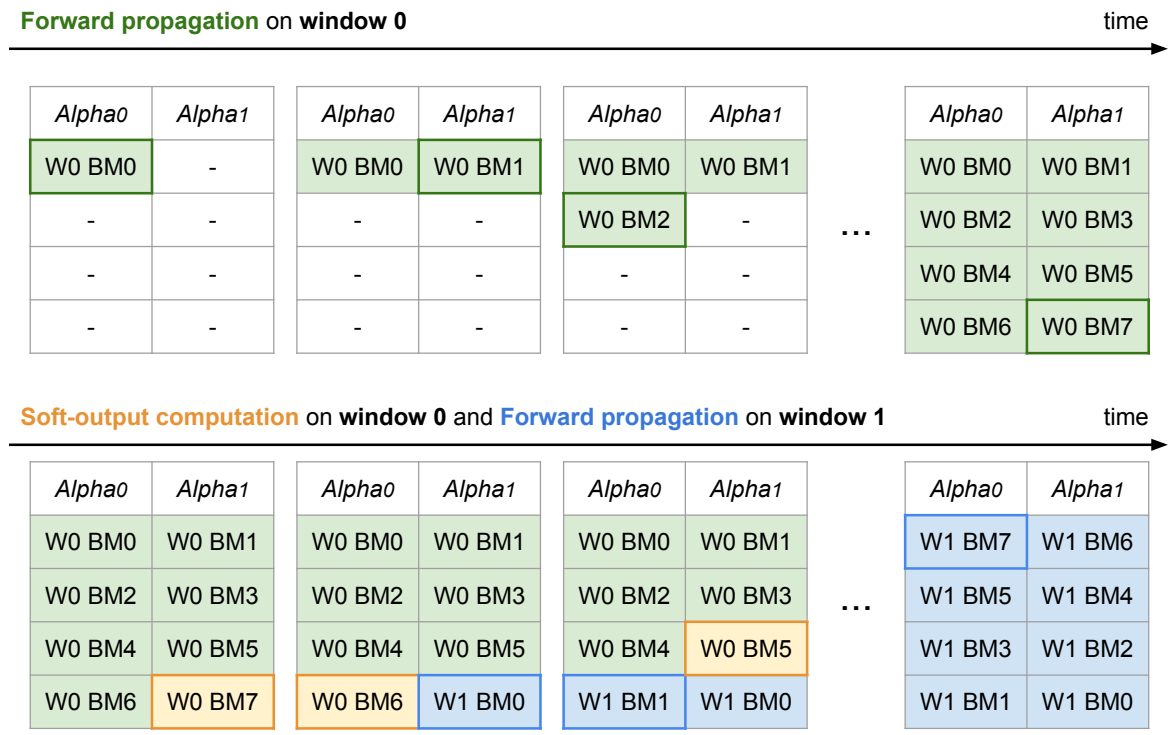

Figure 8.11: Alpha memory access policy example, considering two windows with 8 Trellis-sections each.

NII-Memory

The initialization memory is also presenting conflict issues, especially when the end section of a given window is reached. As a matter of fact, it is required to simultaneously load new initialization metrics and store the final metrics from a given window. The following solution is proposed.

Since this superimposition is found only once per analyzed window, it is possible to organize the two required operations in distinct clock cycles. When the last beta values from a window are computed, they are stored in a set of temporary registers. Meanwhile, the NII-Memory is accessed in order to load the initialization metrics. During the next clock cycle, the temporary stored beta metrics can be saved in the memory, without generating any conflict. The block scheme in figure 8.12 is summarizing this access policy.

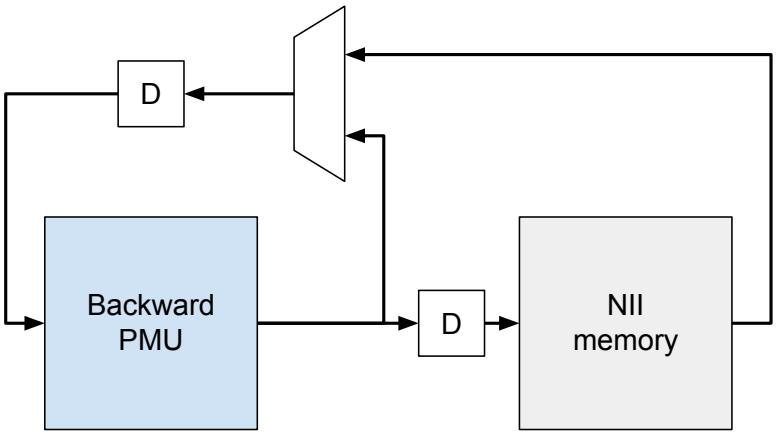

Figure 8.12: NII-MEM access policy block scheme.

8.4.4 GE Model

After detailing the memories structure and organization, it is necessary to estimate the memory area impact, expressed in GE. The first analyzed contribution is the presence of the crossbar, a logic component necessary to correctly manage the access to the extrinsic information. Following the same approach introduced during the logic analysis, synthesis operations on different crossbars have been performed, in order to estimate its area as function of the data parallelism and the number of memory partitions. Moreover, different technologies have been compared. The reference crossbar, reported in table 8.12, is considering $w_{ext} = 7$ and a total of 16 partitions.

	Technology		
	65 nm	90 nm	45 nm
Crossbar			
Area [um^2]	4199.0	7360.8	2162.0
Area [GE]	2915.6	2608.4	2709.3

Table 8.12: Crossbar area synthesis results on different technologies.

Also in this case, referencing the 65 nm technology, the relative GE area variation is below 10%, if compared to the other technologies. Moreover, a proportionality is observed between the area and the extrinsic data bitwidth w_{ext}. Concerning the amount of partitions, the GE area is presenting a quadratic dependence from it. For instance, doubling the amount of partitions is increasing the crossbar area by a factor 4.

Concerning the different memory instances, it is useful to establish a GE model which is able to estimate the total area as function of the memory size, expressed in number of

bits. In this work, all the considered memories are *single-port SRAM*, implemented on the 65 nm technology.

Given a generic memory model, the GE/bit parameter can be computed as follows.

$$GE/bit = \frac{Area(GE)}{words \cdot bits}$$

Analyzing some library memory models, it is interesting to notice how area is affected by the presence of peripheral circuits, such as column decoders or row multiplexers. Following this consideration, a possible expression for the memory area is reported.

$$Area(GE) = A_{cell} \cdot (words \cdot bits) + overhead(words, bits)$$

The first part of the equation is expressing the *cells area*, which is obtained by multiplying the memory size in bits by the static cell area A_{cell}. As noticeable, the overhead added by the peripheral circuits is function of the memory size. More specifically, considering the analyzed technology, a significant sensitivity has been observed as function of the required number of words. Indeed, the higher the amount of words, the lower the relative overhead contribution affecting the total area. Some reference examples, extracted from available memory models, are reported in table 8.13. The parameter *cell efficiency* is representing the ratio between the static cells area and the total one.

		Technology 65 nm		
Words	**Bits**	**Area [um^2]**	**Cell Efficiency**	**GE/bit**
64	88	18734.3	16%	2.31
256	20	9584.6	28%	1.30
2048	25	47923.2	55%	0.65
4096	16	53791.9	63%	0.57

Table 8.13: GE/bit data on different library memory models, considering the 65 nm technology.

As expected, memories including a larger amount of words are more efficient in terms of area occupation. Consequently, the GE/bit parameter is presenting important variations, which are required to be modeled in order to introduce an accurate memory area estimation.

In this work, the following equation has been employed to approximate the GE/bit variation against the involved amount of words.

$$GE/bit = GE_{cell} + \frac{1}{k_1 \cdot ln(k_2 + |k_3 \cdot words|)}$$

The parameter GE_{cell} is representing the static cell area expressed in GE. As visible, the key-point of the model is to move the GE/bit parameter toward the ideal GE_{cell} value by increasing the amount of words and, therefore, the cell efficiency. It can be noticed how the dependency from the *words* parameter is not linear. Moreover, it is necessary to fit the curve considering a set of given true points, by tuning the k constants. Considering the memory data presented in table 8.13, the fit depicted in figure 8.13 has been obtained.

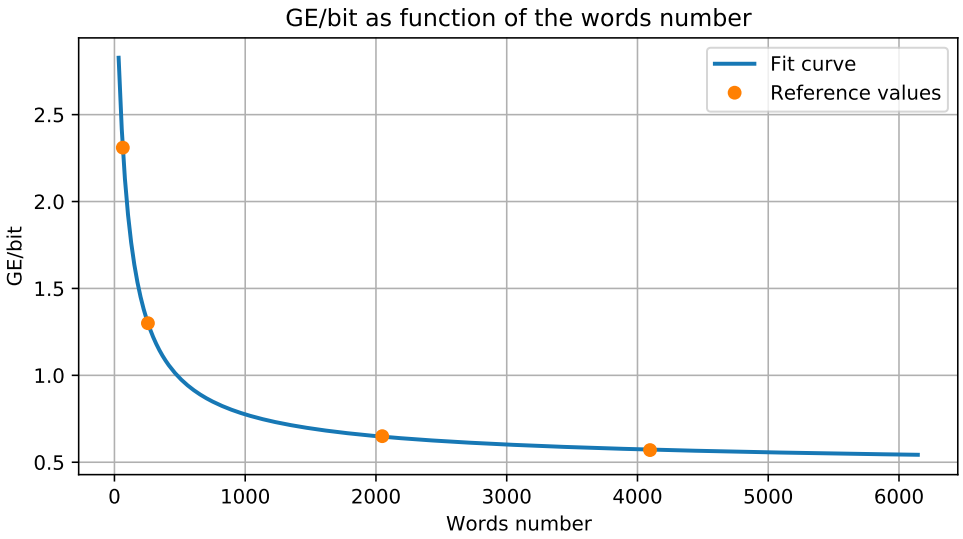

Figure 8.13: GE/bit variation model for library memories, as function of the number of words.

As visible, the curve is well fitting the given reference points. However, the amount of available memory data is limited, therefore it is necessary to consider a degree of uncertainty on the obtained results.

It is interesting to notice how, considering words numbers above 1024, the curve can be well approximated with a linear model, which is expected to be simpler and faster to be employed. Moreover, it could be interesting to compare memory models obtained on different technologies, to improve the results' generalization.

It is now possible to explicit a further negative effect deriving from memory partitioning, since smaller memory partitions are expected to have a larger impact on the total area.

Since in this work it is often required to implement memories with limited sizes, it is fundamental to consider the presence of *synthesized memories*, which are replacing library memories when the required size is below a specified boundary. The switch point between the two memory models can be extracted from the library memory documentation, since a minimum required size is typically declared. In this work, considering a 65 nm static memory library, the boundary is set to 32 minimum words. As a consequence, below the

expressed value, memories must be synthesized.

Synthesizing memories is introducing a further drawback in terms of complexity, since Flip-Flops are employed to store bit informations instead of static memory cells. Also in this case, synthesis operations have been performed in order to study the GE/bit variation as function of the memory size. The results are summarized in table 8.14.

		Technology 65 nm	
Words	**Bits**	**Area [um²]**	**GE/bit**
4	32	1929.8	10.47
8	32	3420.0	9.28
16	32	6709.2	9.10
32	32	13315.3	9.03
16	8	1767.6	9.59
16	16	3439.4	9.33
16	32	6805.1	9.23
16	64	13507.0	9.16

Table 8.14: GE/bit variation on synthesized memories, considering the 65 nm technology.

As noticeable, the GE/bit parameter variations are not significant, while exploring different sizes. If considering an average value of 9.5 GE/bit, the maximum relative excursions are around 10%. The main cause for the limited variation is found in the large GE_{cell} value, considering a Flip-Flop. The introduced error can be considered acceptable, since also the library memory model is affected by a degree of uncertainty.

The two models for library and synthesized memories can be combined, as illustrated in figure 8.14. It can be noticed how the switching point between the two models is found at 32 words. As visible, it is inconvenient to employ synthesized memories: considering the switching point, the GE/bit value for a synthesized memory is approximately three times the library one. However, in some cases, the use of small size memories can't be avoided, forcing the synthesized implementation.

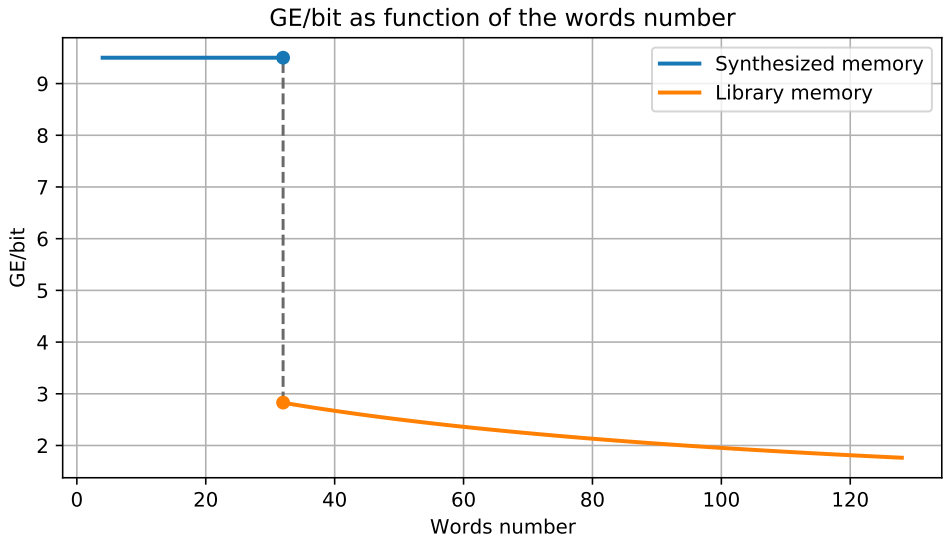

Figure 8.14: GE/bit complete model, considering both library and synthesized memories.

8.5 Throughput

The last parameter to be modeled is the expected throughput from the Turbo-decoder. The general throughput equation for a PMAP-based architecture is reported below.

$$Th = \frac{K \cdot f \cdot log_2(radix)}{(Kp + WS) \cdot n_{HI}}$$

The parameters K, Kp and WS are expected to change when exploring the solutions space. Moreover, also the radix-order is considered a degree of freedom. The Half-Iteration number has been fixed to 12 for all the architectures, aiming to obtaining independent comparisons.

The working clock frequency strongly depends on the critical path, assumed to be found in the PMU, since it is not pipelinable. With the aim of introducing a fair comparison, a critical path increment estimation has been carried out in this work, including different radix orders. Considering the logic employed in the PMU, it is possible to identify a cascade of fundamental operators defining the critical path. Since those operators have already been synthesized during the logic area characterization, it is enough to interconnect them and perform again synthesis operations. Delays are expected to be function of the employed technology, for this reason the *critical path increment factor*, referred to a radix-2 architecture, is estimated.

The sequences of logic blocks found on the critical paths are represented in figure 8.15. As noticeable, two possibilities are available for the radix-8 and radix-16 architectures.

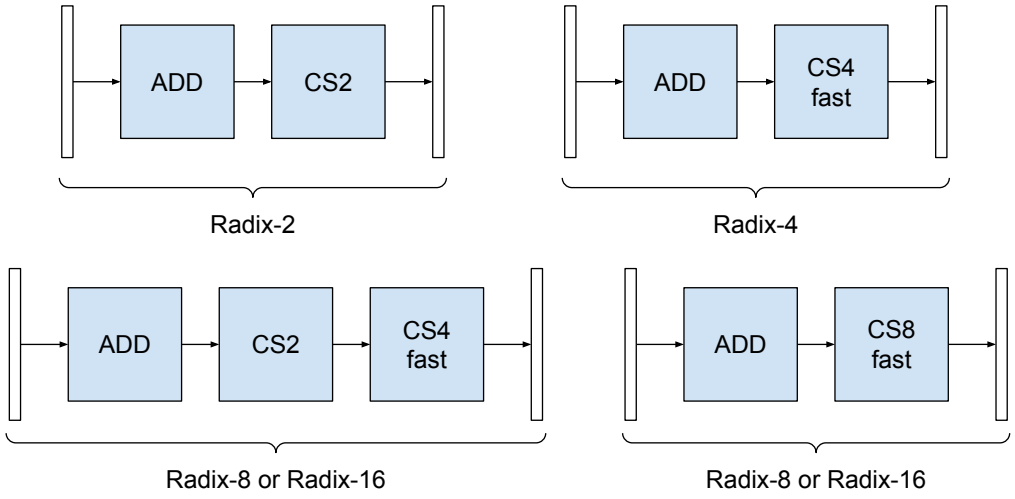

Figure 8.15: Logic units located on the critical path considering different radix-orders.

In table 8.15, critical path synthesis results are summarized, considering a bitwidth equal to 11 bits. As mentioned, the radix-2 critical path has been considered as reference. It can be highlighted how comparing different solutions by employing the CP increment factor is producing similar results on different technologies, improving the model generalization. Indeed, considering the 65 nm technology, relative variations below 10% are found against the other reported technologies.

	Technology					
	65 nm	90 nm	45 nm	65 nm	90 nm	45 nm
Radix	**Critical Path [ns]**			**Radix-2 CP Ratio**		
2	1.46	0.91	1.37	1	1	1
4	1.99	1.26	1.80	1.36	1.38	1.31
8/16 (CS4-fast)	3.37	2.13	3.09	2.31	2.34	2.25
8/16 (CS8-fast)	2.64	1.69	2.34	1.81	1.86	1.71

Table 8.15: Critical path synthesis data on different technologies, considering different radix-orders and data expressed on 11 bits.

The radix-4 architecture is introducing a relative increment around 35%, while the radix-8 solution based on CS4-fast units is more than doubling the original critical path, with a relative increment around 130%. The last proposed architecture, employing CS8-fast units, is limiting the degradation on the critical path, lowering the relative increment to values around 80%.

In the throughput model included in the analysis tool, the working clock frequency is computed starting from a reference frequency found for a radix-2 architecture. Then, a decrement factor is applied, depending on the implemented radix-order. Since the 65 nm technology is the reference one, the radix-2 maximum clock frequency is $f_{max} = 1/(1.46$ ns$) \approx 685$ MHz. Therefore, the reference working frequency has been selected equal to 600 MHz. Table 8.16 is summarizing the choices for the working clock frequencies.

Radix	**Decrement Factor**	**Clock Frequency [MHz]**
2	1	600
4	1.36	441
8/16 (CS4-fast)	2.31	259
8/16 (CS8-fast)	1.81	331

Table 8.16: Working clock frequencies included in the developed model considering different radix-orders, with a bitwidth equal to 11 bits.

If different bitwidths are employed, the declared frequencies are required to be properly scaled. In this work, a proportionality between critical paths and bitwidths has been considered, since ripple-carry logic is employed. However, a limited degree of uncertainty is introduced in this estimation, especially when LUTs are included.

8.6 Uncertainty

As mentioned during logic, memory and throughput analysis, different error sources could affect the reliability of the results declared by the model. A possible way to observe their effect on the final computed indicators by employing the *deterministic uncertainty propagation*.

The deterministic approach is assuming to work with a generic quantity y expressed through a mathematical model $y = f(x_1, x_2, ...x_n)$. The variables included in the model are affected by *absolute uncertainties* δx_i. Assuming that $\delta xi \ll xi$, the uncertainty δy can be estimated as follows.

$$\delta y = \sum_{i=1}^{n} \left| \frac{dy}{dx_i} \right|_{x_i} \cdot \delta x_i$$

The introduced assumption is necessary due to the linearization applied by the partial derivatives. For this reason, it is fundamental to consider the propagated uncertainty results as valid only if limited errors are affecting the parameters. In this work, errors are

found on throughput Th, logic area A_L and memory area A_M. The error propagation has been applied to the area efficiency equation, following the approach introduced above.

The deterministic approach is capable to estimate errors on single computed quantities. However, the developed analysis tool aims to compare different solutions, employing the computed indicators. Therefore, it is fundamental to understand how uncertainties are affecting comparisons between architectures or, in other words, if the choice of an efficient solution is robust against errors.

As an example, given a set of specifications, it can be assumed that the analysis model declared a radix-4 implementation as the most efficient choice. In order to prove the reliability of this solution, it is necessary to check if the presence of errors is potentially able to change the model result, involving for example a different radix-order.

In order to perform a *reliability analysis* over comparisons based on the area efficiency, a brute-force approach is proposed in this work, performed in three steps.

1. Starting from the declared errors on throughput, logic area and memory area, three arrays of values are derived. Each array is including the variation of the analyzed quantity around its nominal value. For instance, considering a logic area equal to 100 GE and a 20% relative error, the array is ranging between 80 GE and 120 GE.
2. Considering the three obtained arrays, all the possible set of combinations are derived. This step can be algorithmically implemented by employing three nested *for loops*. The model is tested on each single set and the provided results are stored.
3. Observing the results obtained from the previous step, it is possible to state if the model converged to a specific solution (reliable result) or if the best solutions are spread over different architectures.

It is remarkable how the computational cost of this approach is exponentially increasing as function of the amount of parameters affected by errors, since all the possible combinations are required to be tested. Moreover, also the amount of elements per array is dramatically affecting the performance of this analysis.

8.7 Full Architecture

A generic block scheme for a radix-2 SISO architecture is illustrated in figure 8.16. It highlights the interconnections between logic blocks and memories. When considering higher radix-orders, the SISO architecture is modified in terms of logic and memory, following the models described in the previous sections.

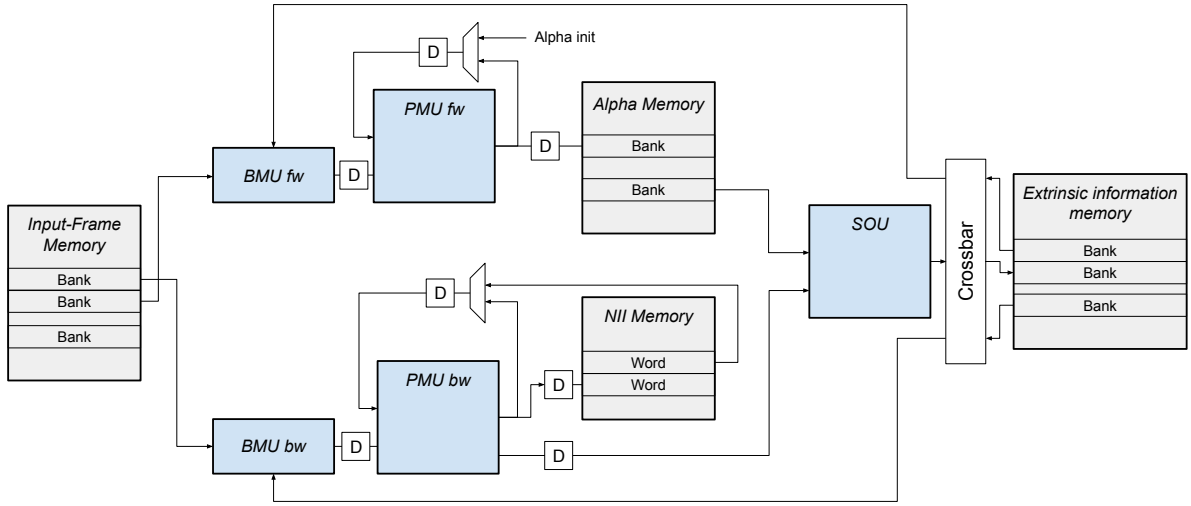

Figure 8.16: Radix-2 SISO block scheme, showing connections between logic units and memories.

As a final aspect, Decoder-level parallelization is considered. As highlighted in Chapter 5, Decoder concurrency is a fundamental key-point to satisfy very high-throughput requirements. Moreover, as a further advantage, some architectural elements can be shared among different Turbo-decoder instances, since they are simultaneously performing the same operations. If the amount of decoders is high enough, the area impact of the shared elements can be neglected. Following this consideration, in this work, the control logic and the memories required to store interleaving data are neglected from the total area contribution, since they are meant to be shared among concurrent decoders.

Furthermore, Decoder-level concurrency is opening the possibility to avoid the introduction of multiple global memories for each Turbo-decoder. Indeed, the same memories can be shared, by including in a single word all the data required to be dispatched to the several Turbo-decoder instances. A generic block scheme of a Concurrent-PMAP architecture is shown in figure 8.17.

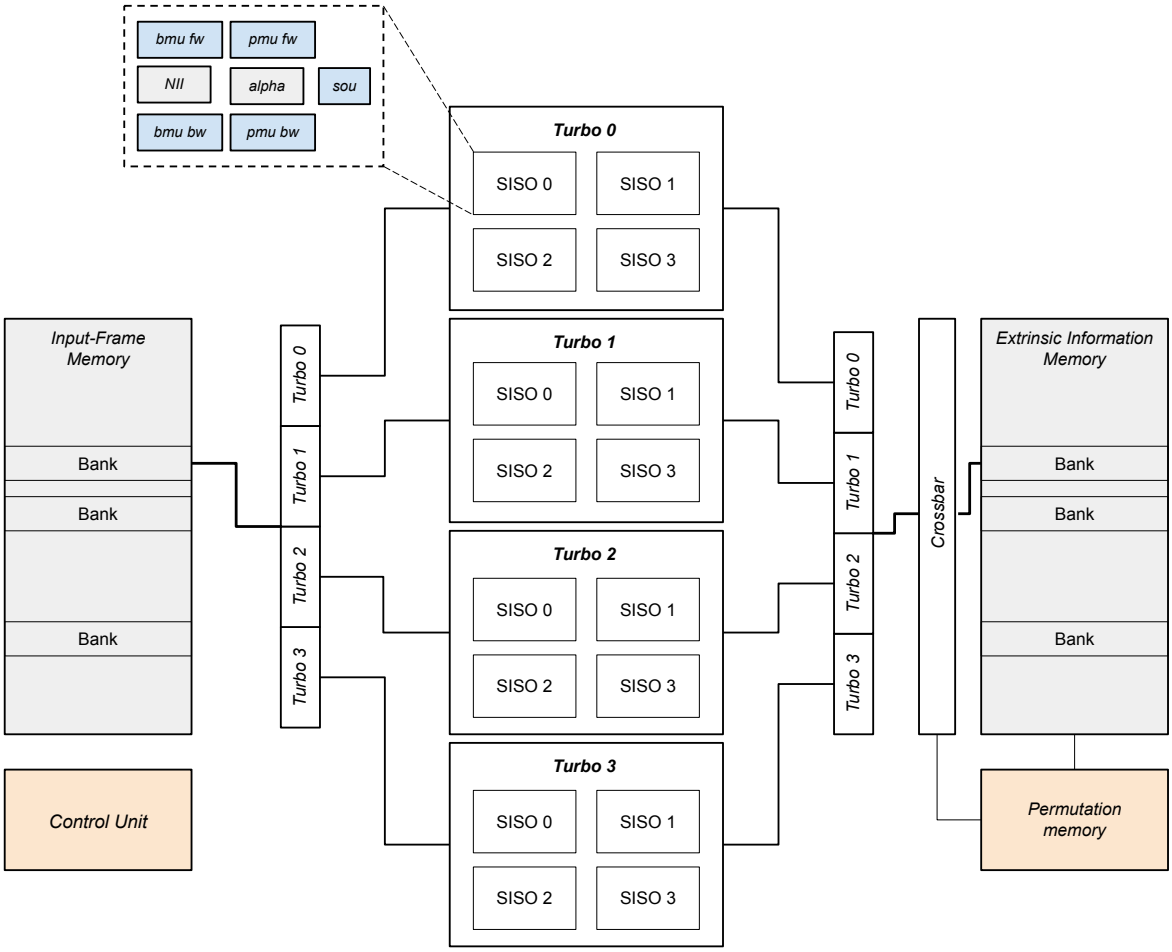

Figure 8.17: Concurrent-PMAP block scheme, with shared elements and including unique global memories.

Chapter 9

Model Results

This chapter is first introducing some specifications on the developed analysis tool, highlighting its main features. Then, a solutions space exploration will be presented, analyzing how indicators are effected by several degrees of freedom. Moreover, some guidelines to select area efficient architectures will be introduced, including the radix-order choice. Proceeding in the chapter, two specific solutions will be analyzed and compared in details. Then, limitations on high radix-order architectures will be discussed, highlighting the necessary improvements. The last part of the chapter is fully dedicated to explore very high-throughput solutions, considering concurrent Decoder-level parallelization applied on a PMAP architecture.

9.1 Tool Organization

The developed analysis tool is based on a group of functions able to compute the necessary parameters and interact in order to formulate the required results. The obtained data have the possibility to be exported as text files or graphically visualized through plots. Functions are grouped in sub-sets in order to distinguish the field in which they operate, as represented in figure 9.1. From the *model parameters* group, it is possible to customize the input data employed by the tool, as well as control the type of analysis to be performed and set options on how the results will be exported.

The *tasks* group is including all the possible types of analysis that can be launched. The tool has two main options: *single analysis* and *multiple analysis*. The former is capable, given a unique set of input parameters, to explicit the most-efficient radix order and provide details about the implementation. If the *reliability* option is selected, the provided solution is tested against errors superimposed to throughput, logic area and memory area. On the other hand, multiple analysis are meant to be employed for exploration purposes, in order to study throughput and area efficiency variation within a set of architectures,

highlighting the guidelines to select efficient implementations.

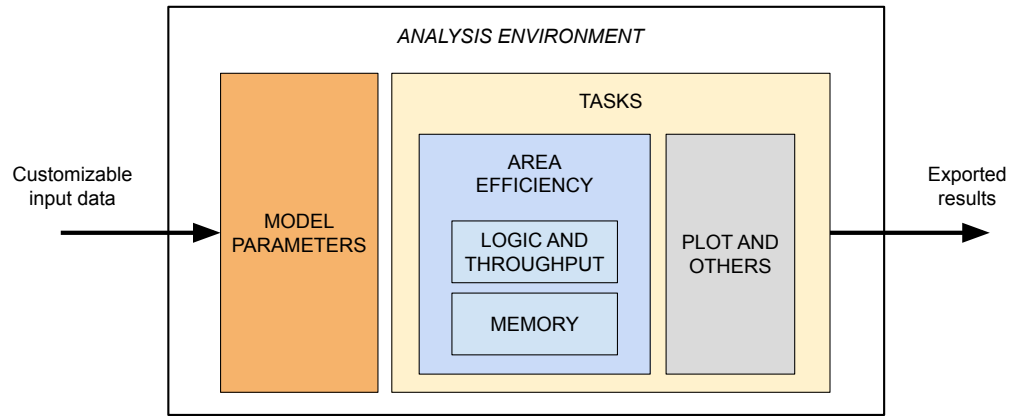

Figure 9.1: Analysis tool block scheme, representing the main function groups included.

Most of the data collected in the previous chapter are employed by the *area efficiency* group, which is including a further separation between logic and memory related functions.

More details on additional features will be given in the next sections, while analyzing part of the exported results.

9.2 Solutions Space Exploration

As a first step in this analysis, it is interesting to analyze throughput and area efficiency variations against some important specifications for the PMAP architecture. Therefore, several *multiple analysis* have been launched, focusing around the parameters listed in table 9.1. In the analyzed architectures, the CS4-fast based radix-8 PMU has been considered, depicted in figure 8.4.

Parameter	**Symbol**	**Default Value**	**Variation Range**
Frame-Size	K	6144	[1024, 8192]
Sub-Frame-Size	Kp	256	[128, 2048]
Window-Size	WS	32	[16, 128]
Channel LLR bitwidth	w	6	[4, 16]

Table 9.1: Parameters default values and variation ranges for the proposed analysis.

While a parameter is moved in the proposed range, the other ones are fixed on their default values. In this way, it is possible to study the effect of each single quantity.

The first analyzed parameter is the Frame-Size. The analysis results are illustrated in figure 9.2. As visible from the top chart, the area efficiency is presenting a monotonic decreasing curve. The reason behind this trend is the presence of the crossbar in the architecture, since its area is quadratically depending on the amount of SISO-decoders $N = K/Kp$. If the crossbar is neglected, throughput, logic area and memory area would present a proportionality to the Frame-Size, maintaining the area efficiency constant. Observing just the throughput curve, the latter is presenting a continuous increment, since the amount of implemented SISO modules is increasing with the Frame-Size.

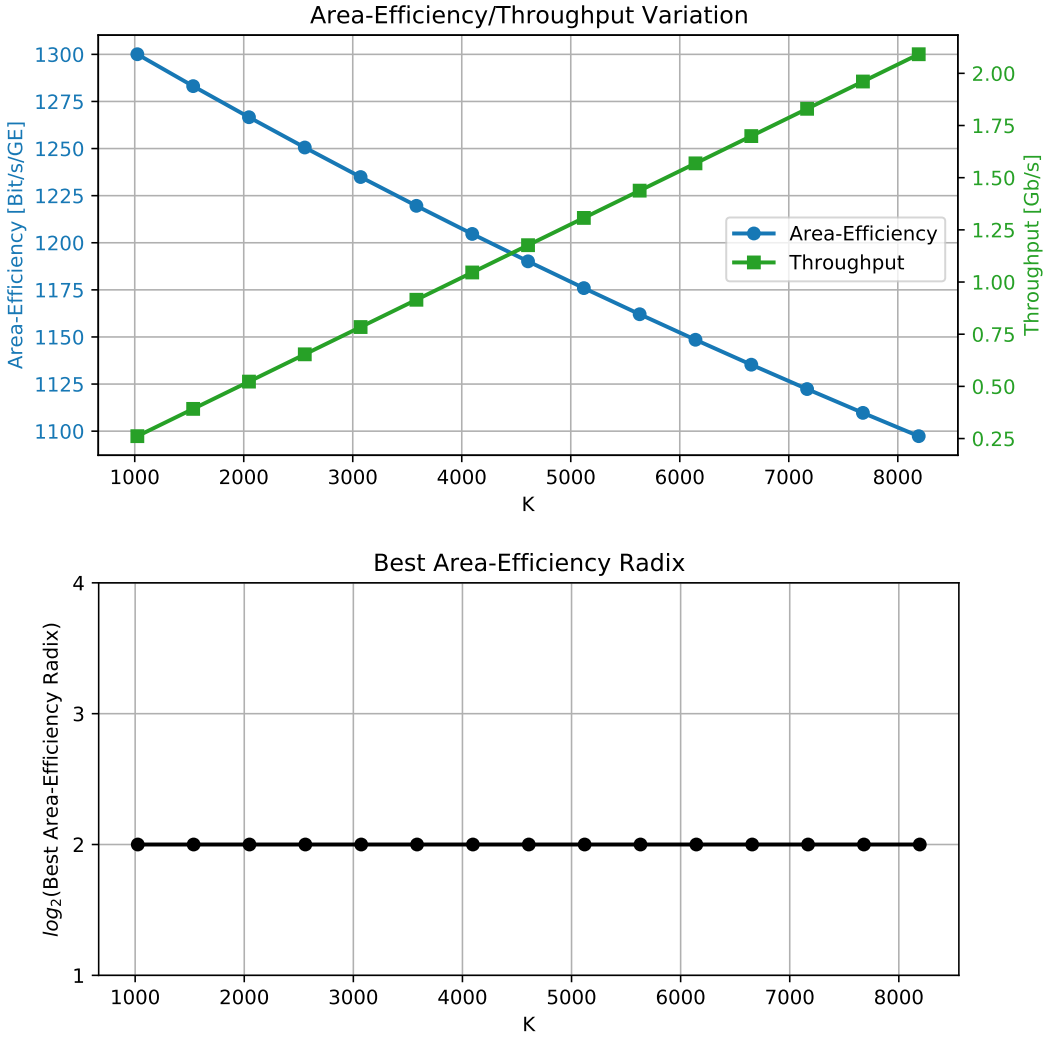

Figure 9.2: Area-Efficiency, Throughput variation (Top) and Most Efficient Radix-Order (Bottom) as function of the Frame-Size K.

The analysis on the Sub-Frame-Size is depicted in figure 9.3. As expected, the throughput is monotonically decreasing, since the number of SISO-decoders operating in parallel is reduced. The declared most efficient radix-order is remaining constant to a radix-4 implementation. The area efficiency is presenting a maximum point, located around $Kp = 256$. As a consequence, once the other parameters have been fixed, it is possible

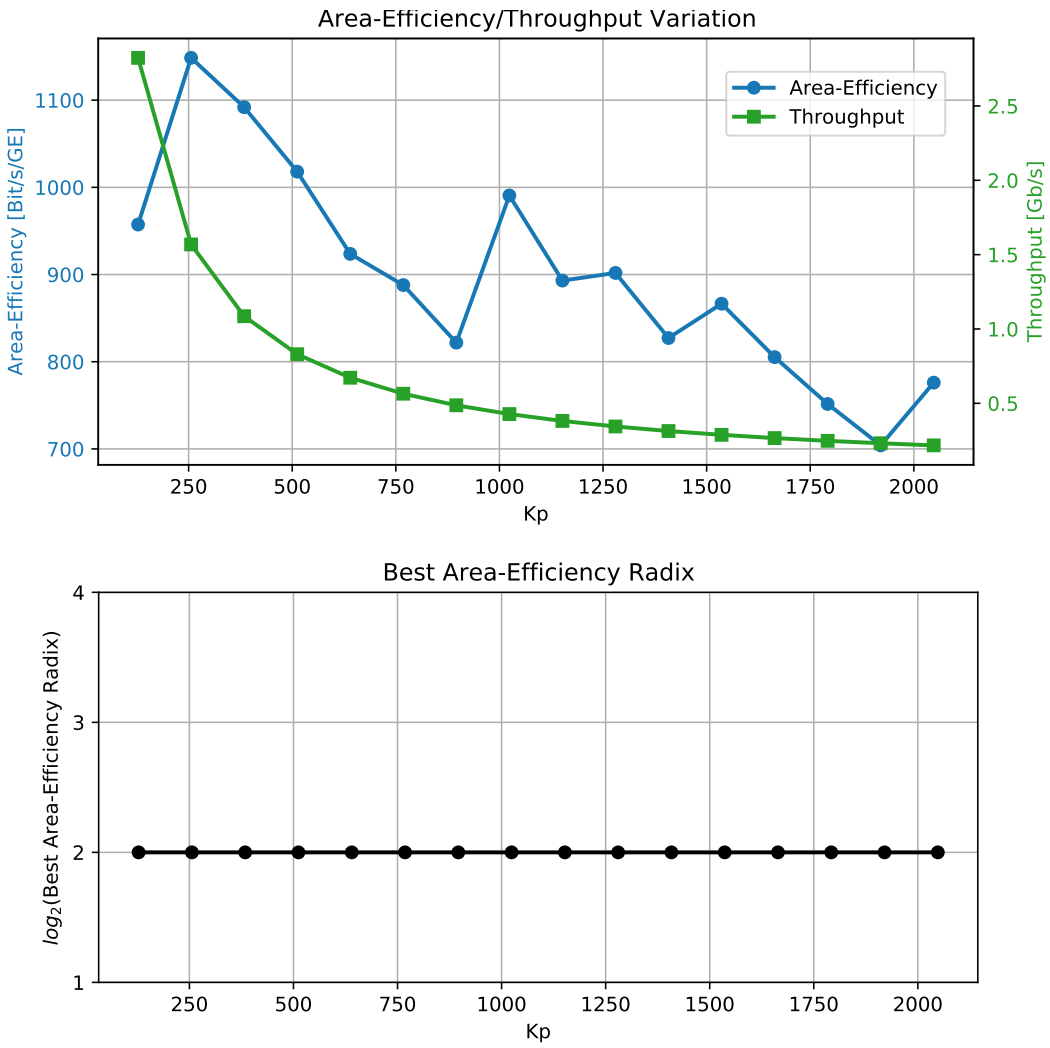

Figure 9.3: Area-Efficiency, Throughput variation (Top) and Most Efficient Radix-Order (Bottom) as function of the Sub-Frame-Size Kp.

to find the optimal Sub-Frame-Size, maximizing the area efficiency. Furthermore, it is interesting to notice how this last curve is including several relative maximum points. As a first consideration, it is important to notice that Kp values which are divisors of the Frame-Size K are optimal choices concerning the area efficiency. Indeed, the efficiency boosts are located at Kp values satisfying this property. Moreover, a larger increment can be noticed around $Kp = 1024$. This further efficiency boost is ensured by the possibility to implement the NII-Memory employing a library model, avoiding the synthesis and reducing the impact on the total area.

The Window-Size variation is analyzed in figure 9.4. Increasing WS, the throughput curve is mostly presenting a reduction, as a consequence of the Forward-Backward scheduling. A fundamental turning point is located around $WS = 64$. Indeed, for $WS \geq 64$, a library model is available for the Alpha-Memory implementation. Therefore, the overall memory impact is greatly reduced and, consequently, the declared most efficient radix-order is

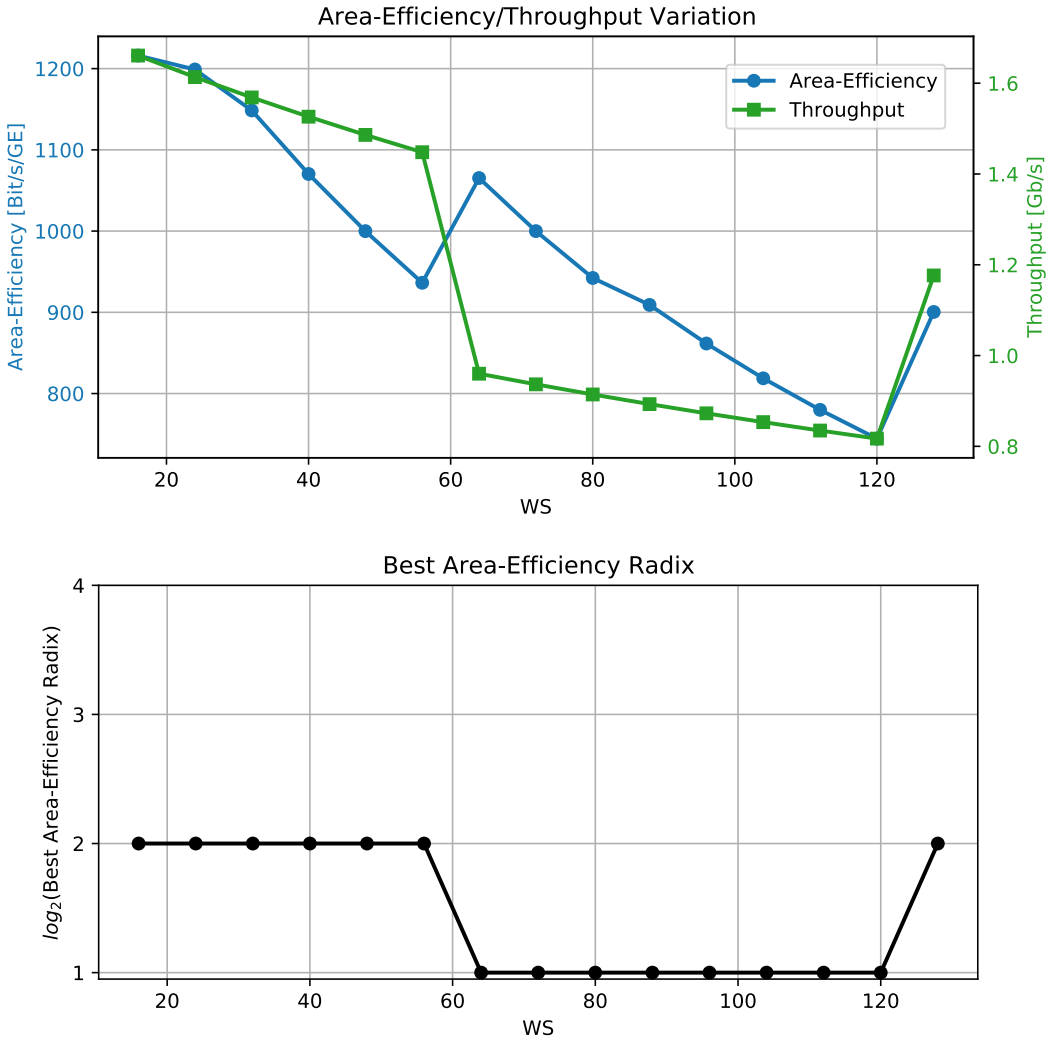

Figure 9.4: Area-Efficiency, Throughput variation (Top) and Most Efficient Radix-Order (Bottom) as function of the Window-Size WS.

moving from 4 to 2. However, if the employed memory model is constant, incrementing the windows size is increasing the memory impact and reducing the overall efficiency. Indeed, it can be noticed how for $WS > 120$, the best-radix order is returning to a radix-4 implementation. Also in this case, efficiency boosts are found when the Window-Size is a divisor of the Sub-Frame-Size Kp.

The last analyzed variation, depicted in figure 9.5, is considering the channel LLR bitwidth w. As covered in Chapter 6, the bitwidths for all the other internal quantities are fully derivable from w, therefore they are automatically computed by the analysis tool.

As expected, the throughput is presenting a monotonic reduction, due to the additional propagation delay found in the PMU. Moreover, also the area efficiency is characterized by a continuous decrement, since both logic and memory areas are negatively affected by larger bitwidths. Both those area contributions are expected to grow linearly as function

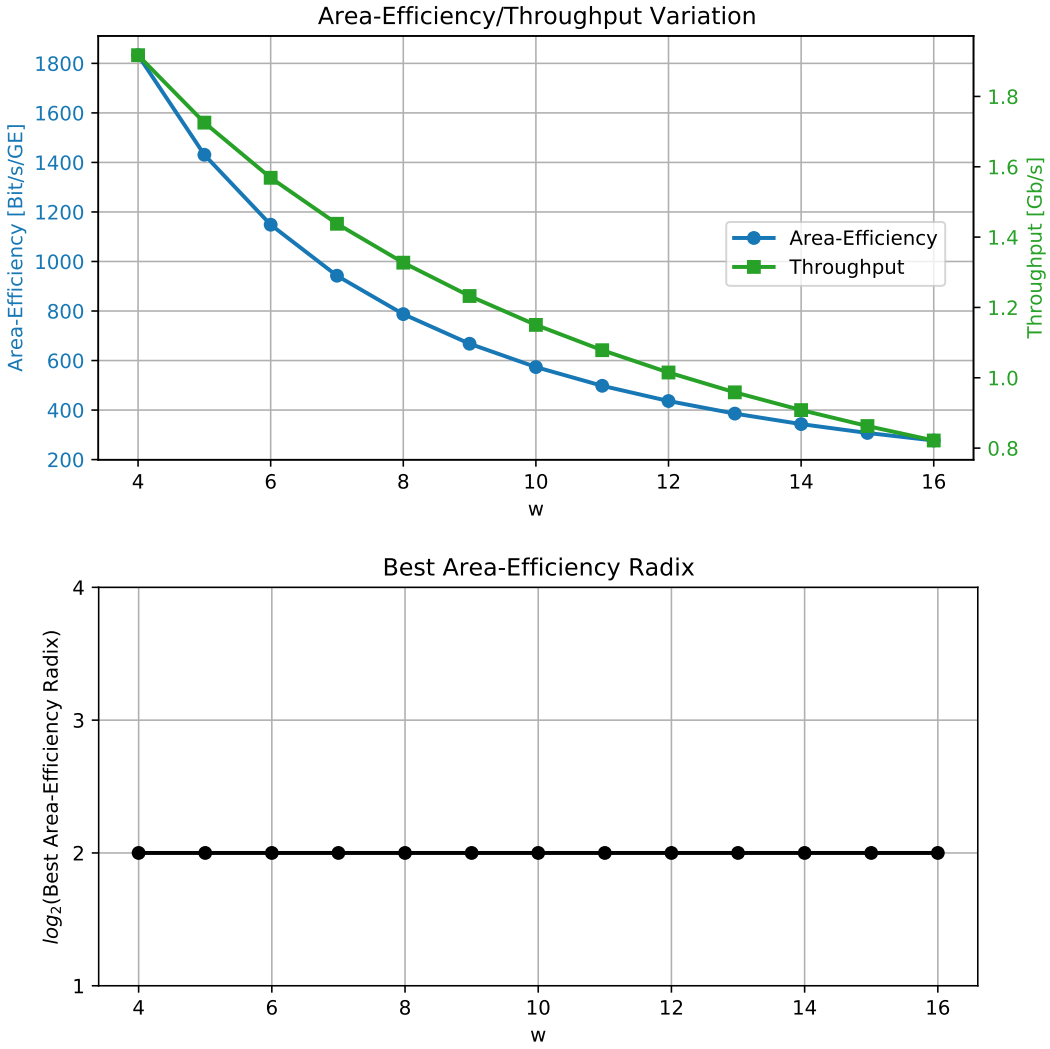

Figure 9.5: Area-Efficiency, Throughput variation (Top) and Most Efficient Radix-Order (Bottom) as function of the Channel LLR bitwidth w.

of w, therefore the most-efficient radix order is not affected by any variation, as visible in the bottom chart.

After analyzing the effects of the proposed quantities one by one, the tool has the possibility to study a set of architectures obtained by changing multiple parameters at the same time. This feature is useful to better explore the solutions space and understand which are the optimal choices for area efficient architectures. The selected variables in this case are the Frame-Size K, the Sub-Frame-Size Kp and the Window-Size WS. As noticeable, the channel LLR bitwidth is not present, since, as visible in figure 9.5, it is not expected to cause any interesting variation. The considered ranges are expressed in table 9.2.

The first two graphs in figures 9.6 and 9.7 are summarizing the area efficiency variation, as well as the most-efficient radix orders that have been selected. From the chart represented

Parameter	Symbol	Start	Stop	Step
Frame-Size	K	2048	6144	2048
Sub-Frame-Size	Kp	128	512	128
Window-Size	WS	32	128	32

Table 9.2: K, Kp and WS proposed variation ranges for the global analysis.

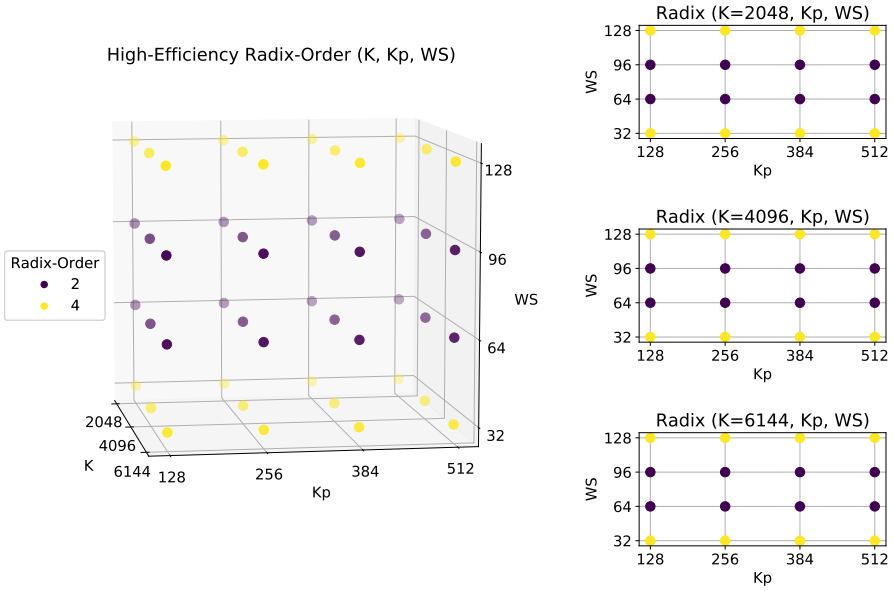

Figure 9.6: Most efficient radix-orders as function of K, Kp and WS.

in figure 9.6, it is possible to appreciate the efficient radix distribution, selecting for each architecture the radix-order expected to maximize the area efficiency. As visible, only radix-2 and radix-4 approaches have been declared, discarding higher radix orders. On the other hand, considering the chart in figure 9.7, high-efficiency solutions can be easily spotted. As expected, optimal choices for high-efficiencies are involving low K and WS values. Moreover, the optimal Sub-Frame-Size is expected to be around $Kp = 256$. An alternative parametric representation for the area efficiency distribution is included in figure 9.8.

Once a solution has been extracted from the efficiency chart, it is possible to observe the correspondent radix-order on the other presented graph. It is interesting to notice how efficient solutions are not relying on a unique radix-order. For instance, considering $Kp = 256$ and $WS = 32$, a radix-4 implementation is required. On the other hand, changing WS to 64 is involving a radix-2 architecture.

A similar analysis can be carried out considering throughput, as shown in figure 9.9. As expected, high-throughputs Turbo-decoders are considering high Frame-Sizes and small Sub-Frame and Window sizes. Moreover, it is noticeable how high throughputs are ranging

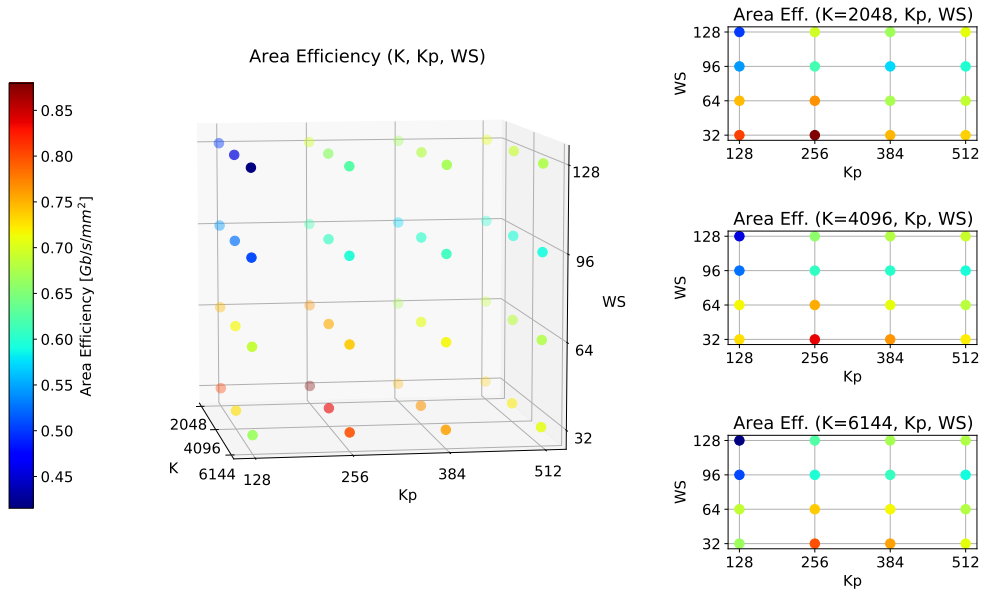

Figure 9.7: Area efficiency as function of K, Kp and WS, using the radix-orders indicated in 9.6.

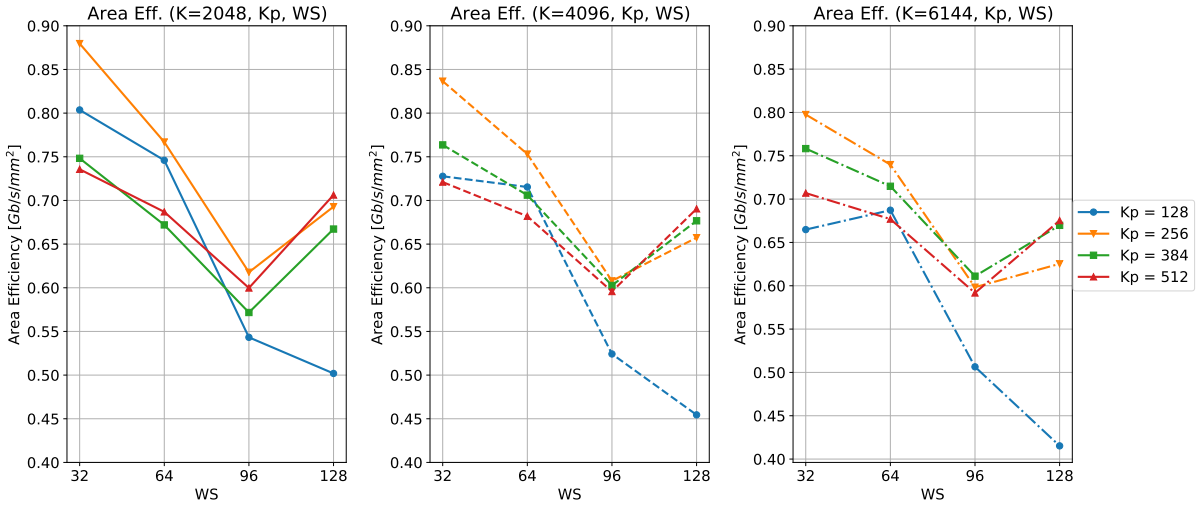

Figure 9.8: Area efficiency as function of K, Kp and WS - Parametric plot.

between 1 Gb/s and 2 Gb/s, typical values for PMAP-based architectures [23].

If just a single Turbo-decoder is employed, the highest-throughput solution can be found by directly considering the depicted graph. However, when Decoder-level parallelism is involved, area efficiency must be considered as an additional parameter. Ideally, if an infinite amount of area is available, the best throughput implementation is corresponding to the one maximizing the area efficiency.

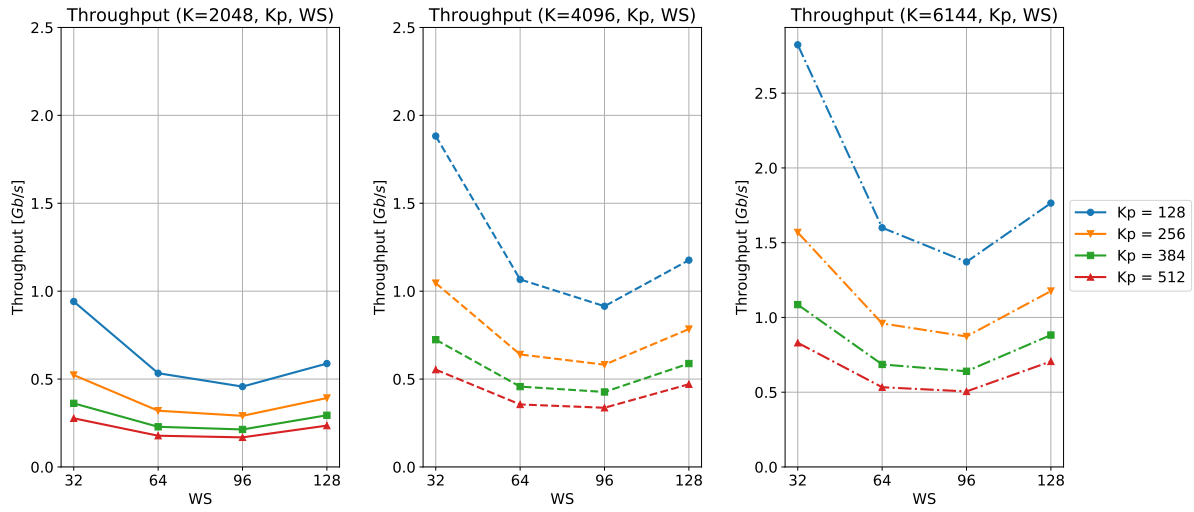

Figure 9.9: Throughput as function of K, Kp and WS.

In a realistic scenario, different throughput distributions are found as function of the available area. In figure 9.10 two examples are presented, considering different restrictions on the maximum areas. The computed throughputs are calculated considering the amount of parallel Turbo-decoders available in the given area.

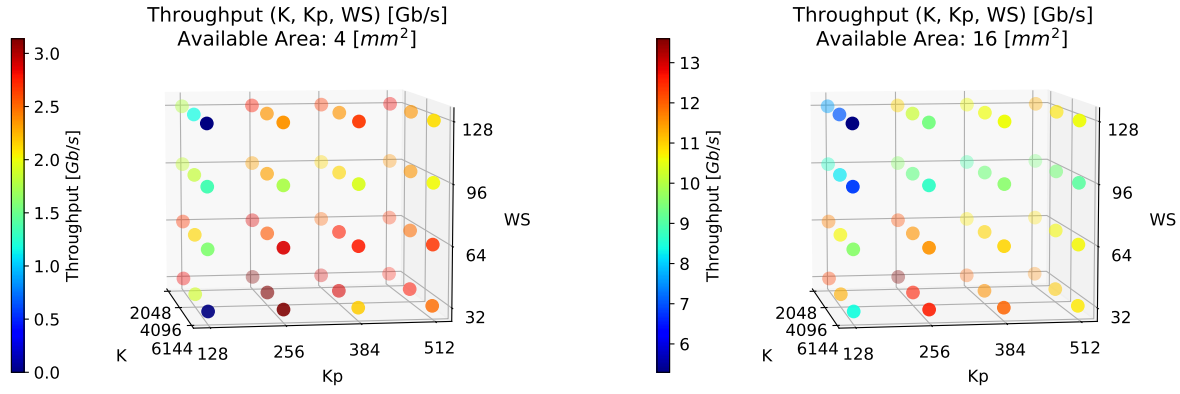

Figure 9.10: Maximum throughput with parallel Turbo-decoders on a given area as function of K, Kp and WS. Two areas are considered: 4 mm^2 (Left) and 16 mm^2 (Right).

As visible, considering small areas is limiting the amount of parallel Turbo-decoders, generating several high-throughput solutions located in different positions. On the other hand, if the given area is high enough, the parallelization is moving the best choices

toward the results stated by the area efficiency analysis.

If throughputs in the order of hundreds of Gb/s are required, Decoder-level parallelization is a fundamental step to be considered. Therefore, the obtained results are proving the importance of the area efficiency parameter in this scenario.

9.3 Efficient Solutions

Supposing to fix the Frame-Size to $K = 6144$, for flexibility purposes, it is possible to search for area efficient solutions by employing the graphs presented in figures 9.6 and 9.7. To simplify the representation, the plane corresponding to $K = 6144$ has been isolated and represented in 9.11.

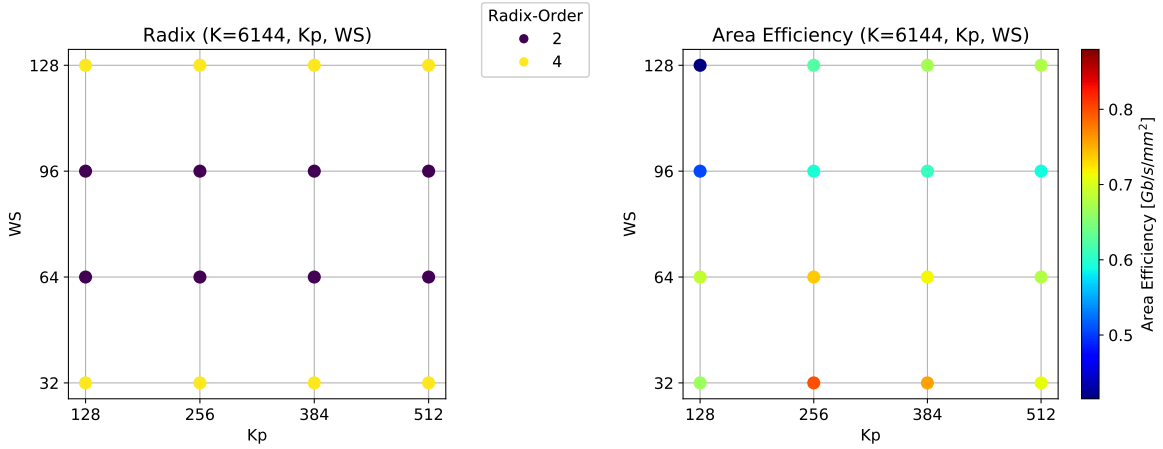

Figure 9.11: Most efficient radix-orders (Left) and area efficiency (Right) as function of Kp and WS, considering $K = 6144$.

Two solutions from the presented set have been selected, both considering an optimal Sub-Frame-Size $Kp = 256$. Regarding the Window-Size, the first implementation is considering $WS = 32$, while the second one $WS = 64$. As noticeable from the left chart, those solutions are expected to be optimized following two different radix orders. The following subsections are detailing the *single analysis* performed on the two architectures.

9.3.1 Architecture 1

This solution is characterized by a Window-Size equal to 32. The full set of specifications has been included in the model and a single analysis has been launched. As expected, the analysis tool declared the radix-4 implementation as the most efficient one. Some

comparison details for the different radix-orders are included in table 9.3, considering a single Turbo-decoder instance.

Radix	**Logic Area**	**Memory Area**	**Throughput**	**Area Efficiency**
	[GE]	[GE]	[Gb/s]	[Gb/s/mm^2]
2	106137 (8%)	1135160 (92%)	1.07	0.60
4	324264 (23%)	1041527 (77%)	1.57	0.80
8	741633 (32%)	1539499 (68%)	1.39	0.42
16	1580466 (46%)	1848461 (54%)	1.85	0.37

Table 9.3: Comparison indicators for *Architecture 1* considering different radix-orders.

Observing the data reported in the table, it is remarkable how the overall area is typically dominated by memories. Just for the radix-16 implementation, logic and memory contributions are expected to be similar. It is interesting to notice how the best area efficiency choice is not involving the maximum throughput among the analyzed solutions. A strong area efficiency penalty is found on the highest radix orders, due to the significant limitations imposed by the logic complexity and the critical path.

Further details on logic and memory, extracted from the analysis tool results, are reported in tables 9.4 and 9.5, considering the proposed radix-4 solution.

Unit	**SISO % Logic Area**
BMU	9%
PMU	61%
SOU	30%

Table 9.4: Logic units area distribution in a SISO-decoder, considering *Architecture 1* and a radix-4 implementation.

As visible, the PMU is expected to be the dominant contribution in the logic area. Regarding memories, it is possible to observe how the Input-Frame-Memory and the Alpha-Memory are covering more than 50% of the total memory area. As expected, the crossbar area contribution is not negligible. It is remarkable how this solution is implementing the Global memories employing library models, while the Local ones are forced to be synthesized, dramatically increasing their area impact.

It is now necessary to analyze the effect of uncertainty over the proposed radix-4 solution. The *reliability analysis* approach presented in Chapter 8 has been applied, considering the relative errors indicated in table 9.6.

Contribution	Model	Words	Bits	Banks	% Memory Area
IN-FR-MEM	*Library*	64	18	96	25%
EXTR-INF-MEM	*Library*	43	7	144	11%
ALPHA-MEM	*Synthesis*	8	88	48	31%
NII-MEM	*Synthesis*	8	88	24	15%
CROSSBAR	-	-	-	-	18%

Table 9.5: Memory sizes, partitioning and area distribution considering *Architecture 1*.

Parameter	Relative Error
Throughput	10%
Logic Area	20%
Memory Area	30%

Table 9.6: Considered relative errors for the reliability analysis on *Architecture 1*.

The considered uncertainties are aiming to prove the solution robustness against different technologies. Moreover, a larger error on the Memory Area has been included to consider the presence of pipeline registers inside the architecture, which have not been included in the analysis model. By testing all the possible combinations, the reliability analysis produced the results reported in figure 9.12.

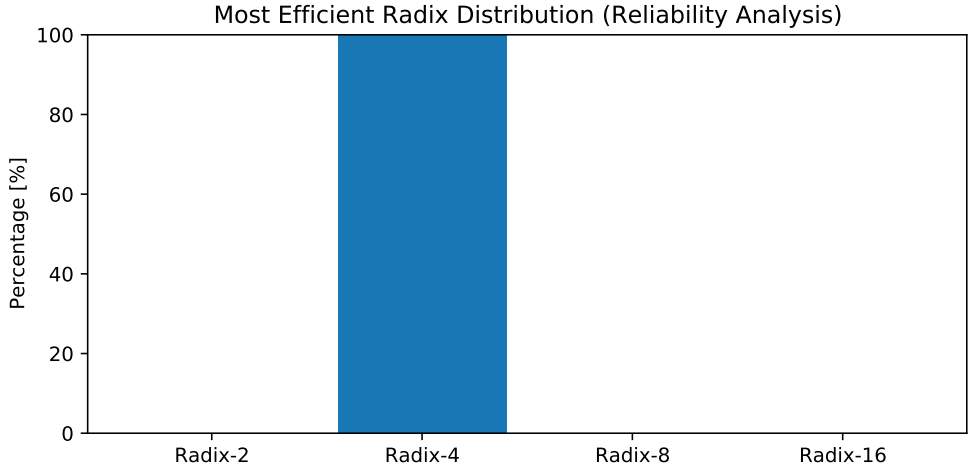

Figure 9.12: Reliability analysis results considering *Architecture 1*.

As visible, even considering the superimposed errors, the analysis model converged to a radix-4 implementation, without any percentage of cases declaring a radix-2 architecture. Therefore, the radix-4 order can be considered a robust choice.

The effects of errors are depending on the differences between efficiencies of the compared architectures. In this case, the radix-4 implementation is presenting a difference which is large enough in order to be considered a reliable solution. On the other hand, if similar efficiencies are found, it is expected a non-convergence from the model, even considering small errors.

9.3.2 Architecture 2

The second analyzed solution is introducing a Window-Size equal to 64. Also in this case, a dedicated single analysis has been launched. The fundamental results are reported in tables 9.7, 9.8 and 9.9.

Radix	**Logic Area**	**Memory Area**	**Throughput**	**Area Efficiency**
	[GE]	[GE]	[Gb/s]	[Gb/s/mm^2]
2	106137 (11%)	795014 (89%)	0.96	0.74
4	324264 (20%)	1282295 (80%)	1.41	0.61
8	741633 (30%)	1659883 (70%)	1.25	0.36
16	1580466 (44%)	1936013 (56%)	1.66	0.33

Table 9.7: Comparison indicators for *Architecture 2* considering different radix-orders.

Unit	**SISO % Logic Area**
BMU	4%
PMU	54%
SOU	42%

Table 9.8: Logic units area distribution in a SISO-decoder, considering *Architecture 2* and a radix-2 implementation.

In this case, the most efficient architecture is following a radix-2 implementation. Since the logic units are including a small amount of operators, the Turbo-decoder area is almost completely defined by memories.

Observing table 9.9, it is noticeable how a library model is available for the Alpha-Memory implementation. Since Alpha memories are representing a significant percentage of the total memory area, their implementation through library models is guaranteeing a high efficiency for the radix-2 approach. As a matter of fact, the radix-4 implementation is not capable to guarantee this possibility.

Contribution	Model	Words	Bits	Banks	% Memory Area
IN-FR-MEM	*Library*	128	18	48	25%
EXTR-INF-MEM	*Library*	86	7	72	11%
ALPHA-MEM	*Library*	32	88	48	48%
NII-MEM	*Synthesis*	4	88	24	10%
CROSSBAR	-	-	-	-	6%

Table 9.9: Memory sizes, partitioning and area distribution considering *Architecture 2*.

Furthermore, this implementation is relaxing the crossbar specifications, since a lower amount of partitions is required for the Extrinsic-Information-Memory. Indeed, as visible, the crossbar impact on the area is drastically reduced if compared to a radix-4 implementation.

A reliability analysis has been launched also for this solution, considering the relative errors indicated in table 9.6. As noticeable in figure 9.13, all the performed tests are declaring a radix-2 implementation as the most efficient one. Therefore, the radix choice robustness has been proved.

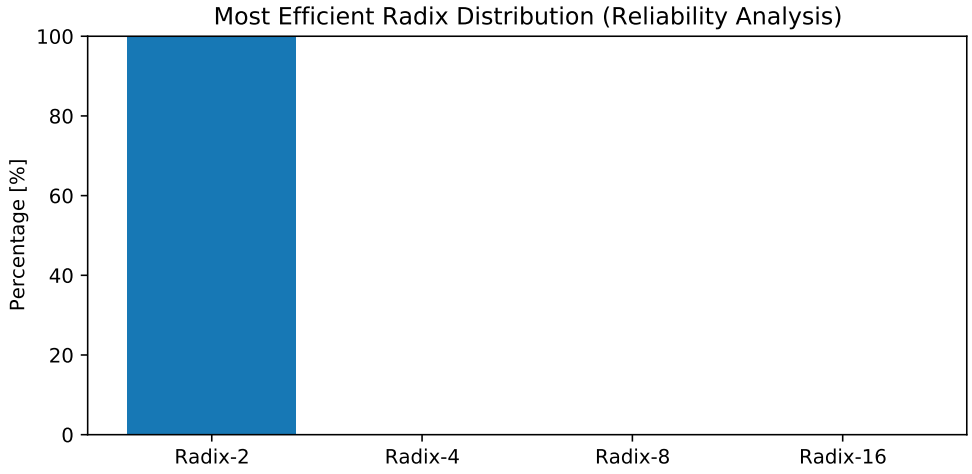

Figure 9.13: Reliability analysis results considering *Architecture 2*.

The presented examples have shown the fundamental role of the Alpha-Memory in the choice of the most efficient radix-order. Indeed, since it is representing a large percentage of the total memory area, its implementation is heavily affecting the achievable area efficiency.

In the next chapter, the model results obtained for the two considered solutions will be compared with synthesis data deriving from a full architectural description, in order to

highlight the model accuracy.

9.4 High-Radix Limitation

As visible from the data presented during the previous sections, radix-8 and radix-16 architectures are not considered efficient choices. Indeed, observing a generic radix comparison, for instance table 9.3, it is noticeable how area efficiencies for the radix-8 and radix-16 solutions are significantly lower if compared to the other available radix-orders.

Efficiency is mostly limited by the critical path increment, caused when a radix-8 PMU is involved. This statement can be proved by comparing different radix-orders while removing the critical path increment factors. In other words, it is assumed the all the radix solutions are working with the same clock frequency, which has been selected equal to 259 MHz, from table 8.16. A global analysis has been launched, exploring the same architectural space defined in table 9.2. The obtained results are shown in figure 9.14.

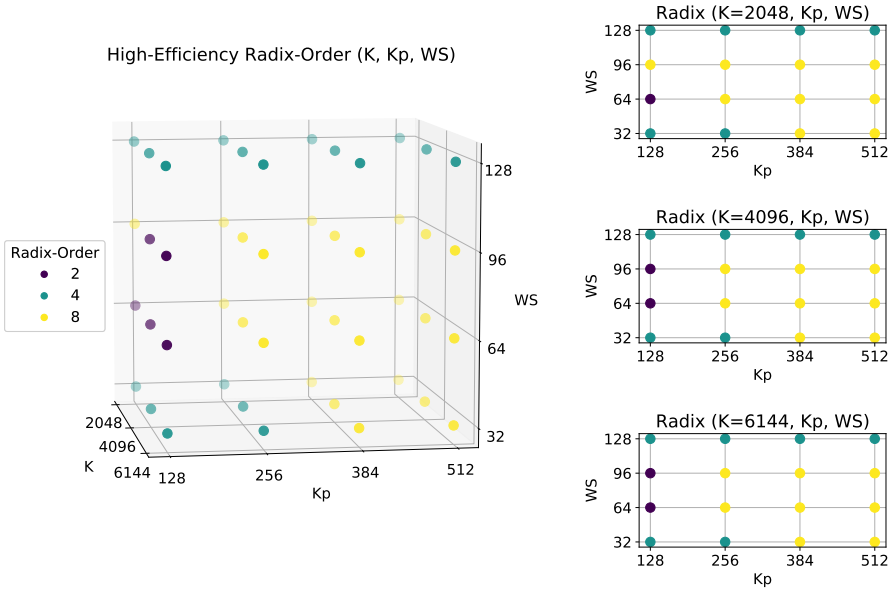

Figure 9.14: Most efficient radix-orders as function of K, Kp and WS, assuming the same working clock frequency for all the radix-orders.

As visible, the depicted chart is opening the possibility for efficient radix-8 implementations. However, these results are valid only under the assumption of a fixed clock frequency, a condition that may be requested in some practical cases. Although, in this work, the interest is to find the most efficient solutions and, as a consequence, the working frequency is representing an important degree of freedom to be included in the study.

The results obtained from the last analysis are suggesting that decreasing the PMU critical path for high radix-orders could open the possibility for their efficient use. As mentioned, during the performed analysis, the CS4-based CS8 model has been employed. In this scenario, a switch to the CS8-fast model has been performed, in order to test the consequences of the critical path reduction. Therefore, an additional global analysis has been launched, restoring the clock frequency dependence from the radix-order.

The results are identical to the ones shown in figure 9.6, therefore radix orders higher than 4 have been discarded from the efficient solutions set. However, a slight improvement in the area efficiency can be noticed by performing a single analysis. For instance, considering $K = 6144$, $Kp = 256$ and $WS = 32$, a comparison between the different CS8 models can be carried out.

Radix	**CS8 Model**	**% Log. Area**	**% Mem. Area**	**Area Efficiency**
				[Gb/s/mm^2]
8	CS4-based	32%	68%	0.42
8	CS8-fast	39%	61%	0.48
16	CS4-based	46%	54%	0.37
16	CS8-fast	50%	50%	0.44

Table 9.10: CS8 model comparisons on radix-8 and radix-16 architectures.

The results reported in table 9.10 are showing how employing the proposed CS8-fast unit is improving the area efficiency. However, the efficiencies are still limited if compared to radix-2 and radix-4 implementations. As visible, an important drawback is found on the logic area contribution, since its percentage over the total area is significantly increased, due to the larger amount of employed operators.

Following this direction, it is required to find CS8 solutions which are presenting an optimal trade-off between area occupation and critical path. An interesting possibility could be to reduce the number of CS2 units in the CS8-fast architecture, lowering the impact on critical path and area, while introducing an acceptable degradation on the error-correction performances.

As a conclusion, the presented results are showing how radix-2 and radix-4 implementations are valuable choices, capable to introduce an optimal trade-off between throughput and complexity. Depending on the logic and memory organization, one among those two solutions can be selected in order to maximize the area efficiency. However, higher radix-orders should not be automatically discarded, since possible improvements on the logic architecture could lead to acceptable area efficiencies. Moreover, a further advantage of high radix-order solutions is the latency reduction, which is fundamental for part of the presented use cases by the EPIC project.

9.5 Very High Throughputs

In this section, the Concurrent-PMAP model is pushed toward high-throughputs, in order to better understand its limitations and study its applicability. The EPIC project, following the purpose of a realistic implementation, indicated constraints on some Key Performance Indicators [11]. A maximum *area* of 10 mm^2 is declared, with a correspondent maximum *area efficiency* equal to 100 $Gb/s/mm^2$.

Therefore, considering the 65 nm technology, it is possible to estimate how many Turbo-decoders can be implemented in a given area equal to 10 mm^2, highlighting the throughput distribution over different architectures. A global analysis has been launched, fixing the maximum area size and allowing the usage of concurrent Turbo-decoders. The produced results are shown in figure 9.15.

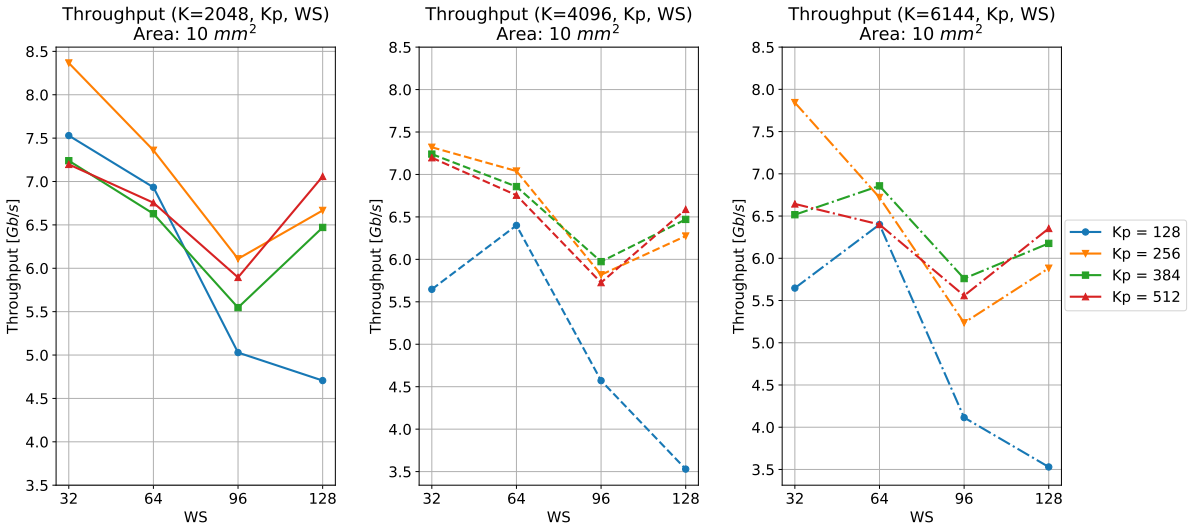

Figure 9.15: Maximum throughput with concurrent PMAPs on a 10 mm^2 area. Technology: 65 nm.

As visible, parallelizing PMAP architectures is producing proportional benefits in terms of throughput. The expected maximum achievable throughput is above 8 Gb/s. The dimension of the single PMAP-decoder is stating how many instances can be found in the given area. As a matter of fact, the amount of instances is strongly depending on the employed technology.

The analysis tool has the capability to estimate results on different technologies, starting from the reference one and considering a scaling factor on the GE area. As a general rule, given the *scaling factor* S, which is the size ratio between transistors implemented

with different technologies, the area is expected to scale as $1/S^2$. However, this simple scaling rule is inaccurate if applied to recent technologies [29], since not all the quantities are expected to scale as function of a unique scaling factor S. Large differences from the ideal scaling model can especially be found considering technologies below 45 nm (high-k dielectrics introduction) and below 20 nm (multi-gate technology introduction). As a consequence, the authors in [29] proposed a method to properly determine scaling factors, while keeping into account the issues mentioned above. Part of the area scaling factors indicated in the article have been extracted and reported in table 9.11, considering as a starting node the 65 nm technology.

Starting Node	**Desired Node**	**Area Scaling Factor**
65 nm	65 nm	1
65 nm	45 nm	1.5
65 nm	32 nm	3.3
65 nm	28 nm	4.6[1]
65 nm	20 nm	7.1
65 nm	16 nm	7.9
65 nm	14 nm	8.7
65 nm	10 nm	15
65 nm	7 nm	25

Table 9.11: Area scaling factors from the 65 nm technology to other technological nodes.

In order to perform a comparison with the Unrolled-X-MAP architecture, presented in table 7.1 during the State-of-the-Art review, a scaling factor has been introduced to properly estimate the results on a 28 nm technology. Moreover, the number of half-iterations and the maximum working clock frequency are considered equal to the UXMAP implementation. The global analysis results are represented in figures 9.16 and 9.17.

As visible from the efficiency plot, area efficiencies higher than 7.5 Gb/s/mm^2 are expected to be reached, comparable to the 6.2 Gb/s/mm^2 value declared for the UXMAP architecture. Moreover, the achievable throughput on the same maximum area has been compared: the Concurrent-PMAP approach is expected to produce throughputs above 100 Gb/s, also comparable with the throughput declared for the reference UXMAP.

As a direct result, concurrent Decoder-level parallelism should not be discarded as a valuable solution for high-throughput requirements. It is remarkable how these results should not be used as a detailed comparison metric between the two architectures, since the area efficiencies are expected to be overestimated by the model, as detailed in the

[1]Scaling factor obtained by linearly interpolating the 32 nm and 20 nm results.

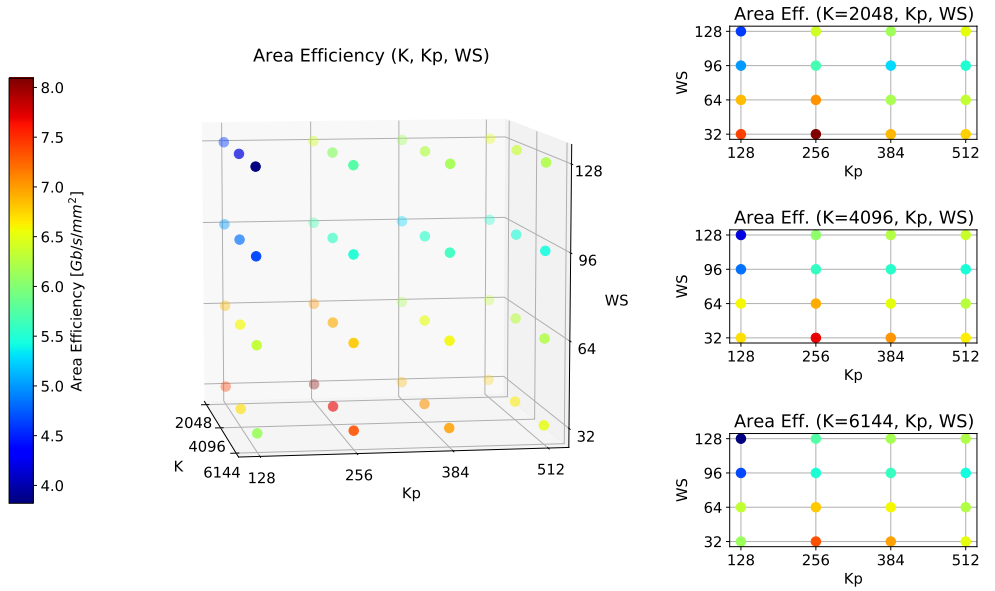

Figure 9.16: Area efficiency distribution as function of K, Kp and WS. Technology: 28 nm (Scaled).

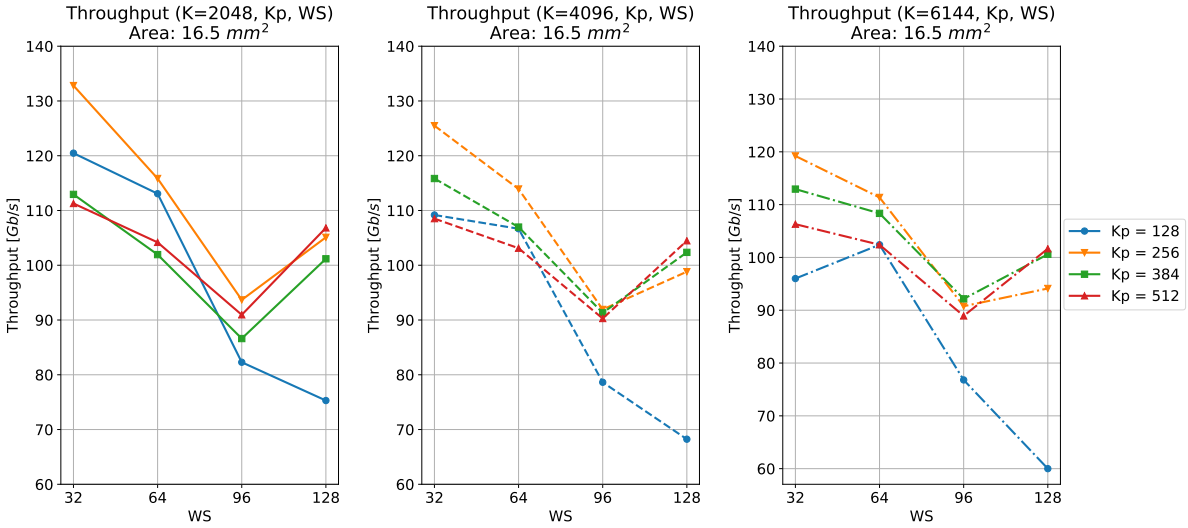

Figure 9.17: Maximum throughput with concurrent PMAPs on a 16.5 mm^2 area. Technology: 28 nm (Scaled).

next chapter. Therefore, the aim of this analysis is simply to prove the suitability of the proposed architecture and introduce the need for a more detailed comparison against the UXMAP approach.

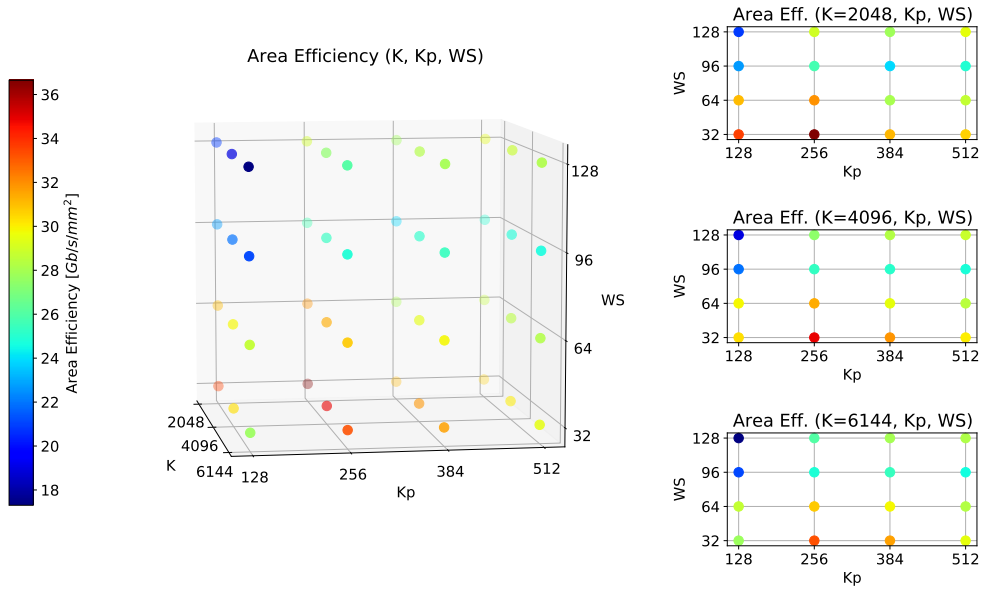

Figure 9.18: Area efficiency distribution. Technology: 7 nm (Scaled).

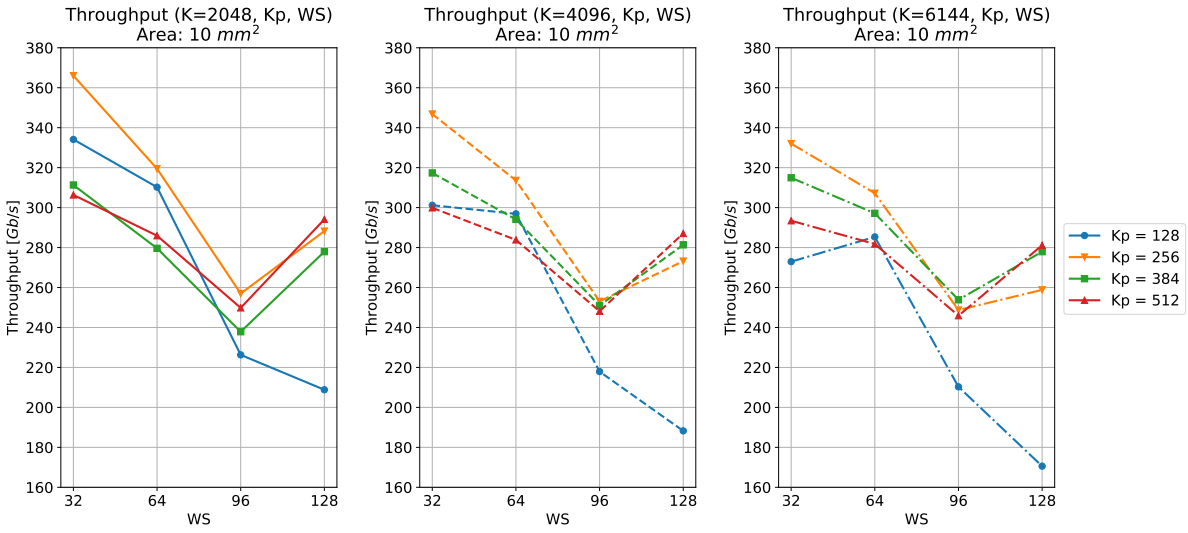

Figure 9.19: Maximum throughput with concurrent PMAPs on a 10 mm^2 area. Technology: 7 nm (Scaled).

Furthermore, it is interesting to scale the solutions space to the 7 nm technology, considered as a reference technology for comparisons by the EPIC project. Moreover, a maximum area of 10 mm^2 has been considered and the maximum working clock frequency has been fixed to 1000 MHz, following the EPIC comparison methodology. Area efficiency and throughput distributions are reported in figures 9.18 and 9.19.

The area efficiency is still far from the 100 Gb/s/mm^2 limit, as well as the throughput from 1 Tb/s. Consequently, further architectural and algorithmic improvements are required. However, it is fundamental to consider an important feature of the concurrent-PMAP architecture, which is the possibility to improve the area efficiency by reducing the required amount of iterations, according to the expected BER. For instance, modifying the total iteration number from 6 to 4 is guaranteeing a 50% boost on the area efficiency. Moreover, modifying the half-iteration number is not impacting on the total required area, in opposition with the UXMAP approach.

Chapter 10

Model Validation

This chapter aims to validate part of the obtained results from the analysis tool, relying on synthesis operations. First, the main logic units for radix-2, 4 and 8 architectures will be considered, comparing both areas and critical paths. Moreover, their correct functionality will be also tested. Then, both SISO and Turbo decoders architectures will be synthesized, aiming to prove the results estimated in Chapter 9 for *Architecture 1* and *Architecture 2*.

10.1 Logic Units Validation

As mentioned in the introduction, BMU, PMU and SOU have been described in hardware following different radix-orders. In particular, the radix-8 PMU has been implemented relying on CS4-fast operators. As a first step, the correct functionality has been tested by employing a set of testbenches per each radix-order. Reference input data and expected results, considering a frame size $K = 6144$, have been extracted from the C-model[1], also employed for the simulations presented in Chapter 6. A group of testbenches on a given radix-order is organized as illustrated in figure 10.1: the results from each logic block are employed to emulate a full half-iteration on the given architecture, checking the correctness of the final produced LLR. Moreover, each block is also singularly tested in the process.

All the performed testbenches confirmed the correct functionality of the involved logic units. Therefore, it is possible to proceed with the synthesis operations, in order to detail informations on area and critical path. All the synthesis proposed in this chapter are performed employing the 65 nm technology.

[1]Turbo-Decoder model available at the VLSI group - Politecnico di Torino

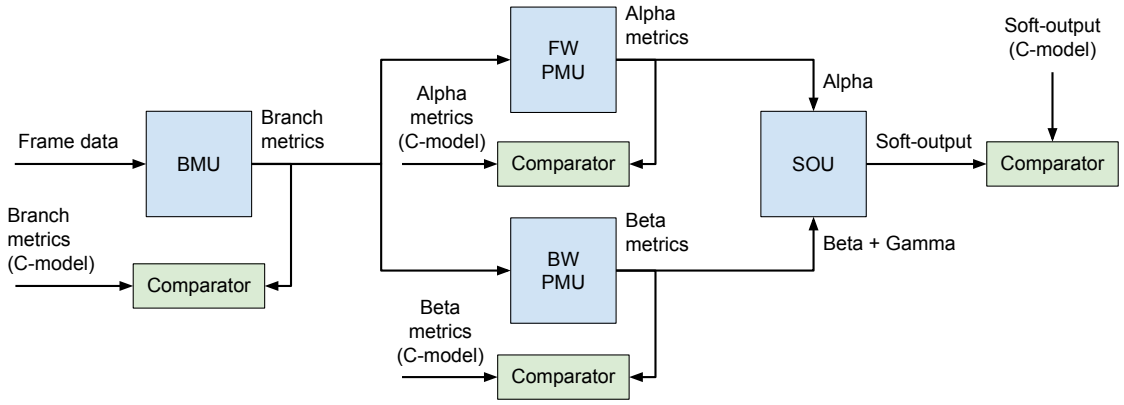

Figure 10.1: Testbench block scheme for logic units.

First, a study on the area occupation is detailed in table 10.1, including a comparison with the analysis tool results. As visible, the estimations are accurate, since the maximum introduced error is below 1.5%. Consequently, the proposed approach to estimate the computational units' area can be considered reliable.

Radix	**Logic Unit**	**Area synthesis [GE]**	**Area model [GE]**	**% Error**
2	BMU	82	82	0.0%
2	PMU (FW/BW)	1196	1194	0.2%
2	SOU	1875	1871	0.2%
4	BMU	598	600	0.3%
4	PMU (FW/BW)	4097	4155	1.4%
4	SOU	4008	4001	0.2%
8	BMU	3075	3075	0.0%
8	PMU (FW/BW)	8034	8084	0.6%
8	SOU	8600	8584	0.2%

Table 10.1: Logic units area comparison between model results and synthesis results.

Proceeding with the next step, critical delays have been extracted from the synthesized designs, aiming to verify the critical path increment factors employed by the model. Comparison data are reported in table 10.2. The estimations performed on the critical path increment, against the reference radix-2 architecture, have been proved to be valid. Moreover, as visible, pipeline stages are required in the SOU architecture, in order to bound the critical path to the PMU. The same consideration is expected to be extended also for the BMU, when high radix-orders are involved.

			Radix-2 CP Increment Factor		
Radix	**Logic Unit**	**Critical Path [ns]**	**Synthesis**	**Model**	**% Error**
2	BMU	0.75	-	-	-
2	PMU (FW/BW)	1.46	1	1	0.0%
2	SOU	5.02	-	-	-
4	BMU	1.07	-	-	-
4	PMU (FW/BW)	1.99	1.36	1.36	0.0%
4	SOU	6.41	-	-	-
8	BMU	1.63	-	-	-
8	PMU (FW/BW)	3.37	2.31	2.31	0.0%
8	SOU	7.81	-	-	-

Table 10.2: Critical path synthesis data and comparison with the increment factors employed in the developed model.

Developing a complete hardware description of the logic units is a time-consuming task. Therefore, obtaining accurate estimations starting from the included basic operators is a remarkable advantage.

10.2 Architectures Validation

This section aims to verify the results obtained for the two area efficient solutions presented in Chapter 9. Therefore, it is necessary to fully describe in hardware a PMAP Turbo-decoder architecture.

First, the SISO structure has been defined, including additional hardware elements necessary to fully control the algorithm application. Additional multiplexers were added, as well as layers of pipeline registers to separate the logic stages. Moreover, also the Extrinsic-Scaling-Factor logic has been included. In this phase, all the synthesized local memories are added to the architecture's description. Then, multiple SISO instances are included in the Turbo-decoder, introducing Global memories and the required crossbar. Due to the limited amount of available library memory models, their contribution has been included considering the estimations produced by the analysis tool.

10.2.1 Architecture 1

Considering this solution, it is desired to verify the area efficiency improvement deriving from using a radix-4 implementation against a radix-2 one. Therefore, the two architectures have been described. As mentioned, it is necessary to collect library memory data before proceeding with the synthesis. In this case, the memories relying on library models are the Input-Frame-Memory and the Extrinsic-Information-Memory. Their data are included in table 10.3.

Radix	**Memory**	**Size**	**Banks**	**GE/Bit**	**Area [mm^2]**
2	IN-FR-MEM	128x18	48	1.76	0.280
2	EXTR-INF-MEM	86x7	72	2.07	0.129
4	IN-FR-MEM	64x18	96	2.31	0.368
4	EXTR-INF-MEM	43x7	144	2.62	0.164

Table 10.3: Library memory models for *Architecture 1*.

Then, radix-2 and radix-4 implementations have been synthesized, and area reports have been generated. After the synthesis, the library memory areas have been added. From the gathered informations, it is possible to separate the logic and memory contributions in the total area, as reported in tables 10.4 and 10.5.

Radix-2			
Quantity	**Synthesis Data**	**Model Data**	**% Error**
Total Area [mm^2]	1.943	1.787	8.0%
Logic Area [mm^2]	0.186 (10%)	0.153 (9%)	17.7%
Memory Area [mm^2]	1.757 (90%)	1.634 (91%)	7.0%
Area Efficiency [Gb/s/mm^2]	0.549	0.597	8.7%

Table 10.4: *Architecture 1*, Radix-2 - model and synthesis data comparison.

As expected, observing table 10.4, the synthesized area is larger, due to the additional logic and pipeline registers, not included in the analysis model. The two mentioned contributions are generating errors respectively on logic and memory areas. As visible, the logic is particularly affected, since in a radix-2 architecture the computational logic complexity is limited. However, the error impact on the efficiency estimation is below 10%.

Similar considerations are found for the radix-4 case, except for the error effect on the total logic area, which is limited. Indeed, the higher logic complexity of the radix-4 architecture is guaranteeing robustness against possible overheads. The introduced area

Radix-4			
Quantity	**Synthesis Data**	**Model Data**	**% Error**
Total Area [mm^2]	2.202	1.967	10.7%
Logic Area [mm^2]	0.506 (23%)	0.467 (24%)	7.7%
Memory Area [mm^2]	1.696 (77%)	1.500 (76%)	11.6%
Area Efficiency [Gb/s/mm^2]	0.712	0.797	11.9%

Table 10.5: *Architecture 1*, Radix-4 - model and synthesis data comparison.

efficiency error is below 12%. The presented comparisons highlight the importance of detailing the analysis model, if really accurate estimations are desired. However, the presented errors are acceptable considering the purpose of this work, which is purely centered around exploring different solutions.

It is interesting to notice how the best radix-order choice is strongly reliable, if the *area efficiency improvement ratio* is compared. The model is predicting that employing a radix-4 architecture is improving the efficiency with a factor 1.34 against the radix-2 solution. The same factor, computed employing synthesis data, is 1.30, introducing a 3.1% error. As a final consideration, the expected efficiency improvement from selecting the right radix-order is around 30%.

10.2.2 Architecture 2

Also in this case, the first step consists in characterizing memories which are requiring a library model. As a difference with respect to *Architecture 1*, the radix-2 implementation is ensuring a library model also for the Alpha-Memory. The obtained data are included in table 10.6.

Radix	**Memory**	**Size**	**Banks**	**GE/Bit**	**Area [mm^2]**
2	IN-FR-MEM	128x18	48	1.76	0.280
2	EXTR-INF-MEM	86x7	72	2.07	0.129
2	ALPHA-MEM	32x88	48	2.83	0.551
4	IN-FR-MEM	64x18	96	2.31	0.368
4	EXTR-INF-MEM	43x7	144	2.62	0.164

Table 10.6: Library memory models for *Architecture 2*.

Logic and memory data extracted from synthesis are included in tables 10.7 and 10.8.

Radix-2			
Quantity	**Synthesis Data**	**Model Data**	**% Error**
Total Area [mm^2]	1.468	1.298	11.6%
Logic Area [mm^2]	0.186 (13%)	0.153 (12%)	17.7%
Memory Area [mm^2]	1.282 (87%)	1.145 (88%)	10.7%
Area Efficiency [Gb/s/mm^2]	0.654	0.740	13.1%

Table 10.7: *Architecture 2*, Radix-2 - model and synthesis data comparison.

As visible in 10.7, the errors on the estimated quantities are exceeding 10%. As a general rule, the smaller the logic and memory areas, the larger their sensitivities to possible overheads. Depending on how the architecture is organized, it may be necessary to include the contribution of the additional logic and pipeline registers. Indeed, this work proved the significant area impact introduced by Flip-Flop memories. Therefore, if a large amount of quantities are required to be pipelined, it may be necessary to model the required registers. In case the architecture is based on forwarding data by means of pipelines (XMAP and UXMAP), an accurate characterization of the included registers is mandatory to introduce acceptable errors.

Radix-4			
Quantity	**Synthesis Data**	**Model Data**	**% Error**
Total Area [mm^2]	2.534	2.313	8.7%
Logic Area [mm^2]	0.506 (20%)	0.467 (20%)	7.7%
Memory Area [mm^2]	2.028 (80%)	1.846 (80%)	9.0%
Area Efficiency [Gb/s/mm^2]	0.557	0.610	9.5%

Table 10.8: *Architecture 2*, Radix-4 - model and synthesis data comparison.

As expected, the radix-4 solution is more robust against the overhead added to the logic area. Consequently, a smaller error is found on the total area estimation.

Also in this case, the area efficiency improvement factor, considering the choice of a radix-2 architecture against a radix-4 one, is expressing high reliability. Indeed, the model is proposing an improvement factor equal to 1.21, while a value equal to 1.17 is derived from synthesis data, introducing a 3.4% error. Overall, in the considered scenario, the radix-2 solution is expected to introduce a 15% higher efficiency if compared to the radix-4 implementation.

10.2.3 Conclusions

This section proved how the developed model can be considered reliable on the choice of the most efficient radix-order. Therefore, it can be used to test further architectural or algorithmic improvements for high-radix implementations. More generally, area efficiencies comparisons are expected to produce reliable results.

On the other hand, single area efficiency computations could be improved by adding further details in the area estimation model. Fundamental error sources are the additional logic necessary in the SISO architecture and the included pipeline registers. By now, a limited degree of underestimation must be considered on the total areas provided by the analysis tool, consequently affecting the area efficiencies.

Chapter 11

Conclusions and Future Works

This chapter aims to first summarize and review the results presented in this work, focusing on their impact on modern Turbo-decoder architectures. Moreover, possible steps toward future works will be discussed, starting from the previously analyzed results.

11.1 Conclusions

In the first part of this work, a review on Turbo-Codes has been detailed, focusing on the specific use of the LTE/UMTS standard code. The usage of the Max-Log-Map decoding algorithm, including the ESF, has been justified, considering the optimal trade-off between BER and complexity.

Then, a set of important parallelization degrees has been presented, focusing on their impact on throughput, area, BER and flexibility. Moreover, a critical review of the State-of-the-Art architectures has been included, highlighting the implemented parallelization degrees. In this scenario, the UXMAP architecture is a promising candidate for high-throughput requirements. However, also PMAP-based solutions with high degrees of parallelism should be considered: this work proposed a detailed analysis on the Concurrent-PMAP architecture, which is including several PMAP-decoders working in parallel. Moreover, its suitability for high-radix orders has been studied.

A generic comparison model has been developed, aiming to fully explore a given architectural space, generalizing the results against the employed technology. The central role of the area efficiency parameter has been highlighted, proving its importance in scenarios including high degrees of parallelization. The steps necessary to build the comparison tool have been covered, highlighting the necessary architectural choices for both logic and memory organization. More specifically, several techniques and methods have been introduced in this work, in order to face issues presented by employing high radix-orders.

The results have shown how radix-2 and radix-4 architectures are suitable implementations, capable to introduce optimal area efficiencies. Depending on the specifications, one among these two radix approaches should be selected. Moreover, it has been proved how the memory organization is playing a fundamental role in the choice of the most efficient radix-order. On the other hand, higher-order implementations are limiting the achievable area efficiency, mainly due to their critical paths. However, it has been proved how a critical path reduction is potentially opening the possibility for efficient radix-8 architectures. Therefore, they should not be a-priori discarded, also considering the latency reduction that they are able to guarantee.

The area efficiency variation has been studied as function of fundamental PMAP parameters, considering the effect of each single quantity separately. Therefore, the required choices directed toward efficiency maximization have been detailed. It has been proved how, following the estimated results, the Concurrent-PMAP architecture is expected to reach comparable throughputs and area efficiencies if confronted with the UXMAP solution. Therefore, the suitability of PMAP-based architectures in the high-throughput scenario has been proved. Moreover, considering a technological scaling estimation, it has been noticed how the presented Concurrent-PMAP architecture is not expected to fully satisfy the throughput requirements expected for the next future use cases. Indeed, maximum area efficiencies below 36 Gb/s/mm^2 are expected to be reached on the 7 nm technology. Consequently, further architectural and/or algorithmic improvements are required.

As a last aspect, the accuracy of the developed model has been analyzed, confirming its suitability for comparison purposes among different architectures. The relative uncertainties on the provided numerical results have been detailed, highlighting the possible steps to further improve the model accuracy.

11.2 Future Works

The presented comparison model has been developed considering a set of assumptions, which are limiting the architectural space to be explored. Possible future works could consider an extension of the model, including additional degrees of freedom. For instance, interesting results could be obtained by comparing the usage of different scheduling policies. Moreover, accuracy improvements could be introduced by modeling the presence of pipeline registers in the architecture.

Interesting steps could be also considered toward the implementation of high radix-order architectures. In this scenario, efforts may be spent in reducing both critical path and logic complexity, aiming to introduce efficient solutions. A more detailed analysis could be carried out around fast Compare-and-Select operators, including the possibility to approximate their results. Moreover, the use of high-radix solutions could be investigated

on larger architectural spaces, including for instance different scheduling policies.

As highlighted, the Concurrent-PMAP architecture is expected to represent a feasible candidate for high-throughput solutions. Therefore, a better comparison is desired between PMAP and XMAP based approaches, aiming to highlight advantages and disadvantages. Following this consideration, the analysis model could be extended in order to fully include architectures based on the XMAP model.

Finally, new algorithmic and architectural solutions are expected to be introduced in the next future, aiming to satisfy the high-throughput demand. New proposed approaches could be discriminated following the proposed method, saving time on their full architectural description.

Appendix A

Recursive Trellis Exploration

When exploring high-radix orders, it is useful, especially for hardware description purposes, to gather informations on the operations required to be performed by the logic units. Moreover, it is interesting to observe the full set of paths considered by the decoding algorithm. Following these purposes, a Trellis-Diagram exploration approach is proposed.

As a first aspect, it is necessary to model the Trellis-Diagram state transitions in a generic Trellis-section. In this work, the reference LTE/UMTS standard code has been considered, however the approach can be also extended to other convolutional codes. As illustrated in figure A.1, a proper function, called **next_state()**, is defined in order to model the state transitions. Considering the Trellis-Diagram, the function requires the current state and the codeword bits to select the proper destination state, which is returned as a result. For instance, **next_state(0, 01)** will return a destination state equal to **5**.

An additional function, named **code_bits()**, is required to return the available codeword bits, given a generic present state. For instance, given the state **0**, two possible codewords are returned: **00** and **11**.

By employing the presented functions, it is possible to select a starting state and recursively explore all the possible paths deriving from it. The key idea is to first call **code_bits()** to gather informations on the possible codeword bits, and then **next_state()** to obtain the successive states in the Trellis-Diagram. This sequence of operations can be recursively repeated on the new obtained states, considering them as starting states.

A *stop condition* is required in order to block the recursion. Therefore, the recursive exploration should be aware of the amount of Trellis-sections to be explored, which is bounded by radix-order. For instance, a radix-4 exploration is requiring only two consecutive Trellis-sections to be analyzed.

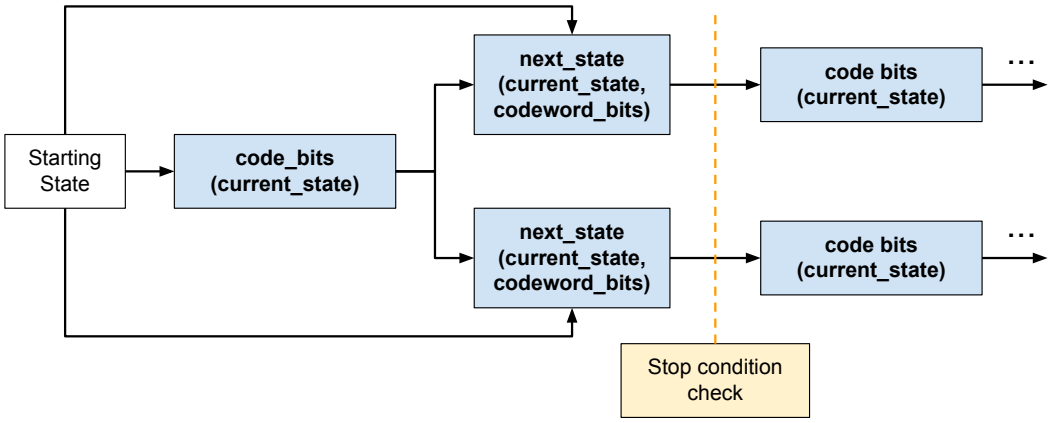

Figure A.1: Trellis-Diagram recursive exploration algorithm - block scheme.

The presented approach is meant to be repeated for each possible starting state, obtaining the full list of paths in the Trellis-Diagram. Each path is described by a sequence of codeword bits. As a first important information, it is possible to analyze if redundant paths, sharing the same codeword bits, are found in the analysis. Considering the reference code, redundant paths are found for both radix-2 and radix-4 approaches.

Moreover, it is possible to collect useful data in order to simplify the implementation of the three fundamental logic units included in a SISO-decoder. Considering the Branch Metric Unit (BMU), it is desired to compute all the combined branch metrics, introducing the minimum amount of additions. During the Trellis-Diagram exploration, it is possible to rely on a data structure storing the partially computed metrics (sum of radix-2 branches). When a combined branch is required to be computed, a new addition is introduced only if the necessary result is not found inside the mentioned data structure.

Regarding the Path Metric Unit (PMU), it is useful to group paths that are sharing the same starting or destination states, depending on the direction of the propagation. For instance, if the Forward recursion is performed, 8 set of paths sharing the same destination states can be identified.

Following the Soft Output Unit (SOU) model employed in this work, it is useful to identify several groups of paths sharing the same systematic sequences.

If required, other useful informations can be collected. Moreover, this approach can be also employed to compare Trellis-Diagrams belonging to different convolutional codes. In this work, the logic units description is relying on data collected from the Trellis-exploration, which saved a significant amount of time on the hardware description.

Appendix B

Fast CS Operators

As presented in Chapter 8, critical path reduction is a fundamental improvement necessary for the Path Metric Unit (PMU). In this context, the main logic operator is the Compare-and-Select (CS2) unit, capable to discriminate the maximum value among two input operands. This appendix is covering two fundamental aspects: first, a possible critical path reduction for the CS2 operator is presented, relying on look-ahead logic. Then, fast Compare-and-Select operators working with an arbitrary number of input operands are discussed.

B.1 Look-Ahead Logic CS2

As depicted in figure 8.3, a basic CS2 implementation is including a subtractor, capable to generate the logic signal employed by the selection multiplexer. It is remarkable how the subtraction result is not fully required for the correct functionality of the unit. Indeed, just the Most-Significant-Bit (MSB) is employed to drive the multiplexer.

Starting from this consideration, the subtractor architecture can be directly simplified, in order to avoid the computation of the unnecessary result bits. More specifically, relying on look-ahead logic, a critical path reduction is available with a small impact on the area.

Given two N-bits operands a and b, considering a generic bit i in the sequence, it is possible to estimate in parallel the *generate* and *propagate* signals g_i and p_i, as reported below. This first computation is illustrated in figure B.1, considering operands expressed on 11 bits, typical bitwidth for state-metrics.

$$g_i = a_i \cdot b_i$$

$$p_i = a_i \oplus b_i$$

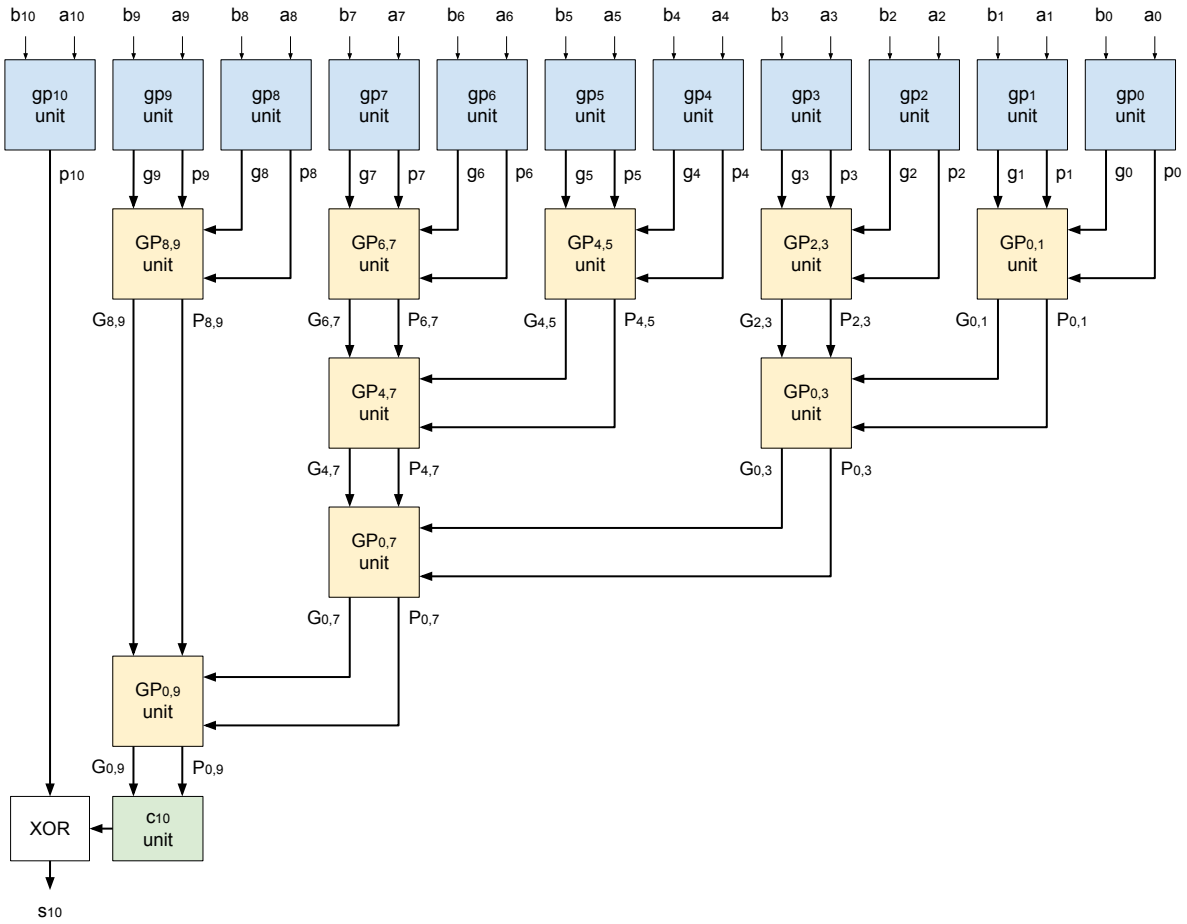

Figure B.1: Look-Ahead based CS2.

A generic sum bit s_i can be computed starting from the propagate bit p_i and the input carry c_i, as shown in the equation below. Typically, all the input carry values for each stage are required to be correctly computed to obtain the complete result.

$$s_i = p_i \oplus c_i$$

However, in this case, just the MSB is necessary, therefore it is enough to compute the logic value of the input carry for the last stage, c_{N-1}. A fast computation can be achieved relying on *Generate-Propagate-Block* units, organized in a tree-like structure as shown in B.1. The generated signals $G_{x,y}$ and $P_{x,y}$ are embedding information about the carry generation/propagation between the sum stages x and y. In other words, they are extending the concepts expressed by g_i and p_i over multiple bits. Moreover, as visible, those signals can be combined in order to extend their application ranges. The operations performed by a GP-Unit are reported.

$$G_{i,k} = G_{j+1,k} + P_{j+1,k} \cdot G_{i,j}$$
$$P_{i,k} = P_{i,j} \cdot P_{j+1,k}$$

Once $G_{0,N-2}$ and $P_{0,N-2}$ have been computed, it is possible to estimate c_{N-1}, knowing the logic value of the first input carry c_0.

$$c_{N-1} = G_{0,N-2} + P_{0,N-2} \cdot c_0$$

Since a subtraction is performed, c_0 is set equal to 1, in order to correctly complement the second operand. Therefore, the equation can be further simplified. Finally, a XOR gate is included to produce the output logic result.

As visible, most of the included operators are working in parallel layers, reducing the propagation delay. More specifically, since the GP-Blocks are following a tree-like disposition, the critical path is expected to logarithmically depend on the number of bits N. This advantage is introduced at a cost of a higher complexity in the carry propagation network, especially if compared to a ripple-carry based approach. However, part of the logic included in a complete look-ahead subtractor is neglected, since it is not necessary to fully compute the result.

Synthesis operations have been performed on two CS2 architectures, employing the standard subtractor approach and the look-ahead implementation. The bitwidth has been fixed to 11 bits and the 65 nm technology has been employed. Area and critical path comparisons have been extracted and reported in table B.1.

CS2 Model	**Area [um^2]**	**Critical Path [ns]**
Subtractor-Based	93.24	1.36
Look-Ahead-Logic	99.36	0.97

Table B.1: Area and critical path comparisons between CS2 models based on subtractors and look-ahead logic.

As visible, a 28% reduction factor is found on the critical path, which is compensated by a 7% area increment. Therefore, the Look-Ahead solution is introducing a better area efficiency, if compared to the classic subtractor implementation.

B.2 Multiple Inputs CS

Considering radix-orders higher than 2, it is fundamental to work on the delay introduced by Compare-and-Select operators working with more than two operands, aiming to reduce the Path Metric Unit (PMU) critical path. As presented in Chapter 7, a tree-like

structure including CS2 operators is discarded, considering this purpose. A better option is proposed in [24], in which a CS4-fast architecture, depicted in figure 8.4, is presented.

In this work, the same approach is extended, aiming to introduce fast Compare-and-Select operators working with a generic amount of N input operands. The first step consists in fixing the amount of parallel comparisons to be performed employing CS2 units. Observing the CS4-fast implementation, a total of 6 CS2 units are required. As visible, all the possible couple of operands are required to be compared. A possible generic equation to derive the amount of CS2 comparisons as function of N is indicated below.

$$\text{CS2 number} = \sum_{i=1}^{N-1} i$$

The next step consists in specifying the logic meaning of the comparisons results. An example, considering the CS4-fast architecture, is shown in table B.2.

It is now necessary to establish the content of the LUT. The algorithmic steps shown in figure B.2 are proposed: starting from the logic results table, it is possible to define the function **comp_res()**, capable to return the array of logic results, given a set of input operands. In other words, it is emulating the parallel CS2 operators. Starting from the same set, another defined function, **max()**, is in charge of extracting the maximum value, returning the correct selection signal for the multiplexer.

CS2 ID	**Input A**	**Input B**	**Logic Result**
0	Operand 0	Operand 1	Op. 0 > Op. 1
1	Operand 0	Operand 2	Op. 0 > Op. 2
2	Operand 0	Operand 3	Op. 0 > Op. 3
3	Operand 1	Operand 2	Op. 1 > Op. 2
4	Operand 1	Operand 3	Op. 1 > Op. 3
5	Operand 2	Operand 3	Op. 2 > Op. 3

Table B.2: Logic results from the set of 6 comparisons included in a CS4-fast operator.

The proposed steps are meant to be applied to all the possible combinations of input sets. The latter can be obtained considering all the permutations of the N-elements array [0, 1, 2 ... N-1]. Consequently, the number of input sets to be tested is N!, which is also representing the number of entrances for the LUT. For instance, considering a CS4-fast operator, 4! = 24 entrances are required.

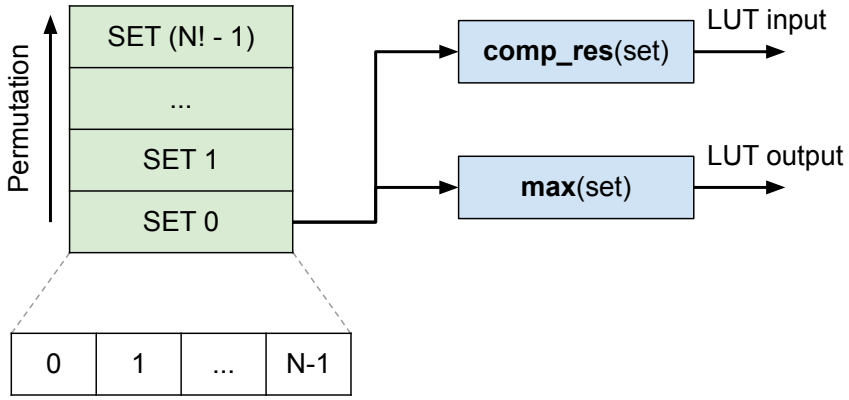

Figure B.2: Proposed steps to compute the LUT content - block scheme.

However, when increasing the amount of input operands, it is not feasible to describe in hardware a LUT to drive the multiplexer. The main reason is the number of required entrances, which is dramatically increasing with N. For instance, a CS8-fast operator would require 8! = 40320 entrances. Therefore, a different approach is proposed, aiming to implement a proper logic function necessary to drive the multiplexer.

Given a generic operand, N-1 logic results from the comparison table are associated to it. For instance, considering the *Operand 2* in table B.2, the comparisons 1, 3 and 5 are including it. Starting from this set of comparisons results, it is possible to derive a logic function stating if the operand under analysis is satisfying the *maximum condition*. The implementation is requiring a set of **AND** operations among the logic results of the involved comparisons. Moreover, if the operand under analysis in connected at the input B, a **NOT** logic operator is required before the correspondent logic result, in order to express it correctly. Proceeding with the *Operand 2* example, the *maximum* logic function is expressed below, indicating with CS_n the logic results of the compare and select operations.

$$\text{Maximum(Op. 2)} = \mathbf{NOT}(CS_1)\ \mathbf{AND}\ \mathbf{NOT}(CS_3)\ \mathbf{AND}\ CS_5$$

The *maximum* logic function can be derived for each operand in the set. Assuming that the i^{th} operand is the maximum one, the output of the i^{th} *maximum* logic function will assume a logic value equal to 1. At the same time, all the other *maximum* functions in the set will output a logic value equal to 0. Therefore, an N-input encoder is capable to directly generate the selection signal necessary to drive the multiplexer. A block scheme of the presented method is depicted in figure B.3.

The CS8-fast operator employed in this work has been developed following this proposed approach. Moreover, the correctness of the provided results has been verified through a set of testbenches on the architecture.

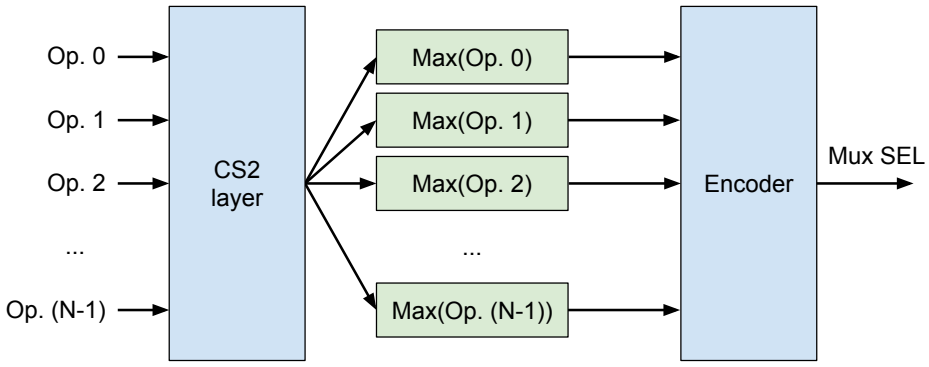

Figure B.3: CSN-fast, proposed implementation block scheme.

It is interesting to compare area and critical path of CS4-fast operators implemented either with a LUT or following the proposed method. Therefore, the synthesis results have been collected and reported in table B.3, considering the 65 nm technology and data expressed on 11 bits.

Approach	**Area [um^2]**	**Critical Path [ns]**
LUT-based	432.36	1.88
Encoder-based	425.16	1.87

Table B.3: Area and critical path comparisons between the two available approaches to realize a CS4-fast operator.

As visible, the post-synthesis results are presenting a strong similarity. Consequently, the two approaches can be employed equivalently.

Appendix C

Radix and Scheduling Aware Memory-Mapping

As introduced in Chapter 8, the authors in [28] proposed a generic approach to generate a conflict-free memory mapping, considering any permutation law. As mentioned, the approach is meant to be applied on a PMAP-based architecture, including N Kp-bits sub-frames. The access policy for the N memory banks can be derived starting from a defined *mapping matrix*, which is requested to satisfy conflict-free properties when accessed in natural order (columns) or interleaved order (tiles). Hence, columns and tiles are representing groups of simultaneously accessed memory banks.

The article is assuming to work with a radix-2 approach. Moreover, the usage of a specific scheduling policy is not specified. The algorithm for the mapping matrix derivation is described, following two fundamental steps. The first step is guaranteeing the derivation of a partially filled mapping-matrix. The second step is aiming to complete the void matrix cells, guaranteeing to respect the conflict-free property on columns and tiles.

Before proceeding, it is useful to graphically analyze how the mapping matrix is accessed in time, both during natural and interleaved processing. Following the same example reported in figure 8.9, the parallel accesses in time are highlighted in figure C.1. As visible, a possible way to extract the conflict-free matrix regions is to analyze, during the processing, where accesses are performed.

When radix orders higher than 2 are employed, multiple adjacent columns are expected to be accessed at the same time. Larger tiles are also expected to be generated, as a consequence of the higher amount of required memory banks. The same conflict-free properties are required to be satisfied by larger groups of cells in the matrix. A radix-4 matrix organization example is shown in figure C.2. For the sake of simplicity, only the natural processing is represented. Indeed, the same concepts can be applied to the interleaved data sequence, in order to identify the tile-set. Moreover, all the numeric

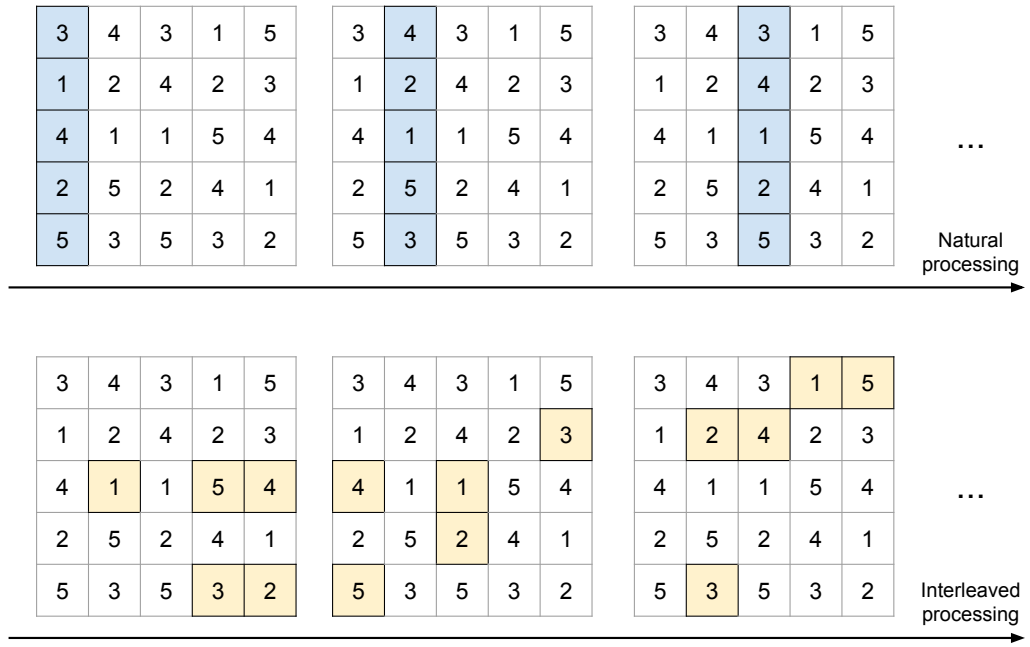

Figure C.1: Natural and interleaved parallel accesses in time - Radix 2, no scheduling policy.

values included in cells are to be intended as a mapping example, since they are not related to any permutation law.

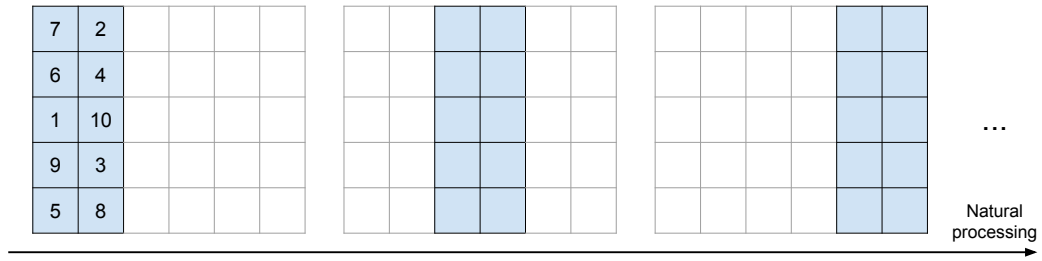

Figure C.2: Natural parallel accesses in time - Radix 4, no scheduling policy.

A specific scheduling policy can be also considered while mapping the memory accesses. Also in this case, it is necessary to highlight where the parallel accesses are located in time on the mapping matrix. For instance, the Forward-Backward scheduling is requiring three parallel accesses to simultaneously perform the forward/backward propagations and store the soft-output values. An example is shown in figure C.3, highlighting the different windows included in a sub-frame. A radix-2 approach is assumed in this case.

As visible, it is fundamental to highlight in which time instants the parallel accesses are requested by the scheduling algorithm. Therefore, it is fundamental to take into account

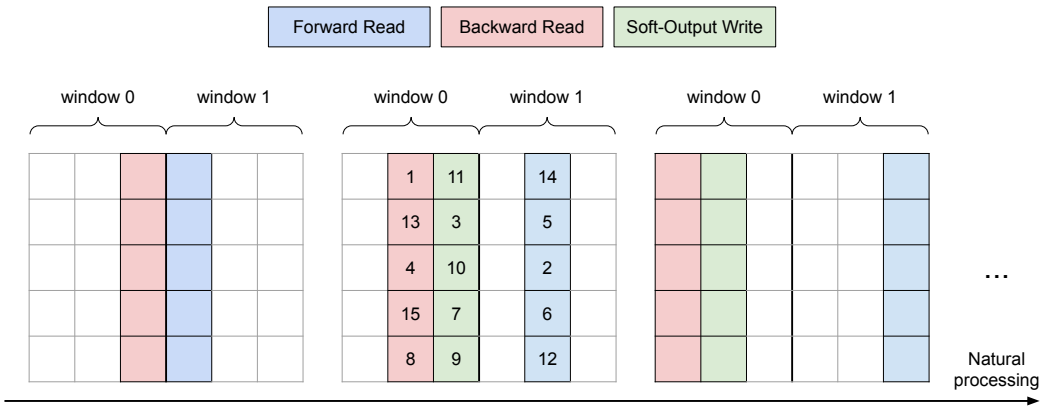

Figure C.3: Natural parallel accesses in time - Radix 2, Forward-Backward scheduling.

the eventual presence of pipeline stages, which are capable to further separate in time the required parallel accesses.

Moreover, it is possible to combine the requirements of a high-radix approach and a scheduling policy. Considering the natural processing, the scheduling algorithm is generating a set of simultaneously accessed columns, typically separated. On the other hand, the radix order is potentially increasing the amount of adjacent columns to be considered. For instance, a radix-4 approach is shown in figure C.4, considering a Forward-Backward scheduling.

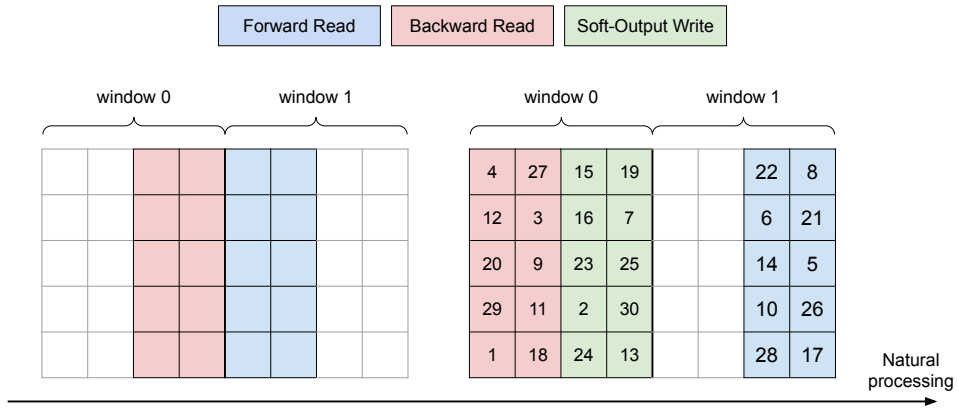

Figure C.4: Natural parallel accesses in time - Radix 4, Forward-Backward scheduling.

It is noticeable how the general steps for the mapping matrix derivation, presented in the reference article, can be still applied. As a difference, the conflict-free requirements to be satisfied are adapted to the radix order and the scheduling policy.

Bibliography

[1] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1". In: *Proceedings of ICC '93 - IEEE International Conference on Communications*. Vol. 2. 1993, 1064–1070 vol.2. DOI: 10.1109/ICC.1993.397441.

[3] Claus Kestel, Matthias Herrmann, and Norbert When. "When Channel Coding Hits the Implementation Wall". In: *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*. 2018, pp. 1–6. DOI: 10.1109/ISTC.2018.8625324.

[4] Vinh Hoang Son Le. "Design of Next-Generation Tbps Turbo Codes". PhD thesis. 2020.

[5] *Universal Mobile Telecommunications System (UMTS); Multiplexing and channel coding (FDD) (3GPP TS 25.212 version 16.0.0 Release 16)*. 2020. URL: https://www.etsi.org/deliver/etsi_ts/125200_125299/125212/16.00.00_60/ts_125212v160000p.pdf.

[6] *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (3GPP TS 36.212 version 13.10.0 Release 13)*. 2020. URL: https://www.etsi.org/deliver/etsi_ts/136200_136299/136212/13.10.00_60/ts_136212v131000p.pdf.

[7] L. Bahl et al. "Optimal decoding of linear codes for minimizing symbol error rate (Corresp.)" In: *IEEE Transactions on Information Theory* 20.2 (1974), pp. 284–287. DOI: 10.1109/TIT.1974.1055186.

[8] P. Robertson, E. Villebrun, and P. Hoeher. "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain". In: *Proceedings IEEE International Conference on Communications ICC '95*. Vol. 2. 1995, 1009–1013 vol.2. DOI: 10.1109/ICC.1995.524253.

[9] J. Vogt and Adolf Finger. "Improving the max-Log-MAP turbo decoder". In: *Electronics Letters* 36 (Dec. 2000), pp. 1937–1939. DOI: 10.1049/el:20001357.

[10] Emmanuel Boutillon, Catherine Douillard, and Guido Montorsi. "Iterative Decoding of Concatenated Convolutional Codes: Implementation Issues". In: *Proceedings of the IEEE* 95.6 (2007), pp. 1201–1227. DOI: 10.1109/JPROC.2007.895202.

[11] *B5G Wireless Tb/s FEC KPI Requirement and Technology Gap Analysis*. 2018. URL: https://epic-h2020.eu/downloads/EPIC-D1.2-B5G-Wireless-Tbs-FEC-KPI-Requirement-and-Technology-Gap-Analysis-PU-M22.pdf.

[12] E. Boutillon, W.J. Gross, and P.G. Gulak. "VLSI architectures for the MAP algorithm". In: *IEEE Transactions on Communications* 51.2 (2003), pp. 175–185. DOI: 10.1109/TCOMM.2003.809247.

[13] Stefan Weithoffer et al. "Advanced Hardware Architectures for Turbo Code Decoding Beyond 100 Gb/s". In: *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. 2020, pp. 1–6. DOI: 10.1109/WCNC45663.2020.9120779.

[14] A.P. Hekstra. "An alternative to metric rescaling in Viterbi decoders". In: *IEEE Transactions on Communications* 37.11 (1989), pp. 1220–1222. DOI: 10.1109/26.46516.

[15] M.J. Thul et al. "A scalable system architecture for high-throughput turbo-decoders". In: *IEEE Workshop on Signal Processing Systems*. 2002, pp. 152–158. DOI: 10.1109/SIPS.2002.1049701.

[16] A. Worm, H. Lamm, and N. Wehn. "A high-speed MAP architecture with optimized memory size and power consumption". In: *2000 IEEE Workshop on SiGNAL PROCESSING SYSTEMS. SiPS 2000. Design and Implementation (Cat. No.00TH8528)*. 2000, pp. 265–274. DOI: 10.1109/SIPS.2000.886725.

[17] Robert G. Maunder. "A Fully-Parallel Turbo Decoding Algorithm". In: *IEEE Transactions on Communications* 63.8 (2015), pp. 2762–2775. DOI: 10.1109/TCOMM.2015.2450208.

[18] Stefan Weithoffer et al. "25 Years of Turbo Codes: From Mb/s to beyond 100 Gb/s". In: *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*. 2018, pp. 1–6. DOI: 10.1109/ISTC.2018.8625377.

[19] Christoph Roth et al. "Efficient Parallel Turbo-Decoding for High-Throughput Wireless Systems". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.6 (2014), pp. 1824–1835. DOI: 10.1109/TCSI.2013.2290831.

[20] Stefan Weithoffer, Frederic Pohl, and Norbert Wehn. "On the applicability of trellis compression to Turbo-Code decoder hardware architectures". In: *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*. 2016, pp. 61–65. DOI: 10.1109/ISTC.2016.7593077.

[21] An Li et al. "VLSI Implementation of Fully Parallel LTE Turbo Decoders". In: *IEEE Access* 4 (2016), pp. 323–346. DOI: 10.1109/ACCESS.2016.2515719.

[22] Stefan Weithoffer et al. "Advanced Hardware Architectures for Turbo Code Decoding Beyond 100 Gb/s". In: *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. 2020, pp. 1–6. DOI: 10.1109/WCNC45663.2020.9120779.

[23] Stefan Weithoffer et al. "Fully Pipelined Iteration Unrolled Decoders the Road to TB/S Turbo Decoding". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 5115–5119. DOI: 10.1109/ICASSP40776.2020.9053453.

[24] Christoph Studer et al. "Design and Implementation of a Parallel Turbo-Decoder ASIC for 3GPP-LTE". In: *IEEE Journal of Solid-State Circuits* 46.1 (2011), pp. 8–17. DOI: 10.1109/JSSC.2010.2075390.

[25] Oscar Sánchez et al. "High speed low complexity radix-16 Max-Log-MAP SISO decoder". In: *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*. 2012, pp. 400–403. DOI: 10.1109/ICECS.2012.6463718.

[26] Oscar David Sanchez Gonzalez. "Towards higher speed decoding of convolutional turbocodes". PhD thesis. Télécom Bretagne, Université de Bretagne-Sud.

[27] Ajit Nimbalker et al. "Contention-Free Interleavers for High-Throughput Turbo Decoding". In: *IEEE Transactions on Communications* 56.8 (2008), pp. 1258–1267. DOI: 10.1109/TCOMM.2008.050502.

[28] A. Tarable and S. Benedetto. "Mapping interleaving laws to parallel turbo decoder architectures". In: *IEEE Communications Letters* 8.3 (2004), pp. 162–164. DOI: 10.1109/LCOMM.2004.823364.

[29] Aaron Stillmaker and Bevan Baas. "Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm". In: *Integration* 58 (2017), pp. 74–81. ISSN: 0167-9260. DOI: https://doi.org/10.1016/j.vlsi.2017.02.002.